Primjena umjetne inteligencije u kartaškoj igri

Blaić, Krešimir

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 682

PRIMJENA UMJETNE INTELIGENCIJE U KARTAŠKOJ IGRI

Krešimir Blaić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 682

PRIMJENA UMJETNE INTELIGENCIJE U KARTAŠKOJ IGRI

Krešimir Blaić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 682

Pristupnik:	**Krešimir Blaić (0036519428)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	izv. prof. dr. sc. Marko Čupić

Zadatak: **Primjena umjetne inteligencije u kartaškoj igri**

Opis zadatka:

Igranje kartaških igara ljudima je uvijek bio jedan od načina zabave. Postoji mnoštvo različitih vrsta kartaških igara - s potpunom informacijom, s djelomičnom informacijom, s elementima slučajnosti, za različit broj igrača i slično. U situaciji kada osoba nema partnera za igru, zanimljivo bi bilo raspolagati računalnim igračem protiv kojeg je moguće igrati. U okviru ovog diplomskog rada potrebno je proučiti i opisati različite vrste kartaških igara prema njihovim karakteristima. Potom je za odabranu kartašku igru potrebno napraviti prototipnu računalnu implementaciju igrača koji na zadovoljavajućoj razini može igrati igru protiv ljudskog igrača. Potrebno je opisati razvijenu implementaciju i napraviti njezino vrednovanje.

Rok za predaju rada: 28. lipnja 2024.

Zahvaljujem mentoru izv. prof. dr. sc. Marku Čupiću na mentorstvu i savjetima.

Sadržaj

Popis slika

1. Uvod

Igranje kartaških igara ljudima je uvijek bio jedan od načina zabave. Kartaška igra je svaka igra koja koristi igraće karte kao primarni uređaj za igranje. Postoji mnoštvo zanimljivih kartaških igara pa je lako razumjeti zašto je kartanje postalo vrlo omiljeno u društvu.

Općenito se igre mogu podijeliti prema nekoliko kriterija. Jedan od njih bi bio *broj igrača* budući da neke igre imaju fiksan broj igrača, dok kod nekih igara broj može varirati i to najčešće u rasponu od 2 do 4 igrača. Ovisno o dostupnosti relevantnih informacija, postoje igre s *potpunom informacijom* (primjerice Šah ili Križić-kružić), gdje su svakom igraču poznate sve informacije, te igre s *nepotpunom informacijom* u kojima se tijekom igre otkrivaju dodatne informacije. Rijedak je slučaj kartaške igre s potpunom informacijom pa su kartaške igre uglavnom igre s nepotpunom informacijom zato što primjerice igrač želi pobijediti u rundi i zna svoje karte, ali ne zna protivnikove karte. Ovisno o utjecaju slučajnosti, igre se dijele na *determinističke* i *stohastičke*. U determinističkim igrama poput Šaha, Poveži 4 ili Križić-kružić nema slučajnosti, dok u stohastičkim igrama postoji utjecaj slučajnosti. U kartaškim igrama je stohastičnost prisutna kod izvlačenja karata iz promiješanog špila, dok bi na primjer kod društvenih igara stohastički element bio bacanje kockice. Osim navedenih kriterija, kartaške igre moguće je podijeliti po načinu igre. Igre nadigravanja su igre u kojima postoje runde i koje završavaju kad igrači odigraju sve svoje karte. Primjeri takvih igara su Briškula i Bela. Igre razmjene karata su velika skupina igara u kojima igrači razmjenjuju karte s drugim igračima ili kartama koje su “na stolu". Primjeri takvih igara popularnih kod nas su Crni Petar i Kocka - igra u kojoj je potrebno sakupiti četiri karte iste jačine i signalizirati svom suigraču da ih držite u ruci bez da protivnici primijete. Poker i Blackjack primjeri su usporednih kartaških igara u kojima se uspoređuju vrijednosti ruku kako bi se odredio pobjednik i to su uglav-

nom kockarske igre. Igre raspoređivanja karata su igre u kojima je cilj složiti karte po nekom kriteriju. Primjeri takvih igara su Pasijans i FreeCell.

Na ovim prostorima najpopularnije kartaške igre su Bela, Briškula i Trešeta. Sve te igre su istog tipa - stohastičke igre s nepotpunom informacijom. Cilj ovog rada je opisati i razviti sustav za kartašku igru Briškulu koji na zadovoljavajućoj razini može igrati igru protiv ljudskog igrača. Ovaj sustav je osmišljen za situacije kada osoba nema partnera za igru. U nastavku će biti opisan tijek razvoja programskog rješenja te korišteni algoritmi.

2. Kartaška igra Briškula

Briškula (tal. *Briscola*) je talijanska kartaška igra koja je rasprostranjena na hrvatskom priobalju i otocima. Najčešće se igra s tršćanskim kartama i u parovima, iako ju može igrati od 2 do 5 igrača. Postoje dvije verzije igre:

- obična briškula - u svakoj "ruci" svaki igrač baca jednu kartu
- dupla briškula - u svakoj "ruci" svaki igrač baca dvije karte

Obična briškula najčešće se igra u parovima (četiri igrača), a dupla briškula u dvoje.

2.1. Karte

U jednom špilu ima 40 karata koje su podijeljene u 4 boje i 10 jačina. Boje tršćanskih karata su:

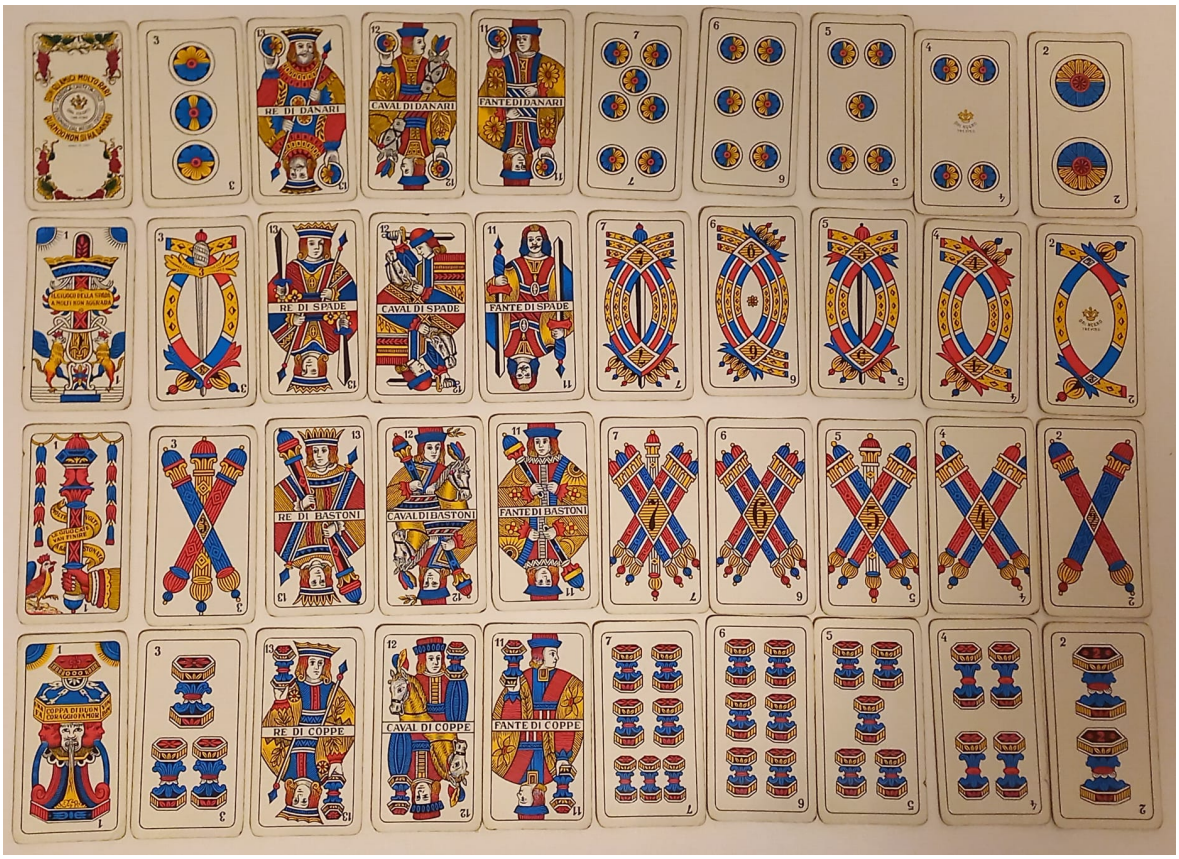
- dinari – karte sa simbolom starorimskog novca
- špade – karte sa simbolom mačeva
- baštoni – karte sa simbolom štapa
- kupe – karte sa simbolom kaleža vina

Karte poredane po jačini u briškuli, počevši od najjače su:

- as (1)
- trica (3)
- kralj (13)

- konj (12)
- fanat (11)
- sedmica (7)
- šestica (6)
- petica (5)
- četvorka (4)
- dvica (2)

gdje su u zagradama navedeni brojevi koji se nalaze na kartama. Karte su prikazane na slici ispod, poredane po bojama (od gore prema dolje po redoslijedu kojim su navedene) i po jačini (s lijeva na desno).



Slika 2.1. Karte za briškulu

2.2. Miješanje i dijeljenje karata

Karte se miješaju u smjeru kazaljke na satu što bi značilo da nakon što je odigrana jedna partija, špil karata za iduću partiju miješa prvi igrač s lijeve strane. Isto vrijedi i za dijeljenje karata pa će tako prvi igrač s lijeve strane prilikom dijeljenja prvi dobiti karte, a igrač koji je miješao zadnji dobiva karte (u slučaju dva igrača igrač koji miješa prvo dijeli karte protivniku pa onda sebi). Nakon miješanja karata, a neposredno prije dijeljenja igrač koji je miješao prijašnju partiju (ili prvi igrač s desne strane ako se radi o prvoj partiji) ili prediže karte ili udara rukom po njima što bi značilo da "tuče".

Ako igrač prediže karte, u običnoj briškuli se svakom igraču naizmjenično dijeli jedna karta sve dok igrači nemaju tri karte u ruci. U duploj briškuli se u tom slučaju igračima naizmjenično dijele dvije karte sve dok igrači u ruci nemaju četiri karte. Važno je napomenuti da kada se podijeli pola karata, jedna karta se okreće licem prema gore i stavlja na stol. Na tu kartu se postavlja ostatak špila kad se podijele sve početne karte i ona predstavlja adut za tu partiju.

Ako igrač "tuče", u običnoj briškuli se odmah podijele tri karte svakom igraču dok se u duploj briškuli dijele četiri karte svakom igraču. U ovom slučaju se karta koja predstavlja adut za tu partiju okreće nakon što su svim igračima podijeljene sve početne karte.

