# Primerjava genetskega algoritma in spodbujevanega učenja pri računalniškem iskanju strategije igre Nim

Raziskovalna naloga

2023

# Primerjava genetskega algoritma in spodbujevanega učenja pri računalniškem iskanju strategije igre Nim

Raziskovalna naloga

*Področje:*
Računalništvo in informatika

*Avtor:*
Gregor Bokal
8. razred

*Mentorstvo in strokovni pregled:*
prof. Andreja Urh
dr. Drago Bokal

*Lektura:*
Ljudmila Bokal

*Šola:*
Osnovna šola Alojzija Šuštarja
(Štula 23, Ljubljana - Šentvid)

2023

# Primerjava genetskega algoritma in spodbujevanega učenja pri računalniškem iskanju strategije igre Nim

Raziskovalna naloga

*Področje:*
Računalništvo in informatika

*Avtor:*
Učenec 8. razreda

2023

# Vsebina

# Kazalo slik



# Kazalo tabel



# Kazalo algoritmov

# Povzetek

Igra Nim je matematična igra za *n* igralcev, pri kateri je na polju nekaj žetonov. Ko je igralec na potezi, lahko s polja vzame od 1 do *k* žetonov (število *k* se med igro ne spreminja). Tisti igralec, ki vzame zadnji žeton, je poražen.

S pomočjo matematike lahko najdemo in dokažemo optimalno strategijo za to igro, ko jo igrata 2 igralca. Zanima nas pa tudi, kako bi optimalno strategijo pridobil računalnik, zato naredimo Python program, ki simulira igralce, da ti igrajo igro.

Med igralci, ki jih računalnik simulira, so nekateri taki, ki imajo točno določeno strategijo. Poleg teh pa so tudi igralci, ki nimajo točno določene strategije. Ti začnejo igrati povsem naključno, nato pa svojo strategijo skozi odigrane igre dopolnjujejo s *spodbujevanim učenjem*. Na ta način lahko računalnik poišče optimalno strategijo na podlagi tega, koliko žetonov je na polju pred začetkom. Toda spodbujevano učenje je primernejše za iskanje strategije, kadar igro igrata dva igralca. Zato računalnik sprogramiramo tako, da čim uspešnejšo strategijo poiščemo tudi z *genetskim algoritmom*. Oba načina tudi primerjamo in ugotovimo, da je spodbujevano učenje primernejše za iskanje strategije za igro z dvema igralcema, genetski algoritem pa za iskanje strategije pri igri z več igralci.

Python program, ki ga uporabljamo za simuliranje igre in za strojno učenje, pa ne uporabimo le zase, ampak naredimo tudi Telegram bota, preko katerega lahko program uporabljajo tudi drugi. Na njem lahko igramo Nim, izvajamo simulacije iger in se spoznavamo z umetno inteligenco.

# 1. Uvod

Dandanes postaja v svetu vse pomembnejša umetna inteligenca. Na spletni strani Evropske komisije (Evropska komisija 2023) zasledimo, da je umetna inteligenca obravnavana znotraj *politike ključnih omogočitvenih tehnologij*. Med drugim piše, da dobro usklajena uporaba umetne inteligence lahko družbi prinese pomembne izboljšave. Pomaga nam lahko pri doseganju podnebnih in trajnostnih ciljev ter prinese odmevne inovacije v zdravstvu, izobraževanju, prometu, industriji in številnih drugih sektorjih.

Jasno je, da umetna inteligenca vse bolj prodira v naše vsakdanje življenje. Če hočemo, da nam umetna inteligenca koristi, jo je treba dobro razumeti. V naši raziskovalni nalogi pokažemo, da je že v osnovni šoli mogoče umetno inteligenco razumeti do te mere, da z njo obdelamo enostavno matematično igro Nim.

Tako kot vsaki končni strateški matematični igri med dvema igralcema, tudi za igro Nim med dvema igralcema obstaja optimalna strategija. Že lansko leto smo v matematični raziskovalni nalogi (Bokal, 2022) s pomočjo matematike poiskali optimalno strategijo za igro Nim, ki jo igrata dva igralca, in njeno pravilnost tudi dokazali. Zanima nas pa tudi, kako bi pa do optimalne strategije prišel računalnik.

Obravnavamo dva načina, s katerima računalnik išče strategijo — *spodbujevano učenje*, pri katerem računalnik večkrat simulira igro in po vsaki izvedbi nekoliko prilagodi strategijo, da le-ta postaja učinkovitejša, in pa *genetski algoritem*, pri katerem računalnik simulira celo populacijo igralcev, ki med seboj igrajo igro, nato pa v naslednjo generacijo preda potomce le najuspešnejših. Tako na podoben način, kot se to v naravi dogaja pri evoluciji, sčasoma postanejo igralci vse bolj prilagojeni. V raziskovalni nalogi želimo ugotoviti, kakšne prednosti nam daje posamezen način in kateri je za iskanje optimalne strategije primernejši in kdaj.

# 2 Teoretični del

## 2.1 Znani rezultati

### 2.1.1 Igra in strategije

V raziskovalni nalogi obravnavamo matematično igro Nim, ki jo lahko igrata dva ali več igralcev. Pri tej igri imamo igralno polje, na katerem je nekaj žetonov. Igralci igrajo izmenično. Vsak v svoji potezi s polja vzame od enega do *n* žetonov. Tisti igralec, ki vzame zadnji žeton, izgubi igro.

Pri igranju te igre lahko uporabljamo veliko različnih *strategij*. V literaturi (Jamnik, 1973) izvemo, da je strategija navodilo, ki ga uporabljamo pri igranju igre in nam določi, katero izmed možnih dejanj naj izvedemo v različnih okoliščinah, oziroma s kakšno verjetnostjo naj izberemo določeno dejanje. Pri igri Nim se mora igralec v svoji potezi odločiti, koliko žetonov bo vzel. To pomeni, da mora strategija za igro Nim za vsako *situacijo* (v našem primeru je to število žetonov, ki so na začetku poteze na polju) dati podatek, s kolikšno verjetnostjo naj igralec vzame koliko žetonov. Temu naboru verjetnosti za mogoče odločitve, rečemo *porazdelitev*. Porazdelitev za strategijo pri igri Nim, kjer lahko igralec vzame največ 2 žetona, je lahko na primer taka:

$$\text{porazdelitev pri } n \text{ žetonih } (n > 1) = \begin{pmatrix} 1 & 2 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Ta porazdelitev velja takrat, kadar je na polju več kot 1 žeton. Pove nam, da je verjetnost, da bo igralec vzel en žeton, 0,5, in prav tako 0,5, da bo vzel dva. Ta matematično natančen zapis v raziskovalni nalogi zapisujemo kot Python seznam, pri katerem vrednost na indeksu *i* predstavlja verjetnost, da vzamemo *i + 1* žeton:

$$\text{porazdelitev pri } n \text{ žetonih } (n > 1) = [0.5,\ 0.5]$$

Pri tem zapisu nam prva številka pove verjetnost, da igralec vzame en žeton, druga pa verjetnost, da igralec vzame 2 žetona. Če igro igramo tako, da lahko vzamemo tudi več žetonov, verjetnosti za to nanizamo po vrsti, a v raziskovalni nalogi takih primerov ne opisujemo.

### 2.1.2 Strategije igre Nim

Kot smo že omenili, je igro Nim mogoče igrati z veliko različnimi strategijami. V raziskovalni nalogi obravnavamo naslednje:

#### 2.1.2.1 Naključna strategija

Igralec, ki uporablja *naključno strategijo*, se bo za vsako od možnih dejanj odločil z enako verjetnostjo. Če lahko igralec vzame največ dva žetona in je na polju več kot en žeton, bo njegova porazdelitev [0.5, 0.5].

