Računalna igra koja povezuje stvarni i virtualni svijet ostvarena u Unity okolini

Maletić, Ivan

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:375790

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-09-13**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Ivan Maletić

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

Doc. dr. sc. Tomislav Stipančić, dipl. ing. Ivan Maletić

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem svojoj obitelji na svakom obliku podrške tijekom dosadašnjeg studija te mentoru doc. dr. sc. Tomislavu Stipančiću, dipl. ing., na stručnim sugestijama, pomoći i pristupačnosti tijekom pisanja završnog rada.

Ivan Maletić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove.
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Ivan Maletić** JMBAG: **0035216743**

Naslov rada na hrvatskom jeziku: **Računalna igra koja povezuje stvarni i virtualni svijet ostvarena u Unity okolini**

Naslov rada na engleskom jeziku: **A computer game that connects the real and virtual world implemented in a Unity environment**

Opis zadatka:

Suvremene tehnike za vizualizaciju informacija omogućuju njihov prikaz tako da kontekstualno i intuitivno budu lakše shvaćene od strane ljudi. Na taj je način omogućeno kreiranje računalnih igara koje vjerno simuliraju objekte ili pojave iz stvarnog svijeta. Također, koristeći vizijski ili neki drugi senzor moguće je povezati procese iz stvarnog svijeta s onima koji se odvijaju na računalu.
U radu je potrebno modelirati računalnu igru u Unity programskoj okolini koja između ostaloga omogućuje ostvarivanje interakcije između stvarnog i virtualnog svijeta. Koristeći metode računalnog vida (OpenCV) i vizijski senzor potrebno je ostvariti računalno prepoznavanje kružnih objekata u sklopu realne okoline. Na temelju lokacije prepoznatog objekta te dinamike njegovog kretanja potrebno je računalno modelirati uzročnu vezu koja bi se potom koristila prilikom modeliranja dinamike procesa u računalnoj igri u sklopu virtualne okoline. Dobiveno softversko rješenje je potrebno eksperimentalno evaluirati u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:
30. 11. 2021.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

SADRŽAJ I
POPIS SLIKA II
POPIS TABLICA III
POPIS OZNAKA IV
1. UVOD 1
2. UNITY OKOLINA 3
3. RAČUNALNI VID 8
4. NEURONSKE MREŽE 13
5. GENETSKI ALGORITAM 21
6. MODELIRANJE IGRE UNUTAR UNITY OKOLINE 26
7. SUSTAV AUTOMATSKOG IGRANJA 34
8. ZAKLJUČAK 38
LITERATURA 39
PRILOG 40

POPIS SLIKA

Slika 1 - Grafičko korisničko sučelje Unity okoline [1] 3
Slika 2 - Mogućnosti unutar hijerarhije 4
Slika 3 - Komponenta *Rigidbody* 6
Slika 4 - Dijagram procesa rada unutar Unity okline 7
Slika 5 - HSV prostor boja 9
Slika 6 - RGB prostor boja 9
Slika 7 - Primjena linearnog filtra [4] 10
Slika 8 – Lijevo: ulazna slika, desno: Gaussov filtar primijenjen na ulaznu sliku [4] 11
Slika 9 - Vizualizacija rezultata Harrisove detekcije [4] 12
Slika 10 - Struktura biološkog neurona 13
Slika 11 - Struktura umjetnog neurona [5] 13
Slika 12 - Struktura neuronske mreže 14
Slika 13 - Pseudo kod genetskog algoritma 22
Slika 14 - Ruletno pravilo s tri člana 23
Slika 15 - Rekombinacija u točki, lijevo - par roditelja, desno - par potomaka 24
Slika 16 - Bit-flip mutacija 24
Slika 17 - Dovršena igra 26
Slika 18 - Linija prikaza smjera 27
Slika 19 - Prikaz krutih granica igre unutar prozora scene 28
Slika 20 - Prikaz glavne scene s gumbom pauze 30
Slika 21 - Izbornik pauze 31
Slika 22 - Komponenta gumba te značajka *On Click ()* 31
Slika 23 - Okružje glavnog izbornika 32
Slika 24 - Okružje uputa za igru 32
Slika 25 - Houghova transformacija (lijevo – rubovi, desno – kružnice konstantnog radijusa za svaku točku ruba) 35
Slika 26 - Rezultati učenja (lijevo - prosječna funkcija podobnosti generacije, desno – broj poteza najbolje jedinke generacije) 36

POPIS TABLICA

Tablica 1 - prikaz i jednadžbe aktivacijskih funkcija..... 15

POPIS OZNAKA

Oznaka	Jedinica	Opis
$f(i,j)$	-	vrijednost piksela ulazne slike F na poziciji i, j
$g(i,j)$	-	vrijednost piksela izlazne slike G na poziciji i,j
K	-	skalarna konstanta množenja
L	-	skalarna konstanta zbrajanja
K	-	matrica jezgre
$f(x)$	-	Gaussova funkcija
σ	-	varijanca Gaussove funkcije
μ	-	srednja vrijednost Gaussove funkcije
E	-	Harrisova funkcija
w	-	matrica prozora
v	-	pomak
u	-	pomak
$I(x,y)$	-	vrijednost piksela u točki x, y
z_1	-	Ulaz u aktivacijsku funkciju prvog neurona prvog skrivenog sloja
$\theta_{v,1}$	-	pomak prvog neurona u skrivenom sloju
x_i	-	i-ti neuron ulaznog sloja
$v_{1,i}$	-	težina i-tog neurona ulaznog sloja koja utječe na prvi neuron skrivenog sloja
$\vec{x}_i$	-	ulaz skupa za učenje
$\vec{y}_i$	-	željeni izlaz iz mreže za ulaz $\vec{x}_i$
w_{ij}^k	-	težine između čvora j u sloju l_k i čvora i u sloju l_{k-1}
b_i^k	-	pomaci čvora i u sloju l_k
X	-	skup podataka veličine N ulazno-izlaznih parova
$E(X,\theta)$	-	funkcija greške
$\hat{\vec{y}}_i$	-	izračunati izlaz neuronske mreže
α	-	stopa učenja
θ^t	-	skup parametara neuronske mreže u iteraciji t
a_i^k	-	zbroj sume produkata težina i aktivacija sloja l_{k-1} za čvor i u sloju l_k s pomakom čvora i u sloju l_k
r_{k-1}	-	broj čvorova u sloju l_{k-1}
o_j^{k-1}	-	izlaz čvora j u sloju l_{k-1}
δ	-	greška
$g_o(x)$	-	aktivacijska funkcija izlaznog sloja
p_i	-	vjerojatnost svake jedinke da bude odabrana kao roditelj
f_i	-	vrijednost funkcije podobnosti za jedinku i

SAŽETAK

Tema ovog završnog rada je modeliranje računalne igre u koja se sastoji od 3 kruga u ravnini. Samo modeliranje ostvareno je u programskoj okolini Unity. Nakon modeliranja igre potrebno je izraditi sustav koji ovisno o pozicijama krugova automatski odigrava poteze umjesto ljudskog igrača. U tome služe OpenCV (knjižnica programskih funkcija usmjerena na računalni vid) te neuronska mreža čije se učenje izvršava pomoću genetskog algoritma.

Ključne riječi: Unity okolina, umjetna inteligencija, računalni vid, neuronske mreže, genetski algoritam.

SUMMARY

The subject matter of this undergraduate thesis is modeling a game consisted of three circles on a screen. The modeling itself is achieved inside of Unity environment. After modelling the game, the task is to make a system which, based on position of circles, automatically plays a turn instead of a human player. For that purpose system uses OpenCV (library of programming functions aimed towards computer vision) and neural network whose learning is based on genetic algorithm.

Key words: Unity environment, artificial intelligence, computer vision, neural networks, genetic algorithm.

1. UVOD

Za napredak polja strojarstva zaslužne su i ostale grane znanosti te inženjerstva.

Primjena umjetne inteligencije na područje konstruiranja omogućuje u krajnjem pogledu mehanički jednostavniju konstrukciju obzirom na broj pomičnih dijelova (samim time i smanjenje vibracija) te ubrzava sustav. Uz zadavanje uvjeta i ograničenja, proces dizajniranja konstrukcije može se prepustiti sustavima potpomognutim ili u potpunosti baziranim na umjetnoj inteligenciji.

Kada je riječ o igranju igara, umjetna inteligencija pojavljuje se početkom 50-ih godina 20. stoljeća. Prvi softver koji je uspio usavršiti neku igru (eng. *Tic-Tac-Toe*, hrv. Križić-Kružić) razvio je A.S. Douglas kao dio doktorske disertacije.

Alan Turing, smatran ocem računarske znanosti, prenamijenio je Minimax algoritam u svrhu igranja šaha. Početkom godine 1996. IBM-ovo računalo Deep Blue nadvladalo je trenutnog svjetskog prvaka (tada je to bio Gari Kasparov) po prvi puta u partiji šaha. U današnje doba, najbolji svjetski igrači praktički ne mogu konkurirati sustavima umjetne inteligencije.

1.1.Pregled poglavlja

Rad je podijeljen u sljedeća poglavlja:

- Prvo poglavlje stavlja u korelaciju polje strojarstva i umjetne inteligencije te govori o počecima primjene i kratko prolazi kroz povijest igranja igara od strane umjetne inteligencije.
- Drugo poglavlje, *Unity okolina*, predstavlja osnovne principe rada u programskoj okolini *Unity*.
- Treće poglavlje, *Računalni vid*, prikazuje povijesni pregled polja računalnog vida te osnovne metode za obradu slike (filtri, detekcija rubova).
- U četvrtom poglavlju, *Neuronske mreže,* prikazan je princip rada neuronskih mreža, sastavljena je tablica najčešće korištenih aktivacijskih funkcija te je opisano općenito učenje neuronskih mreža, kao i metoda učenja putem povratne propagacije.
- U petom poglavlju, *Genetski algoritam*, kratko je prikazana sfera evolucijskih algoritama te je razrađen genetski algoritam po njegovim koracima.
- Šesto poglavlje, *Modeliranje igre unutar Unity okoline,* prikazuje cjelokupni tok rada unutar *Unity* okoline kako bi se razvila igra, kao i razvoj svih potrebnih skripti.

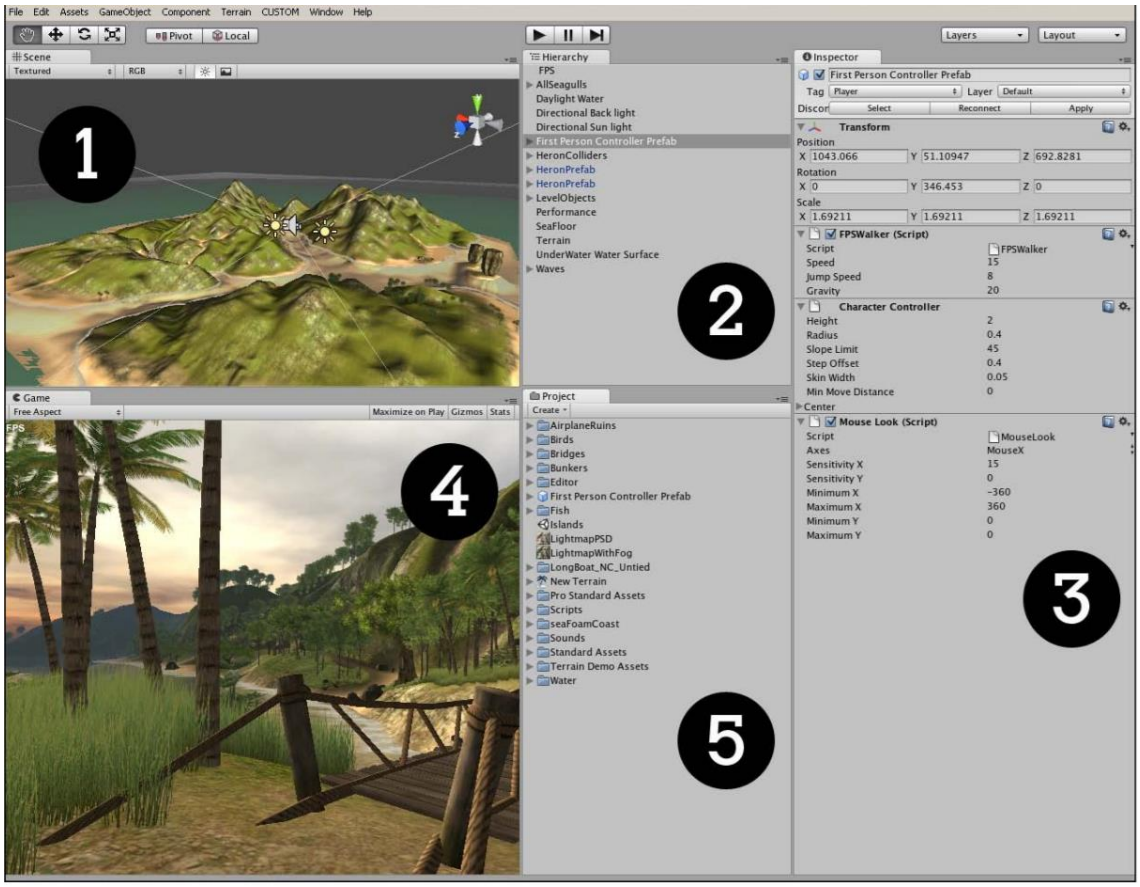
- Sedmo poglavlje, *Sustav automatskog igranja*, opisuje razvoj sustava računalnog vida za prepoznavanje objekata igre te implementaciju neuronske mreže i algoritma njenog učenja, te definira odnos između sustava računalnog vida i sustava učenja.
- Osmo poglavlje, *Zaključak*, komentira modeliranje igre i rezultate učenja (koji su prikazani u sedmom poglavlju) te predlaže rješenja za napredak algoritma učenja.
- Nakon zaključka, prikazan je popis literature te prilog koji sadrži skripte korištene za izradu igre te sustava učenja, kao i grafički prikaz rezultata.

2. UNITY OKOLINA

Razvijen od strane Unity Software Inc, Unity je okolina za razvoj i testiranje projekata računalnih igara, animacija, filmova ili projekata inženjerske namjene. Na intuitivan način, unutar okruženja mogu se u interakciju postavljati objekti grafike, fizike, zvuka, animacija i mnoštvo ostalih.

2.1. Elementi Unity okoline

Slika 1 prikazuje pet osnovnih elemenata grafičkog korisničkog sučelja okoline.



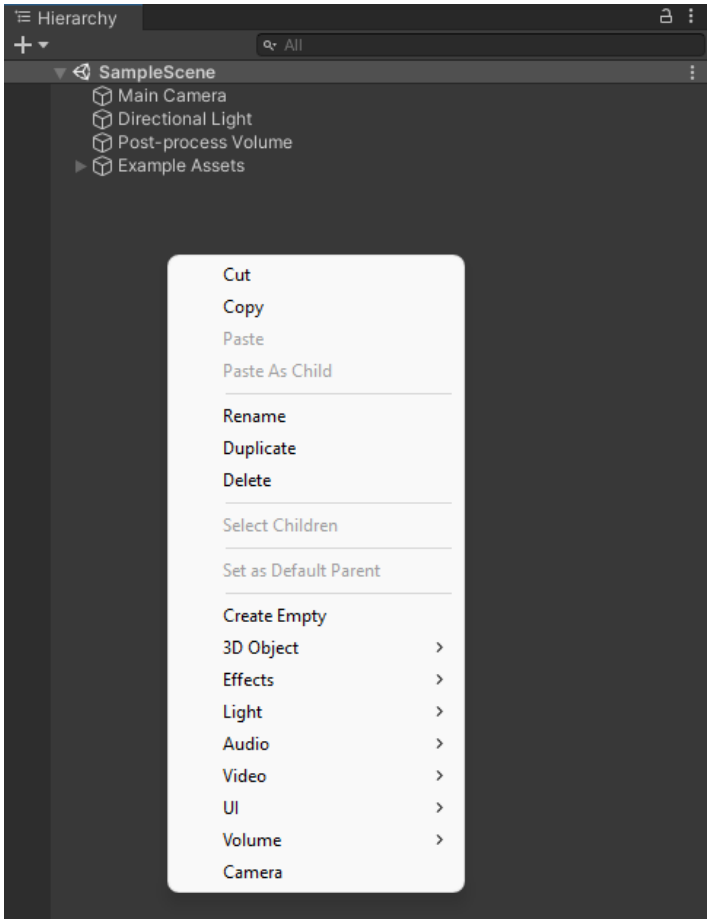
Slika 1 - Grafičko korisničko sučelje Unity okoline [1]

2.1.1. Scena

Područje scene (Slika 1, broj 1) omogućuje upravljanje objektima unutar igre, kao i kamerom te osvjetljenjem. Elementi se mogu pomicati, rotirati, skalirati, moguće im je promijeniti veličinu itd. Sve navedeno moguće je izvršiti po odabranoj osi ili slobodnom smjeru. Unutar scene postoji mogućnost rotiranja i uvećavanja pogleda na konstrukciju igre što omogućuje detaljno promatranje svakog objekta te korelacije s okolinom istog.