2.3. Tijek igre

Jedna odigrana igra se naziva *partija* i ona završava kada se odigraju sve karte iz špila. Partija se sastoji od "ruku" u kojima igrači bacaju karte ovisno o verziji igre. U običnoj briškuli svaki igrač u jednoj "ruci" baca jednu kartu, a u duploj igrači naizmjenično bacaju po jednu kartu sve dok svaki igrač ne baci točno dvije karte. Nakon što su u jednoj "ruci" bačene sve karte, određuje se pobjednik te "ruke" po pravilima koji su opisani u idućem poglavlju. Pobjednik prvi vuče zamjensku kartu iz špila pa nakon njega ostali igrači redom u smjeru kazaljke na satu. Broj karata koji svaki igrač vuče iz špila također ovisi o verziji igre pa će tako svaki igrač u običnoj briškuli vući jednu kartu, a u duploj briškuli dvije karte, ali naizmjenično.

2.4. Pravila igre

Da bi igrač pobijedio mora skupiti barem 60 bodova ili "punata" od mogućih 120 bodova koliko iznosi zbroj svih karata koje donose bodove. Brojevi bodova karata su idući:

- as - 11 bodova
- trica - 10 bodova
- kralj - 4 boda
- konj - 3 boda
- fanat - 2 boda
- sedmica, šestica, petica, četvorka, dvica - 0 bodova

Igrač koji je skupio barem 61 bod je sigurno pobjednik partije. Partija može završiti i neriješeno na način da igrači imaju svaki po 60 bodova. Igrači skupljaju bodove tako da osvajaju "ruke". Logično bi bilo pretpostaviti onda da što više "ruku" igrač osvoji da je veća šansa za pobjedu, no to često nije slučaj zbog karata koje vrijede 0 bodova i zbog toga što se jako vrijedne karte (as i trica) često čuvaju za završnicu igre, odnosno predzadnju i zadnju "ruku" pa te dvije "ruke" znaju presuditi cijelu partiju jer nose jako puno bodova.

Osvojiti "ruku" znači baciti najjaču kartu u toj "ruci". Rečeno je već kako se pri dijeljenju jedna karta okrene licem prema gore i ona predstavlja adut, odnosno druge karte se mogu "ubiti" bojom aduta. "Ubijanje" u kontekstu ove igre znači bacanje jače karte od karte koja je trenutno najjača. Prva bačena karta u "ruci" je najjača i ona se može ubiti samo:

- jačom kartom iste boje
- kartom boje aduta

pri čemu je jačina karata već objašnjena u poglavlju o kartama. Ako je bačena neka karta aduta, ona se može ubiti samo jačom kartom te boje. Nakon što su bačene sve karte u "ruci" (broj karata ovisi o verziji briškule i broju igrača), određuje se koje je karta najjača te igrač koji je bacio tu kartu nosi sve bodove i prvi vuče nove karte iz špila nakon čega

prvi baca kartu u novoj "ruci". Nakon što su odigrane sve karte, igrači računaju broj bodova koji su ostvarili te je završena jedna partija i pobjednik je igrač koji ima najviše bodova.

2.5. Rječnik za briškulu

U nastavku su dani neki specifični izrazi koji se koriste u ovoj kartaškoj igri:

- ruka – jedno bacanje karata u partiji
- zog – boja karata
- punat – bod
- karik, karig – karta velike vrijednosti (as ili trica)
- lišine – karte čija je vrijednost bodova 0 (7, 6, 5, 4, 2)
- peškanje – uzimanje karte iz špila
- štrocanje – bacanje karika kad je ruka osvojena
- motiranje – signaliziranje suigraču

2.5.1. Motiranje

Motiranje se koristi kada se briškula igra u parovima. Motiranjem igrač svom suigraču govori koji kartu od aduta ima u ruci. Motiraju se karte koje imaju neku bodovnu vrijednost što bi značilo da se "lišine" ne motiraju. Tajni znakovi ili "moti" su idući:

- as - nakratko napućiti usne
- trica – namignuti
- kralj - podizanje obrva
- konj – podizanje jednog ramena
- fanat - pokazivanje vrha jezika

3. Umjetne neuronske mreže

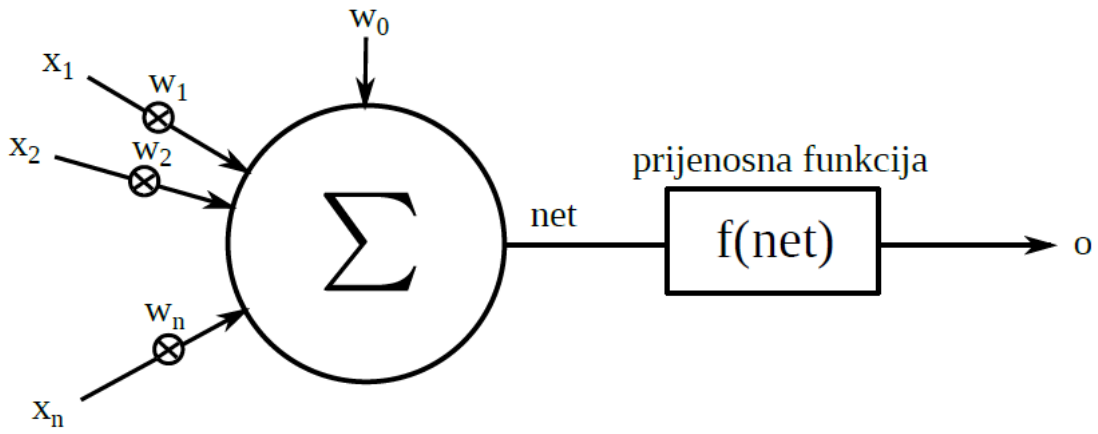
Umjetne neuronske mreže ili samo neuronske mreže su složeni računalni modeli inspirirani biološkim neuronima koji oponašaju rad ljudskog mozga i koji se koriste za obradu informacija. Jedna od glavnih karakteristika neuronskih mreža je njihova sposobnost učenja iz podataka, što im omogućava da prepoznaju obrasce, prilagode svoje parametre kroz iterativne procese te poboljšaju točnost svojih predikcija [3]. Osim za probleme klasifikacije i regresije, ove mreže koriste se u različitim područjima poput računalnog vida, obrade prirodnog jezika, prepoznavanja uzoraka i još mnogo toga. Prije detaljnog opisivanja takvih modela važno je definirati što je to *umjetni neuron* - osnovna gradivna jedinica umjetnih neuronskih mreža.

3.1. Umjetni neuron

Neuron imitira biološki neuron kojeg čine dendriti, tijelo neurona i aksoni pa se tako i umjetni neuron sastoji od nekoliko ključnih komponenti: *ulaznih vrijednosti*, odgovarajućih *težina*, *pomaka* te *izlazne vrijednosti*.

3.1.1. Struktura neurona

Po uzoru na dendrite biološkog neurona, umjetni neuron ima ulazne vrijednosti pomoću kojih prima informacije od prethodnih neurona ili od programa ako se neuron nalazi u ulaznom sloju. Svaki ulaz ima pripadajuću težinu koja određuje utjecaj tog ulaza na izlaz neurona. Težina se množi s ulaznom vrijednosti pa će pozitivne težine povećavati utjecaj ulaznih vrijednosti. Zbroj svih ulaznih vrijednosti pomnoženih s pripadajućim težinama, na koji se dodaje pomak (engl. *bias*), predstavlja akumuliranu vrijednost koja se naziva *net*. Kako bi se dobio konačan izlaz neurona, akumulirana vrijednost se propušta kroz *aktivacijsku funkciju*.



Slika 3.1. Umjetni neuron [1]

Na slici 3.1. prikazan je model umjetnog neurona, a u nastavku je dana formula za računanje akumulirane vrijednosti:

$$net = \sum_{i=1}^{n} x_i \cdot w_i + w_0$$

gdje x_i predstavlja ulazne vrijednosti, w_i težine, a w_0 pomak. Kada se izračuna net, izlaz neurona dobiva se primjenom prijenosne funkcije:

$$o = f(net)$$

osim u neuronima koji se nalaze u izlaznom sloju (ne primjenjuje se prijenosna funkcija).

3.1.2. Prijenosne funkcije

Prijenosne ili aktivacijske funkcije određuju kako se akumulirana vrijednost skalira na određeni interval. Najčešće korištene prijenosne funkcije su:

- funkcija identiteta

$$f(net) = net \tag{3.1}$$

- funkcija skoka

$$f(net) = step(net) = \begin{cases} 0 & \text{ako } net < 0 \\ 1 & \text{ako } net \geq 0 \end{cases} \tag{3.2}$$

- sigmoidalna funkcija

$$f(net) = step(net) = \frac{1}{1 + e^{-net}} \tag{3.3}$$

- tangens hiperbolni

$$f(net) = \tanh(net) = 2 * sigm(2 * net) - 1 \tag{3.4}$$

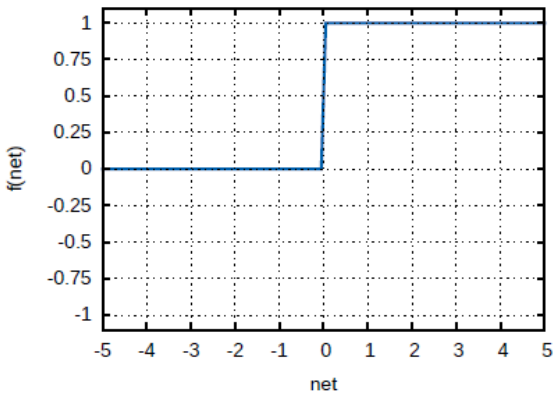
- zglobnica

$$f(net) = \max(0, net) \tag{3.5}$$

- propusna zglobnica

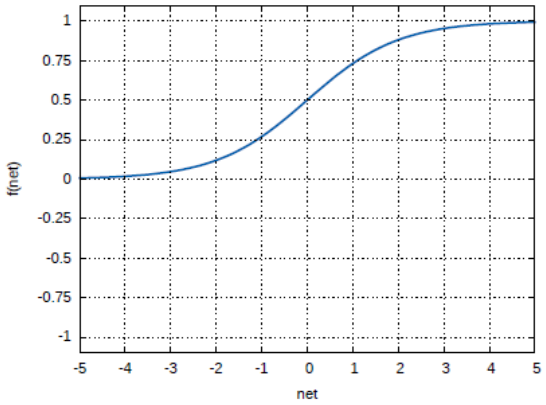
$$f(net) = \begin{cases} net & \text{ako } net > 0 \\ 0.01 * net & \text{ako } net \leq 0 \end{cases} \tag{3.6}$$

U nastavku je dan primjer funkcije skoka i sigmoidalne funkcije koje su možda naizgled slične, ali se ipak značajno razlikuju.



Slika 3.2. Funkcija skoka [2]

Funkcija skoka ima problematičnu karakteristiku za postupke učenja koji se oslanjaju na derivacije jer je njena derivacija iznosi 0, osim za točku $net = 0$. Sigmoidalna funkcija se može smatrati generalizacijom funkcije skoka, pri čemu se njena vrijednost postupno mijenja od 0 do 1 kao što je prikazano na slici 3.3. Ključna prednost sigmoidalne funk-



Slika 3.3. Sigmoidalna funkcija [2]

cije je njena derivabilnost, što omogućava primjenu algoritama za učenje temeljenih na gradijentnom spustu.