#### 2.1.2.2 Optimalna strategija

Optimalna strategija je tista, ki nas, kadar je mogoče, pripelje do zmage. Z matematiko je mogoče najti in dokazati optimalno strategijo za igro Nim. V literaturi (Bokal, 2022) je dokazano, da je optimalna strategija za igro Nim, ki jo igrata dva igralca, ki lahko v svoji potezi vzameta največ dva žetona, taka:

- porazdelitev za $3 \times n + 2$ žetonov = [1, 0],
- porazdelitev za $3 \times n + 3$ žetonov = [0, 1],
- porazdelitev za $3 \times n + 1$ žeton, ne obstaja.

Ker za $3 \times n + 1$ žeton optimalna strategija ne obstaja, v raziskavi uporabljamo dve različni *zmagovalni strategiji*.

#### 2.1.2.3 Zmagovalna strategija

Zmagovalna strategija ima za *obvladljive situacije* takšno porazdelitev kot optimalna strategija. Kadar pa le-ta ne obstaja, ima porazdelitev [1, 0].

#### 2.1.2.4 Zmagovalna naključna strategija

Kadar je mogoče, tudi ta strategija uporabi porazdelitev zgoraj omenjene optimalne strategije, kadar pa to ni mogoče, se igralec, ki uporablja to strategijo, odloči naključno. Torej je tu porazdelitev za $3 \times n + 1$ žeton = [0.5, 0.5].

### 2.1.3 Računalniški igralci

Za izvajanje eksperimentov in preizkušanje strategij naredimo Python program, ki simulira igranje igralcev. Simulira lahko več različnih igralcev, vsak od njih pa uporablja svojo strategijo.

#### 2.1.3.1 Živi igralec

Tega igralca upravlja človek. V raziskavi ga navadno ne uporabljamo.

#### 2.1.3.2 Naključni igralec

Ko je na potezi naključni igralec, se za število vzetih žetonov odloči povsem naključno. Uporablja naključno strategijo.

#### 2.1.3.3 Naključni uteženi igralec

Naključni uteženi igralec se prav tako odloča naključno. Lahko pa mu damo razne „uteži“, s katerimi mu spremenimo porazdelitev za izbrane odločitve. Tako mu lahko mi določimo, kakšna bo njegova strategija.

#### 2.1.3.4 Zmagovalni igralec

Zmagovalni igralec uporablja prej omenjeno zmagovalno strategijo. Torej igra optimalno, kadar pa to ni mogoče, vzame en žeton.

#### 2.1.3.5 Zmagovalni naključni igralec

Ta igralec uporablja zmagovalno naključno strategijo. Tudi on, kadar je mogoče, igra optimalno, sicer pa se odloča naključno.

#### 2.1.3.6 Zmagovalni uteženi igralec

Zmagovalni uteženi igralec se obnaša enako kot Zmagovalni naključni igralec — uporablja zmagovalno naključno strategijo. Razlika tema igralcema je, da se ta odloča s pomočjo porazdelitev, Zmagovalni naključni pa s pogojnim stavkom (if, elif, else).

#### 2.1.3.7 Izgubljivi uteženi igralec

Tudi ta igralec se odloča na podlagi porazdelitev. Pri njem so le te nastavljene tako, da bo, če je le mogoče, izgubil. Tudi tega igralca v raziskavi ne uporabljamo.

#### 2.1.3.8 Učljivi igralec

Učljivi igralec nima točno določene strategije, saj si jo sam pridobiva s spodbujevanim učenjem, ki je podrobneje opisano v poglavju o umetni inteligenci in v razdelku *Spodbujevano učenje za igro Nim*. Gre za to, da ta igralec najprej igra z naključno strategijo, nato pa jo po vsaki igri, ki jo odigra, nekoliko popravi, da je bolj učinkovita.

#### 2.1.3.9 Genetski igralec

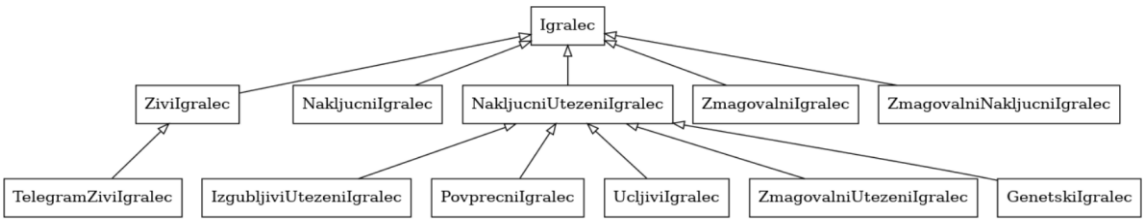
Genetski igralec ima vse funkcije, ki jih potrebuje za to, da lahko sodeluje v genetskem algoritmu. Podrobneje ga opišemo v poglavju o genetskem algoritmu.

#### 2.1.3.10 Povprečni igralec

Ta igralec tudi nastopa v genetskem algoritmu. Tako kot lahko damo Naključnemu uteženemu igralcu nekaj uteži za eno strategijo, pa damo Povprečnemu igralcu veliko strategij. On nato iz vseh danih strategij oblikuje eno — povprečno.

### 2.1.4 Igralci kot Python razredi

Vsak od teh igralcev je v programu predstavljen kot Python razred. Ker pa imajo vsi igralci dokaj podobne funkcije, so nekateri igralci *izpeljani* iz drugih igralcev. To pomeni, da *podrazred* od *nadrazreda* funkcije *podeduje* in jih nato prilagodi po svojih potrebah. Na spodnjem diagramu lahko vidimo, v kakšnih odnosih so razredi.




*Slika 1: UML diagram igralcev, narisan z orodjem Pylint.*

Vidimo, da so vsi igralci izpeljani iz razreda *Igralec*. Ta razred sam po sebi ne dela ničesar. Njegova naloga je, da ostali igralci od njega *podedujejo* funkcije, ki jih potrebujejo čisto vsi igralci. Neposredno iz razreda *Igralec* so izpeljani Živi igralec, Naključni igralec, Naključni uteženi igralec, Zmagovalni igralec in Zmagovalni naključni igralec.

Med drugimi funkcijami, ki jih ti igralci podedujejo, je tudi funkcija *poteza(ne_vzeto)*. Kot parameter dobi število ne vzetih žetonov, torej tistih, ki so še zmeraj na polju. Vsak od teh igralcev na podlagi te številke na svoj način izbere, koliko žetonov vzame. Živi igralec število prebere s funkcijo *input()*, Naključni igralec s funkcijo *random.randint()*, Zmagovalna igralca pa s pomočjo pogojnih stavkov ugotovita, koliko žetonov morata vzeti.

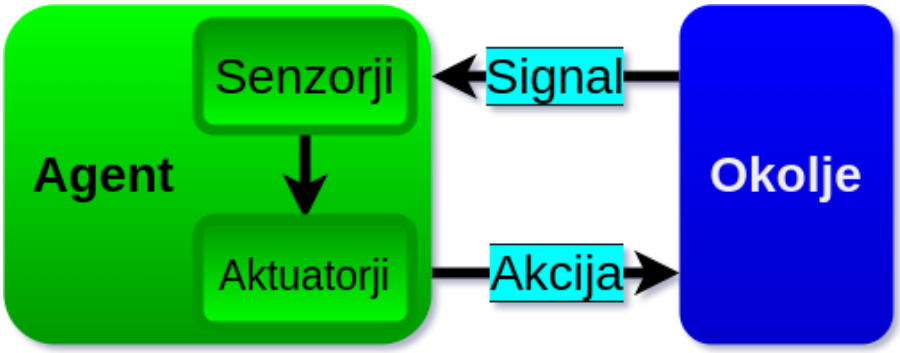
Veliko preostalih igralcev pa je izpeljanih iz Naključnega uteženega igralca, ker ta igralec uporablja strategijo s porazdelitvami. Ta je v programu predstavljena kot slovar, ki ima za ključe število žetonov na polju, vsak od teh ključev pa ima za vrednost seznam, ki predstavlja porazdelitev. Takšna strategija je lahko na primer:

$$\text{strategija} = \{1 : [1.0, 0.0], 2 : [1.0, 0.0], 3 : [0.0, 1.0], 4 : [0.5, 0.5]\}$$

Tak sistem poteze, ki upošteva porazdelitve v strategiji, potrebujejo tudi številni drugi igralci, ki so zato izpeljani iz Naključnega uteženega. To so Učljivi igralec, Genetski igralec, Povprečni igralec, Zmagovalni uteženi igralec in Izgubljivi uteženi igralec.