2.1.2. Hijerarhija

Unutar hijerarhije (Slika 1, broj 2) izlistani su svi objekti igre koji su instancirani. Objekti mogu stajati samostalno, no postoji i mogućnost stavljanja objekata međusobno u roditelj-dijete strukturu (eng. *parent-child*) unutar koje je ugniježđeni objekt unutar hijerarhije dijete. Mijenjanjem pozicije, orijentacije, veličine ili sličnog unutar roditelja automatski utječe i na dijete.



Slika 2 - Mogućnosti unutar hijerarhije

Slika 2 između ostalog prikazuje skupine objekata koje je moguće dodati u hijerarhiju. Skupina *UI* služi većinom za stvaranje menija unutar igre, sadrži objekte tipa tekst, slika, gumb, pokretna poluga, polje unosa, panel, platno, padajući izbornik itd. Skupina *3D Object* služi za dodavanje objekata raznih 3D tijela, a daljnom rekonstrukcijom te manipuliranjem 3D tijela mogu se dobiti i 2D oblici.

2.1.3. Preglednik

Osnovni sastavni element objekta igre jest komponenta, a vrijednost elementa komponente mijenja se u pregledniku (Slika 1, broj 3). Zadana komponenta svakog objekta jest transformacija gdje se također može modificirati pozicija, razmjer, orijentacija itd. Za razliku od scene, u hijerarhiji se eksplicitno unosi vrijednost u polje koje je potrebno mijenjati.

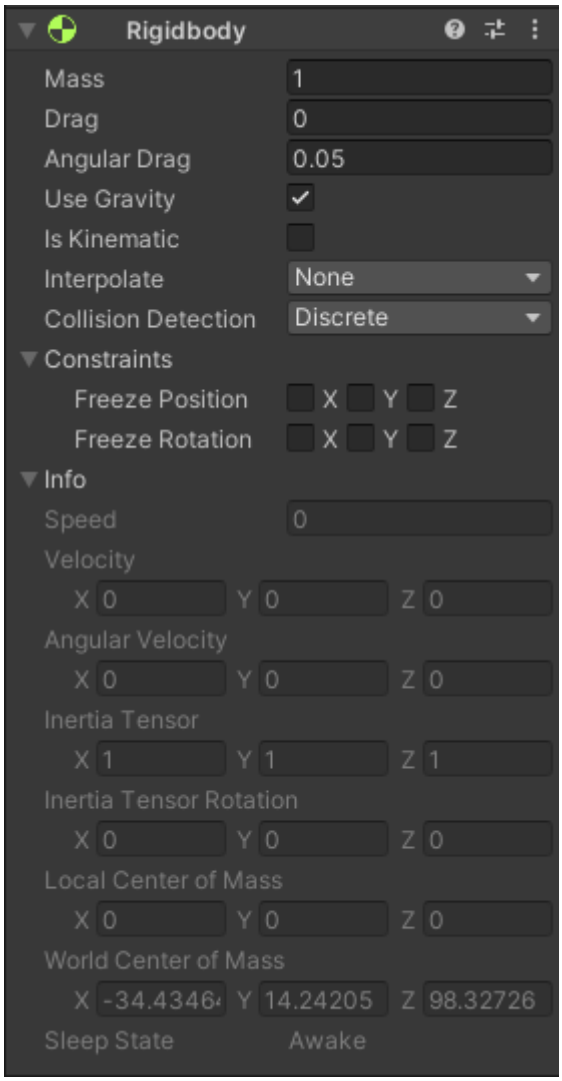
Neke od osnovnih skupina komponenti i komponente:

- Skupina efekti
  - Prikazatelj linija (eng. *Line renderer)* – Prikazuje liniju između seta zadanih točki, moguće je mijenjati debljinu, stil, boju linije itd.
  - Vizualni efekt

- Skupina fizika
  - Kruto tijelo (eng. *Rigid body*) – Ukoliko objekt tijekom igre ostvaruje interakciju s ostalim objektima iz okoline, njegova svojstva (poput mase, trenja, utjecaja gravitacije i ostalo) moficiraju se unutar komponente *Rigidbody* (Slika 3).

- Skupina mreža (eng. *mesh*)
  - Prikazatelj mreže – Komponenta zaslužna za krajnji prikaz površine objekta. Unutar komponente određuje se materijal objekta.

- Skripte – Esencijalni dio razvijanja igre, unutar Unitya smatran kao komponenta radi jednostavnijeg tijeka rada. Za pisanje skripti mogu biti korišteni jezici C#, JavaScript ili Boo. Za ubrzavanje procesa rada koristi klasa *Monobehaviour* iz koje mogu proizlaziti klase koje stvara korisnik, omogućuje pristup i modificiranje komponenti i objekata igre unutar skripte.



Slika 3 - Komponenta *Rigidbody*

2.1.4. Igra

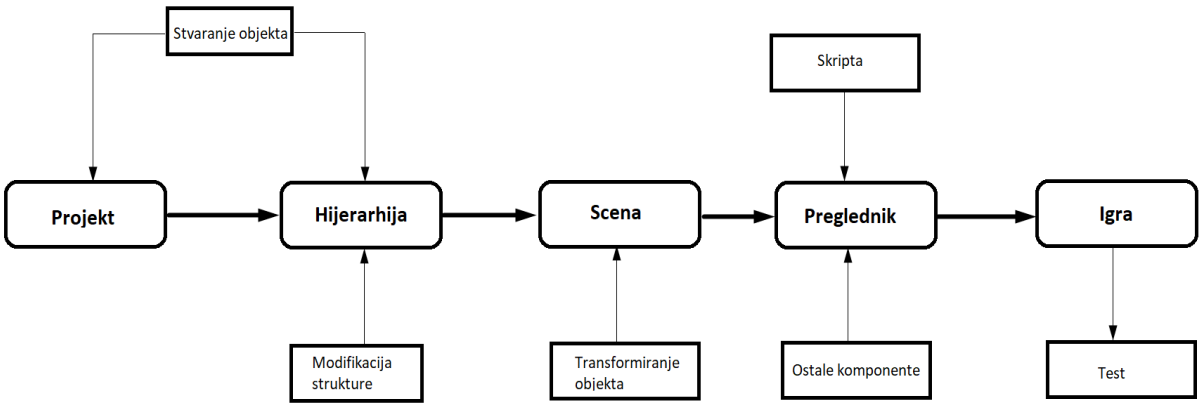
Klikom na tipku *Play* unutar okoline, pokreće se prozor igre (Slika 1, broj 4) koji služi kao realističan test igre tijekom razvoja iste. Unutar njega mogu se mijenjati razmjeri rezolucije kako bi korisnik provjerio izgled igre na različitim platformama te uređajima. Bilo koje izmjene u igri tijekom pokrenutog testa biti će vraćene na početno stanje jednom kada test završi.

2.1.5. Projekt

Element projekt (Slika 1, broj 5) sadrži svaku datoteku i objekt unutar projekta igre bez obzira bio on inicijaliziran unutar igre ili ne. Osim fontova, materijala, korisnikovih modificiranih objekata, scena, skripti i slika, unutar projekta nalaze se i zadani te instalirani paketi okoline.

2.2.Tok rada unutar okoline

Tipičan tok rada unutar okoline jest da se objekt uvede u hijerarhiju iz projekta (ili se stvori unutar hijerarhije), nakon toga da se modificira unutar scene i dodaju mu se komponente unutar preglednika.



Slika 4 - Dijagram procesa rada unutar Unity okline

3. RAČUNALNI VID

Računalni vid je polje umjetne inteligencije koje omogućuje računalima i sistemima razaznati značajne informacije iz digitalnih slika, videozapisa te ostalih vizualnih izvora, poduzeti radnje ili dati preporuke ovisno o dobivenoj informaciji. Za razliku od čovjeka, koji se koristi mrežnicom, optičkim živcima te vizualnim korteksom, računala „vide“ pomoću kamera, podataka i algoritama.

3.1.Povijesni pregled područja

Početkom razvoja područja računalnog vida smatra se eksperiment iz godine 1959. kada je skupina neurofiziologa prikazala mački set slika u pokušaju otkrivanja njenog odziva na različite oblike i ulaze. Otkriveno je kako je mačka prvo reagirala na oštre rubove i linije što je značilo da općenito obrada slike počinje istim elementima.

Otprilike istovremeno, razvijena je prva tehnologija skeniranje slike koja je omogućila računalu da prihvati i prevede sliku u digitalni oblik. Godine 1963. računala su po prvi put uspjela transformirati 2D slike u 3D oblike.

Umjetna inteligencija kao akademsko područje pojavljuje se 60-ih godina 20. stoljeća i od tada počinje zadatak da računala interpretiraju ljudski vid.

U godini 1974. predstavljena je prva tehnologija prepoznavanja znakova (OCR) koja je mogla prepoznati tekst bilo kojeg stila i fonta. Tehnologija inteligentnog prepoznavanja znakova (ICR) koristi neuronske mreže kako bi obavila zadatak. Nakon savladavanja prepoznavanja znakova krenulo se u druga područja kao što su procesiranje dokumenata, prepoznavanje registracijskih oznaka, strojno prevođenje itd.

Neuroznanstvenik David Marr je 1982. godine ustanovio da vid funkcionira hijerarhijski i uveo algoritme za otkrivanje rubova, kutova, krivulja i drugih oblika, dok je računalni znanstvenik Kunihiko Fukushima razvio mrežu stanica koje mogu raspoznati uzorke.

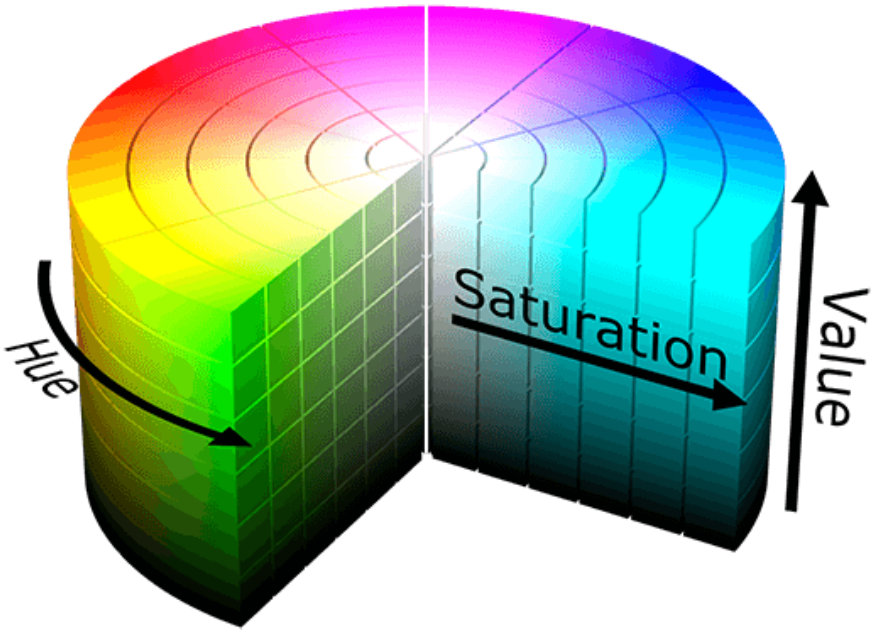
Do 21. stoljeća fokus područja bio je na prepoznavanju objekata, a godine 2001. predstavljene su prve aplikacije za prepoznavanje lica.

U današnje vrijeme računalni vid nalazi primjenu u poljima autonomnih vozila, inteligentnim transportnim sustavima, medicini, proizvodnoj industriji, prometu i dr.

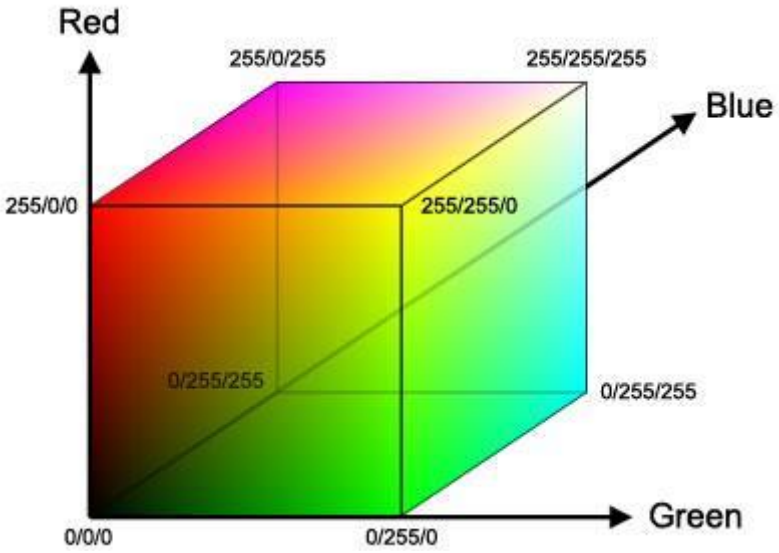
3.2. Obrada slike

Osnovni element u obradi digitalne slike jest piksel, stoga svaka slika poprima oblik matrice s različitim vrijednostima piksela. Najpopularniji prostori boja su RGB i HSV.

Prostor RGB predstavlja spektar boja preko vrijednosti crvene (eng. *Red*), zelene (eng. *Green*) i plave (eng. *Blue*), svake u rasponu od 0 do 255, dok je HSV prostor boja interpretiran u obliku valjka gdje je svaka boja predstavljena kutom (eng. *Hue*), vrijednošću (eng. *Value*) te zasićenošću (eng. *Saturation*). Slika se može prikazati u nijansama sive u rasponu od 0 do 255 gdje 0 označava crnu, a 255 bijelu boju.



Slika 5 - HSV prostor boja



Slika 6 - RGB prostor boja

3.2.1. Filtri

Filtri modificiraju sliku u svrhu davanja željene informacije ili da postane korisna ostalim zadacima sustava. Mogu obavljati funkciju otklanjanja šumova iz slike, izdvajanja rubova, zamagljenja slike, otklanjanja neželjenih objekata itd.

Potreba za filtrima javlja se pošto razni faktori uzrokuju šumove, npr. slika koja prikazuje veoma osvjetljeno područje sadrži mnogo svijetlog i mnogo tamnog područja, stoga sustav računalnog vida može doći do krivih zaključaka.

3.2.1.1. Linearni filtri

Najjednostavnija vrsta filtra gdje je svaka vrijednost piksela pomnožena skalarom, zapisuje se u sljedećem obliku:

$$g(i,j) = K \times f(i,j) \quad (1)$$

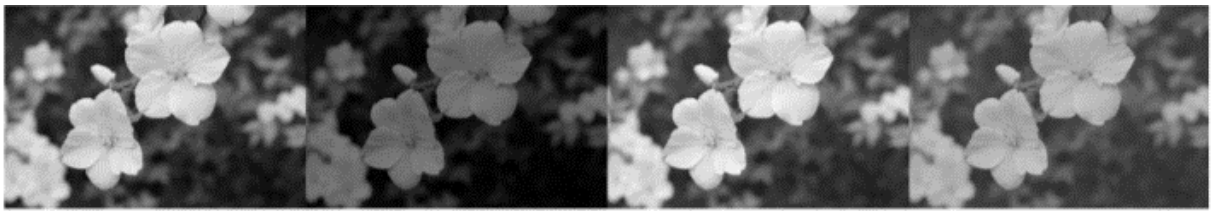
Gdje su:

- $f(i,j)$ - vrijednost piksela ulazne slike F na poziciji i,j
- $g(i,j)$ - vrijednost piksela izlazne slike G na poziciji i,j
- K – skalarna konstanta množenja.

Svakom pikselu, osim množenja sa skalarom, može biti zbrojena ili oduzeta skalarna vrijednost:

$$g(i,j) = K \times f(i,j) + L \quad (2)$$

Gdje je L skalarna konstanta zbrajanja.



Slika 7 - Primjena linearnog filtra [4]

Na slici 7, ulazna slika je prva s lijeva (prikazana u nijansama sive), a parametri filtra na slikama desno su kako slijedi:

- K = 0,5 L=0
- K = 0,7 L=10
- K = 0,7 L=25

3.2.1.2. 2D linearni filtri

Prethodna vrsta filtra bila je zasnovana na točkama, no pikseli slike imaju informacije o pikselima u svojoj okolini. U principu, 2D linearni filtri koriste matricu jezgre (eng. *kernel*) koja se superponira na originalnoj slici, zatim se uzima umnožak odgovarajućih piksela i vraća zbroj svih umnožaka. Proces se ponavlja pomicanjem matrice jezgre duž redaka i stupaca slike.

3.2.1.3. Gaussov filter

Koristi se za uglađivanje slika, a efekt se postiže umanjivanjem vrijednosti elemenata matrice jezgre prema njenim rubovima. Tipičan primjer matrice jezgre 5x5 dan je u sljedećoj jednadžbi:

$$\mathbf{K} = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \tag{3}$$

Elementi matrice određuju se po Gaussovoj razdiobi, čija je formula sljedeća:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \times \mathrm{e}^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{4}$$

Gdje je σ varijanca, a μ srednja vrijednost.