Nelinearne prijenosne funkcije su ključne za modeliranje složenih nelinearnih odnosa u podacima, omogućujući neuronskim mrežama da rješavaju probleme kao što su linearno-neodvojivi razredi (funkcija XOR) koji se ne mogu riješiti linearnim funkcijama.

3.2. Umjetna neuronska mreža

Sad kada je opisan neuron koji je osnovna gradivna jedinica, lako je razumjeti što su to neuronske mreže. *Umjetne neuronske mreže* (engl. *artificial neural network* ili skraćeno ANN) su složeni sustavi koji se sastoje od međusobno povezanih neurona koji su raspoređeni u više slojeva. Postoje tri osnovne vrste slojeva u umjetnim neuronskim mrežama:

- ulazni sloj
- skriveni sloj
- izlazni sloj

3.2.1. Ulazni sloj

Ulazni sloj je početni sloj mreže koji sadrži neurone zadužene za primanje ulaznih podataka. Ovi neuroni ne provode nikakvu dodatnu obradu podataka već jednostavno prenose podatke dalje u mrežu. Svaki neuron u ulaznom sloju odgovara jednoj komponenti ulaznog vektora podataka.

3.2.2. Skriveni slojevi

Skriveni slojevi su slojevi smješteni između ulaznog i izlaznog sloja. Ovi slojevi su ključni za sposobnost mreže da uči složene obrasce i odnose u podacima. Skriveni slojevi obrađuju podatke kroz različite kombinacije težina i prijenosnih funkcija te transformiraju ulazne podatke u oblik koji je pogodan za konačnu klasifikaciju ili regresiju. Važno je istaknuti da broj skrivenih slojeva i broj neurona u svakom sloju mogu značajno utjecati na performanse mreže.

3.2.3. Izlazni sloj

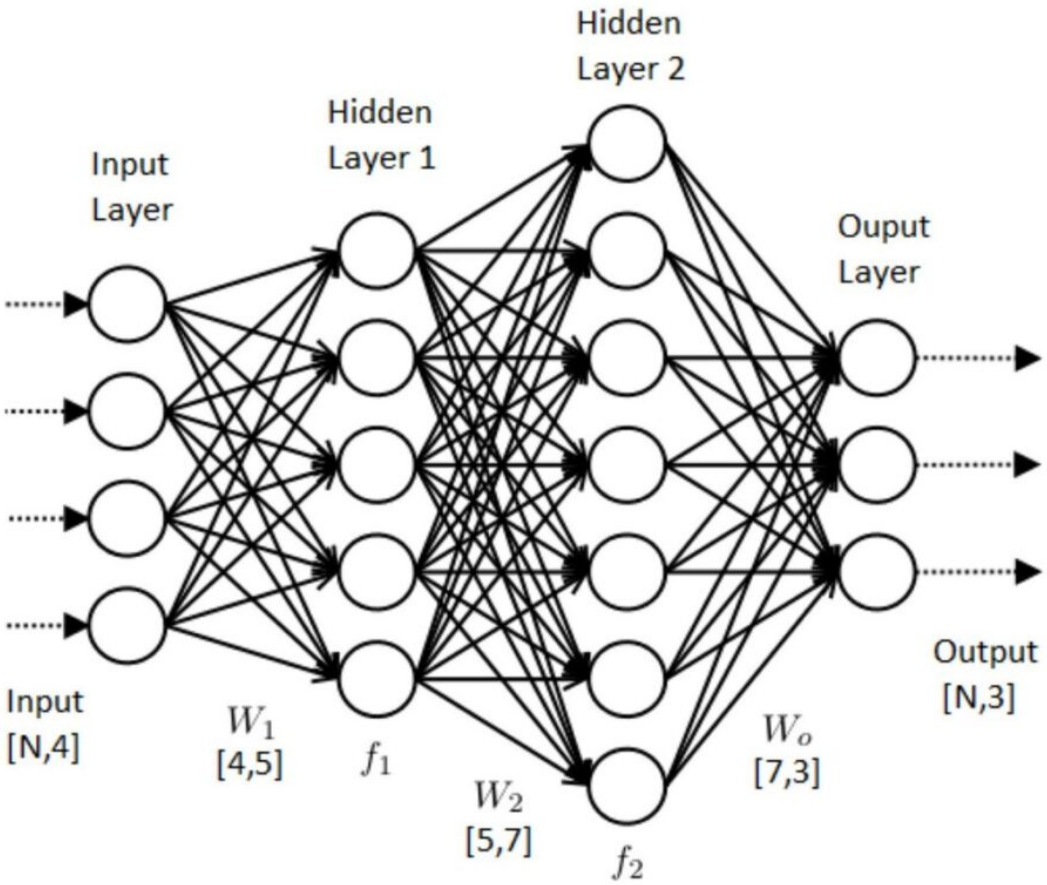
Izlazni sloj je završni sloj neuronske mreže. Neuroni u izlaznom sloju generiraju konačni izlaz mreže, koji može biti u obliku klasifikacije ili regresije, ovisno o specifičnom zadatku. Broj neurona u izlaznom sloju obično odgovara broju različitih klasa ili varijabli koje mreža treba predvidjeti. Izlazi neurona u ovom sloju mogu se dalje obraditi kako bi se dobio konačan rezultat, kao što je vjerojatnost pripadnosti određenoj klasi ili predviđena vrijednost kontinuirane varijable.

3.2.4. Vrste neuronskih mreža

Pojam arhitektura neuronske mreže odnosi se na to kako su neuroni međusobno povezani te od koliko se slojeva sastoji mreža. Na primjer, ako kažemo da je neuronska mreža 5 x 3 x 2 x 3, to bi značilo da mreža ima četiri sloja: jedan ulazni, dva skrivena i jedan izlazni sloj. U tom slučaju bi se ulazni sloj sastojao od pet neurona, prvi skriveni sloj od tri neurona, drugi skriveni sloj od dva neurona i u izlaznom sloju bi bila tri neurona. Ovisno o povezanosti neurona između slojeva, razlikujemo više vrsta neuronskih mreža.

Unaprijedne neuronske mreže (engl. *Feedforward Neural Networks* ili skraćeno FNN) predstavljaju najjednostavniji tip neuronskih mreža gdje podaci teku u jednom smjeru: od ulaznog sloja pa kroz jedan ili više skrivenih slojeva sve do izlaznog sloja. U ovim mrežama nema povratnih veza, što znači da informacije teku samo unaprijed, bez mogućnosti vraćanja ili ponovnog korištenja izlaznih podataka u prethodnim slojevima. FNN su pogodne za rješavanje problema klasifikacije i regresije gdje je odnos između ulaza i izlaza ne ovisi o vremenu. Zbog svoje jednostavnosti, FNN su često prvi izbor za mnoge osnovne zadatke strojnog učenja. Na slici 3.4. dan je primjer unaprijedne neuronske

mreže s dva skrivena sloja.



Slika 3.4. Unaprijedna neuronska mreža

Povratne neuronske mreže (engl. *Recurrent Neural Networks* ili skraćeno RNN) imaju strukturu koja omogućava povratne veze, što znači da podaci mogu ići u oba smjera. Ova karakteristika omogućava RNN-ovima da zapamte informacije iz prethodnih koraka i koriste ih za obradu trenutnih ulaznih podataka. Zbog ove sposobnosti, RNN-ovi su posebno učinkoviti za obradu sekvencijalnih podataka i vremenskih serija gdje trenutni izlazi ovise o prethodnim ulazima. RNN-ovi se koriste u aplikacijama poput prepoznavanja govora, obrade prirodnog jezika i predikcije vremenskih serija. Međutim, zbog svoje složenije strukture, RNN-ove je teže trenirati i skloni su problemima poput eksplodirajućih gradijenata.

Osim osnovnih FNN i RNN mreža, postoje i napredni oblici neuronskih mreža kao što su *konvolucijske neuronske mreže* (CNN) koje su specijalizirane za obradu slika, te *dugoročne kratkoročne memorijske mreže* (LSTM) koje su vrsta RNN-a dizajnirana za bolje rukovanje dugoročnim ovisnostima u podacima.

3.3. Učenje umjetne neuronske mreže

Kao što je već rečeno, neuronske mreže imaju sposobnost učenja iz podataka. Učenje se može odvijati na različite načine pa ovisno o informacijama koje su dostupne tijekom učenja razlikujemo nekoliko načina učenja neuronskih mreža.

3.3.1. Načini učenja

Nadzirano učenje (engl. *supervised learning*) je metoda u kojoj se neuronska mreža trenira pomoću skupa ulazno-izlaznih parova i za cilj ima naučiti mapirati ulazne podatke na ispravne izlazne vrijednosti. Svaki ulazni primjer dolazi s odgovarajućom izlaznom vrijednosti.

Nenadzirano učenje (engl. *unsupervised learning*) je metoda gdje mreža uči iz podataka koji nemaju pripadajuće izlazne vrijednosti. Mreža nema nikakvih dodatnih podataka osim samih uzoraka i pokušava pronaći pravilnosti u danim uzorcima kako bi se na primjer oni mogli grupirati na temelju sličnosti ili kako bi im se smanjila dimenzionalnost.

Podržano učenje je metoda u kojoj mreža uči putem interakcije s okolinom. Mreža prima povratne informacije u obliku nagrada ili kazni na temelju svojih akcija, i prilagođava svoje težine kako bi maksimizirala ukupnu nagradu [1].

3.3.2. Nadzirano učenje

Umjetne neuronske mreže (ANN) prolaze kroz dvije ključne faze:

1. Faza učenja (treniranja)

2. Faza obrade podataka (eksploatacije)

Pod pojmom učenje neuronske mreže misli se na iterativni proces prilikom kojeg se na ulaz dovode podaci koji se propuštaju kroz mrežu te se ovisno o dobivenim izlazima i očekivanim izlazima radi korigiranje težina. Cilj je prilagoditi težine veza između neurona kako bi mreža mogla što točnije predvidjeti izlaze na temelju novih ulaznih podataka, odnosno glavni cilj je naučiti neuronsku mrežu da dobro generalizira. Nakon što

su predočeni svi podaci iz skupa za učenje, kažemo da je završila jedna epoha. Ovisno o tome kad se radi korekcija težina, razlikujem tri metode učenja:

1. Pojedinačno učenje (engl. *on-line*): U ovoj metodi težine se prilagođavaju nakon svakog pojedinačnog uzorka što omogućava brzu prilagodbu težina.