## 2.2 Umetna inteligenca, spodbujevano učenje

Umetna inteligenca je obsežno področje računalništva. Literatura (Guid in Strnad, 2007) navaja, da bi jo lahko opisali kot študij *inteligentnih agentov*, ki s *senzorji* zaznavajo in sprejemajo podatke iz okolice in na podlagi le teh z *aktuatorji* izvedejo akcije. Agent je umeščen v okolje, v katerem samostojno deluje.

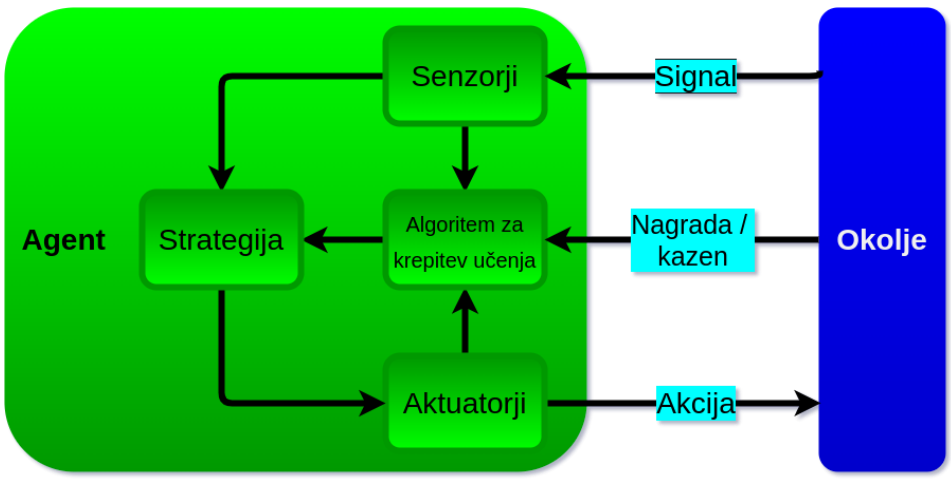



*Slika 2: Delovanje inteligentnega agenta.*

Poznamo več različnih vrst inteligentnih agentov:

- **Preprosti odzivni agenti**, ki izbirajo akcije na osnovi signala.
- **Odzivni agenti, temelječi na modelu**, ki izbirajo akcije na osnovi zavedanja, kaj njegove akcije naredijo v okolju in zgodovine zaznavanja signala.
- **Agenti, temelječi na cilju**, ki poleg prej omenjenega pri izbiri akcije upoštevajo tudi cilj.
- **Agenti, temelječi na koristi**, ki tako kot agenti, temelječi na cilju, iščejo cilj. Ti agenti pa pri tem uporabijo najugodnejšo pot.
- **Učeči se agenti**, ki se učijo, kako v dani situaciji najbolje odreagirati.

Delovanje učečega se agenta prikazuje spodnja slika:




*Slika 3: Delovanje učečega se agenta.*

Učeči se agent od okolja prejme signal (v našem primeru je to število žetonov na polju), agent pa se nanj odzove tako, kot od njega zahteva strategija. Ta se pa spreminja glede na to, ali je v igri zmagal ali ne (agenta se nagradi ali kaznuje). Tak način učenja imenujemo *spodbujevano učenje*.

Na Wikipediji (Wikipedija, Spodbujevano učenje, 2023) piše, da gre pri spodbujevanem učenju za strojno učenje, čigar cilj je priučiti ali optimizirati vedenje na podlagi povratne informacije (nagrajevanja oz. kaznovanja). Učeči se agent, ki se uči s spodbujevanim učenjem, torej v zanki izbira možnosti, ki so mu v dani situaciji na voljo. Če se izkaže, da je izbral pravilno možnost, je nagrajen, sicer pa kaznovan. Agent želi biti čim večkrat nagrajen, zato po vsaki povratni informaciji svojo strategijo prilagodi — če je nagrajen, poveča verjetnost, da se naslednjič odloči enako, sicer pa verjetnost za kaj takega zmanjša.

## 2.3 Genetski algoritem

Genetski algoritem je eden izmed evolucijskih algoritmov. V literaturi (Kavran, 2008) piše, da evolucijski algoritmi jemljejo ideje iz žive narave, kjer vse temelji na nenehnem spreminjanju živih bitij in borbi za obstanek. Evolucija tako poskrbi, da so bitja v daljšem časovnem obdobju vedno bolj izpopolnjena in prilagojena na okolico. V računalništvu uporabljamo evolucijske algoritme takrat, ko ne poznamo optimalnih rešitev in smo zadovoljni že z rešitvami, ki so optimalni rešitvi blizu. V našem primeru je to takrat, ko igro Nim igra več igralcev kot dva.

Genetski algoritem se začne z generiranjem začetne populacije, ki je oblikovana čisto naključno. Po generiranju sledi vstop v zanko, ki se izvaja tako dolgo, dokler ni izpolnjen kriterij zaustavitve. Ta je odvisen od trenutne populacije in drugih parametrov. V tej zanki najprej generacija vstopi v okolje (v našem primeru je to, da igrajo igro), na koncu pa oblikujejo novo generacijo. To poteka v treh fazah: selekciji, križanju in mutaciji.

Selekcija v naravi deluje kot izbira staršev in je povezana z uspešnostjo teh dveh posameznikov. To pomeni, da imajo posamezniki, ki so bolje prilagojeni okolju, večjo verjetnost za obstanek.

Proces, ki skrbi za izmenjavo genetskih informacij, je križanje. Pri križanju gre za mešanje genetskih informacij staršev, ki dajejo v pravi kombinaciji potomcu neko novo kvaliteto. Križanje skrbi tudi za raznolikost populacije.

Drugi vir raznolikosti v populaciji je mutacija, ki pomeni naključno spremembo genskega materiala posameznika.