Matrica **K** koristi se u normaliziranoj formi, a efekt uglađivanja također ovisi o veličini matrice. Primjena Gaussovog filtra uklanja komponente visoke frekvencije što rezultira uklanjanjem oštrih rubova.

Slika 8 – Lijevo: ulazna slika, desno: Gaussov filtar primijenjen na ulaznu sliku [4]

3.2.2. Detekcija kuteva

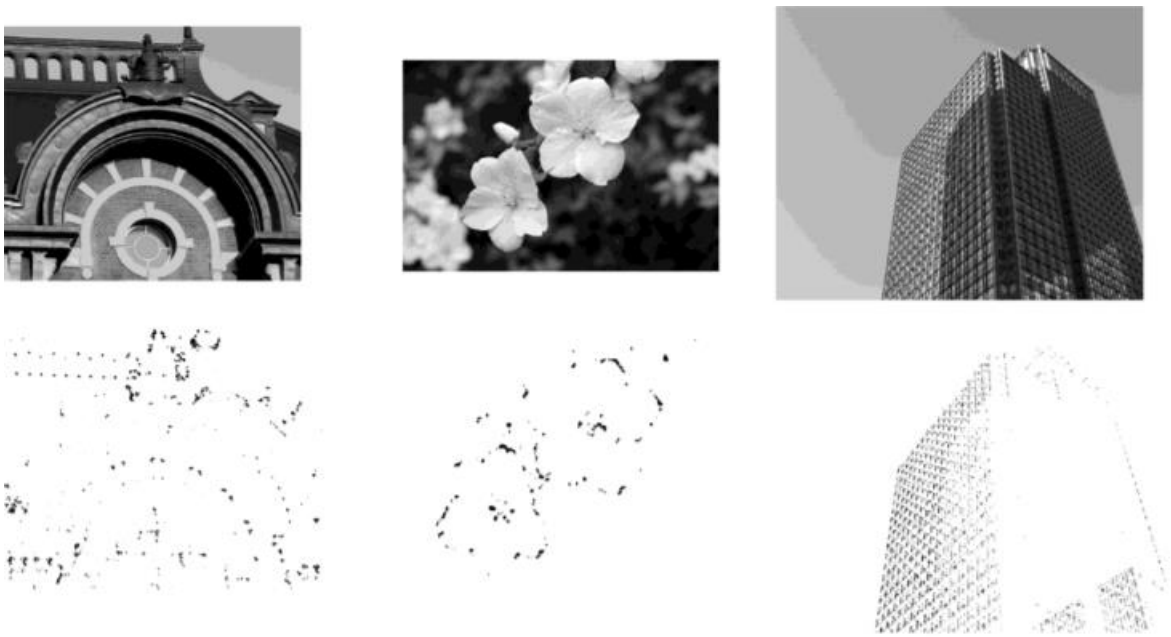
Za detekciju kuteva može se koristiti Harrisova detekcija. Prvo se definira matrica prozora, manja od veličine ulazne slike. Generalna ideja je prekriti dio matrice ulaza matricom prozora i promatrati prekriveni dio matrice ulaza. Matrica prozora nakon promatranja pomiče se nakon promatranja dalje dok ne prijeđe cijelu ulaznu sliku. Moguća su tri različita scenarija tijekom pomicanja:

1. Ukoliko je površina ravna, ne vidi se promjena u području prozora, neovisno o smjeru kretanja (jer ne postoje niti rubovi niti kutevi).
2. Ako je matrica prozora prekrila ulaznu sliku na dijelu ruba, područje prozora ne mijenja se pomicanjem jedino u smjeru ruba.
3. Ukoliko se u području prozora nalazi kut koji je presjek dva ruba, najvjerojatnije će doći do promjene pomicanjem.

Matematički opis funkcije dan je sljedećom jednadžbom:

$$E[u,v] = \sum_{x,y} w(x,y)[I(x+u)(y+v) + I(x,y)] \tag{5}$$

Gdje je w prozor, u i v pomaci, $I(x,y)$ vrijednost piksela u točki x, y.

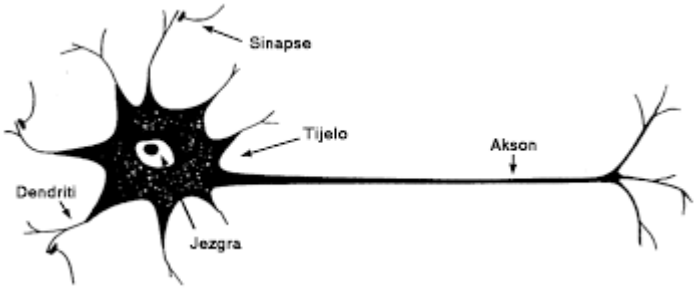


Slika 9 - Vizualizacija rezultata Harrisove detekcije [4]

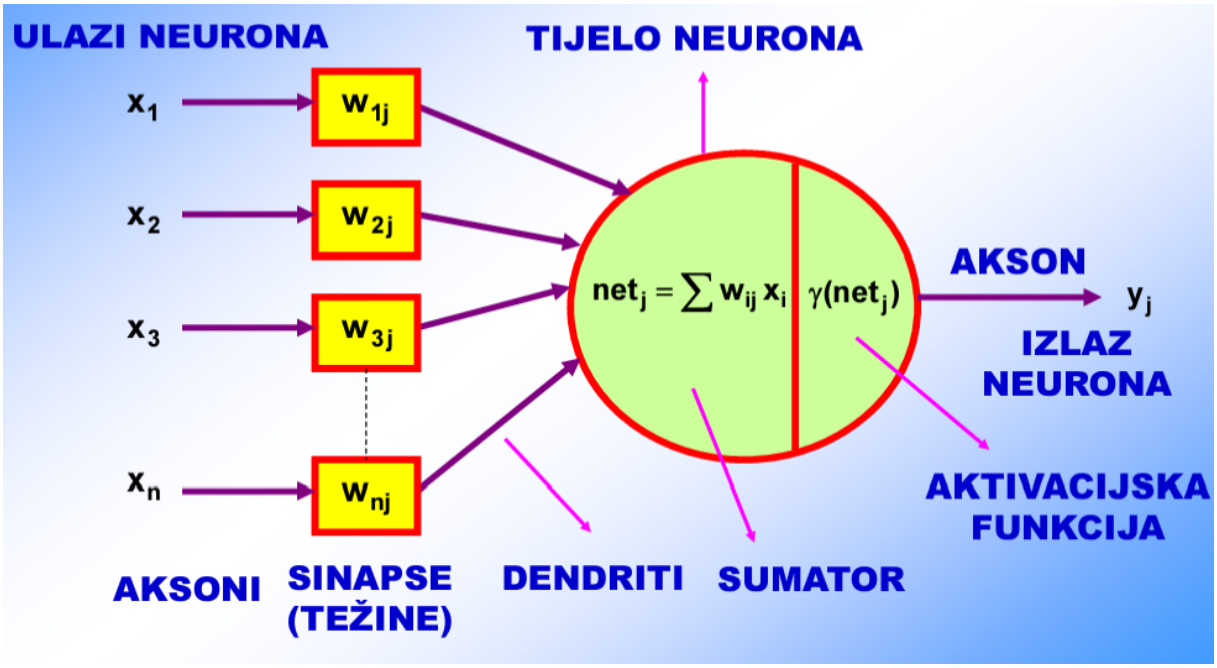
4. NEURONSKE MREŽE

Neuronske mreže (poznate i pod nazivom umjetne neuronske mreže) su podskup područja strojnog učenja i srž algoritama dubokog učenja. Ime i struktura inspirirani su ljudskim mozgom – oponašaju način kako umjetni neuroni komuniciraju.

Svaka umjetna neuronska mreža sastavljena je od slojeva s čvorovima te obavezno sadrži ulazni sloj, jedan ili više skrivenih slojeva te izlazni sloj. Svaki čvor (umjetni neuron) spojen je sa svakim neuronom iz sljedećeg sloja (ukoliko postoji) preko veze koja sadrži težinu. Također, da bi se dodatno oblikovali izlazi iz neurona, koriste se pomaci – skalarne veličine koje se dodaju umnošku neurona iz prethodnog sloja i njihovih težina.

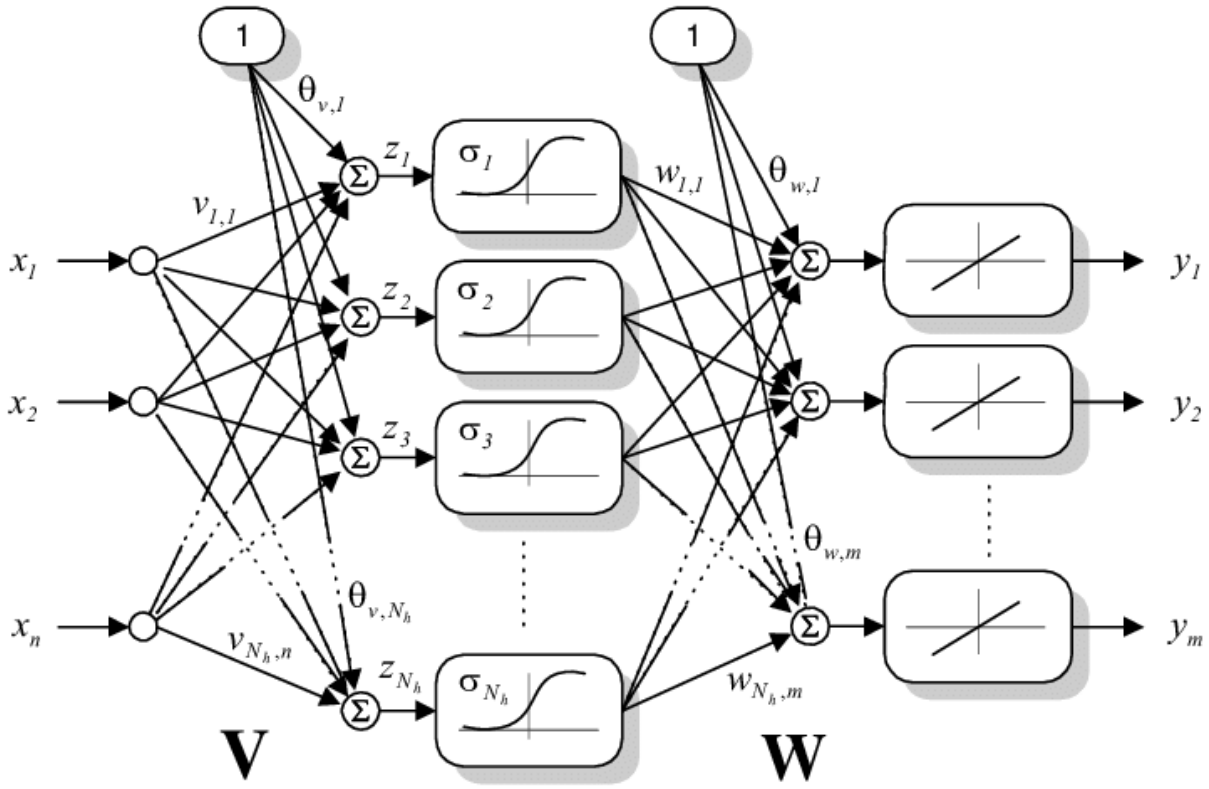


Slika 10 - Struktura biološkog neurona



Slika 11 - Struktura umjetnog neurona [5]

4.1.Princip rada



Slika 12 - Struktura neuronske mreže

Na prethodnoj slici prikazana je umjetna neuronska mreža s n ulaza (x_1 do x_n), skrivenim slojem od N_h neurona i m izlaza (y_1 do y_m). Svaki neuron s ulaza množi se s onoliko težina koliko je neurona u sljedećem sloju, a svaki neuron skrivenog i izlaznog sloja zbraja odgovarajuće težine te pomak.

Za prvi neuron iz skrivenog sloja:

$$z_1 = 1 \cdot \theta_{v,1} + \Sigma_{i=1}^{n} x_i \cdot v_{1,i} \tag{6}$$

Gdje su:

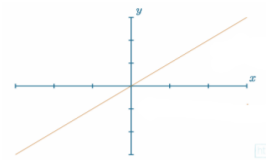
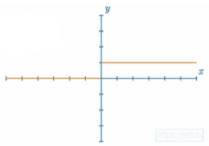
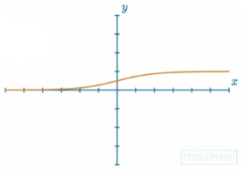
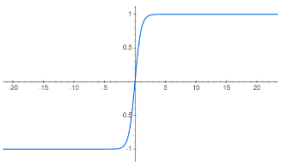
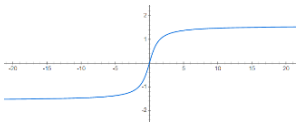
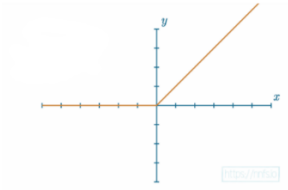
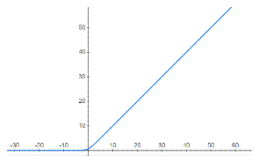
- $\theta_{v,1}$ – pomak prvog neurona u skrivenom sloju
- x_i – i-ti neuron ulaznog sloja
- $v_{1,i}$ – težina i-tog neurona ulaznog sloja koja utječe na prvi neuron skrivenog sloja

Nakon provedbe sumacije, iznos svakog neurona prolazi kroz aktivacijsku funkciju, na slici 12 to je sigmoid aktivacijska funkcija (σ_1, σ_{N_h}) za skriveni sloj te linearna aktivacijska funkcija za izlazni sloj. Ako neuron ne zadrži aktivacijsku funkciju, naziva se ADALINE(eng. *Adaptive Linear Neuron*). ADALINE zapravo samo računa težinsku sumu ulaza.

4.2.Aktivacijske funkcije

Aktivacijske funkcije koriste se kako bi oblikovale izlaz iz neurona. Kako bi neuronska mreža mogla mapirati nelinearne funkcije, koriste se nelinearne aktivacijske funkcije u najmanje 2 skrivena sloja (najčešći slučaj).

Tablica 1 - prikaz i jednadžbe aktivacijskih funkcija

Ime	***Prikaz***	***Jednadžba***
Linearna		$f(x) = x$
Funkcija skoka ***(Funkcija praga)***		$f(x) = \begin{cases} 0 \; za \; x < 0 \\ 1 \; za \; x \geq 0 \end{cases}$
Sigmoid		$f(x) = \dfrac{1}{1 + e^{-x}}$
Tangens hiperbolni		$f(x) = tanh\,(x) = \dfrac{2}{1 + e^{-2x}} - 1$
Arkus tangens		$f(x) = tan^{-1}(x)$
Ispravljena linearna jedinična (ReLU)		$f(x) = \begin{cases} 0 \; za \; x < 0 \\ x \; za \; x \geq 0 \end{cases}$
SoftPlus		$f(x) = ln\,(1 + e^{x})$

Da bi se imitirao rad ljudskog mozga (aktivirana ili neaktivirana veza), u prošlosti je korištena funkcija skoka, no pošto je davala šture informacije (nije bilo sigurno koliko je neuron bio blizu aktivacije ili deaktivacije) pribjeglo se sigmoid funkciji. Optimiranje mreže nakon toga postaje fleksibilnije.

ReLU aktivacijska funkcija korisna je i raširena najviše zbog brzine računanja vrijednosti funkcije te učinkovitosti.

4.3.Postupak učenja mreže

U manjim mrežama koje obavljaju zadatke interpretacije unaprijed zadanih logičkih funkcija i koriste „jednostavniju“ aktivacijsku funkciju (npr. funkcija skoka) postoji mogućnost praćenja rada mreže u svakom koraku. U slučaju većih mreža i složenijih aktivacijskih funkcija gubi se zor nad načinom obrade podataka mreže. U takvim situacijama, prvo se definira arhitektura mreže i nakon toga se obavlja postupak učenja ili treniranja mreže. Specifičnost učenja neuronske mreže jest ta što su informacije o postupku učenja implicitno spremljene u težine i pomake mreže u svakom koraku učenja. Učenje se provodi sve dok rezultat izlaza iz mreže, provjeren na skupu podataka, ne bude zadovoljavajuć.

Ovisno o vrsti podataka, postoje dva načina učenja mreže:

1. Nadgledano učenje (eng. *supervised learning*) – za svaki ulazni skup postoji očekivani par u vidu izlaza iz mreže
2. Nenadgledano učenje (eng. *unsupervised learning*) – učenje se provodi bez poznatog izlaza za ulazni skup

Učenje mreže često se odvija s tri odvojena skupa podataka – skup za učenje, skup za testiranje te skup za validaciju. Podaci iz skupa za učenje služe za podešavanje težinskih faktora, primjeri iz skupa za testiranje ulaze u mrežu (s trenutnim težinskim faktorima) kako bi se proces učenja zaustavio u trenutku degradacije performanski. Točnost se naposlijetku provjerava skupom za validaciju.