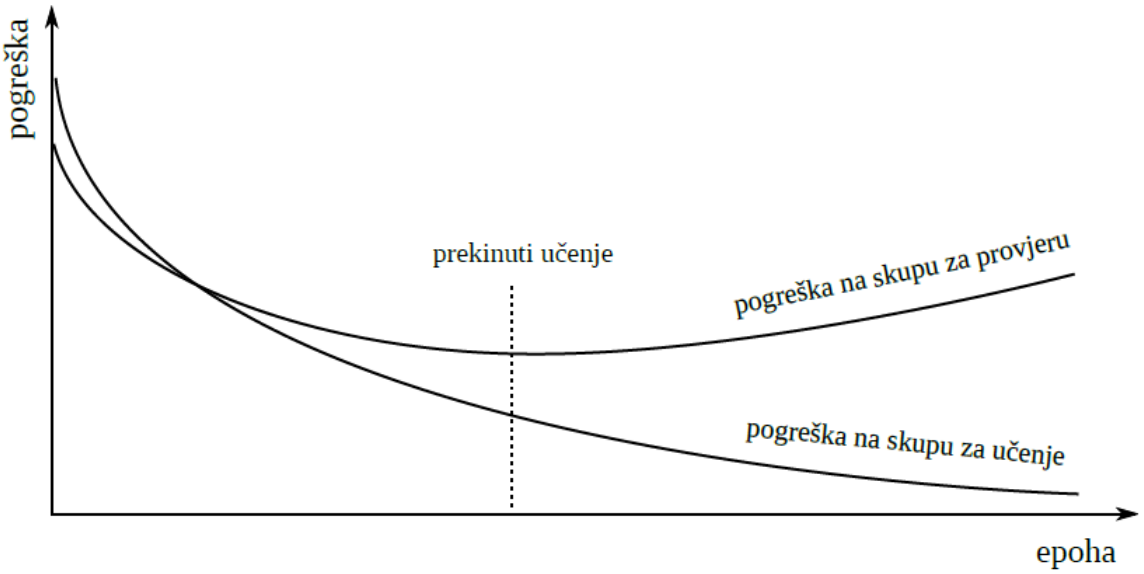
2. Učenje s minigrupama (engl. *mini-batches*): Težine se ažuriraju nakon što se obradi skupina uzoraka (minigrupa). Ova metoda predstavlja kompromis između pojedinačnog i grupnog učenja, kombinirajući prednosti obiju metoda.

3. Grupno učenje (engl. *batch*): U ovoj metodi težine se prilagođavaju tek nakon što se obradi cijeli skup uzoraka. Ovaj pristup zahtijeva više računalnih resursa i vremena.

Nakon što je model naučen, ulazi u fazu obrade podataka gdje prima dosad neviđene podatke za koje on daje predikciju. Ako model dobro generalizira, trebao bi davati ispravne predikcije za neviđene podatke. Složenost modela ima velik utjecaj na ponašanje modela pa će tako modeli koji su jednostavni imati jako loše generalizacijske sposobnosti jer nisu dovoljno složeni da bi mogli nešto naučiti na temelju podataka za učenje. Složeniji modeli mogu naučiti izvrsno generalizirati što rezultira odličnim rezultatima na neviđenim primjerima. Model ne smije biti previše složen jer onda dolazi do glavnog problema strojnog učenja - prenaučenosti (engl. *overfitting*). Model se previše prilagodi podacima za učenje za koje daje točne predikcije, što rezultira lošim predikcijama na neviđenim podacima. Takav model naravno nije dobar, cilj je izbjeći prenaučenost, a jedna od metoda koja procjenjuje učinkovitost modela na neviđenim podacima naziva se metoda unakrsne provjere [5].

3.3.3. Metoda unakrsne provjere

Unakrsna provjera (engl. *cross-validation*) je tehnika koja se koristi u strojnom učenju za procjenu generalizacijske sposobnosti modela. Budući da neviđeni podaci nisu dostupni, ideja je dio dostupnih primjera odvojiti i ponašati se kao da su oni neviđeni podaci na kojima će se model testirati. Podaci se dijele na *skup za učenje* i *skup za testiranje*, obično u omjeru 70:30. Nakon što model prilagodi težine ovisno o podacima iz skupa za učenje, testira se sposobnost generalizacije na skupu za testiranje.



Slika 3.5. Unakrsna provjera [3]

Budući da složenost modela može značajno utjecati na ponašanje modela, važno je odrediti pravu složenost. Zbog tog se uvodi još jedan skup podataka - *skup za provjeru* (engl. *validation set*). Dostupni podaci se dijele na tri dijela: većinski dio podataka ide u skup za učenje, dok podjednak broj primjera ide u skup za testiranje i skup za učenje pa podjela može biti na primjer 70:15:15 (70% skup za učenje, 15% skup za provjeru i 15% skup za testiranje) ili 40:30:30. Najbolji model se određuje tako da se svi modeli uče na skupu za učenje pa se njihova točnost testira na skupu za provjeru. Kao što je vidljivo na slici 3.5., učenje bi trebalo prekinuti kada pogreška počne rasti na skupu za provjeru. Model koji ima najbolje rezultate na tom skupu se uzima kao model s najboljom generalizacijom te se on potom testira na skupu za testiranje.

3.3.4. Algoritam propagacije pogreške unatrag

Algoritam propagacije pogreške unatrag (engl. *Backpropagation*) je temeljni algoritam za treniranje umjetnih neuronskih mreža koji se sastoji od dva glavna koraka: unaprijedni prolaz (engl. *forward pass*) i prolaz unatrag (engl. *backward pass*). Ovaj algoritam omogućava mreži da uči iz pogrešaka, prilagođavajući težine veza između neurona kako bi se minimizirala ukupna pogreška mreže. Backpropagation koristi gradijentni spust (engl. *gradient descent*) kako bi iterativno smanjio funkciju pogreške. Funkcija pogreške (engl. *loss function*) mjeri razliku između stvarnih vrijednosti i predviđenih vrijednosti koje model generira. Glavna svrha funkcije pogreške je kvantificirati koliko su predikcije

modela netočne. Jedna od najčešće korištenih funkcija pogreške u kontekstu umjetnih neuronskih mreža je srednja kvadratna pogreška (engl. *Mean Squared Error* - MSE). Definira se kao:

$$E = \frac{1}{2 \cdot N} \sum_{s=1}^{N} \sum_{i=1}^{N_0} (t_{s,i} - o_{s,i})^2$$

gdje:

- E je ukupna funkcija pogreške,
- N je broj primjera,
- N_0 je broj izlaznih neurona,
- $t_{s,i}$ je stvarna vrijednost za i-ti izlazni neuron na s-tom uzorku,
- $o_{s,i}$ je predviđena vrijednost za i-ti izlazni neuron na s-tom uzorku,
- $t_{s,i} - o_{s,i}$ je razlika između stvarne i predviđene vrijednosti za i-ti izlazni neuron na s-tom uzorku,
- $(t_{s,i} - o_{s,i})^2$ je kvadrat te razlike, koji osigurava da su sve pogreške pozitivne i naglašava veće pogreške,
- Faktor $\frac{1}{2}$ se često uključuje iz matematičkih razloga kako bi se olakšala izvedba derivata tijekom postupka gradijentnog spusta.

Na ulaz prethodno inicijalizirane neuronske mreže (postavljene su sve težine) se dovedu podaci koji se propuštaju kroz ostale slojeve mreže. Neuroni u svakom sloju računaju akumuliranu vrijednost koju potom propuštaju kroz aktivacijsku funkciju da bi dobili izlaz koji šalju u idući sloj i tako sve do izlaznog sloja. Kada neuroni izlaznog sloja generiraju podatke na svom izlazu, kažemo da je napravljen unaprijedni prolaz.

U prolasku unatrag, pogreška se propagira unatrag kroz mrežu kako bi se ažurirale težine. Prvo je potrebno odrediti pogreške neurona izlaznog sloja koje se računaju po formuli:

$$\delta_i^K = o_{s,i} \cdot (1 - o_{s,i}) \cdot (t_{s,i} - o_{s,i})$$

gdje:

- δ_i^K je pogreška i-tog neurona u izlaznom sloju K,
- $o_{s,i}$ je stvarni izlaz i-tog neurona za uzorak s,
- $t_{s,i}$ je očekivani izlaz i-tog neurona za uzorak s,
- $t_{s,i} - o_{s,i}$ je razlika između stvarnog i očekivanog izlaza.

Pogreške neurona izlaznog sloja su potrebne za izračun pogrešaka neurona skrivenog sloja. Neovisno o tome koliko skrivenih slojeva postoji, ideja je uvijek ista: sumiraj umnožak svih veza prema neuronima u sloju ispred s pogreškama tih neurona te sumu pomnoži s izlazom neurona i razlikom (1 - izlaz neurona). Formalno to izgleda ovako:

$$\delta_i^{(k)} = y_i^{(k)} \cdot (1 - y_i^{(k)}) \cdot \sum w_{i,d} \cdot \delta_d^{(k+1)}$$

gdje:

- $\delta_i^{(k)}$ je pogreška i-tog neurona u sloju k,
- $y_i^{(k)}$ je izlaz i-tog neurona u sloju k,
- $w_{i,d}$ je težina između i-tog neurona u sloju k i d-tog neurona u sloju $k+1$,
- $\delta_d^{(k+1)}$ je pogreška d-tog neurona u sloju $k+1$.

Zadnji i najvažniji korak je korekcija težina. Težine se korigiraju po idućoj formuli:

$$w_{i,j}^{(k)} = w_{i,j}^{(k)} + \eta \cdot y_i^{(k)} \cdot \delta_j^{(k+1)}$$

gdje:

- $w_{i,j}^{(k)}$ je težina između i-tog neurona u sloju k i j-tog neurona u sloju $k+1$,
- η je stopa učenja,

- $y_i^{(k)}$ je izlaz i-tog neurona u sloju k,
- $\delta_j^{(k+1)}$ je pogreška j-tog neurona u sloju $k+1$.

Za pomak se koristi malo drugačija formula jer je on svojstvo jednog neurona:

$$w_{0,j}^{(k)} = w_{0,j}^{(k)} + \eta \cdot \delta_j^{(k+1)}$$

gdje:

- $w_{0,j}^{(k)}$ je pomak za j-ti neuron u sloju k,
- η je stopa učenja,
- $\delta_j^{(k+1)}$ je pogreška j-tog neurona u sloju $k+1$.

Algoritam backpropagation je ključan za učinkovito treniranje neuronskih mreža jer omogućava mreži da iterativno prilagođava svoje težine [2]. Težine definiraju svojstva modela. Ako su težine dobro postavljene, model će moći vrlo dobro generalizirati.