## 2.4 Uporaba za igro Nim

### 2.4.1 Spodbujevano učenje za igro Nim

Pri spodbujevanem učenju, ki je obravnavano tudi v naši lanski raziskovalni nalogi (Bokal, 2022) se Učljivi igralec uči na podlagi odigranih iger. Torej z enim od preostalih simuliranih igralcev najprej večkrat igra to igro. Igrati začne s prej omenjeno naključno strategijo, nato pa si po vsaki potezi zabeleži, koliko žetonov je vzel pri kolikšnih žetonih na polju. Če v igri zmaga, poveča verjetnost, da v prihodnjih igrah v teh situacijah spet vzame toliko žetonov, kot jih je vzel zdaj, poleg tega pa tudi zmanjša verjetnost, da vzame število žetonov, ki jih v igri ni vzel. Če je v igri poražen, stori obratno. Za lažjo predstavo si oglejmo preprost primer dveh iger za dva igralca, pri katerih so na polju trije žetoni, vsak od igralcev pa lahko v svoji potezi vzame največ dva žetona. Recimo, da igro začne Učljivi igralec. Prva igra bi potem lahko bila taka:

*Tabela 1: Razlaga spodbujevanega učenja (prva igra).*

| Strategija pred igro | Igra | Rezultat igre | Strategija po igri |
|---|---|---|---|
| Porazdelitev pri 1 žetonu = $[1,0]$<br>Porazdelitev pri 2 žetonih = $[\frac{1}{2},\frac{1}{2}]$<br>Porazdelitev pri 3 žetonih = $[\frac{1}{2},\frac{1}{2}]$ | 1. Učljivi vzame 1 žeton.<br>2. Soigralec vzame 1 žeton.<br>3. Učljivi vzame zadnji žeton. | Učljivi igralec je poražen. | Porazdelitev pri 1 žetonu = $[1,0]$<br>Porazdelitev pri 2 žetonih = $[\frac{1}{2},\frac{1}{2}]$<br>Porazdelitev pri 3 žetonih = $[\frac{1}{4},\frac{3}{4}]$ |

Igralec po igri spozna, da se mu, ko ima pred seboj tri žetone, ne splača vzeti enega. Ker si tega ne upa z gotovostjo trditi, še ne onemogoči tega, da bi spet kdaj vzel en žeton, ampak verjetnost za kaj takega vseeno močno zmanjša. S situacijo, ko ima pred seboj dva žetona, se ne sreča, zato porazdelitev ostaja enaka. Ko ostane le še en žeton, ne more vzeti dveh, zato ostane porazdelitev tudi tu enaka. Učljivi igralec naslednjo igro začne igrati s strategijo, ki jo je razvil v prejšnji igri. Če je prva igra takšna, kot je v našem primeru, je druga najverjetneje taka:

*Tabela 2: Razlaga spodbujevanega učenja (druga igra).*

| Strategija pred igro | Igra | Rezultat igre | Strategija po igri |
|---|---|---|---|
| Porazdelitev pri 1 žetonu = $[1,0]$<br>Porazdelitev pri 2 žetonih = $[\frac{1}{2},\frac{1}{2}]$<br>Porazdelitev pri 3 žetonih = $[\frac{1}{4},\frac{3}{4}]$ | 1. Učljivi vzame 2 žetona.<br>2. Soigralec vzame zadnji žeton. | Učljivi igralec zmaga. | Porazdelitev pri 1 žetonu = $[1,0]$<br>Porazdelitev pri 2 žetonih = $[\frac{1}{2},\frac{1}{2}]$<br>Porazdelitev pri 3 žetonih = $[\frac{1}{8},\frac{7}{8}]$ |

Učljivi igralec je sedaj pridobil strategijo, s katero pri treh žetonih na polju že zelo verjetno vzame dva žetona. S ponavljanjem iger postaja verjetnost, da v takšnih okoliščinah vzame en žeton, vse manjša.

### 2.4.2 Genetski algoritem za igro Nim

Osrednji del raziskovalne naloge je genetski algoritem. V njem sodelujejo tako imenovani Genetski igralci. Genetski igralci delujejo podobno kot Naključni uteženi igralci, le da imajo nekaj dodatnih funkcij za potrebe genetskega algoritma.

Na začetku se ustvari populacija Genetskih igralcev, vsak od njih pa ima naključno narejeno strategijo (to ne pomeni, da imajo naključno strategijo, ki ima vse verjetnosti enake, pač pa, da je vsaka porazdelitev naključno narejena). Tako imamo prvo, začetno populacijo igralcev, vsak od njih pa ima svojo strategijo.

Ko je populacija ustvarjena, igralci začnejo igrati igro. Poljuben igralec (označimo ga z A) z vsakim drugim igralcem (označimo jih z B) odigra deset iger. Od tega pri pet igrah začne igralec A, pri preostalih 5 pa igralec B. Po vsaki igri si vsak igralec pri sebi zabeleži, ali je zmagal, ali ne.

Ko so igre končane, je čas za oblikovanje nove generacije igralcev. Najprej se izvede selekcija — program razvrsti igralce po vrsti glede na to, kako so bili v igrah uspešni (o tem, kaj pomeni, da je igralec uspešen, kasneje). Program potem podvoji dva najboljša igralca in ju nespremenjena prenese v naslednjo generacijo, izmed preostalih pa s tretjo funkcijo, ki jo vidimo na **algoritmu 1**, izbere igralce, potomci katerih bodo šli v naslednjo generacijo.

*Algoritem 1: Selekcija.*

```
def posortiraj_igralce(self, igralci): # Ta funkcija kot parameter dobi seznam vseh igralcev v populaciji.
    posortirano = sorted(igralci, key=lambda igralec: POGOJ ZA USPEŠNOST, reverse=True)
    return posortirano # Na koncu vrne seznam vseh igralcev, ki so posortirani glede na to, kako so bili uspešni.


def izberi_igralca(self, igralci):
    # Ta funkcija poskrbi, da je večja verjetnost, da bo šel v naslednjo generacijo uspešnejpi igralec.
    return random.choices(population=igralci, weights=[POGOJ ZA USPEŠNOST])[0]


def selekcija(self, igralci): # Ta funkcija kot parameter dobi posortirane igralce.
    izbranci = [] # Najprej naredimo seznam igralcev, potomci katerih bodo šli v naslednjo generacijo.
    while len(izbranci) < self.stevilo_igralcev - self.velikost_elite: # Seznam igralcev bomo sedaj napolnili.
        izbran_igralec = self.izberi_igralca(igralci) # 1. Izberemo igralca.
        igralci.remove(izbran_igralec) # 2. Izbranega igralca izbrišemo iz seznama (neizbranih) igralcev.
        izbranci.append(izbran_igralec) # 3. Izbranega igralca dodamo v seznam izbrancev.
    return izbranci # Funkcija nato vrne seznam izbranih igralcev.
```

Nato se začne križanje. Program najprej naključno izbere po dva igralca (verjetnost, da program izbere uspešnejšega igralca, je večja od verjetnosti, da bo izbran manj uspešen igralec). Zopet imenujmo enega igralec A, drugega pa igralec B. V tej fazi program s kombiniranjem strategij igralca A in igralca B naredi dve novi strategiji. Kombiniranje strategij si oglejmo na primeru:

*Tabela 3: Razlaga kombiniranja strategij (pred križanjem).*

| Strategija A | Strategija B |
|---|---|
| {1: [1.0, 0.0], 2: [0.4, 0.6], 3: [0.7, 0.3], 4: [0.3, 0.7]} | {1: [1.0, 0.0], 2: [0.8, 0.2], 3: [0.1, 0.9], 4: [0.5, 0.5]} |

Torej imamo dva igralca in vsak od njiju ima svojo strategijo. Ti dve strategiji program spremeni v dve novi strategiji (imenujmo ju C in Č). To naredi tako, da gre v zanki od porazdelitev za en žeton do porazdelitve za (v našem primeru) štiri žetone, in pri tem porazdelitve naključno shrani v strategijo C ali v strategijo Č. Po kombiniranju strategij A in B v tabeli lahko nastaneta npr. takšni strategiji C in Č:

*Tabela 4: Razlaga kombiniranja strategij (po križanju).*

| Strategija C | Strategija Č |
|---|---|
| {1: [1.0, 0.0], 2: [0.4, 0.6], 3: [0.1, 0.3], 4: [0.3, 0.7]} | {1: [1.0, 0.0], 2: [0.8, 0.2], 3: [0.1, 0.9], 4: [0.5, 0.5]} |

Vidimo, da se porazdelitve prejšnjih strategij enakomerno razporedijo v novi strategiji. Tiste porazdelitve, ki jih je strategija C prevzela od strategije A, je strategija Č prevzela od strategije B in obratno. Ko sta novi strategiji ustvarjeni, program naredi dva nova Genetska igralca in vsakemu dodeli svojo strategijo.