Svaki korak učenja gdje se podešavaju težinski faktori naziva se iteracija, a cjelokupni skup za učenje naziva se epoha.

Ovisno o broju primjera koje mreža prihvaća tijekom jedne iteracije razlikuje se pojedinačno te grupno učenje. Tijekom pojedinačnog učenja (eng. *on-line training*) tijekom jedne iteracije

mreži se predočava samo jedan primjer za učenje, dok se kod grupnog učenja (eng. *off-line training*) u jednoj iteraciji mreži predočavaju svi primjeri za učenje.

4.3.1. Povratna propagacija

Nadgledano učenje moguće je provesti pomoću povratne propagacije koristeći gradijentni spust. Suština rješavanja problema leži u minimizaciji funkcije greške. Riječ „povratna“ stoji u imenu metode pošto se najprije izračuna gradijent izlaznog sloja, a nakon toga se računa za pretposlijednji sloj i tako redom unatraške po slojevima duž cijele mreže.

Uvjeti za kalkulaciju putem povratne propagacije:

1. **Skup podataka za učenje** sadržan od ulazno-izlaznih parova $(\vec{x}_i, \vec{y}_i)$ gdje je $\vec{y}_i$ željeni izlaz iz mreže za ulaz $\vec{x}_i$. Skup podataka veličine N ulazno-izlaznih parova označava se s $X = \{(\overrightarrow{x_1}, \overrightarrow{y_1}), \dots, (\overrightarrow{x_N}, \overrightarrow{y_N})\}$.
2. **Unaprijedno aktivirana neuronska mreža** (ona u kojoj veze između čvorova ne čine krug) čiji je cjelokupni skup parametara označen s θ. Primarni interes u povratnoj propagaciji zauzimaju težine w_{ij}^k (težine između čvora j u sloju l_k i čvora i u sloju l_{k-1}) i pomaci b_i^k(pomaci čvora i u sloju l_k).
3. **Funkcija greške**, $E(X, \theta)$ – definira iznos greške između željenog izlaza $\vec{y}_i$ i izračunatog izlaza $\hat{\vec{y}}_i$ neuronske mreže za ulaz $\vec{x}_i$ za set ulazno-izlaznog para $(\vec{x}_i, \vec{y}_i) \in X$ i parametara mreže θ u danoj iteraciji.

Učenje mreže pomoću gradijentnog spusta zahtjeva izračun gradijenta funkcije greške u odnosu na težine w_{ij}^k i pomake b_i^k. Nakon toga, ovisno o stopi učenja α, svaka iteracija učenja utječe na parametre mreže (θ):

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X, \theta^t)}{\partial \theta} \tag{7}$$

Gdje je θ^t skup parametara neuronske mreže u iteraciji t gradijentnog spusta.

Funkcija greške zapravo je srednja kvadrirana pogreška (eng. *mean squared error*):

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \tag{8}$$

gdje je y_i ciljana vrijednost izlaza mreže za ulazno-izlazni par $(\overrightarrow{x_i}, y_i)$, a $\hat{y}_i$ izračunata vrijednost izlaza mreže za ulaz $\overrightarrow{x_i}$.

U izvođenju algoritma povratne propagacije koristi se lančano pravilo i pravilo umnoška diferencijalnog računa. Primjena navedenih pravila ovisna je o derivaciji aktivacijske funkcije, stoga diskontinuirane funkcije ne dolaze u obzir (npr. funkcija skoka).

Konvencija – pomak b_i^k za čvor i u sloju l_k ukomponiran je u težine kao w_{0i}^k s fiksim izlazom nultog čvora sloja l_{k-1} iznosa 1 ($o_0^{k-1} = 1$):

$$w_{0i}^k = b_i^k \tag{9}$$

Tako suma produkata čvora zbrojena s pomakom (član koji ulazi u aktivacijsku funkciju) postaje:

$$a_i^k = b_i^k + \sum_{j=1}^{r_{k-1}} w_{ji}^k o_j^{k-1} = \sum_{j=0}^{r_{k-1}} w_{ji}^k o_j^{k-1} \tag{10}$$

Gdje su:

- a_i^k – zbroj sume produkata težina i aktivacija sloja l_{k-1} za čvor i u sloju l_k s pomakom čvora i u sloju l_k
- r_{k-1} – broj čvorova u sloju l_{k-1}
- w_{ji}^k – težina između čvora j u sloju l_{k-1} te čvora i u sloju l_k
- o_j^{k-1} – izlaz čvora j u sloju l_{k-1}.

Funkcija greške minimizira računanjem $\frac{\partial E}{\partial w_{ij}^k}$ za svaku težinu w_{ij}^k. Budući da se za skup od N ulazno-izlaznih parova funkcija greške može izračunati posebno za svaki ulazno-izlazni par, spomenuta derivacija može se također računati za svaki par i na kraju zbrojiti (derivacija sume jednaka je sumi derivacija):

$$\frac{\partial E(X,\theta)}{\partial w_{ij}^k} = \frac{1}{N}\sum_{d=1}^{N} \frac{\partial}{\partial w_{ij}^k}\left(\frac{1}{2}(\hat{y}_d - y_d)^2\right) = \frac{1}{N}\sum_{d=1}^{N} \frac{\partial E_d}{\partial w_{ij}^k} \tag{11}$$

Za pojednostavljivanje izraza u računanju jedne iteracije, ispušta se indeks d, stoga funkcija greške za jedan ulazno-izlazni par postaje:

$$E = \frac{1}{2}(\hat{y} - y)^2 \tag{12}$$

Za računanje derivacije funkcije greške u odnosu na težinu koristi se lančano pravilo:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{ij}^k} \tag{13}$$

Prvi član poprima nakon toga oblik (naziva se greška):

$$\delta_j^k \equiv \frac{\partial E}{\partial a_j^k} \tag{14}$$

Drugi član, uz pomoć izraza (10) poprima vrijednost:

$$\frac{\partial a_j^k}{\partial w_{ij}^k} = O_i^{k-1}. \tag{15}$$

Izlazni sloj

Za slučaj izlaznog sloja s jednim slojem čvorom, potrebno je odrediti δ_1^m gdje m označava izlazni sloj. Zgodno je preformulirati izraz funkcije greške:

$$E = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(g_o(a_1^m) - y)^2. \tag{16}$$

Gdje je $g_o(x)$ aktivacijska funkcija izlaznog sloja.

Primjenom parcijalne derivacije i korištenjem lančanog pravila:

$$\delta_1^m = (g_0(a_1^m) - y)g_o'(a_1^m) = (\hat{y} - y)g_o'(a_1^m). \tag{17}$$

U konačnici, izraz (13) poprima oblik:

$$\frac{\partial E}{\partial w_{i1}^m} = \delta_1^m o_i^{m-1} = (\hat{y} - y)g_o'(a_1^m)o_i^{m-1}. \tag{18}$$

Skriveni slojevi

Budući da se skriveni slojevi sadrže od više čvorova, greška poprima oblik sume:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial E}{\partial a_l^{k+1}}\frac{\partial a_l^{k+1}}{\partial a_j^k} \tag{19}$$

Vidljivo je kako se u izrazu za grešku u sloju l_k nalazi greška u sloju l_{k+1}:

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1}\frac{\partial a_l^{k+1}}{\partial a_j^k} \tag{20}$$

Izraz koji ulazi u aktivacijsku funkciju u čvoru l u sloju l_{k+1}:

$$a_l^{k+1} = \sum_{j=1}^{r^k} w_{jl}^{k+1} g\left(a_j^k\right) \tag{21}$$

Gdje je $g(x)$ aktivacijska funkcija sloja l_k.

Nakon toga, izraz za parcijalnu derivaciju iz izraza (20) postaje:

$$\frac{\partial a_l^{k+1}}{\partial a_j^k} = w_{jl}^{k+1} g'\left(a_j^k\right) \tag{22}$$

Na posljetku, ubacivanjem (22) u (20):

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1} g'\left(a_j^k\right) = g'\left(a_j^k\right) \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1} \tag{23}$$

Time su se stekli uvjeti za računanje parcijalne derivacije funkcije greške u odnosu na težine za skrivene slojeve:

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} = g'\left(a_j^k\right) o_i^{k-1} \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1} \tag{24}$$

Sada je jasno po čemu je povratna propagacija dobila ime – greška se računa prvo za izlazni sloj, a zatim unatrag po skrivenim slojevima uz pomoć greške iz sljedećeg sloja.

Nakon kalkulacije parcijalne derivacije (24) težine se mijenjaju ovisno o istoj te stopi učenja po iteracijama pomoću gradijentnog spusta sve dok se ne uđe u područje lokalnog minimuma ili se ne ispuni uvjet konvergencije:

$$\Delta w_{ij}^k = -\alpha \frac{\partial E(X, \theta)}{\partial w_{ij}^k} \tag{25}$$

5. GENETSKI ALGORITAM

U mnogim tehničkim izumima čovjek je imitirao prirodu, tako je od promatranja šišmiša i odašiljanja te reflektiranja zvučnih valova nastao radar, oponašane su ribe kako bi se razvila podmornica itd. Prirodna evolucija vrsta može se promatrati kao proces prilagodbe okolini i optimiranja pogodnosti jedinke da preživi. Pandan navedenog u tehničkom svijetu su evolucijski algoritmi. Za evolucijske algoritme kaže se da upravo obavljaju optimizaciju ili uče obavljati zadatke s mogućnošću evolucije.

Postoje 3 glavne značajke:

1. **Populacija** – U biologiji predstavlja grupu jedinki, a u evolucijskim algoritmima grupu rješenja koja paralelno uče rješavati problem ili optimizirati rješenje.
2. **Dobrota/kvaliteta** (eng. *fitness*) - Svaka jedinka (svako rješenje) posjeduje svoj genotip, a evaluacija rješenja naziva se funkcija dobrote/funkcija kvalitete. Evolucijski algoritmi preferiraju one jedinke koje kotiraju visoko u evaluaciji rješenja što je temelj za optimiranje i konvergenciju algoritma.
3. **Varijacija** – Kako bi se prekrio što veći predio prostora rješenja, jedinke prolazi brojne varijacijske operacije koje oponašaju promjene u genima.

Osim strategije evolucije, genetskog programiranja, evolucijskog programiranja, diferencijalne evolucije, vrsta evolucijskog algoritma je i genetski algoritam.

5.1. Infrastruktura genetskog algoritma

Prvi korak u rješavanju problema genetskim algoritmom inicijalizacija je parametara algoritma. Bitno je definirati genotip jedinke, odrediti veličinu populacije (broj jedinki), stopu rekombinacije, stopu mutacije, broj generacija, uvjete za prekid algoritma itd.

Nakon toga potrebno je generirati nultu generaciju od jedinki čiji se genotip sastoji od uniformno raspodijeljenih slučajnih brojeva i evaluirati jedinke kako bi se dobila funkcija podobnosti svake od njih.

Sljedeća faza ponavlja se po generacijama sve dok se ne aktivira uvjet za izlaz iz algoritma (ukoliko postoji) ili se ne dosegne maksimalan broj generacija.

Iz cijele populacije najprije je potrebno slučajnim izborom izabrati jedinke koje će stvarati potomke, a potom iz skupa slučajnih jedinki izabrati parove jedinki koji će izvršiti

rekombinaciju i time stvoriti potomke. Svaki novi potomak podliježe mutaciji te se dodaje u novu populaciju.

Nakon stvaranja potomaka, svaka jedinka nove populacije šalje se na evaluaciju.

Na posljetku, stara populacija zamjenjuje se novom.

```
POČETAK
        INICIJALIZIRAJ populaciju slučajnih kandidata;
        OCIJENI svakog kandidata
        PONAVLJAJ DO (UVJET PREKIDA ZADOVOLJEN)
                1 IZABERI roditelje;
                2 REKOMBINIRAJ parove roditelja;
                3 MUTIRAJ rezultirajuće potomke;
                4 OCIJENI nove kandidate;
                5 IZABERI jedinke za novu generaciju;
        prekid
KRAJ
```

Slika 13 - Pseudo kod genetskog algoritma [11]

Naravno, kako bi se ubrzao proces evolucije, postoje metode koje odstupaju od generalne ideje.

5.2. Selekcija

Jedinke s većom vrijednošću funkcije dobrote posjeduju veću vjerojatnost da postanu roditelji, no čak i one jedinke s najmanjim vrijednostima funkcije dobrote posjeduju (malenu) šansu da postanu izabrane. Upravo stohastički izbor sprječava zapinjanje u lokalnom optimumu tijekom traženja rješenja.

5.2.1. Ruletno pravilo

Naziv za ovaj način odabiranja roditelja dolazi iz popularne igre na sreću – ruleta, koja je također stohastička. Za razliku od ruleta, gdje svako od 37 polja ima istu vjerojatnost da bude odabrano, u ruletnom pravilu genetskog algoritma vjerojatnost da će polje biti odabrano proporcionalna je funkciji dobrote.

Naime, za ruletno pravilo s n članova, prvi korak zahtjeva izbor n jedinki iz postojeće generacije, a nakon toga definiranje vjerojatnosti p_i svake jedinke da bude odabrana:

$$p_i = \frac{f_i}{\Sigma_{i=1}^{n} f_i} \tag{26}$$

Gdje f_i predstavlja vrijednost funkcije podobnosti za jedinku i. Nije teško pokazati kako je $\Sigma_{i=1}^{n} p_i = 1$, a upravo zbog te relacije može se definirati ruletni krug gdje svako polje ima kut $2\pi \cdot p_i$:

Slika 14 - Ruletno pravilo s tri člana [11]

Na ovaj način, moguće je i nekoliko puta u istoj generaciji izabrati iste jedinke za roditelje. Budući da je riječ o stohastičkom odabiru, čak i ako jedinka ima visoku vrijednost funkcije dobrote, a samim time i visoku vjerojatnost da bude odabrana, izbor ne mora pasti na nju. Fenomen gdje se takav slučaj dogodi jedinki do isti broj puta veličine populacije tokom svake generacije naziva se selekcijski pomak.

5.3.Varijacijski operatori

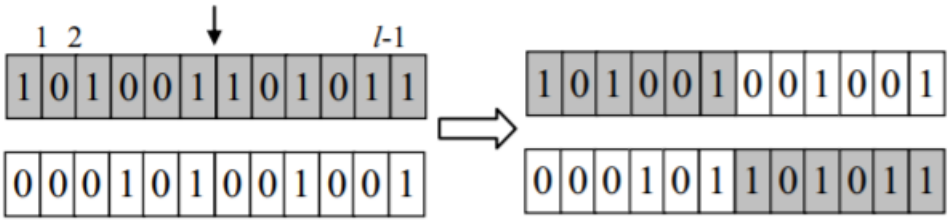
Mnogo je varijacijskih operatora pomoću kojih se mogu izmijeniti informacije izabranih roditelja, no kada je riječ o izmjeni gena među dvjema jedinkama, govori se o rekombinaciji. Ukoliko se mijenja genotip jedinke, u pitanju je mutacija.

5.3.1. Rekombinacija

Postoje dva načina za odabir parova roditelja za rekombinaciju:

1. Promiješati slučajnim redoslijedom skup roditelja iz selekcije te slati parove jedinki u rekombinaciju – jedinka 1 s jedinkom 2, jedinka 3 s jedinkom 4 itd.
2. Stvara se slučajni cijeli broj između 1 i broja jedinki u populaciji, naziva per. per(i)=j označava da je i −ti element permutacije j −ti element iz skupa roditelja. Nakon odabira u rekombinaciju se šalju parovi jedinki per(1) i per(2), zatim per(3) te per(4) itd.

Sljedeći korak je odrediti slučajno mjesto rekombinacije u svakom paru roditelja u rasponu od 1 do (duljina genotipa-1). Stvaraju se dva potomka na sljedeći način – genotip prvog potomka jednak je prvom roditelju do mjesta rekombinacije, a nakon mjesta rekombinacije genotip istog jednak je genotipu drugog roditelja. Analogno, geni u genotipu drugog potomka do mjesta rekombinacije imaju iste vrijednosti kao i oni na istim mjestima u genotipu drugog roditelja, a nakon mjesta generacije, genotip je isti kao i kod prvog roditelja. Takav način rekombinacije naziva se rekombinacija u točki.

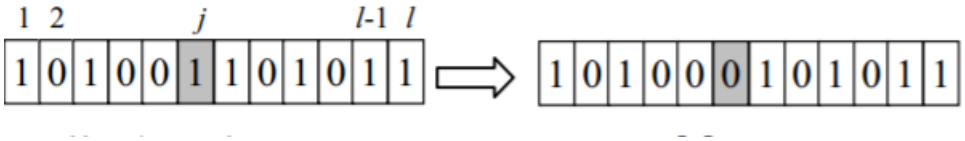


Slika 15 - Rekombinacija u točki, lijevo - par roditelja, desno - par potomaka

Ovisno o problemu, određuje se stopa rekombinacije – postotak generacije nastao rekombinacijom. Ukoliko je podešen na 0% izostaje postupak rekombinacije te su svi potomci kopirani upravo od roditelja, a ukoliko je stopa rekombinacije 100% znači da svaki roditelj ulazi u paru u rekombinaciju.