4. Algoritam Minimax

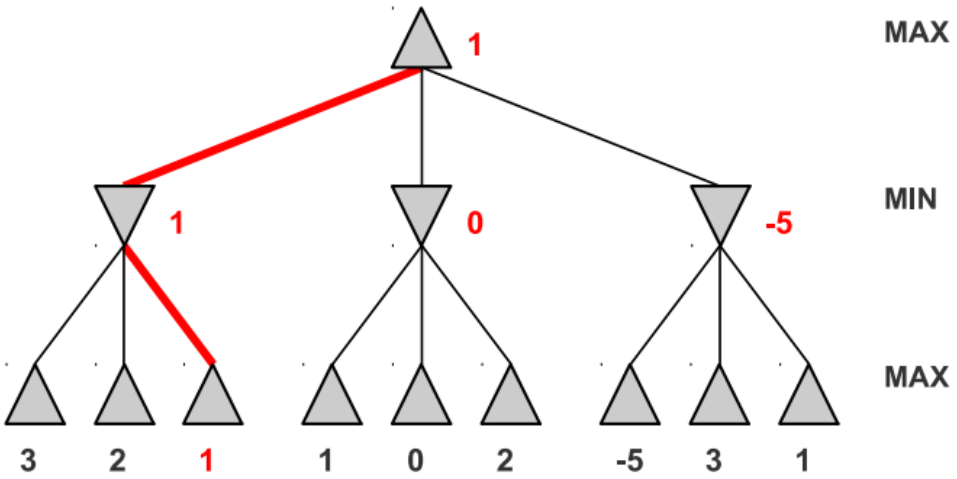
Minimax je algoritam za donošenje odluka koji se koristi u determinističkim igrama za dva igrača s potpunom informacijom i sumom nula. Igre sa sumom nula su one u kojima je dobitak jednog igrača jednak gubitku drugog igrača. Ukupna suma dobiti i gubitka u igri uvijek je nula. Primjerice, ako jedan igrač pobijedi i osvoji +1 bod, drugi igrač gubi i dobiva -1 bod. Cilj minimax algoritma je minimizirati maksimalni mogući gubitak, odnosno maksimizirati minimalni dobitak u igri, osiguravajući optimalnu strategiju za oba igrača. U ovom algoritmu, jedan igrač se smatra maksimizatorom (Max), dok se drugi igrač smatra minimizatorom (Min). Koraci algoritma su sljedeći:

1. Izgradnja stabla igre
    - Stablo igre predstavlja sve moguće poteze koje igrači mogu povući od trenutnog stanja igre do kraja igre. Svaki čvor u stablu predstavlja stanje igre, a grane predstavljaju moguće poteze.
2. Procjena vrijednosti čvorova
    - Listovi stabla (krajnja stanja igre) procjenjuju se pomoću funkcije evaluacije koja određuje korisnost tih stanja za igrača Max.
    - Ako je Max na potezu, cilj je maksimizirati vrijednost čvora. Ako je Min na potezu, cilj je minimizirati vrijednost čvora.
3. Rekurzivni prolaz kroz stablo
    - Algoritam rekurzivno prolazi kroz stablo od listova prema korijenu. Na svakom nivou, Max ili Min biraju potez koji maksimizira ili minimizira vrijednost čvora, ovisno o tome koji je igrač na potezu.

4. Odabir optimalnog poteza

    - U korijenskom čvoru, optimalni potez za igrača Max je onaj koji vodi do čvora s maksimalnom vrijednosti.

Primjer ovog algoritma prikazan je na slici 4.1. Listovi stabla su evaluirana stanja koja imaju neke vrijednosti. Jednu razinu iznad nalaze se čvorovi koji predstavljaju Min igrača. Njegov cilj je minimizirati maksimalni mogući gubitak pa će on od svih mogućnosti odabrati onu najmanju. Skroz lijevi čvor u stablu igrača Min bira minimalnu vrijednost iz skupa (3, 2, 1) pa će odabrati vrijednost 1. Isto tako, čvor u sredini uzima minimum iz skupa (1, 0, 2) što je vrijednost 0, dok skroz desni Min čvor bira vrijednost -5. Razinu iznad u stablu se nalazi igrač Max čiji je cilj maksimizirati minimalnu dobit. On može birati između vrijednosti (1, 0, -5) i odlučuje se za maksimalnu vrijednost koja iznosi 1.



Slika 4.1. Minimax algoritam [4]

Problem kod ovog algoritma je što igrač svaki put kada je na redu mora računati optimalnu strategiju. To može biti dosta zahtjevno zbog vremenske složenosti, jer broj stanja igre eksponencijalno ovisi o broju mogućih poteza. U praksi su takve odluke vremenski ograničene pa je potrebno skratiti broj stanja koja se pretražuju. To je moguće postići ograničavanjem dubine pretraživanja stabla i podrezivanjem stabla.

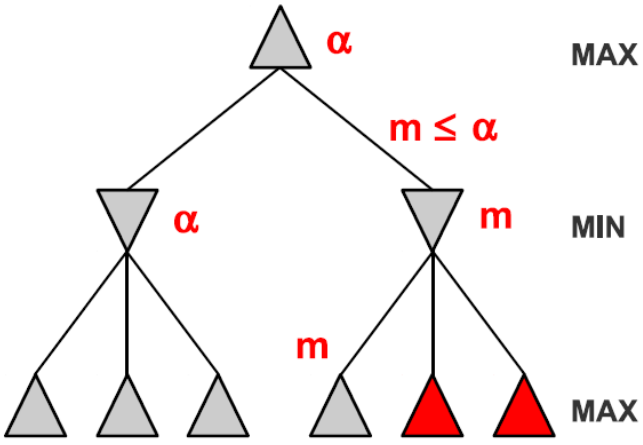
4.1. Alfa beta podrezivanje

Alfa-beta podrezivanje je optimizacija minimax algoritma koja značajno smanjuje broj čvorova koje treba procijeniti u stablu igre. Ova tehnika eliminira grane stabla koje ne mogu utjecati na konačnu odluku, čime se smanjuje vrijeme izvršavanja algoritma.

- Alfa predstavlja najbolji (najveći) rezultat koji igrač Max može zagarantirano ostvariti u tom trenutku
- Beta predstavlja najbolji (najmanji) rezultat koji igrač Min može zagarantirano ostvariti u tom trenutku

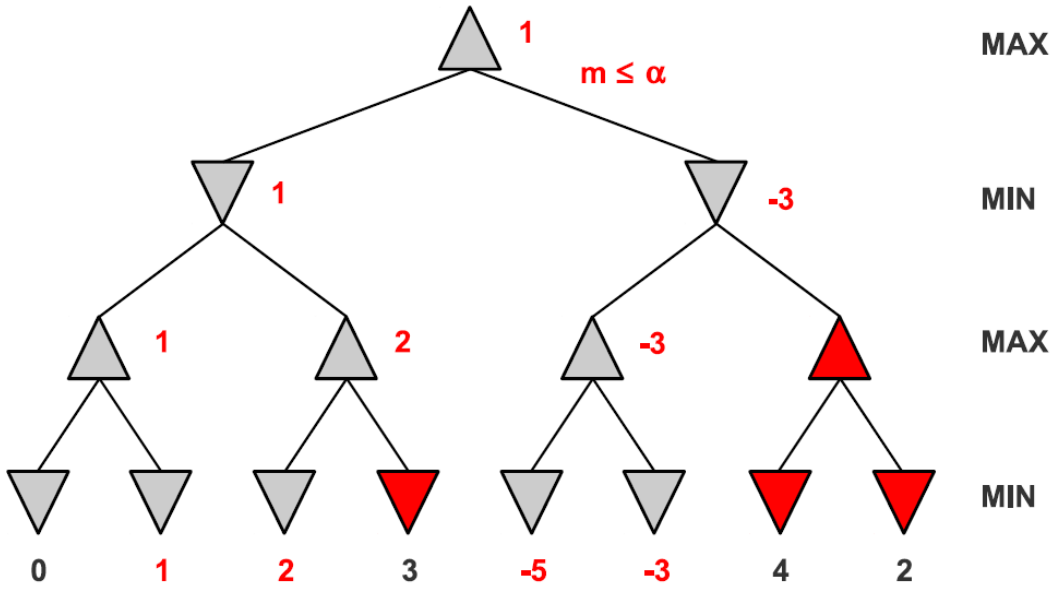
Ovaj algoritam se sastoji od nekoliko koraka:

1. Inicijalizacija
    - Na početku se postavi: $\alpha = -\infty$ i $\beta = +\infty$
2. Rekurzivni prolaz kroz stablo
    - Dok se rekurzivno prolazi kroz čvorove, ažuriraju se vrijednosti α i β na temelju minimizirajućih i maksimizirajućih poteza.
3. Podrezivanje grana
    - Ako se u bilo kojem trenutku utvrdi da trenutni čvor ne može poboljšati trenutnu α ili β vrijednost, ta grana se podrezuje (ne evaluira se dalje).



Slika 4.2. Alfa-podrezivanje [4]

Vizualni prikaz alfa-podrezivanja prikazan je na slici 4.2. Alfa-podrezivanje se radi ako se podrezuje ispod Min čvora, kao što je vidljivo na slici. Konkretan primjer prikazan na slici 4.3. objašnjen je u nastavku.

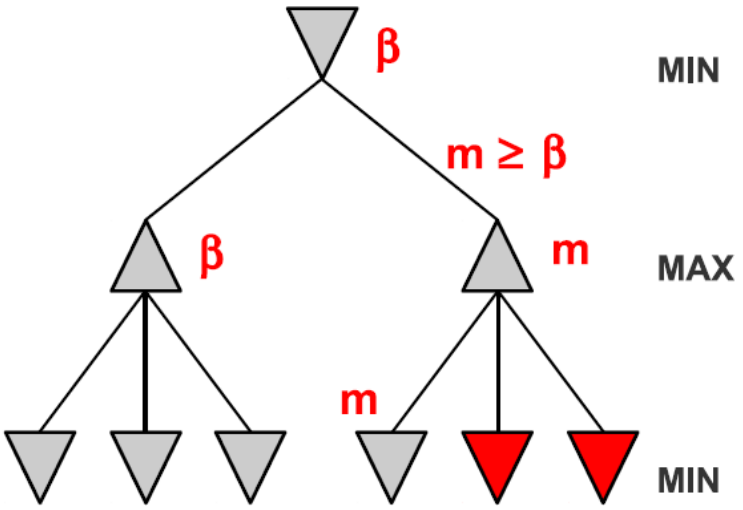


Slika 4.3. Primjer alfa-podrezivanja [4]

Recimo da Max igrač istražuje lijevo i desno podstablo. Nakon istraživanja lijevog stabla dobijemo nekakvu vrijednost, nazovimo ju alfa koja u ovom slučaju iznosi 1. Krenemo istraživati desno podstablo i ako čvor Min u lijevom podstablu ima vrijednost koja je manja ili jednaka vrijednosti alfa, prekidamo istraživanje desnog podstabla. Zašto je to tako? Znamo da je nakon nas na redu igrač Min koji će odabrati najlošiji potez za nas i u ovom slučaju on zna da može ostvariti vrijednost -3. Ako čvor Min shvati da može ostvariti manju vrijednost od tog, primjerice -5, on će odabrati tu vrijednost jer je to stanje gore za nas kao igrača Max. Budući da najveća vrijednost koju će ostvariti igrač Min iznosi -3, možemo prekinuti pretraživanje jer je ta vrijednost manja od vrijednosti alfa pa sigurno znamo da nećemo naći bolje rješenje.