Tako, kot smo pokazali na zgornjem primeru, program ustvari igralce za novo populacijo. Program za križanje je prikazan tudi na **algoritmu 2**.

*Algoritem 2: Križanje.*

```
def kombiniraj_strategiji(self, strategija_a, strategija_b, verjetnosti): # Ta funkcija skombinira dve strategiji.
    # Tu naredimo dva prazna slovarja za novi strategiji
    strategija_a2 = {}
    strategija_b2 = {}
    for a in range(1, len(list(strategija_a.keys())) + 1): # Nato slovarja napolnimo.
        if verjetnosti[a - 1] == 0:
            strategija_a2[a] = strategija_a[a]
            strategija_b2[a] = strategija_b[a]
        else:
            strategija_a2[a] = strategija_b[a]
            strategija_b2[a] = strategija_a[a]
    return strategija_a2, strategija_b2 # Na koncu funkcija vrne novi strategiji.


def krizanje(self, izbranci): # Ta funkcija kot parameter dobi izbrane igralce.
    nova_generacija = [] # Najprej naredimo prazen seznam za novo generacijo.
    a = 0
    while len(nova_generacija) < self.stevilo_igralcev - 2: # Senam sedaj napolnimo z novimi igralcimi
        # V ta seznam shranimo prekrižani strategiji, dobljeni z zgornjo funkcijo.
        novi_strategiji = self.kombiniraj_strategiji(izbranci[a].strategija, izbranci[a + 1].strategija,
                                                     [random.randint(0, 1) for a in range(self.zetoni)])
        for x in novi_strategiji: # Novi strategiji nato dodelimo dvema novima igralcema.
            nov_igralec = igralci_s_porazdelitvijo.GenetskiIgralec(self.max_vzeto, self.zetoni,
                                                                   self.stevilo_igralcev_v_igri * self.vzv * self.stevilo_igralcev)
            nov_igralec.strategija = x
            nova_generacija.append(nov_igralec)
        a += 2
    return nova_generacija # Funkcija vrne prekrižane igralce, ki se v novi generaciji pridružijo najboljšima dvema.
```

Da v Genetskem algoritmu ne obravnavamo vedno istih porazdelitev na različnih koncih, vsak igralec svojo strategijo še naključno *mutira* tako, kot vidimo na **algoritmu 3**.

*Algoritem 3: Mutacija.*

```
def mutiraj_kljuc(self, porazdelitev, max_sprememba): # Ta funkcija naključno spremeni izbrano porazdelitev
    nova_porazdelitev = [] # Najprej naredimo seznam za novo porazdelitev.
    porazdelitev = porazdelitev[:-1] # Obstoječi verjetnosti zamenjamo.
    for verjetnost in porazdelitev: # Ta zanka gre čez vse verjetnosti v porazdelitvi.
        # Najprej izračunamo, za koliko se bo vrednost spremenila.
        sprememba = min(1, max(0, random.random() * max_sprememba - max_sprememba / 2))
        if sum(nova_porazdelitev) + verjetnost + sprememba > 1:
            verjetnost = 0 # Ker mora biti vsota obeh verjetnosti v porazdelitvi 1 in tu preprečimo težave s tem.
        else:
            verjetnost += sprememba # Če težav ni, se verjetnost spremeni kot smo načrtovali.
        nova_porazdelitev.append(verjetnost) # Nova verjetnost se doda v seznam „nova_porazdelitev“

    nova_porazdelitev.append(1 - sum(nova_porazdelitev)) # Poleg izračunane verjetnosti dodamo še nasprotno.
    return nova_porazdelitev

def mutiraj(self, verjetnost, velikost_mutacije):
    # To funkcijo ima vsak Genetski igralec. Skrbi za mutacijo strategije.
    for a in self.strategija.keys(): # Ta zanka gre čez vse porazdelitve v strategiji.
        if random.random() <= verjetnost: # Pri naključnih porazdelitvah se zgodi sprememba.
            self.strategija[a] = self.mutiraj_kljuc(self.strategija[a], velikost_mutacije)
```

Na tak način se ustvari nova populacija, ki je enako številčna kot prejšnja. V to populacijo so z največjo verjetnostjo predani *potomci* najuspešnejših iz prejšnje, torej so igralci vedno bolj prilagojeni. Cikel se nato ponavlja, dokler ne doseže omejitve za število generacij. Algoritem se lahko tudi predčasno konča, če so igralci v populaciji dovolj podobni. Podobnost igralcev izračunamo tako, da ustvarimo Povprečnega igralca, ki mu kot parameter damo strategije vseh igralcev v populaciji. Ko je Povprečni igralec narejen, s pomočjo Pitagorovega izreka za več dimenzij vsak igralec izračuna, kolikšna je razdalja med njegovo strategijo in strategijo povprečnega igralca. Program nato izračuna povprečje vseh teh razdalj in tako ve, kako se igralci v populaciji razlikujejo med seboj.

Ker pa se lahko zgodi, da se zaradi določenih okoliščin drugačnost med igralci v populaciji ne zmanjšuje, je trajanje algoritma tudi omejeno na število ponovitev cikla.

Igralci se pri oblikovanju nove generacije razvrstijo glede na uspešnost. Kaj pomeni, da so igralci uspešni, pa lahko določimo sami, zato je mogoče ta genetski algoritem izvesti na več različnih načinov. V raziskavi ga izvajamo na tri načine:

- Način, kjer igralčeva uspešnost raste z deležem njegovih zmag med vsemi odigranimi igrami.

- Način, kjer igralčeva uspešnost raste z deležem njegovih zmag med tistimi igrami, kjer je bil na potezi drugi (ker je na začetku igre v genetskem turnirju deset žetonov, to pomeni, da ima igralec, ki igro začne, manj možnosti).

- Način, kjer je igralec bolj uspešen, če je njegova strategija bolj podobna prej omenjeni optimalni strategiji. Ta uporabljamo zato, da preverimo, če genetski algoritem deluje.

# 3 Eksperimentalni del

## 3.1 Struktura programa

Iskanje strategije s spodbujevanim učenjem in z genetskim algoritmom primerjamo v že večkrat omenjenem Python programu, ki simulira igralce in igro. Za simulacijo spodbujevanega učenja in genetskega algoritma uporabljamo dva različna sistema, ki ju opišemo v naslednjih razdelkih.

### 3.1.1 Struktura spodbujevanega učenja

Pri izvajanju znanstvenih poskusov velja, da več kot izvedemo ponovitev, zanesljivejši so rezultati. Ker je za to, da računalnik najde strategijo s spodbujevanim učenjem, potreben le en agent (Učljivi igralec), si lahko privoščimo, da program izvede več ponovitev učenj in to pod različnimi pogoji. S tem pridobimo veliko podatkov. Program podatke pridobiva z naslednjim sistemom:

- Naš najbolj osnoven pojem je *igra*. V eni igri se pomerita Učljivi igralec in eden izmed preostalih. Po koncu igre en igralec zmaga, drugi pa je poražen.

- V eni *tekmi* se izvede veliko iger. Vse igre v eni tekmi odigrata dva **ista** igralca. Torej Učljivi igralec prvo igro v tekmi igra z naključno strategijo, med tekmo pa se le-ta izboljšuje.

- V enem *turnirju* se izvede več tekem. Vsako tekmo igra **nov** par igralcev **istega tipa**. Torej, če imamo turnir, v katerem igrajo npr. Učljivi igralci in Zmagovalni igralci, prvo tekmo odigrata po en Učljivi in po en Zmagovalni igralec. Ko se tekma konča, se začne nova tekma, v kateri se pomerita nov Zmagovalni igralec in nov Učljivi igralec, ki se mora od začetka učiti optimalne strategije.