5.3.2. Mutacija

Nakon što su inicijalizirani u postupku rekombinacije, potomci, svaki za sebe, prolaze kroz postupak mutacije – male, ali značajne promjene u genotipu budući da može garantirati povezanost prostora traženja. Mjesto mutacije određuje se slučajno, a broj gena koji mutiraju ovisi o veličini genotipa. Ukoliko se genotip zapisuje u obliku jedinica i nula (što je čest slučaj), na mjestu mutacije gen na lokaciji mutacije izmjeni se iz 1 u 0 ili obrnuto, što se naziva bit-flip mutacija.



Slika 16 - Bit-flip mutacija

Mutacija je uvedena kako evolucijski proces ne bi zapeo u lokalnom optimumu, ali ukoliko se odvija prečesto, proces se svodi na slučajno pretraživanje. Stoga je uvedena stopa mutacije, broj između 0 i 1 koji određuje koji će postotak potomaka biti mutiran.

5.4.Zamjena populacije

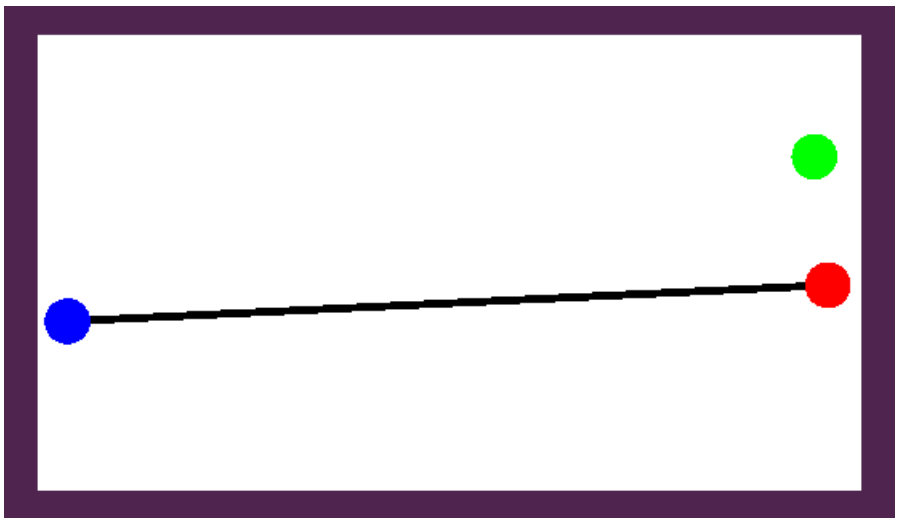
Budući da u prirodi okoliš ima resursa za ograničen broj jedinki, u genetskom algoritmu to zahtjeva konstantnu veličinu populacije. Drugim riječima ne „preživljavaju“ svi jedinke i svi potomci generacije. Nekoliko je osnovnih načina odabira jedinki za sljedeću generaciju:

1. Bazirano na fitnessu – rangiraju se roditelji i potomci po vrijednosti funkcije dobrote te se u sljedećoj generaciji zadržavaju oni s višom vrijednosti funkcije dobrote
2. Bazirano na starosti – Generira se potomaka koliko ima i roditelja te potomci zamjene roditelje
3. Elitizam – kombinacija metode 1 i 2. Nekoliko članova s najvećom podobnošću ostaje u populaciji kroz generacije, dok ostatak populacije nadopunjavaju novonastali potomci.

Bitan element cijele populacije je raznolikost – ukoliko su genotipi jedinki u populaciji u velikoj mjeri slični, ne mogu se stvoriti potomci koji će imati bitno različite karakteristike od roditelja, drugim riječima, na veoma sličan način rješavati će isti problem i dobiti će slične, ako ne iste rezultate i time se gubi mogućnost napretka vrste. Da bi se to izbjeglo, razvijene su metode očuvanja raznolikosti.

6. MODELIRANJE IGRE UNUTAR UNITY OKOLINE

Računalna igra razvijena u sklopu ovog rada sastoji se od 3 kruta tijela oblika kruga (crvene, plave te zelene boje) smještena u istu ravninu i omeđena krutim granicama.



Slika 17 - Dovršena igra

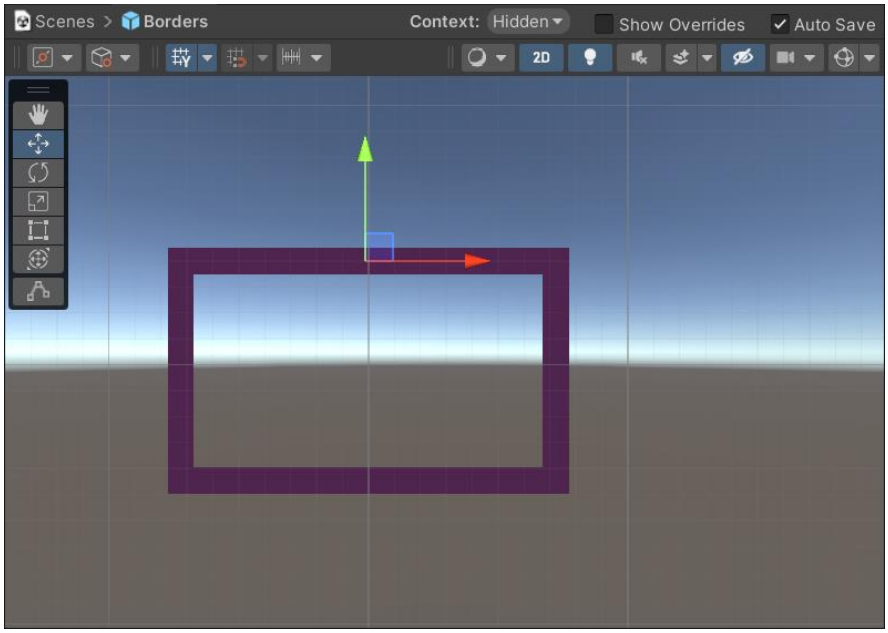
U svakom potezu je cilj krugu određene boje narinuti silu na način da je rezultat njegovog kretanja prolazak kroz crnu spojnu liniju između ostala dva kruga. Sila na krug dodaje se klikom miša na određene koordinate. Početna pozicija igre prikazana je na slici 17 – inicijalno je red na zelenom krugu. Ukoliko zeleni krug ispuni cilj, u sljedećem potezu definirati će se i prikazati na zaslonu crna spojna linija između njega te crvenog kruga, a plavom krugu biti će potrebno narinuti silu. Uspješnim potezom plavog kruga, red pada na crveni, a nakon crvenog ponovno na zeleni i tako u krug. Neuspješnim potezom završava igra, a igrač je postigao rezultat jednak broju uspješnih poteza. Tijekom kretanja, krug koji je na redu može se odbiti od granica prostora igre, ali i od ostala dva kruga koji su u tom trenutku nepomični. Kako bi igrač dobio predikciju o smjeru kretanja kruga, prilikom poteza, umjesto instantnog klika na zaslon, postoji mogućnost držanja pritisnutom lijeve tipke miša prilikom koje se pojavljuje linija prikaza smjera kruga. Pomicanjem miša pomiče se i linija prikaza smjera, puštanjem pritiska s lijeve tipke miša ispaljuje se krug.

Kako bi događanja na zaslonu bila jasnija, odmah po otvaranju praznog projekta, zadanom objektu igre – kameri (unutar Unity okoline nazvanoj *Main Camera*) se mijenja boja unutar komponente *Camera*, značajke Background.

Slika 18 - Linija prikaza smjera

6.1.Modeliranje granice

Prostor igre omeđen je s dva para pravokutnika, kraći se nalaze s lijeve i desne strane prikaza, a dulji vertikalno iznad i ispod zbog razmjera prostora (16:9). Prvi korak, kao što je opisano u poglavlju 1, stvaranje je objekta igre: hijerarhija → desni klik → *2D Object* → *Sprites* → *Square*. U prozoru *Scene* stvoren je objekt kvadra zadane bijele boje te zadanih dimenzija te se transformiranjem pozicije i veličine smješta na lijevi rub prostora igre (gabaritnih dimenzija prikaza kamere), a u komponenti *Sprite renderer* mijenja se značajka boje u RGBA prostoru – proširenju RGB prostora u vidu transparentnosti (eng. *alpha*). Da bi krug tijekom kretanja imao mogućnost odbijanja od granica, dodaje se komponenta ravninskog krutog tijela (eng. *RigidBody2D)*, a kao tip tijela odabire se statičko (eng. *Static*). Budući da je kretanje kruga na potezu diskretizirano skupom točaka – početnom točkom, svakom točkom sudara s drugim objektima igre te krajnjom točkom, dodaje se komponenta pravokutnog sudarača (eng. *Box Collider*) te skripta *CollisionScript* vidljiva u prilogu. Zadaća skripte je poslati informaciju o lokaciji sudara kruga i granice u glavnu skriptu igre (*GameplayScript*, također vidljiva u prilogu), gdje se nakon toga lokacija/lokacije dodaje u listu točaka. Navedeno je ostvareno pomoću svojstava (eng. *Properties*). Nakon svega, potrebno je duplicirati objekt i smjestiti ga u desni rub prostora igre kako bi se stvorila desna granica. Analogni postupak potrebno je ponoviti za gornji i donji rub.



Slika 19 - Prikaz krutih granica igre unutar prozora scene

6.2. Modeliranje krugova

Na analogni način objektima granice inicijaliziraju se objekti krugova, također im se mijenja boja i dodaje komponenta ravninskog krutog tijela. Prednost komponente *Rigidbody2D* jest ta što je kontrolirana od stane fizikalnog mehanizma Unity okoline te posjeduje skriptabilno aplikacijsko programsko sučelje (eng. *API*). Unutar skripte može se dobiti informacija o faktoru trenja tijela, poziciji, kutnoj brzini, masi, rotaciji, centru mase i mnogim drugim komponentama, a također tijelu se može i pridodati sila. Prilikom igre bitno je kreće li se neki krug te koja mu je trenutna pozicija. Bitno je također iskoristiti informaciju o pozicijama druga dva kruga. Potrebno je mijenjati tip krutog tijela tijekom igre – kada krug nije na potezu statičan je, a kada je na potezu tada je dinamičan.

Prvom na redu za igru (zelenom) krugu dodaje se komponenta glavne skripte *GameplayScript* te komponenta prikazivača linija (eng. *Line Renderer*) za prikaz linije smjera. Drugom (plavom) krugu također je pridodana komponenta prikazivača linija kako bi se prikazale spojne linije.

6.3. Skripta GameplayScript

Kao što je već rečeno, glavna skripta igre u kojoj je modelirana dinamika cijele igre naziva je *GameplayScript*. Pri pokretanju skripte inicijaliziraju se objekti igre, polja te varijable koje će

biti korištene kasnije. Unutar metode *Update,* koja se izvodi svaki puta kada se obnovi slika s zaslona, iznova se odvija provjera je li igrač koji je na potezu izvršio ispravan potez.

Ovisno o iznosu cjelobrojne varijable *counter* čija je inicijalna vrijednost 0 ulazi se u razdvojene petlje iste namjene:

- Poziva se metoda *Turn* koja kao ulazne argumente uzima komponente krutih tijela i objekt igre kruga koji je na potezu
- Unutar metode Turn mijenjaju se tipovi krutih tijela te vrši provjera o stanju lijeve tipke miša – ukoliko je ista pritisnuta, modelira se krajnja pozicija linije prikaza smjera (početna pozicija uvijek je pozicija kruga koji je na potezu) pomoću vektora *currentPoint, difference, direction* i *distance*. Nakon izračuna, početna i krajnja točka ulaz su u metodu *RenderLine* koja postavlja točke komponente prikazivača linija i omogućuje prikaz istog na zaslonu. Ukoliko dođe do otpuštanja lijeve tipke miša, mijenja se vrijednost varijable *counter* te se uništava linija prikaza smjera metodom *EndLine*. Ukoliko je krug koji je na potezu u stanju mirovanja, dodaje mu se impuls sile, u smjeru i iznosu ovisno o poziciji miša u trenutku prestanka pritiska.
- Svakim prolaskom kroz metodu *Update* provjerava se je li krug na potezu prošao kroz spojnu liniju središta ostala dva kruga. Za to je zaslužna metoda *IntersectionCheck* koja kao ulazne argumente uzima elemente liste *st_hit_end_#* (za zeleni krug #=1, za plavi #=2, za crveni #=3). Navedena lista preuzima vektore iz liste vektora *listOfAllPlayersPoints* čiji je prvi element početna točka kruga, svaka sljedeća točka je točka sudara s granicom ili drugim krugom, a zadnja točka je ona u kojoj je krug prestao s gibanjem. Unutar metode, radi jednostavnijeg toka rada, poziva se metoda *LineLineIntersection* koja kao argumente uzima poziciju kruga koji je na potezu, poziciju jednog kruga koji nije na potezu te vektore smjera između početne i krajnje pozicije kruga na potezu, kao i vektor smjera između dva kruga koja nisu na potezu. Unutar metode pomoću vektorskog umnoška provjerava se je li došlo do presjeka linija početne i krajnje pozicije kruga koji je na redu te spojne linije između ostala dva kruga pomoću mješovitog umnoška te vraća točku sjecišta ukoliko postoji, ukoliko ne postoji, metoda vraća nul-vektor. Nalaženjem sjecišta rezultat se povećava za jedan i polazi se u sljedeću iteraciju.
- Unutar petlje u svakom trenutku provjerava se brzina kruga na potezu. Ukoliko se krug kreće po zaslonu i počinje zaustavljati u jednom trenutku kada je brzina gotovo

infinitezimalna, znatno se povećava faktor trenja tijela kako bi se tijelo što prije zaustavilo i time ubrzao proces rada.

6.4. Gumb i okruženje pauze

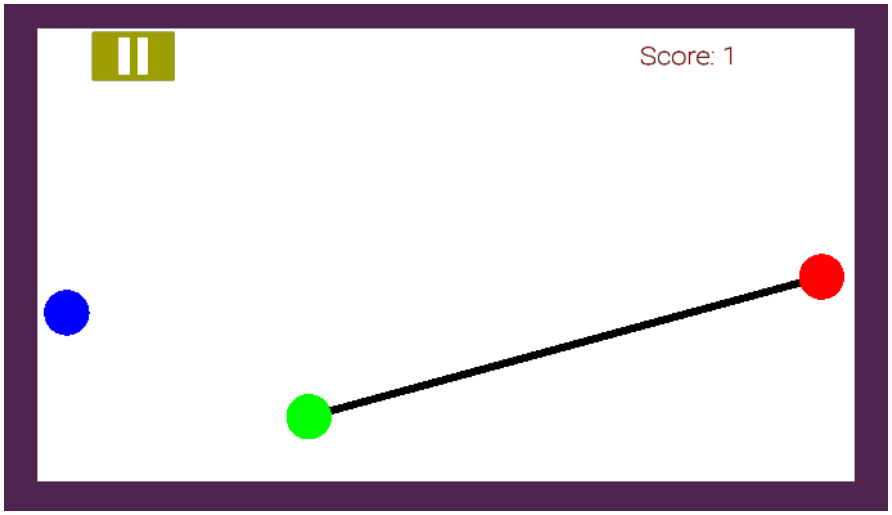
Radi preglednosti projekta, a i dojma da je igra pauzirana, gumb pauze stvara se na novom platnu koji je smješten na novom panelu, dakle, panel je dijete platna. Platno je objekt igre koji brine za poziciju objekata igre tijekom skaliranja prostora igre, dok panel služi za raspodjelu drugih objekata korisničkog sučelja (koji su vezani s panelom u roditelj-dijete strukturi) na zaslonu.