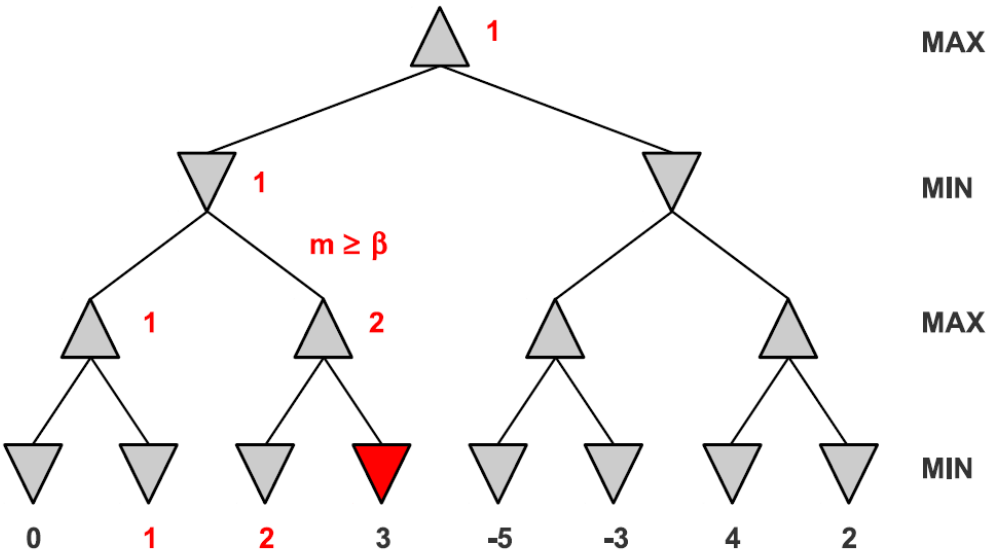
Ako se podrezuje ispod Max čvora, radi se o beta-podrezivanju koje je prikazano na slici 4.4. Situacija je slična, samo što se za ovo podrezivanje gleda iz perspektive igrača Min.

Primjer beta-podrezivanja prikazan je na slici 4.5. Istražimo lijevo podstablo i dobijemo neku vrijednost beta koja u ovom slučaju iznosi 1. Krenemo s istraživanjem desnog



Slika 4.4. Beta-podrezivanje [4]

podstabla i čvor Max nam vrati vrijednost njegovog prvog djeteta koja iznosi 2. Budući da se radi o čvoru Max koji uzima maksimalne vrijednosti, znamo da će on u najgorem slučaju odabrati vrijednost 2. Ako postoji manja vrijednost, on će ju zanemariti dok će veće vrijednosti uzeti, primjerice 5, jer su one bolje za njega i lošije za nas kao igrača Min. Rekli smo da vrijednost beta iznosi 1, a igrač Max ima trenutnu vrijednost 2, koja je veća od vrijednosti beta pa možemo prekinuti pretraživanje jer smo u desnom podstablu već pronašli najmanju vrijednost koja je jednaka ili veća od vrijednosti beta.



Slika 4.5. Primjer beta-podrezivanja [4]

Alfa-beta podrezivanje znatno poboljšava učinkovitost minimax algoritma zbog eliminacije nepotrebnih procjena. Kombinacija minimax algoritma i alfa-beta podrezivanja omogućava optimalnu strategiju za igre s dva igrača, smanjujući vrijeme i resurse potrebne za donošenje odluka.

5. Algoritam Expectiminimax

Expectiminimax algoritam je proširenje minimax algoritma koji se koristi za donošenje odluka u stablima igara koje uključuju elemente nesigurnosti ili šanse, kao što su igre na sreću. Ovaj algoritam kombinira minimax pristup s očekivanim vrijednostima kako bi pružio optimalnu strategiju za igre koje sadrže slučajne događaje, poput bacanja kocke ili izvlačenja karata.

Ovaj algoritam se od minimax algoritma razlikuje po tome što osim čvorova Max i Min, postoji još jedna vrsta čvorova. *Čvorovi šanse* (engl. *Chance čvorovi*) su čvorovi koji predstavljaju slučajne događaje s poznatim vjerojatnostima [6]. Oni izračunavaju očekivanu vrijednost na temelju vjerojatnosti svakog mogućeg događaja. Za čvorove šanse, vrijednost čvora v je:

$$v = \sum_i p_i \cdot v_i$$

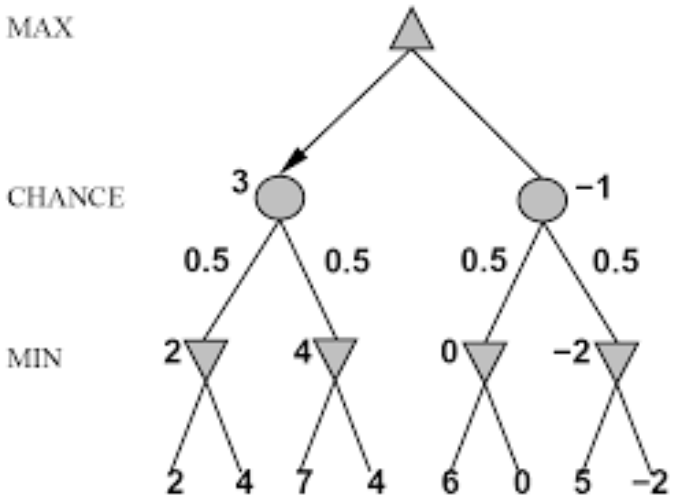
gdje p_i predstavlja vjerojatnost i-tog ishoda, a v_i vrijednost i-tog djeteta čvora.
Koraci algoritma su slični koracima minimax algoritma uz neke promjene:

1. Izgradnja stabla igre
    - Čvorovi šanse dodaju se na mjesta gdje dolazi do slučajnih događaja, s granama koje predstavljaju sve moguće ishode tih događaja.
2. Procjena vrijednosti čvorova
    - Vrijednosti unutarnjih čvorova računaju se na temelju vrijednosti njihovih djece i tipa čvora.

3. Rekurzivni prolaz kroz stablo

    - Algoritam rekurzivno prolazi kroz stablo od listova prema korijenu. Na svakom nivou, vrijednost čvora određuje se ovisno o tome je li čvor Max, Min ili Chance.

4. Odabir optimalnog poteza



Slika 5.1. Expectiminimax algoritam

Primjer ovog algoritma prikazan je na slici 5.1. gdje se vidi modeliranje slučajnosti pomoći čvorova šansi. Uzmimo za primjer lijevo podstablo igrača Max. Znamo da je nakon igrača Max na redu igrač Min koji u ovom slučaju može ostvariti vrijednosti 2 i 4, ali svaku tu vrijednosti može ostvariti s vjerojatnošću koja iznosi 0.5. Čvor šanse računa očekivanu vrijednost po gore navedenoj formuli pa dobijemo ((2*0.5) + (4*0.5)), što iznosi 3 kao što je prikazano na slici.

6. Implementacija

U ovom poglavlju detaljno je opisan proces implementacije kartaške igre Briškula te raznih igrača umjetne inteligencije za koje su korišteni prethodno spomenuti algoritmi i metode učenja.

6.1. Briškula

Prije samog koda za igru, važno je definirati kako su implementirane karte. Svaka karta može imati jačinu i boju, što je implementirano enumeracijama *Rank* i *Suit*, čiji je kod dan u nastavku.

```
public enum Rank {

        AS(1, 11, "As"),
        TRICA(3, 10, "Trica"),
        KRALJ(13, 4, "Kralj"),
        KONJ(12, 3, "Konj"),
        FANAT(11, 2, "Fanat"),
        SEDMICA(7, 0, "Sedmica"),
        SESTCA(6, 0, "Sestica"),
        PETICA(5, 0, "Petica"),
        CETVORKA(4, 0, "Cetvorka"),
        DVICA(2, 0, "Dvica"),;

        private final int rank;
        private final int value;
        private final String name;

public enum Suit {
```

```
    DINARI("Dinari"),
    SPADE("Spadi"),
    BASTONI("Bastoni"),
    KUPE("Kupa"),;

    private final String suit;
```

Klasa *Card* predstavlja implementaciju karte koja osim jačine i boje ima i pripadnu sliku koja može biti vidljiva ovisno o varijabli *isFacedUp*. Varijabla *isFacedUp* simulira je li karta vidljiva ili ne (nalazi se u špilu karata ili u ruci protivnika pa je skrivena). Primjer vidljive karte i poleđine karte kad je ona skrivena je prikazan slikama 6.1. i 6.2., a programski kod za klasu *Card* je dan u nastavku.

Slika 6.1. Okrenuta karta

Slika 6.2. Skrivena karta

```
public class Card {
```

```
    private Suit suit;
    private Rank rank;
    private boolean isFacedUp;
    private ImageIcon image;


    public Card(Rank rank, Suit suit) {
        this.rank = rank;
        this.suit = suit;
        isFacedUp = false;
        initializeImage();
```

Špil karata predstavljen je klasom *Deck* koja sadrži listu karata (klasa *Card*) te ima metode za miješanje špila, kao i mogućnost uzimanja karte s vrha (metoda *drawCard*).

```
public class Deck {

    private List<Card> cards;

    public Deck() {
        cards = new ArrayList<>();
        fillDeck();
    }

    public Deck(Deck deck) {
        cards = new ArrayList<>(deck.getDeck());
    }

    private void fillDeck() {
        for(Suit s : Suit.values())
            for(Rank r : Rank.values())
                cards.add(new Card(r, s));
    }
```