- Naslednjo stopnjo imenujemo *primerjava uspešnosti*. V njej se izvede več turnirjev. Pri tem še vedno tekmujeta enaka tipa igralcev, le da se po vsakem novem turnirju spremeni število žetonov, ki so pred začetkom na polju. S tem preverimo prilagodljivost spodbujevanega učenja.

- Največji sklop predstavlja *raziskava*, v kateri se izvede več primerjav uspešnosti. Ta zagotovi, da se izvedejo vse potrebne kombinacije tipov igralcev.

Pri izvajanju tega programa računalnik zabeleži vse, kar se dogaja, in nato na podlagi tega ustvari diagrame, ki nam pregledno prikažejo najpomembnejše podatke.

### 3.1.2 Struktura genetskega algoritma

Za razliko od spodbujevanega učenja iskanje strategije z genetskim algoritmom ni učenje posameznika, pač pa učenje populacije in posledično traja dlje. Zato ne bomo simulirali ogromnega števila ponovitev genetskega algoritma, ampak bomo vsakega izvedli posebej. Ker pa je že en sam genetski algoritem razdeljen na več delov, tudi tokrat potrebujemo sistem. Naš program uporablja takšnega:

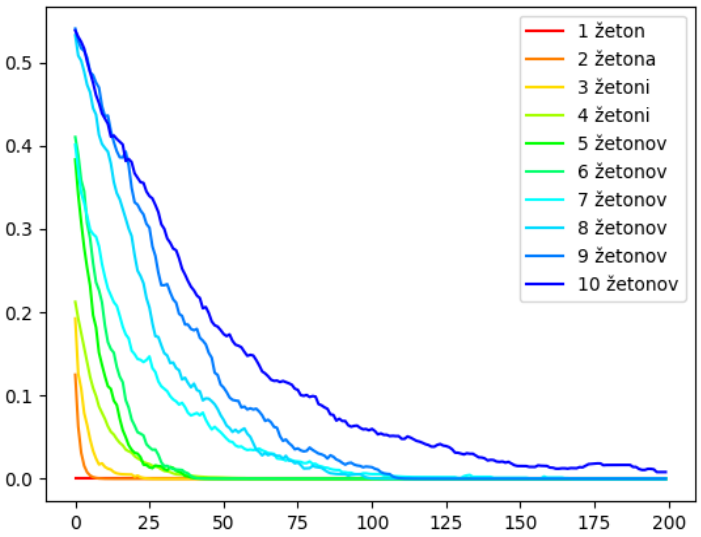
- Še vedno je najbolj osnoven pojem *igra*. V eni igri sta dva genetska igralca. Po koncu igre eden od njiju zmaga, drugi pa izgubi.
- Več iger se izvede v *genetski tekmi*. V njej se izvaja tisti del genetskega algoritma, ko vsak igralec v populaciji z vsakim odigra deset iger.
- Ostali procesi genetskega algoritma se izvedejo v *genetskem turnirju*. Ta poskrbi za generiranje populacije, izvedbo genetske tekme in ustvarjanje populacije (s selekcijo, križanjem in mutacijo).

## 3.2 Rezultati in razprava eksperimentalne raziskave

### 3.2.1 Rezultati spodbujevanega učenja

Ko računalnik izvede prej opisano raziskavo za spodbujevano učenje nastanejo grafi, ki jih opišemo v tem poglavju, povzetem po Bokal, 2022. Najprej pa razložimo pomen grafov:

x = zaporedna igra
y = povprečna razdalja do optimalne strategije (za vsako število začetnih žetonov)



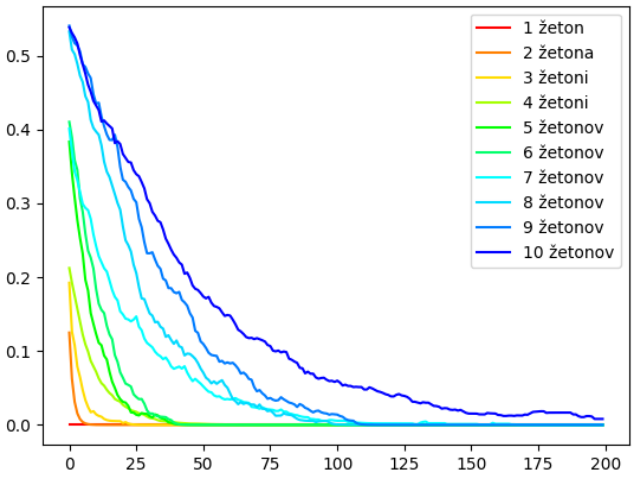

*Slika 4: Primer grafa za prikaz spodbujevanega učenja*

Takšen graf nastane po vsaki primerjavi uspešnosti. V eni primerjavi uspešnosti se izvede več turnirjev in v enem turnirju več tekem. Program si torej v turnirju za vsako tekmo zabeleži, kako je skozi igre v tekmi napredoval učljivi igralec. Nato program izračuna, kako je razdalja do optimalne strategije povprečno padala. *Graf povprečnega učenja* je skupek vseh teh povprečnih krivulj, ki so nastale v turnirjih ene primerjave uspešnosti.

Zdaj sledi ogled grafov povprečnega učenja. Imamo tri grafe, vsak od njih pa prikazuje rezultate za učenje pri enem tipu soigralca.

#### 3.2.1.1 Učljivi in Naključni igralec

Ko Učljivi igralec igra z Naključnim, program naredi takšen graf:

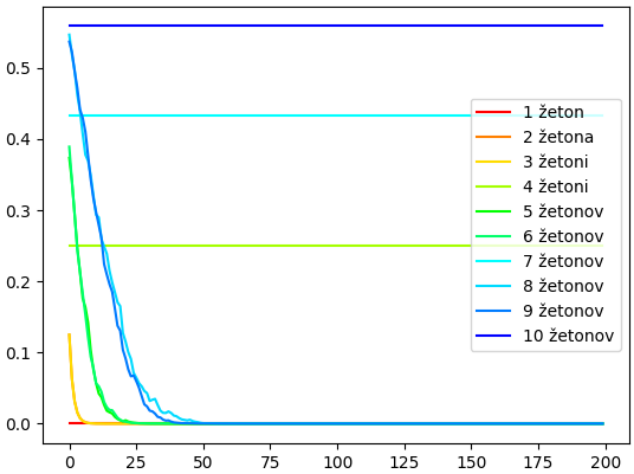



*Slika 5: Povprečna razdalja do optimalne strategije (Učljivi : Naključni)*

Vidimo, da Učljivi igralec vedno pride do optimalne strategije. Graf tudi nazorno prikaže, da več kot je na začetku igre žetonov, več časa potrebuje Učljivi igralec, da se nauči strategije.

#### 3.2.1.2 Učljivi in Zmagovalni naključni igralec

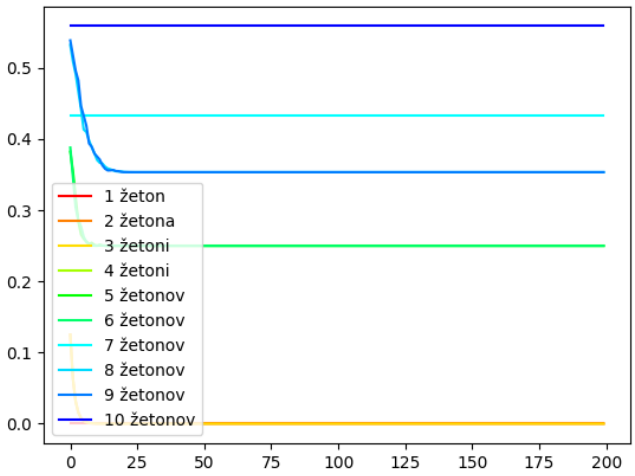
Graf na **sliki 6** nam prikazuje, kako se uči Učljivi igralec, ko igra proti Zmagovalnemu naključnemu. Vidimo, da takrat, ko je na začetku igre 3 × n + 1 žeton, Učljivi igralec optimalne strategije nikoli ne najde, v ostalih primerih pa do nje pride bistveno hitreje kot pri Naključnem.