Stvaranje platna: Hijerarhija → desni klik → *UI* → *Canvas*

Stvaranje panela: Hijerarhija → desni klik na *Canvas* → *UI* → *Panel*

Stvaranje gumba pauze: Hijerarhija → desni klik na *Panel* → *UI* → *Button*

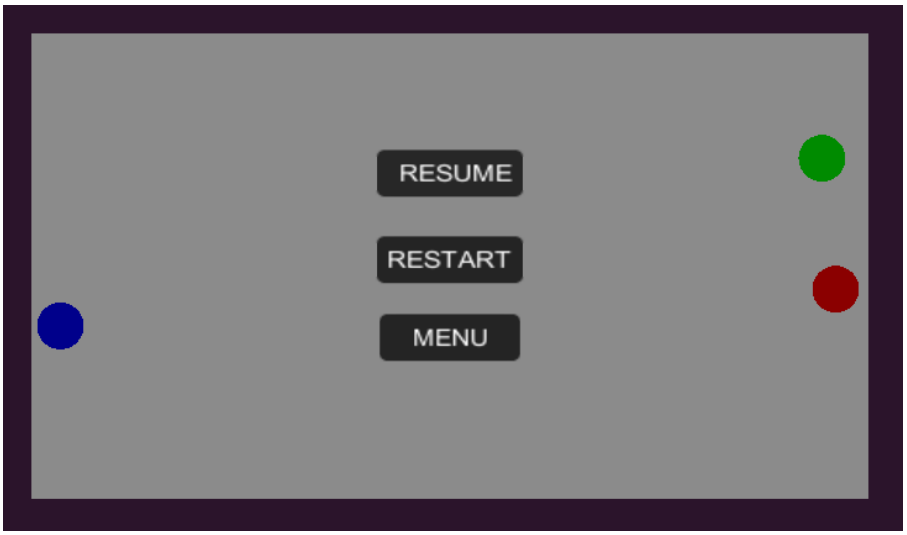
Platno i panel su radi preglednosti istih dimenzija kao i prostor igre, a gumb se smješta u lijevi gornji kut prostora igre, promijeni mu se boja, a kako bi se dobio dojam da se radi o gumbu za pauzu, kao djeca mu se pridodaju dva bijela pravokutnika.



Slika 20 - Prikaz glavne scene s gumbom pauze

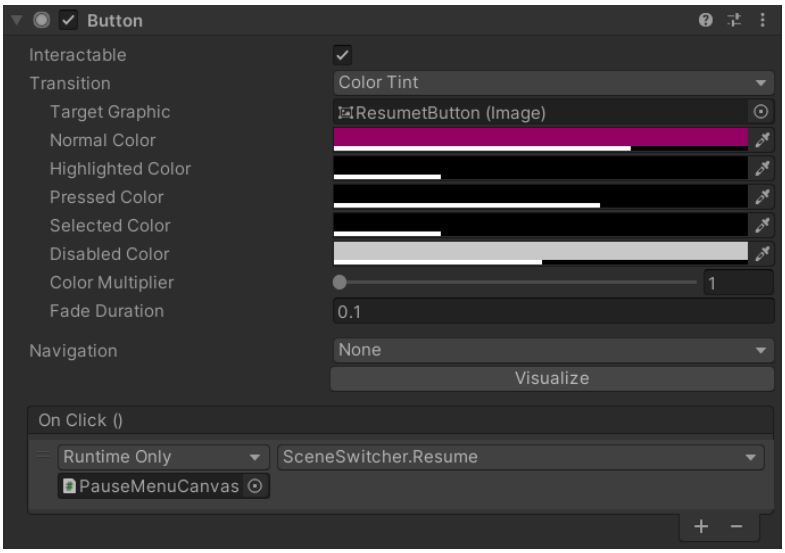
Pritiskom na gumb pauze aktivira se metoda *Pause* iz skripte *SceneSwitcher* (vidljiva u prilogu). Navedena metoda aktivira panel na kojemu su smještena 3 gumba koji nastavljaju, ponovno pokreću ili završavaju igru izlaskom u glavni izbornik. Aktivirani panel je crne transparentne boje stoga se vizualno dobiva dojam nemogućnosti igranja. Da bi se onemogućilo povlačenje

poteza, jednostavno se deaktivira skripta *GameplaySctipt* unutar metode *Pause*. Prikazivač linija se deaktivira radi preglednosti.



Slika 21 - Izbornik pauze

Komponenta gumba za pauzu (kao i komponenta svakog gumba) posjeduje zadanu komponentu *Button* unutar koje je ugrađena značajka *On Click ()* gdje se odabire prvo skripta, a zatim metoda iz skripte koja će se izvršiti pritiskom na gumb.



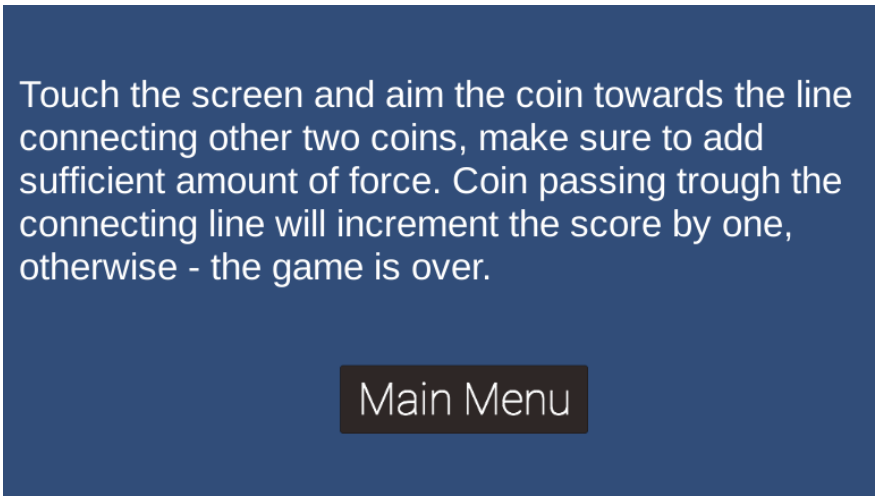
Slika 22 - Komponenta gumba te značajka *On Click ()*

Pritiskom na gumb *RESUME* unutar izbornika pauze revertirati će se promjene rezultirane pritiskom gumba pauze – deaktivira se panel pauze (nestaju tri prikazana gumba, pozadina igre ponovno je bijela) te se aktivira skripta *GameplayScript*. Pritisak na gumb *RESTART* rezultira

istim učinkom kao i pritisak na prethodni gumb, ali postoji dodatak u kojemu se ponovno učitava scena *gameplayScene* (u kojoj se odvija sve do sada navedeno) što znači da se rezultat postavlja na nulu, tj. igra počinje ponovno. Pritisak na gumb *MENU* odvodi sučelje igre u okružje glavnog izbornika igre unutar kojeg postoje opcije pokretanja igre, ulaska u okružje pregleda pravila igre te opcija izlaska iz igre.

Slika 23 - Okružje glavnog izbornika

Prikazani bijeli tekst pri vrhu slike 23 dodan je na način da je panelu unutar okružja dodana komponenta *Text* te je u polje unutar značajke komponente unesen tekst. Pritiskom gumba *How to?* ulazi se u okružje uputa za igru.



Slika 24 - Okružje uputa za igru

Na slici 20 također se može primjetiti prikaz trenutnog rezultata pri gornjem desnom kutu prostora igre. Navedeno je ostvareno stvaranjem platna *ScoreTextCanvas,* dodavanjem panela

ScoreText kojemu je pridodjeljena komponenta teksta te skripta *ScoreGetterScript* (vidljiva u prilogu) koja uzima komponentu teksta kao polje i izmjenjuje ju na način da joj dodaje fiksni string („*Score:* “) zajedno s trenutnom vrijednošću rezultata koja se dobiva iz skripte *GameplayScript.*

6.5. Kraj igre

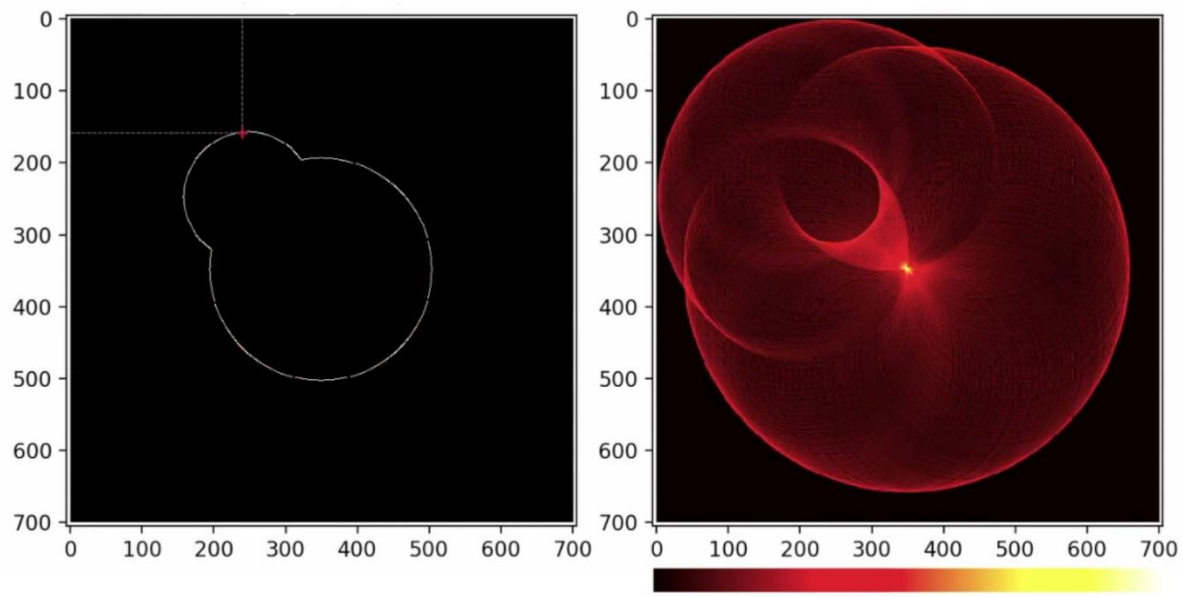
Ukoliko je odigran potez, a rezultat provjere sjecišta linija je nul-vektor, poziva se metoda *EndGame* koja aktivira panel s gumbima za ponovno pokretanje igre te povratka na početni zaslon.

7. SUSTAV AUTOMATSKOG IGRANJA

Sustav koji igra modeliranu igru umjesto ljudskog igrača razvijen je pomoću računalnog vida i neuronske mreže za čije je učenje iskorišten genetski algoritam. Generalna ideja jest da se prepoznaju krugovi pomoću OpenCV knjižnice, neuronska mreža daje koordinate zaslona gdje je potrebno pritisnuti mišem, dok Pythonova knjižnica *pyautogui* obavlja zadatak pritiska miša. Cijeli sustav pisan je u programskom jeziku Python unutrar skripte *raw* (vidljiva u prilogu). Tijekom učenja, podaci o funkciji dobrote jedinki tijekom generacija spremaju se u datoteku kako bi se po završetku učenja simbolički prikazali.

7.1. Sustav računalnog vida

Cijela dinamika prepoznavanja objekata i upravljanje igrom odvija se unutar funkcija *Unit* te *ScreenRecord*. Prostor modelirane igre s pokrenutom glavnom scenom smješta se na zaslonu unutar koordinata *x_start,y_start,x_end,y_end*. Unutar *while True* petlje funkcije *Unit*, očitavaju se vrijednosti piksela navedenog područja u vidu boje unutar RGB prostora te spremaju u obliku matrice. Nakon pauze od 0,4 sekunde u drugu matricu spremaju se ponovno vrijednosti piksela s područja prostora igre. Ukoliko je njihova razlika jednaka nuli znači da nije bilo nikakve promjene unutar igre u navedenom razdoblju, drugim riječima slika miruje. Time se stvara uvjet za pregled objekata prisutnih u igri jer ukoliko slika ne miruje znači da neki od krugova trenutno putuje po zaslonu, a u takvoj situaciji nema smisla ništa provjeravati nego jednostavno treba pričekati zaustavljanje kruga. Ovisno o rezultatu igre koji je spremljen unutar cjelobrojne varijable *number_of_moves*, poziva se funkcija *ScreenRecord* koja kao ulazni argument prima boju kruga koji je trenutno na potezu prikazanu u RGB prostoru. Unutar funkcije ponovno se preuzima slika sa zaslona, prostor boje te slika pretvaraju se u nijanse sive te se slici dodaje filtar za zamagljenje. Za otkrivanje krugova koristi se funkcija knjižnice OpenCV *HoughCircles* koja predaje sliku skupljaču (eng. *accomulator*) gdje se za svaku točku ruba ocrtava kružnica zadanog radijusa (ukoliko je više zadanih radijusa prolazi se svaki radijus zasebno). Na kraju ocrtavanja, središta kružnice ili središta objekata sličnih kružnicama biti će istaknuti unutar rubova. Može se postaviti prag isticanja iznad kojeg se za objekt kaže da je krug. Houghova transformacija može se iskoristiti za traženje linija, ali na nešto drugačiji način. U tom slučaju po uzimaju se točke rubova i prebacuju u polarni prostor gdje cijeli rub predstavlja točku, a točke rubova sinusoide. Ukoliko je rub linija, njegova točku u polarnom prostoru sjecište je mnoštva sinusoida.



Slika 25 - Houghova transformacija (lijevo – rubovi, desno – kružnice konstantnog radijusa za svaku točku ruba)

Nakon pronalaska svih krugova na zaslonu, sustav provjerava za svaki postoji krug njegovu boju, ukoliko je detektirani krug boje krugova igre njegove x i y koordinate dodaju se u listu *coins.* Odmah nakon toga provjerava se postoje li uvjeti za nastavak igre – ukoliko u lista *coins* nema duljinu od 6 elemenata, znači da na zaslonu nisu prikazana 3 kruga igre te se krug prekida. U ovom pristupu pomoglo je što je u slučaju kraja igre na zaslonu prikazan crni transparentni panel stoga se prepoznaju krugovi, no nisu više tražene boje. Slijedi reorganiziranje liste *coins* na način da se koordinate kruga koji je na potezu stavljaju na prva dva mjesta liste. Takva lista potom se šalje u neuronsku mrežu (u funkciju *Network*) kao njen ulaz. Neuronska mreža potom vraća x i y koordinatu gdje je (naredbama iz knjižnice *pyautogui)* potrebno pritisnuti kursorom miša na zaslonu. Svaki puta kada se uspješno klikne mišem, varijable *number_of_moves* i *score* bivaju povećane za jedan.

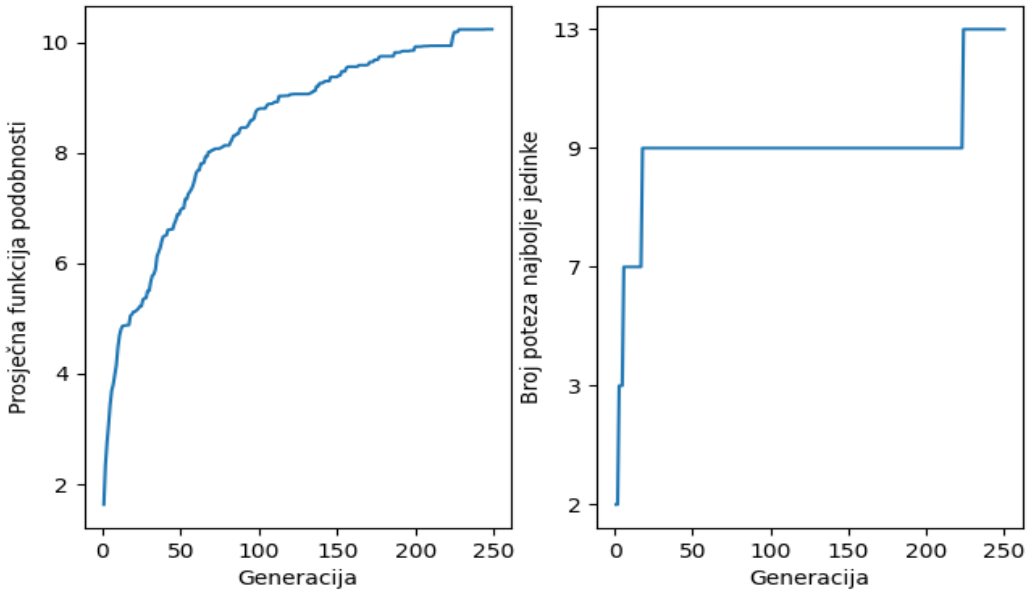
7.2.Učenje neuronske mreže

Prvi korak jest definiranje parametara neuronske mreže. Kao što je ranije spomenuto, ulazni sloj sastoji se od 6 neurona, a izlazni od 2. Postoje i 2 skrivena sloja, prvi od 8, drugi od 4 neurona. Za aktivacijsku funkciju izabrana je sigmoid funkcija (vidjeti Tablica 1). Ulazi u mrežu podijeljeni su s maksimalnim koordinatama prostora igre, a pri aktivaciji zadnjeg sloja brojevi se množe s maksimalnim koordinatama prostora igre. Težine i pomaci mreže prilikom inicijalizacije su slučajni brojevi raspodijeljeni Gaussovom razdiobom između -1 i 1. Upravo

težine i pomaci predstavljaju genotip jedinke duljine 96 gena. Nakon inicijalizacije 40 jedinki, cijela generacija šalje se na evaluaciju unutar funkcije *Generation.* Tamo se (za svaku jedinku posebno) pozove ranije spomenuta funkcija *Unit* koja kao ulazni argument uzima težine i pomake (genotip) jedinke koja vraća funkciju podobnosti jedinke (u vidu varijable *score)*, njene težine i broj poteza (u vidu varijable *number_of_moves*). Funkcija podobnosti jedinke povećava se za jedan svakim uspješnim potezom, umanjuje za 0.3 ukoliko se dogodi situacija da je spojna linija kraća od 6 radijusa krugova te povećava za maksimalno 0.3 (ovisno o duljini spojne linije) ako je spojna linija duža od 6 radijusa krugova. Lista koja sadrži funkcije podobnosti, težine i pomake te broj poteza cijele generacije potom se sortira po iznosu funkcije podobnosti (počevši od najveće) i vraća kao rezultat funkcije *Generation.* Nakon toga dolazi red na stvaranje nove generacije. Navedeno počinje u funkciji *mutation_and_recombination.* Ruletnim pravilom od 4 člana odabire se 40 jedinki iz generacije (neke mogu biti izabrane više od jednom) i šalje u rekombinaciju i mutaciju. Navedena funkcija vraća listu težina i pomaka potomaka jedinki koja se potom šalju na evaluaciju. Za novu generaciju, iz stare generacije zadržava se uvijek 10 najboljih jedinki, dok ostatak generacije čine 30 najboljih jedinki potomaka. Proces se ponavlja za 250 generacija.