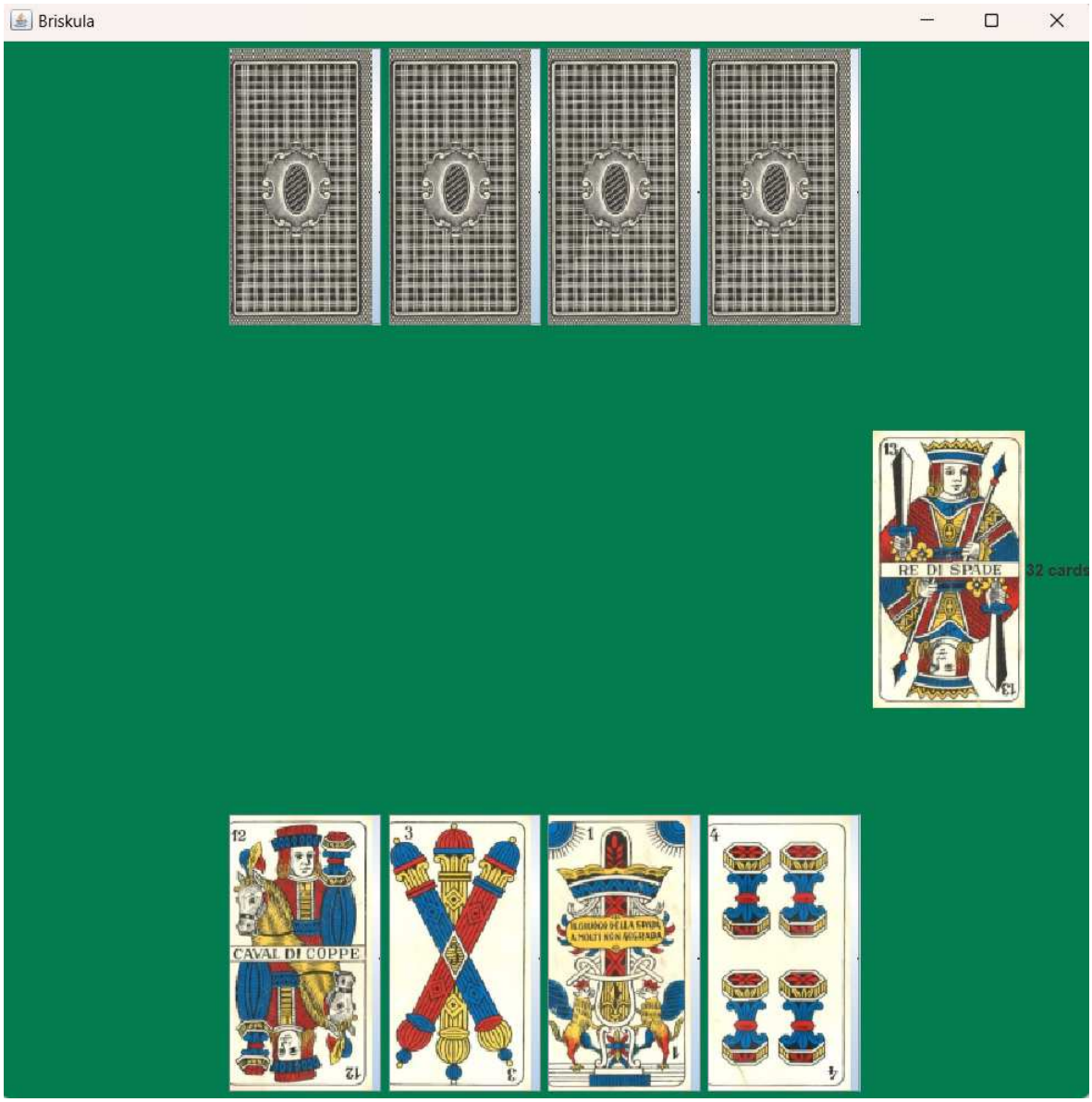
```
    public void shuffleDeck() {
        Collections.shuffle(cards);
    }

    public Card drawCard() {
        if(cards.isEmpty()) {
            throw new IllegalStateException();
        }
        return cards.removeFirst();
    }
```

Implementirana je "dupla briškula" koja je teža varijacija igre budući da svaki igrač u jednoj rundi baca po dvije karte pa se za odlučivanje pobjednika runde razmatraju četiri karte u odnosu na "običnu briškulu" gdje se razmatraju dvije karte (svaki igrač baca po jednu kartu u rundi). U nastavku su dane glavne funkcije koje sadrže svu logiku igre, prva se nalazi u klasi *Briscola* i implementira petlju cijele igre dok se druga nalazi u klasi *Round* i predstavlja logiku jedne runde te na kraju provjerava tko je pobjednik runde. Na slici 6.3. je dan vizualni prikaz jedne runde. S desne strane ekrana se vidi zadnja karta u špilu koja govori koji je adut igre, te se pored nje nalazi broj preostalih karata u špilu.

```
public void play() {
        Collections.shuffle(players);
        initialDeal();
        Card trumpCard = deck.drawCard();
        System.out.println("Zog:␣" + trumpCard.toString());
        deck.addCard(trumpCard);
        initialDeal();
        Round round = new Round(players, trumpCard.getSuit(), sc
           );
        while(!deck.isEmpty()) {
            Player roundWinner = round.play();
            if(roundWinner.getName() != players.get(0).getName()
               ) {
                players.add(players.removeFirst());
            }
```

```
            for(int i = 0; i < 2; i++) {
                for(Player player : players) {
                    player.addToHand(deck.drawCard());
                }
            }
        }

        Player roundWinner = round.play();
        if(roundWinner.getName() != players.get(0).getName()) {
            players.add(players.removeFirst());
        }
        round.setLastHand();
        round.play();

        determineWinner();
        sc.close();
}

public Player play() {
        if(players.size() != 2)
            throw new IllegalStateException();

        for(int i = 0; i < 2; i++)
            for(Player player : players)
                playTurn(player);

        Set<Card> cards = thrownCards.keySet();
        Card highest = null;
        for(Card card : cards)
            if(highest == null || isHigher(card, highest, trump)
               )
                highest = card;
```

```
            Player winner = thrownCards.get(highest);
            winner.addToDiscardCards(List.copyOf(cards));
            thrownCards.clear();
            return winner;
}
```



Slika 6.3. Runda briškule

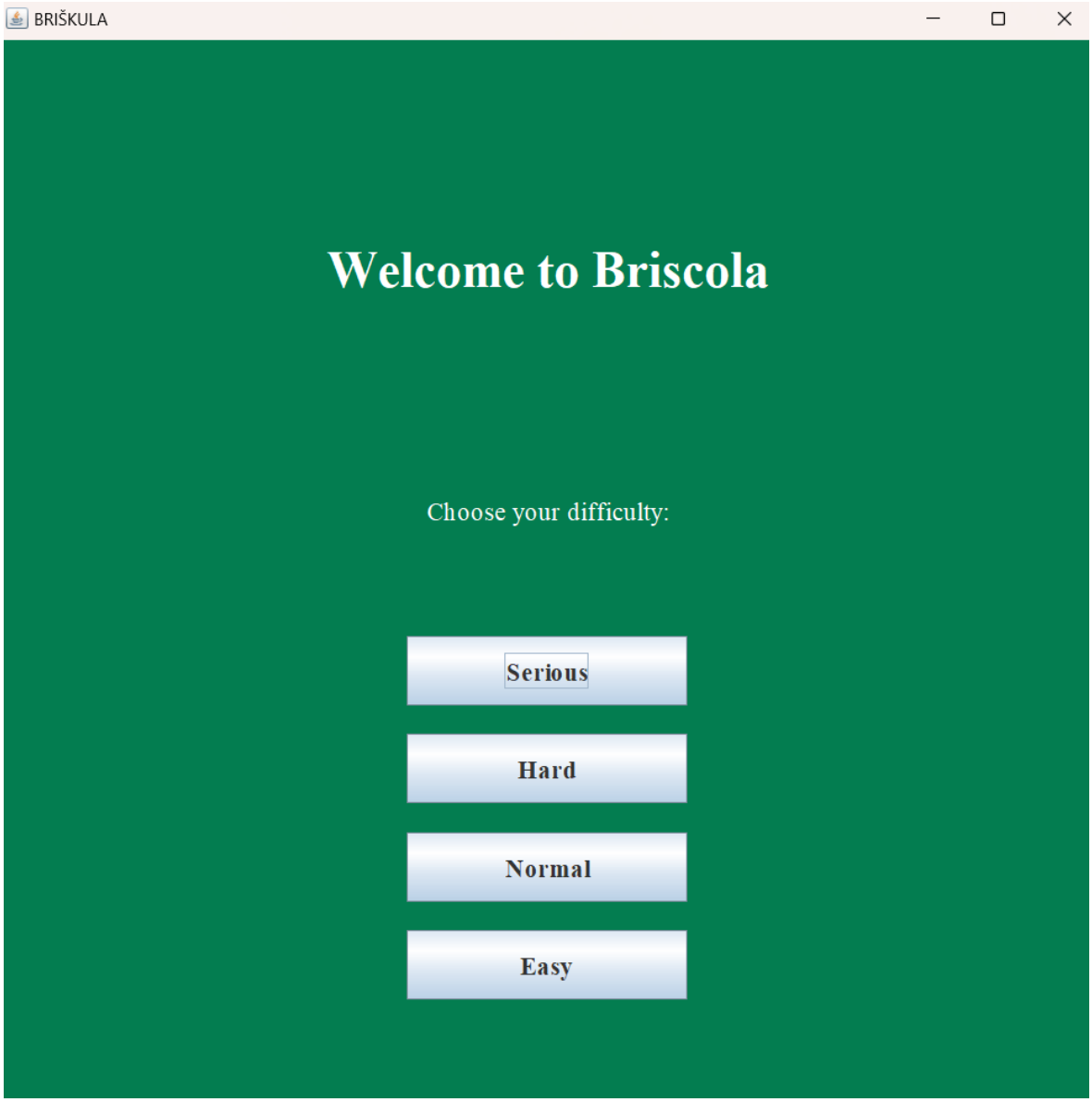
6.2. Igrači

Implementirane su četiri vrste igrača koje predstavljaju četiri različite težine:

- easy (lagano)

- normal (normalno)
- hard (teško)
- serious (ozbiljno - u smislu najteže)

Kada se pokrene igra, korisnik može birati težinu kao što je prikazano na slici 6.4.



Slika 6.4. Igrači

6.2.1. Igrač Easy

Ovaj igrač predstavlja najlakšeg protivnika koji nasumično baca karte. U nastavku je dan primjer koda gdje se bira indeks karte koja će se baciti iz ruke ako igrač ima četiri karte (mogućnosti su 0, 1, 2 i 3).

```
        cardIndex = (int) (Math.random() * 3);
```

6.2.2. Igrač Normal

Ovaj igrač predstavlja protivnika koji zna igrati i koji je dovoljno sposoban zaključiti kada treba "ubiti" ako je to moguće kako bi pobijedio u rundi i osvojio što više bodova. Ovaj igrač implementiran je neuronskom mrežom i treniran je backpropagation algoritmom. Arhitektura neuronske mreže je 10x7x15x14x4, što bi značilo da postoje tri skrivena sloja. Slojevi su implementirani klasom *Layer* čiji je kod dan u nastavku.

```
public class Layer {

        private static Random r = new Random();
        private DoubleUnaryOperator activationFunction = (x ->
           1.0 / (1.0 + Math.exp(-x)));
        private double[][] weights;
        private double[] biases;
        private double[] inputs;
        private double[] outputs;
        private double[] errors;
        private String name;

        public Layer(String name, int inputSize, int
           numberOfNeurons) {
            this.name = name;
            biases = new double[numberOfNeurons];
            weights = new double[numberOfNeurons][inputSize];
            errors = new double[numberOfNeurons];
            outputs = new double[numberOfNeurons];

            for(int i = 0; i < numberOfNeurons; i++) {
                biases[i] = r.nextDouble();
                for(int j = 0; j < inputSize; j++) {
                    weights[i][j] = r.nextDouble();
                }
```

```
        }
    }
```

Neuronska mreža ima deset ulaza koji su redom:

- igračeve karte (4 vrijednosti za 4 moguće karte, 0 označava da karta pod tim rednim brojem ne postoji)
- bačene karte u rundi (4 vrijednosti - svaki igrač treba baciti dvije karte u rundi pa je 4 maksimalni broj bačenih karata)
- broj runde
- karta aduta

Izlazni sloj ima četiri neurona koji predstavljaju indeks karte koju treba baciti. Primjerice, treći neuron izlaznog sloja ima vrijednost koja je blizu 1, dok ostali imaju vrijednost na izlazu koja je blizu 0. To znači da iz ruke treba baciti treću kartu po redu, odnosno kartu koja se nalazi na indeksu 2 (indeksi kreću od 0).