*Slika 6: Povprečna razdalja do optimalne strategije (Učljivi: Zmagovalni naključni).*

#### 3.2.1.3 Učljivi in Zmagovalni igralec

Pričakovali bi, da je z rezultati pri Zmagovalnem igralcu dokaj podobno kot pri Zmagovalnem naključnem igralcu, če ne še boljše. A na **sliki 7** vidimo, da ni tako. Zakaj?




*Slika 7: Povprečna razdalja do optimalne strategije (Učljivi : Zmagovalni).*

Razlog za to, da Učljivi igralec ne pride do optimalne strategije tudi takrat, ko na začetku igre ni 3 × n + 1 žeton, je sledeč: Vedno, ko Zmagovalni igralec pred seboj vidi 3 × n + 1 žeton, vzame enega. Učljivi igralec se zato nikoli ne sreča z nekaterimi situacijami in tako zanje ne pridobi optimalne porazdelitve. Učljivi igralec sicer hitro najde strategijo, s katero lahko premaga Zmagovalnega, ampak zaradi določenosti njegove strategije, končna naučena strategija ni optimalna.

### 3.2.2 Rezultati genetskega algoritma

Za prikaz rezultatov genetskega algoritma uporabimo grafe, predstavljene v tem razdelku. Vsak od njih nam prikazuje, kako so se v enem genetskem turnirju spreminjale naslednje vrednosti:

- **Povprečna razdalja do povprečnega igralca** (ta nam pove, kako so si genetski igralci v generaciji podobni med seboj — program za vsakega igralca izračuna, za koliko se razlikuje od *Povprečnega igralca*, in nato izračuna povprečje dobljenih razdalj).

- **Največja razdalja do povprečnega igralca** (ta prikazuje razdaljo do povprečnega igralca, ki najbolj odstopa).

- **Največji delež zmag** (pove nam, pri kolikšnem deležu odigranih iger je zmagal igralec, ki je zmagal največkrat).

- **Največji delež zmag pri igrah, kjer je bil igralec prvi na potezi** (pove nam, pri kolikšnem deležu iger je zmagal igralec, ki je zmagal največkrat. Pri tej krivulji upoštevamo le tiste igre, pri katerih je bil igralec prvi na potezi).

- **Največji delež zmag pri igrah, kjer je bil igralec drugi na potezi** (pove nam, pri kolikšnem deležu iger je zmagal igralec, ki je zmagal največkrat. Upoštevamo le tiste igre, pri katerih je bil igralec drugi na potezi).

- **Najmanjša razdalja do optimalne strategije** (če sta v eni igri samo dva igralca, si lahko privoščimo, da program izračuna, za koliko se strategije igralcev razlikujejo od optimalne strategije. Ta vrednost nam pove, katera razdalja do optimalne strategije je najmanjša v populaciji).

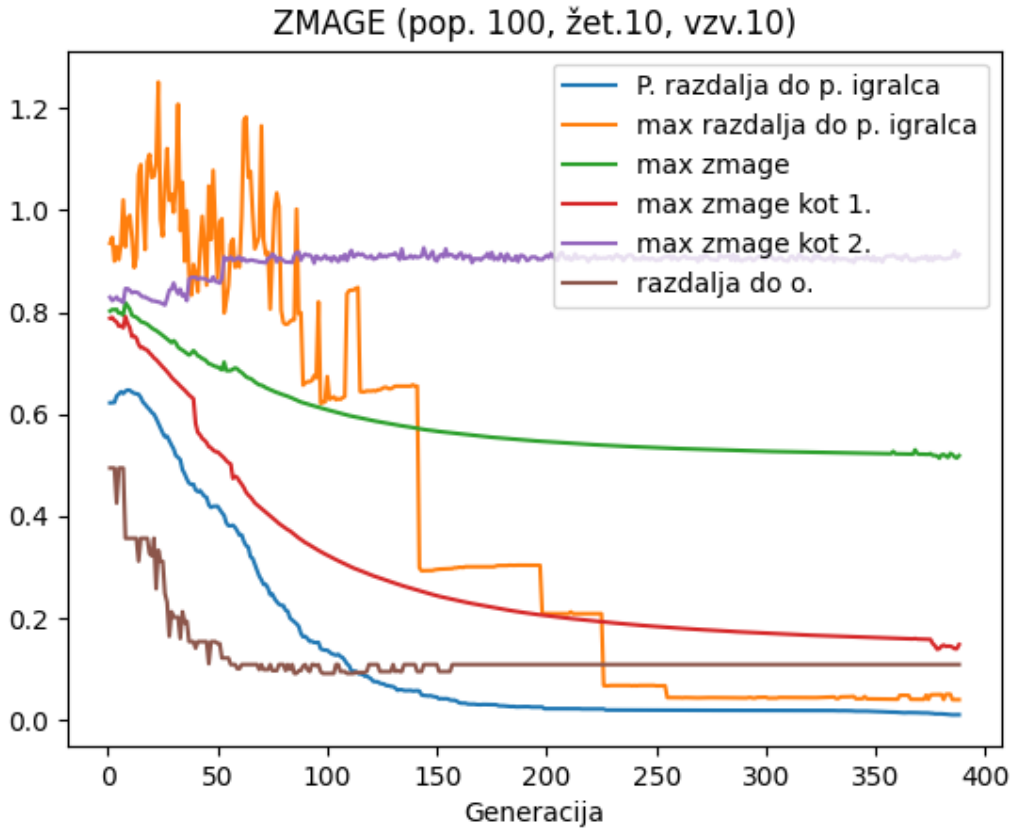
Na podlagi nastalih grafov lahko ugotovimo, kako določitev pogoja, ki pove, kateri igralci bodo predani v naslednjo generacijo, vpliva na to, kako se populacija od generacije do generacije spreminja. V tem razdelku opišemo grafe, ki nastanejo po treh različnih genetskih algoritmih, in sicer:

- Genetski algoritem, pri katerem se v naslednjo generacijo uvrstijo potomci igralcev, ki imajo največji delež vseh zmag.
- Genetski algoritem, ki v naslednjo generacijo pošlje potomce igralcev, ki imajo največji delež zmag pri igrah, kjer so bili na potezi drugi (ker v vseh naštetih turnirjih nastavimo, da je na začetku igre 10 žetonov, imajo igralci, ki igre ne začnejo, več možnosti za zmago).
- Genetski algoritem, pri katerem v naslednjo generacijo tvorijo potomci tistih igralcev, ki imajo čim manjšo razdaljo do optimalne strategije.

Vse te genetske algoritme izvajamo pod enakimi pogoji — pri vseh populacija šteje 100 igralcev, na začetku igre je 10 žetonov in igralci *vsak z vsakim* (vzv.) odigrajo 10 iger. Genetski algoritem se neha izvajati po 500. generaciji, razen če se zaradi velike enakosti med igralci predčasno zaustavi.

#### 3.2.2.1 Vse zmage

Graf na **sliki 8** nastane po izvedbi genetskega turnirja, kjer se v naslednjo generacijo uvrstijo potomci igralcev, ki imajo največji delež zmag (največjega v populaciji prikazuje zelena črta na grafu).