7.3.Rezultati učenja

Funkcija podobnosti te broj poteza svake jedinke svake generacije spremaju se u tekstualnu datoteku *scores_of_gens*.txt. Nakon procesa učenja, u skripti *plt_results* (vidljiva u prilogu) prikupljaju se navedeni podaci te se na grafovima prikazuju prosječna vrijednost funkcije podobnosti po generacijama i rezultati najboljih jedinki po generacijama.



Slika 26 - Rezultati učenja (lijevo - prosječna funkcija podobnosti generacije, desno – broj poteza najbolje jedinke generacije)

Slika 26 lijevo prikazuje karakterističnu krivulju učenja genetskog algoritma – Pri početku učenja izražen je nagli rast funkcije podobnosti, daljnjim učenjem rast biva sve blaži, a pri kraju učenja dolazi gotovo u zasićenje. Isto tako, prema slici 26 desno vidljivo je kako se tijekom učenja najbolje jedinke zadržavaju u populaciji sve dulje i dulje – tj. sve rjeđe se pojavi jedinka koja je svojim svojstvima bolja od svih ostalih iz generacije. Broj poteza najbolje jedinke na kraju učenja jest 13, što se može smatrati sasvim pristojnim rezultatom.

8. ZAKLJUČAK

Igra je uspješno modelirana i ostvaruje sve potrebne zadatke. Kako bi kod bilo pregledniji te lakši za obradu, unutar metode *Update* (skripta *GameplayScript*) valjalo bi implementirati sekvencijalni poziv nove metode koja bi primala informacije o trenutnom stanju unutar okružja igre kao argumente. Analogni pristup može se iskoristiti također u sustavu računalnog vida unutar funkcije *Unit*.

Daljnjim ugađanjem parametara neuronske mreže te algoritma učenja, povećanjem broja jedinki u generaciji, povećanjem broja generacija mogu se očekivati još bolji rezultati. Rezultat specifičnog pristupa pri izradi sustava učenja je jednostavno ugađanje parametara mreže - moguće je dodavanjem elemenata unutar liste *number_of_neurons_in_layer* (na bilo koji indeks osim zadnjeg) moguće je dodavati nove skrivene slojeve s željenim brojem neurona te modificirati broj neurona postojećih skrivenih slojeva.

Također, unutar sustava učenja modifikacijom globalnih cjelobrojnih varijabli *population_size, generation_size, items_to_mutate* i *num_of_children* moguće je izmijeniti veličinu populacije, broj generacija učenja i broj gena za izmjenu te broj jedinki koji ulazi u mutaciju, a promjene se propagiraju unutar cijelog sustava.

Nedostatak pristupa razvijenog u radu jest vrijeme potrebno za evaluaciju svih jedinki. Razlog tomu je pojedinačno učenje, a rezultira vremenom učenja od približno 40 sati za 250 generacija s 40 jedinki. Kada bi se ne bi koristio računalni vid, nego u Unity okolini inicijalizirale jedinke i provelo *on-line* učenje za *n* jedinki, vrijeme učenja bilo bi teoretski i do *n* puta manje.

Umjesto igranja igre, sustav izrađen u sklopu ovoga rada (uz preinake) mogao bi se koristiti za planiranje putanje autonomnog robota. Umjesto prikupljanja informacija o slici na zaslonu, prikupljao bi se signal kamere postavljene iznad radne okoline robota. Potrebno je tada izvršiti klasifikaciju prepreka te slobodnog prostora, a zadatak neuronske mreže je davati iznos signala kojeg je potrebno uputiti motorima robota uz uvjet da ne dođe do kolizija.

LITERATURA

[1] Yannakakis G. N., Togelius J., Artificial Intelligence and Games, Springer, 2018.
[2] Goldstone W., Unity Game Development Essentials, Packt Publishing, Birmingham - Mumbai, 2009.
[3] https://docs.unity3d.com/Manual/index.html, 17.02.2022.
[4] Dadhich A., Practical Computer Vision, Packt Publishing, Bangalore, 2018.
[5] Majetić D., Neuronske mreže (Podloge za predavanja), FSB, Zagreb, 2020.
[6] https://www.ibm.com/cloud/learn/neural-networks
[7] Kinsley H., Kukieła D., Neural Networks from Scratch in Python, Harrison Kinsley, 2020.
[8] Dalbelo Bašić B., Čupić M., Šnajder J., Umjetne neuronske mreže, FER, Zagreb, 2008.
[9] https://brilliant.org/wiki/backpropagation/
[10] Yu X., Gen M., Introduction to Evolutionary Algorithms, Springer, 2010.
[11] Ćurković P., Što je Evolucijski Algoritam? (Podloge za predavanja iz kolegija Umjetna inteligencija), FSB, Zagreb, 2020.

PRILOG

I. Skripta *GameplayScript*

II. Skripta *CollisionScript*

III. Skripta *SceneSwitcher*

IV. Skripta *ScoreGetterScript*

V. Skripta *raw*

VI. Skripta *plt_result*

I.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class GameplayScript : MonoBehaviour
{
    public GameManager gameManager;
    Camera cam;
    public Rigidbody2D rb1;
    public Rigidbody2D rb2;
    public Rigidbody2D rb3;
    private Vector3 currentPoint;
    private Vector3 endpointCoin1;
    private Vector3 callendpointCoin1;
    private Vector3 callendpointCoin2;
    private Vector3 callendpointCoin3;
    private Vector3 endpointCoin2;
    private Vector3 endpointCoin3;
    public Vector2 minPower;
    public Vector2 maxPower;
    private float power = 10f;
    private Vector3 endPoint;
    private Vector2 force;
    private Vector2 steadyState = new Vector2(0, 0);
    public LineRenderer lr;
    public LineRenderer lr1;
    public GameObject Coin1;
    public GameObject Coin2;
    public GameObject Coin3;
    private int counter = 0;
    private Vector3 startPointCoin1;
    private Vector3 callStartPointCoin1;
    private Vector3 startPointCoin2;
    private Vector3 callStartPointCoin2;
    private Vector3 startPointCoin3;
    private Vector3 callStartPointCoin3;
    public int score;
    private Vector2 infintesimalVelocity = new Vector2(0.1f, 0.1f);
    private bool permission1 = true;
    private bool permission2 = true;
    private bool permission3=true;
    public GameObject gameOverPanel;
    private List<Vector3> listOfAllPlayersPoints=new List<Vector3>();
```

```
    private List<Vector3> st_hit_end_1=new List<Vector3>();
    private List<Vector3> st_hit_end_2=new List<Vector3>();
    private List<Vector3> st_hit_end_3=new List<Vector3>();

    public Vector3 ListOfAllPlayersPoints
    {
        set
        {
            listOfAllPlayersPoints.Add(value);
        }
    }
    private float magnitude_of_drag_change=100f;

    private static bool LineLineIntersection(out Vector3 intersection, Vector3 linePoint1, Vector3
lineVec1, Vector3 linePoint2, Vector3 lineVec2)
    {
            Vector3 lineVec3 = linePoint2 - linePoint1;
            Vector3 crossVec1and2 = Vector3.Cross(lineVec1, lineVec2);
            Vector3 crossVec3and2 = Vector3.Cross(lineVec3, lineVec2);
            float planarFactor = Vector3.Dot(lineVec3, crossVec1and2);
            if (Mathf.Approximately(planarFactor, 0f) &&
                !Mathf.Approximately(crossVec1and2.sqrMagnitude, 0f))
            {
                float s = Vector3.Dot(crossVec3and2, crossVec1and2) / crossVec1and2.sqrMagnitude;
                intersection = linePoint1 + (lineVec1 * s);
                return true;
            }
            else
            {
                intersection = Vector3.zero;
                return false;
            }
    }
    private static bool IntersectionCheck(Vector3 Start1,Vector3 End1,Vector3 Start2,Vector3 Start3)
    {
            Vector3 intersection;
            Vector3 aDiff = Start1 - End1;
            Vector3 bDiff = Start2 - Start3;

            if (LineLineIntersection(out intersection, End1, aDiff, Start3, bDiff))
            {
                float aSqrMagnitude = aDiff.sqrMagnitude;
                float bSqrMagnitude = bDiff.sqrMagnitude;
                if ((intersection - End1).sqrMagnitude <= aSqrMagnitude
                     && (intersection - Start1).sqrMagnitude <= aSqrMagnitude
                     && (intersection - Start3).sqrMagnitude <= bSqrMagnitude
                     && (intersection - Start2).sqrMagnitude <= bSqrMagnitude)
```

```
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        return false;
    }

    private void Turn(Rigidbody2D player,GameObject coinNr,Rigidbody2D previousPlayer, Rigidbody2D
nextPlayer)
    {
        player.bodyType = RigidbodyType2D.Dynamic;
        previousPlayer.bodyType = RigidbodyType2D.Static;
        nextPlayer.bodyType = RigidbodyType2D.Static;
        RenderLine(previousPlayer.transform.position, nextPlayer.transform.position, lr1);
        Vector3 startPoint = player.transform.position;
        if (Input.GetMouseButton(0))
        {
        currentPoint = startPoint * 2 -cam.ScreenToWorldPoint(Input.mousePosition);
        currentPoint.z = 0;
        var difference = currentPoint - startPoint;
        var direction = difference.normalized;
        var distance = Mathf.Min(125f, difference.magnitude);
        var endPosition = startPoint + direction * distance;
        RenderLine(startPoint, endPosition, lr);
        }

        if (Input.GetMouseButtonUp(0))
        {
                if (coinNr == Coin1) { counter = 1; }
                if (coinNr == Coin2) { counter = 2; }
                if (coinNr == Coin3) { counter = 0; }
                EndLine(lr);
                endPoint = cam.ScreenToWorldPoint(Input.mousePosition);
                endPoint.z = 0;
                if (player.velocity == steadyState)
                {
                    force = new Vector2(Mathf.Clamp(startPoint.x - endPoint.x, minPower.x, maxPower.x),
Mathf.Clamp(startPoint.y - endPoint.y, minPower.y, maxPower.y));
                    player.drag=0.8f;
                    player.AddForce(force * power, ForceMode2D.Impulse);
                }
        }
    }
```

```
public void Awake()
{
    lr = GetComponent<LineRenderer>();
}
private void RenderLine(Vector3 startPoint, Vector3 endPoint, LineRenderer linerender)
{
    linerender.positionCount = 2;
    Vector3[] points = new Vector3[2];
    points[0] = startPoint;
    points[1] = endPoint;
    linerender.SetPositions(points);
}
public void EndLine(LineRenderer linerender)
{
    linerender.positionCount = 0;
}
public void GameOver()
{
    EndLine(lr1);
    gameOverPanel.SetActive(true);
}
public void Start()
{
    cam = Camera.main;
    rb1 = Coin1.GetComponent<Rigidbody2D>();
    rb2 = Coin2.GetComponent<Rigidbody2D>();
    rb3 = Coin3.GetComponent<Rigidbody2D>();
    lr1 = Coin2.GetComponent<LineRenderer>();
    for (int i = 0; i < 12; i++)
    {
        st_hit_end_1.Add(new Vector3(0,0,0));
        st_hit_end_2.Add(new Vector3(0,0,0));
        st_hit_end_3.Add(new Vector3(0,0,0));
    }

}

public void Update()
{
    //Debug.Log(string.Join(", ", listOfAllPlayersPoints));
    if (counter  == 0)
    {
        if(listOfAllPlayersPoints.Count>0)
        {
        for (int i=1; i<listOfAllPlayersPoints.Count+1;i++)
        {
            st_hit_end_3.Insert(i,listOfAllPlayersPoints[i-1]);
```

```
            }
            }
            if (rb3.velocity.magnitude<magnitude_of_drag_change) rb3.drag=1000;
            endpointCoin3 = rb3.transform.position;
            startPointCoin1 = rb1.transform.position;
            for(int j=0;j<listOfAllPlayersPoints.Count+1;j++)
            {
                st_hit_end_3[0]=startPointCoin3;
                if(st_hit_end_3[j+1]==new Vector3(0,0,0)) break;
                st_hit_end_3[listOfAllPlayersPoints.Count+1]=endpointCoin3;
                callStartPointCoin3=st_hit_end_3[j];
                callendpointCoin3=st_hit_end_3[j+1];
                if (score>0 && permission1 == true && IntersectionCheck(callStartPointCoin3,
callendpointCoin3, endpointCoin1, endpointCoin2))
                {
                    score += 1;
                    permission2 = true;
                    permission1 = false;
                }
            }
            if (rb3.velocity.magnitude < infintesimalVelocity.magnitude)
            {
                if (score % 3 != 0)
                {
                    listOfAllPlayersPoints.Clear();
                    GameOver();
                }
                else
                {
                    listOfAllPlayersPoints.Clear();
                    Turn(rb1, Coin1, rb3, rb2);
                }
            }
            for (int i = 0; i < 12; i++) st_hit_end_3[i]=new Vector3(0,0,0);
        }

        if (counter  == 1)
        {
            if(listOfAllPlayersPoints.Count>0)
            {
                for (int i=1; i<listOfAllPlayersPoints.Count+1;i++)
                {
                    st_hit_end_1.Insert(i,listOfAllPlayersPoints[i-1]);
                }
            }
            if (rb1.velocity.magnitude<magnitude_of_drag_change) rb1.drag=1000;
            endpointCoin1 = rb1.transform.position;
```

```
            startPointCoin2 = rb2.transform.position;
            if(rb1.transform.position.y<-119f&&score==0)
            {
                score += 1;
                permission3 = true;
                permission2 = false;
            }
            for(int j=0;j<listOfAllPlayersPoints.Count+1;j++)
            {
                st_hit_end_1[0]=startPointCoin1;
                if(st_hit_end_1[j+1]==new Vector3(0,0,0)) break;
                st_hit_end_1[listOfAllPlayersPoints.Count+1]=endpointCoin1;
                callStartPointCoin1=st_hit_end_1[j];
                callendpointCoin1=st_hit_end_1[j+1];
                if (permission2 == true && IntersectionCheck(callStartPointCoin1,   callendpointCoin1,
endpointCoin2, endpointCoin3))
                    {
                        score += 1;
                        permission3 = true;
                        permission2 = false;
                    }
            }

            if (rb1.velocity.magnitude < infintesimalVelocity.magnitude)
            {
                if (score % 3 != 1)
                {
                    listOfAllPlayersPoints.Clear();
                    GameOver();
                }
                else
                {
                    listOfAllPlayersPoints.Clear();
                    Turn(rb2, Coin2, rb1, rb3);
                }
            }
            for (int i = 0; i < 12; i++) st_hit_end_1[i]=new Vector3(0,0,0);
        }
        if (counter  == 2)
        {
            if(listOfAllPlayersPoints.Count>0)
            {
            for (int i=1; i<listOfAllPlayersPoints.Count+1;i++)
            {
                st_hit_end_2.Insert(i,listOfAllPlayersPoints[i-1]);
            }
```

```
            }
            if (rb2.velocity.magnitude<magnitude_of_drag_change) rb2.drag=1000;
            endpointCoin2 = rb2.transform.position;
            startPointCoin3 = rb3.transform.position;
            for(int j=0;j<listOfAllPlayersPoints.Count+1;j++)
            {
                st_hit_end_2[0]=startPointCoin2;
                if(st_hit_end_2[j+1]==new Vector3(0,0,0)) break;
                st_hit_end_2[listOfAllPlayersPoints.Count+1]=endpointCoin2;
                callStartPointCoin2=st_hit_end_2[j];
                callendpointCoin2=st_hit_end_2[j+1];
                if (permission3 == true && IntersectionCheck(callStartPointCoin2,
callendpointCoin2, endpointCoin1, endpointCoin3))
                {

                    score += 1;
                    permission1 = true;
                    permission3 = false;
                }
            }
            if (rb2.velocity.magnitude < infintesimalVelocity.magnitude)
            {
                if (score % 3 != 2)
                {
                    listOfAllPlayersPoints.Clear();
                    GameOver();
                }
                else
                {
                    listOfAllPlayersPoints.Clear();
                    Turn(rb3, Coin3, rb2, rb1);
                }
            }
            for (int i = 0; i < 12; i++) st_hit_end_2[i]=new Vector3(0,0,0);
        }
    }
}
```

II.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CollisionScript : MonoBehaviour
{
    // Start is called before the first frame update
    public GameplayScript mainScript = new GameplayScript();
    void OnCollisionEnter2D (Collision2D collision)
    {
        //Debug.Log("Hit");
        foreach (ContactPoint2D contact in collision.contacts)
        {
            //Debug.Log(contact.point);
            Vector3 newPoint= contact.point;
            mainScript.ListOfAllPlayersPoints=newPoint;
        }
    }
}
```

III.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneSwitcher : MonoBehaviour
{
    [SerializeField] GameObject _pauseMenuUI;
    [SerializeField] GameObject _gameOverMenuUI;

    public bool gameIsPaused;
    public void PlayGame()
    {
        SceneManager.LoadScene("gameplayScene");
    }
    public void Instructions()
    {
        SceneManager.LoadScene("Instructions");
    }
    public void MainMenu()
    {
        SceneManager.LoadScene("mainMenu");
    }

    public void QuitGame()
    {
        Application.Quit();
    }

    public void Pause()
    {
        _pauseMenuUI.SetActive(true);
        GameObject.Find("Coin1").GetComponent<GameplayScript>().enabled = false;
        GameObject.Find("Coin1").GetComponent<GameplayScript>().EndLine(GameObject.Find("Coin1").
        GetComponent<GameplayScript>().lr);
        gameIsPaused = true;
    }
    public void Resume()
    {
        _pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameObject.Find("Coin1").GetComponent<GameplayScript>().enabled = true;
        gameIsPaused = false;
    }
    public void Restart()
    {
```

```
        _pauseMenuUI.SetActive(false);
        _gameOverMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameObject.Find("Coin1").GetComponent<GameplayScript>().enabled = true;
        gameIsPaused = false;
        SceneManager.LoadScene("gameplayScene");
    }
}
```

IV.