U nastavku je dan kod klase *AdvancedNeuralNetwork* i njezine metode *train* koja predstavlja implementaciju backpropagation algoritma.

```
public class AdvancedNeuralNetwork {

    private static double ETA = 0.15;
    private Layer hiddenLayer;
    private Layer secondHiddenLayer;
    private Layer thirdHiddenLayer;
    private Layer outputLayer;

    public double train(double[] input, double[] target) {
        double[] actualOutputs = hiddenLayer.calculate(input
            );
        actualOutputs = secondHiddenLayer.calculate(
            actualOutputs);
        actualOutputs = thirdHiddenLayer.calculate(
```

```
    actualOutputs);
actualOutputs = outputLayer.calculate(actualOutputs)
    ;

//pogreske izlaznog sloja
outputLayer.errorsCalculation(target);

thirdHiddenLayer.errorsCalculation(outputLayer.
    getSums());

//pogreske 2. skrivenog sloja
secondHiddenLayer.errorsCalculation(thirdHiddenLayer
    .getSums());

//pogreske 1. skrivenog sloja
hiddenLayer.errorsCalculation(secondHiddenLayer.
    getSums());

//korekcija tezina izlaznog sloja
outputLayer.weightsCorrection(ETA);

thirdHiddenLayer.weightsCorrection(ETA);

//korekcija tezina 2. skrivenog sloja
secondHiddenLayer.weightsCorrection(ETA);

//korekcija tezina skrivenog sloja
hiddenLayer.weightsCorrection(ETA);

double sum = 0.0;
for(int i = 0; i < target.length; i++) {
    double difference = target[i] - actualOutputs[i
        ];
    sum += Math.pow(difference, 2);
```

```
        }

        return sum;
    }
```

6.2.3. Igrač Hard

Ozbiljan igrač zna baciti najjaču kartu u rundi, ali također zna i kalkulirati kada je možda ipak bolje izgubiti rundu kako bi se dobile bolje karte. Čest primjer toga je kada u špilu karata ostanu četiri karte od kojih je zadnja vidljiva i predstavlja adut pa ako se radi o jakoj karti, većina igrača će namjerno izgubiti rundu kako bi zadnji vukli kartu. Za implementaciju ovog igrača korištena je neuronska mreža kao i za normalnog igrača, ali se znanje neuronske mreže kombinira s algoritmom expectiminimax pa ovaj igrač koristi neuronsku mrežu dok se dovoljno ne smanji broj nepoznatih karata, nakon čega se primjenjuje Expectiminimax s ciljem pronalaska najboljeg poteza.

6.2.4. Igrač Serious

Najteži protivnik implementiran je koristeći algoritam minimax. On zna poredak karata u špilu kao i karte koje se nalaze u ruci protivnika pa računa optimalnu strategiju kako bi pobijedio. Iako on zna sve karte, moguće ga je pobijediti ako se sve karte poslože. Sve ovisi o početno podijeljenim kartama pa nije nemoguće da korisnik slučajno odabere najbolju strategiju i sve točno odigra. U praksi je to vrlo teško ostvariti jer jednom kad korisnik pogriješi, ovaj igrač pronalazi najbolje poteze i pobjeđuje. U nastavku je prikazana implementacija algoritma minimax.

```
private static int minimax(GameState state, int depth, boolean
        isMaximizingPlayer, int alpha, int beta) {
    if (isTerminal(state) || depth == 0) {
        return state.evaluateState();
    }

    if (isMaximizingPlayer) {
        int maxEval = Integer.MIN_VALUE;
        for (int i = 0; i < state.getPossibleMoves(); i++) {
```

```
                GameState newState = performMove(state, i);
                int eval = minimax(newState, depth - 1, false,
                   alpha, beta);
                maxEval = Math.max(maxEval, eval);
                alpha = Math.max(alpha, maxEval);
                if (beta <= alpha) {
                    break; // Alpha-beta pruning
                }
            }
            return maxEval;
        } else {
            int minEval = Integer.MAX_VALUE;
            for (int i = 0; i < state.getPossibleMoves(); i++) {
                GameState newState = performMove(state, i);
                int eval = minimax(newState, depth - 1, true,
                   alpha, beta);
                minEval = Math.min(minEval, eval);
                beta = Math.min(beta, minEval);
                if (beta <= alpha) {
                    break; // Alpha-beta pruning
                }
            }
            return minEval;
        }
    }
```

7. Rezultati

Prilikom evaluiranja rezultata gledalo se koliko je igrač ostvario pobjeda, a ne koliko je u prosjeku imao bodova. Napredni igrači često broje osvojene bodove tijekom igre, pa kad osvoje više od 60 bodova znaju da su pobijedili. Takav pristup je korišten i kod igrača koji su implementirani koristeći minimax i expectiminimax.

Kao što je i očekivano, igrač *Easy* daje najslabije rezultate jer nasumično baca karte. Iako bi često znao iznenaditi i baciti najjaču kartu te osvojiti puno bodova, ima jako mali postotak pobjeda.

Igrač *Normal* je ostvario nekoliko pobjeda zahvaljujući neuronskoj mreži koja je uglavnom uspjela naučiti osvajati runde. Neuronska mreža je također naučila bacati karte koje donose puno bodova nakon što igra zadnja u rundi i zna da je s prijašnjom bačenom kartom osvojila rundu. Što se tiče postotka pobjeda, napredak je definitivno vidljiv.

Najbolji igrač koji ne "vara", odnosno koji ne zna poredak karata u špilu i protivnikove karte je igrač *Hard*. Iako dosta karata treba biti bačeno kako bi se počeo primjenjivati ovaj algoritam umjesto izlaza neuronske mreže, na kraju je isplativo jer igrač uspijeva maksimalno iskoristiti preostale runde. Postotak pobjeda je veći nego kod igrača *Normal*.

Najteži protivnik, igrač *Serious*, ima stopostotni učinak iako se doima da ga je moguće pobijediti. U dosta slučajeva je imao malo preko 60 bodova, ali kao što je već rečeno, on zna da ako osvoji preko 60 bodova, dalje ne treba pretraživati prostor stanja.

8. Zaključak

U ovom radu uspješno je implementirana logika kartaške igre Briškule te je napravljena aplikacija koja korisniku omogućuje vizualan prikaz igre. Cilj rada bio je istražiti različite metode umjetne inteligencije i njihovu primjenu na klasičnu kartašku igru pa je napravljeno više igrača koristeći razne metode te su evaluirane performanse svakog pristupa.

Neuronske mreže pokazale su se korisnima za modeliranje kvalitetnog igrača. Treniranje neuronske mreže zahtijevalo je značajnu količinu podataka, ali rezultati su pokazali da mreža može naučiti strategiju iz danih primjera.

Expectiminimax algoritam dodatno je unaprijedio sposobnosti AI igrača dodavanjem elemenata nesigurnosti u odlučivanje. Korištenje expectiminimax algoritma omogućilo je simulaciju slučajnih događaja, poput izvlačenja karata. Ovaj pristup rezultirao je robusnijim i realističnijim modelom.

Implementacija minimax algoritma omogućila je igraču da procijeni sve moguće poteze i odabere optimalni potez na temelju minimizacije maksimalnog gubitka. Ovaj pristup pokazao se učinkovit u situacijama gdje su svi budući potezi predvidivi, omogućujući AI igraču da planira nekoliko poteza unaprijed i donosi optimalne odluke. [5]

Ovaj rad demonstrira kako različite tehnike umjetne inteligencije mogu biti kombinirane za postizanje dobrih rezultata u kompleksnim igrama. Važno je napomenuti da je ovo samo jedan od mogućih načina pristupanja ovom problemu te da ima mjesta za napredovanje i nadogradnju ovog pristupa.

Literatura

[1] Marko Čupić, *Uvod u strojno učenje*. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2020.

[2] Bojana Dalbelo Bašić i Jan Šnajder, "Umjetne neuronske mreže", 2019., Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva.

[3] Marko Čupić, *Umjetne neuronske mreže*. Sveučilište u Zagrebu, Fakultet elektrotehnike računarstva, 2018.

[4] Bojana Dalbelo Bašić i Jan Šnajder, "Igranje igara", 2018., Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva.

[5] ——, "Strojno učenje", 2019., Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva.

[6] Marko Čupić, *Igranje igara*. Sveučilište u Zagrebu, 2020.

Sažetak

Primjena umjetne inteligencije u kartaškoj igri

Krešimir Blaić

Ovaj rad bavi se razvojem igrača za kartašku igru briškulu, koristeći razne metode i algoritme umjetne inteligencije. Objašnjena su pravila kartaške igre i dan je programski kod za važne dijelove. Detaljno su objašnjene umjetne neuronske mreže kao i načini učenja istih, od kojih je najvažniji bio algoritam propagacije pogreške unatrag. Opisan je algoritam minimax koji se koristi u deterministčkim igrama, uz kojeg dolazi i alfa-beta podrezivanje kako bi se smanjio broj stanja koje treba pretražiti. Dan je opis expectiminimax algoritma koji je sličan minimaxu, a koji se koristi u igrama u kojima je prisutan utjecaj slučajnosti. Napravljena su četiri igrača od kojih jedan nasumično baca karte, drugi koristi neuronsku mrežu, treći koristi kombinaciju neuronske mreže i algoritma expectiminimax, a četvrti igrač predstavlja igrača koji zna karte unaprijed i koristi minimax algoritam.

Ključne riječi: umjetna inteligencija; kartaška igra; neuronske mreže; strojno učenje; minimax; expectiminimax

Abstract

Application of artificial intelligence in the card game

Krešimir Blaić

This paper deals with the development of players for the card game "briškula", using various methods and algorithms of artificial intelligence. The rules of the card game are explained and the program code for the important parts is given. Artificial neural networks as well as ways of learning them were explained in detail, the most important of which was the algorithm of backward error propagation. The minimax algorithm used in deterministic games is described, along with alpha-beta trimming to reduce the number of states to be searched. A description of the expectiminimax algorithm is given, which is similar to minimax, and which is used in games where the influence of randomness is present. Four players are created, one of whom randomly throws cards, the second uses a neural network, the third uses a combination of a neural network and the expectiminimax algorithm, and the fourth player represents a player who knows the cards in advance and uses the minimax algorithm.

Keywords: artificial intelligence; card game; neural networks; machine learning; minimax; expectminimax