```
using UnityEngine;
using UnityEngine.UI;
public class ScoreGetterScript : MonoBehaviour
{
    public Text scoreText;
    void Update()
    {
        scoreText.text ="Score: "+
GameObject.Find("Coin1").GetComponent<GameplayScript>().score.ToString();
    }
}
```

V.

```
from dataclasses import dataclass
from glob import glob
from operator import ge
from re import U
from turtle import shape
from cv2 import sqrt
import numpy as np
from PIL import ImageGrab
import cv2
#import matplotlib.pyplot as plt
import time
import pyautogui
from random import randint
import random
import math

#Ovo su maksimalne i minimalne x i y koordinate (Valjalo bi smjestiti prostor igre u tome)
global x_start
x_start=37
global y_start
y_start=55
global x_end
x_end=592
global y_end
y_end=366

#Datoteka u koju se spremaju težine i pomaci zadnje generacije učenja
final_wib_file=open("final_wib.txt","w")

#Datoteka u koju se spremaju fitness funkcije i rezultati po generacijama
scores_of_gens=open("scores_of_gens.txt","w")

#Kalkulacija maksimalne duljine spojne linije (koristi se za određivanje fitness funkcije)
global max_connection_line
max_hipoten_squared=(-557+70)**2+(-340+77)**2
max_connection_line=math.sqrt(max_hipoten_squared)


def Forward(inputs,weights,biases):
    return np.dot(inputs,weights)+biases
#Koristi se sigmoid aktivacijska funkcija
def ActivatinFunction(results):
    return 1/(1 + np.exp(-results))
#Svaki od neurona u sljedeće dvije funkcije smješta se u prostor igre
def FinalActivationX(last_layer_x_neuron):
    global x_end
```

```
    global x_start
    last_layer_x_neuron=last_layer_x_neuron*x_end
    if last_layer_x_neuron>=x_end:last_layer_x_neuron=x_end
    if last_layer_x_neuron<=x_start:last_layer_x_neuron=x_start
    return last_layer_x_neuron
def FinalActivationY(last_layer_y_neuron):
    global y_end
    global y_start
    last_layer_y_neuron=last_layer_y_neuron*y_end
    if last_layer_y_neuron>=y_end:last_layer_y_neuron=y_end
    if last_layer_y_neuron<=y_start:last_layer_y_neuron=y_start
    return last_layer_y_neuron

#U funkciji Network svaka koordinata iz ulaza dijeli se s maksimalnom mogućom i radi se
#forward pass
#Funkcija vraća koordinate ekrana gdje se treba kliknuti
def Network(X):
    global x_end
    global y_end
    for kol in range(6):
        if kol%2==0: X[kol]=(X[kol]/x_end)
        else: X[kol]=(X[kol]/y_end)
    X=np.reshape(X,(1,6))
    global unit_wab
    global number_of_neurons_in_layer

    output_of_a_layer=[]
    output_of_a_layer.append(X)
    X=[]
    for i in range(len(number_of_neurons_in_layer)):

        neurons_in_layer=Forward(output_of_a_layer[i],unit_wab[i*2],unit_wab[(i*2)+1])

        if i <2:activated_neurons_in_layer=ActivatinFunction(neurons_in_layer)
        else:activated_neurons_in_layer=neurons_in_layer

        output_of_a_layer.append(activated_neurons_in_layer)

    finally_activated_x=FinalActivationX(output_of_a_layer[-1][0][0])
    finally_activated_y=FinalActivationY(output_of_a_layer[-1][0][1])
    output_of_a_layer=[]
    return finally_activated_x,finally_activated_y

#Funkcija ScreenRecord uzima sliku s ekrana, provjerava ima li krugova, jesu li ti krugovi crveni,
#plavi i zeleni,
#ako jesu sprema njihove koordinate u listu
#Ako nema crvenih, plavih ili zelenih krugova, znači da je igra gotova
```

```
#Prve dvije koordinate su uvijek koordinate kruga koji je na potezu
#Lista koordinata šalje se u neuronsku mrežu i klikne se na ekran na poziciju outputa mreže

def ScreenRecord(colorofturn):
    while(True):
        img =  np.array(ImageGrab.grab(bbox=(x_start,y_start,x_end,y_end)))
        output=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        gray = cv2.medianBlur(gray, 5)

        circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 17,param1=50, param2=10,
                                   minRadius=2, maxRadius=30)# param1=50, param2=30
        if circles is not None:
            detected_circles = np.uint16(np.around(circles))
            coins=[]
            for (x, y ,r) in detected_circles[0, :]:
                if all(m==n for m,n in zip(output[y,x-1],
                       (0,0,255))) or all(m==n for m,n in zip(output[y,x-1],
                       (255,0,0))) or all(m==n for m,n in zip(output[y,x-1],(0,255,0))):
                        cv2.circle(output, (x, y), r, (0, 0, 0), 3)
                        cv2.circle(output, (x, y), 2, (0, 255, 255), 3)
                        coins.append(x)
                        coins.append(y)
            if cv2.waitKey(25) & 0xFF == ord('q') or len(coins)!=6:
                cv2.destroyAllWindows()
                return False
            for i in range(0,6,2):
                if all(m==n for m,n in zip(output[int(coins[i+1])+3,int(coins[i])+3],colorofturn)):
                        if coins[0]!=coins[i]:
                            coins.insert(0,coins.pop(i))
                            coins.insert(1,coins.pop(i+1))

                        global max_connection_line
                        global score
                        if(score>0):

                            real_hipoten_squared=(int(coins[4])-int(coins[2]))**2+(int(coins[5])-
int(coins[3]))**2
                            real_connection_line=math.sqrt(real_hipoten_squared)

                            if real_connection_line<6*r:score-=0.3
                            else: score+=(real_connection_line/max_connection_line)*0.3
                        x_mouse,y_mouse=Network(coins)

                        pyautogui.click(x_mouse,y_mouse)
```

```
                        pyautogui.click(x_mouse,y_mouse)

            coins=[]
            return True
#Fitness funkcija sastoji se od broja odigranih poteza jedinke
#i zbroja veličina spojne linije prilokom svakog poteza
#To je zapravo varijabla score
def Unit(unit_weight_and_bias):
    global score
    score=0
    #global number_of_moves
    number_of_moves=0
    list_of_wab=[]
    for i in range(len(unit_weight_and_bias)):
        list_of_wab.append(unit_weight_and_bias[i])


    while True:
        first =  np.array(ImageGrab.grab(bbox=(x_start,y_start,x_end,y_end)))
        time.sleep(0.4)
        is_still =  np.array(ImageGrab.grab(bbox=(x_start,y_start,x_end,y_end)))
        difference = cv2.subtract(is_still,first)
        result = not np.any(difference)
        if result is not True:
            continue

        if number_of_moves%3==0 and result:
            if ScreenRecord((0,255,0)):
                score+=1
                number_of_moves+=1
                continue
            else: break
        if number_of_moves%3==1 and result:
            if ScreenRecord((255,0,0)):
                score+=1
                number_of_moves+=1
                continue
            else: break
        if number_of_moves%3==2 and result:
            if ScreenRecord((0,0,255)):
                score+=1
                number_of_moves+=1
                continue
            else: break
    #score-=1 Doing this to avoid div by zero in Recombination
    number_of_moves-=1
```

```
    return score,list_of_wab,number_of_moves
#Svaka jedinka posebno se šalje na evaluaciju(igranje igre) u funkciju Unit
def Generation(wab_of_gen):
    #print("generation function")
    global list_of_scores_and_wab
    global number_of_neurons_in_layer
    list_of_scores_and_wab=[]
    for unit in range(len(wab_of_gen)):
        global unit_wab
        unit_wab=wab_of_gen[unit]
        info=Unit(unit_wab)
        list_of_scores_and_wab.append(info)
        pyautogui.click(215,215)

    #Sorting the list starting with the highest score
    list_of_scores_and_wab.sort(key=lambda y: y[0])
    list_of_scores_and_wab.reverse()


    return list_of_scores_and_wab


#Kako bi se dobili potomci koristi se ruletno pravilo s 4 člana
def mutation_and_recombination(generation):
    global parents_to_next_gen
    global population_size
    new_gen=[]


    def Recombinate(unit_a,unit_b):
        flat_a=[]
        flat_b=[]
        for i in range(len(generation[0][1])):
            flat_w_or_b_a=generation[unit_a][1][i].flatten()
            flat_w_or_b_b=generation[unit_b][1][i].flatten()

            flat_a.append(flat_w_or_b_a)
            flat_b.append(flat_w_or_b_b)
        trully_flat_a=[]
        trully_flat_b=[]
        for element in range(len(flat_a)):
            for num in range(len(flat_a[element])):
                trully_flat_a.append(flat_a[element][num])
        for element1 in range(len(flat_b)):
            for num1 in range(len(flat_b[element1])):
                trully_flat_b.append(flat_b[element1][num1])
        point_of_recombination=randint(0,len(trully_flat_a))
```

```
        flat_child_a=[]
        flat_child_b=[]
        for pp in range(point_of_recombination):
            flat_child_a.append(trully_flat_a[pp])
            flat_child_b.append(trully_flat_b[pp])
        for mm in range(point_of_recombination,len(trully_flat_b)):
            flat_child_a.append(trully_flat_b[mm])
            flat_child_b.append(trully_flat_a[mm])


        #MUTATION
        global items_to_mutate
        a=random.sample(range(0,len(flat_child_a)-2),int(items_to_mutate))
        b=random.sample(range(0,len(flat_child_b)-2),int(items_to_mutate))
        for i in range(len(a)):
            flat_child_a[a[i]]=np.random.uniform(low=-1.0,high=1.0)
            flat_child_b[b[i]]=np.random.uniform(low=-1.0,high=1.0)


        child_shaped_a=[]
        child_shaped_b=[]

        for j in range(len(generation[0][1])):
            shape_of_w_or_b=np.shape(generation[unit_a][1][j])
            point_of_border=shape_of_w_or_b[0]*shape_of_w_or_b[1]
            childs_w_or_b_a=np.reshape(flat_child_a[0:point_of_border],shape_of_w_or_b)
            childs_w_or_b_b=np.reshape(flat_child_b[0:point_of_border],shape_of_w_or_b)
            child_shaped_a.append(childs_w_or_b_a)
            child_shaped_b.append(childs_w_or_b_b)
            del flat_child_a[0:point_of_border]
            del flat_child_b[0:point_of_border]

        return child_shaped_a,child_shaped_b
    global num_of_children
    list_of_parents=[]
    for sth in range(0,num_of_children):
        possib_parents=np.random.randint(population_size,size=4)
        score_of_possib_parents=[]
        for __ in range(4):
            score_of_possib_parents.append(generation[possib_parents[__]][0])
        #print(score_of_possib_parents)
        prob_to_be_parent=[]
        for chos in range(4):
            prob_to_be_parent.append(score_of_possib_parents[chos]/sum(score_of_possib_parents))
        parent=int(random.choices(possib_parents,weights=prob_to_be_parent)[0])
        #parent=list(parent)
        list_of_parents.append(parent)
```

```
    for sth_els in range(0,len(list_of_parents),2):
        child_a,child_b=Recombinate(list_of_parents[sth_els],list_of_parents[sth_els+1])
        new_gen.append(child_a)
        new_gen.append(child_b)

    return new_gen




global unit_wab_gen
global number_of_neurons_in_layer
global population_size
global items_to_mutate
global num_of_children

#Određivanje veličine populacije, broja generacija, broja jedinki za mutaciju, broja potomaka
population_size=40
generation_size=250
items_to_mutate=17
num_of_children=40

#Određivanje broja neurona u skrivenim slojevima
unit_wab_gen=[]
number_of_neurons_in_layer=[8,4,2]
wab_of_a_unit=[]

#Inicijalizacija prve generacije(težina i pomaka mreže) slučajnim brojevima
for unit1 in range(population_size):
    for i in range(len(number_of_neurons_in_layer)):
        if(i==0): wab_of_a_unit.append(np.random.normal(loc=0.0,
scale=1.0,size=(6,number_of_neurons_in_layer[0])))
        else: wab_of_a_unit.append(np.random.normal(loc=0.0,
scale=1.0,size=(number_of_neurons_in_layer[i-1],number_of_neurons_in_layer[i])))
        if(i==len(number_of_neurons_in_layer)-1):
wab_of_a_unit.append(np.zeros((1,number_of_neurons_in_layer[i])))
        else: wab_of_a_unit.append(np.random.normal(loc=0.0,
scale=1.0,size=(1,number_of_neurons_in_layer[i])))
    unit_wab_gen.append(wab_of_a_unit)
    wab_of_a_unit=[]
#Slanje prve generacije na evaluaciju, vraća se fitness funkcija zajedno s težinama i pomacima
gen=Generation(unit_wab_gen)

#Evaluacija svake jedinke po generacijama
for _ in range(generation_size):
    print("\n\ngeneration ",_)
```

```
    L0=["\n\ngeneration ",str(_)]
    scores_of_gens.writelines(L0)
    scores_of_gens.write("\n")
    for i in range(len(gen)):
        print("\nFitness function of unit",i+1, gen[i][0])
        L=[str(gen[i][0])," Fitness function of unit ",str(i+1)]
        scores_of_gens.writelines(L)
        scores_of_gens.write("\n")
        print("Number of moves of unit ",i+1, gen[i][2])
        L1=[str(gen[i][2])," Number of moves of unit ",str(i+1)]
        scores_of_gens.writelines(L1)
        scores_of_gens.write("\n")

    #Pozivanje funkcije mutation_and_recombination da bi se dobili potomci
    children=mutation_and_recombination(gen)

    #Evaluacija potomaka funkcijpm Generation
    eval_of_children=Generation(children)
    for blob in range(len(eval_of_children)):
        gen.append(eval_of_children[blob])

    #Sortitanje generacije po fitness funkciji
    gen.sort(key=lambda y: y[0])
    gen.reverse()
    gen=gen[0:population_size]

final_wib_file.write(str(gen))
final_wib_file.close()
scores_of_gens.close()
```

VI.

```
import matplotlib.pyplot as plt
i=3
sum=0
avg_gen=[]
best_moves=[]
final_wib_file_lines=open("scores_of_gens.txt","r").readlines()
while i<20747:
    if i%83==0:
        avg_gen.append(sum/40)
        sum=0
        i+=3
    else:
        score_of_unit=final_wib_file_lines[i].partition(" ")[0]
        sum+=float(score_of_unit)
        i+=2

for j in range(0,250):
    best_moves.append(final_wib_file_lines[j*83+4].partition(" ")[0])
plt.grid(visible=True, which='both', axis='both')
plt.subplot(121)
plt.plot(range(1,250),avg_gen)
plt.ylabel('Prosječna funkcija podobnosti')
plt.xlabel('Generacija')

plt.subplot(122)
plt.plot(range(1,251),best_moves)
plt.ylabel('Broj poteza najbolje jedinke')
plt.xlabel('Generacija')
plt.show()
```