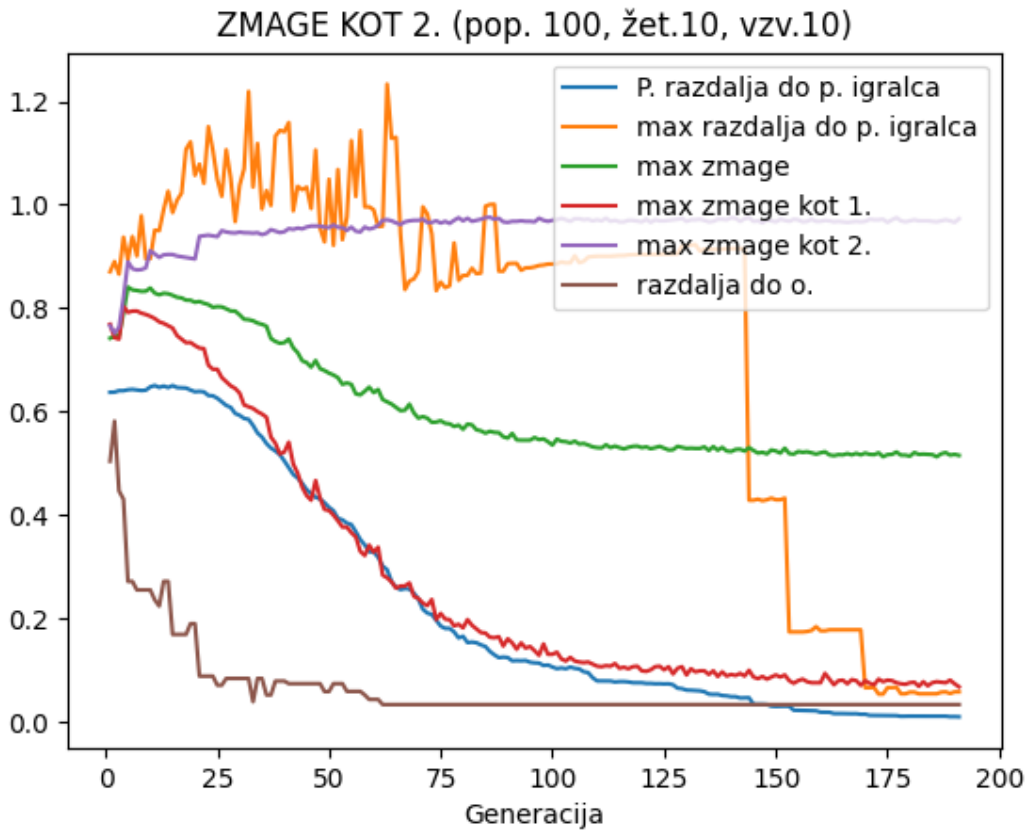

*Slika 8: Genetski turnir (kriterij = vse zmage)*

Z grafa je mogoče razbrati, da od generacije do generacije igralci postajajo podobnejši (to nam prikazujeta modra in oranžna krivulja). Genetski turnir se zaradi prevelike podobnosti ustavi še pred 500. generacijo.

Vidimo tudi, da se razdalja do optimalne strategije (rjava krivulja) okoli petdesete generacije spusti do približno 0,1. Hkrati, kot pada razdalja do optimalne strategije, padata tudi krivulji za delež vseh zmag in za delež zmag, kjer je igralec prvi na potezi. To je zato, ker je, ko so igralci še nevešči optimalne strategije, možno zmagati tudi, ko je igralec prvi na potezi. Ko pa so čez čas v populaciji le še igralci, ki igrajo skoraj optimalno, kaj takega ni več tako lahko doseči. Krivulja, ki prikazuje delež zmag pri igrah, ki jih igralec ne začne, pa ostaja malo pod 1, saj z optimalno strategijo v takih situacijah ni težko zmagati.

#### 3.2.2.2 Zmage kot drugi

Večkrat smo že omenili, da kadar je na polju 3 × n + 1 žeton in igramo proti igralcu, ki uporablja optimalno strategijo, ne moremo zmagati. Takšno je tudi število žetonov 10, ki je na začetku iger v našem genetskem algoritmu. Zato poleg ostalih genetskih algoritmov izvajamo tudi takega, kjer gredo v naslednjo generacijo igralci, ki imajo največji delež zmag le pri igrah, kjer so začeli kot drugi. Po izvajanju takšnega genetskega algoritma nastane graf, ki ga vidimo na **sliki 9**.

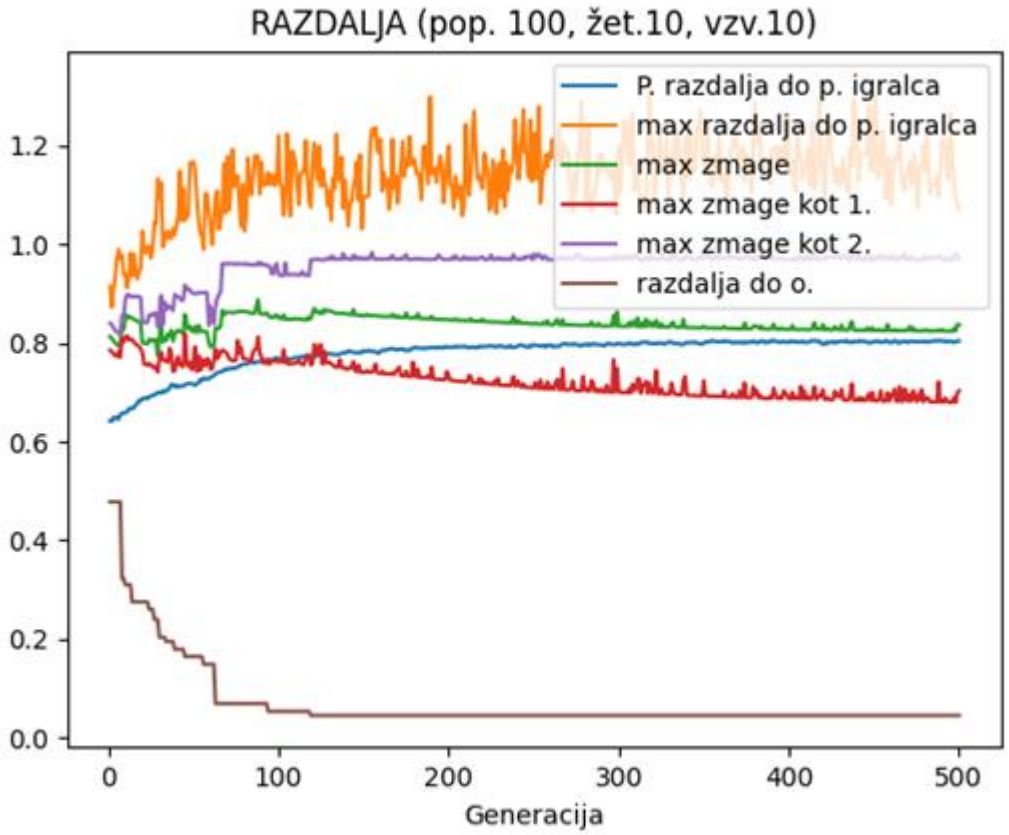



*Slika 9: Genetski turnir (kriterij = zmage kot 2.)*

Krivulje na tem grafu v glavnem težijo v isto smer, kot pri prejšnjem grafu — igralci so med seboj vedno podobnejši (tudi tokrat se je zaradi tega genetski turnir predčasno zaključil), razdalja do optimalne strategije pada itd. Kar je tu drugače, je to, da se razdalja do optimalne strategije na 0,1 spusti že pred 25. generacijo, kasneje se pa spušča še nižje. Torej se igralci tukaj optimalni strategiji hitreje približajo bolj. To pa zato, ker program v prejšnjem primeru (kjer upoštevamo delež vseh zmag) pri ustvarjanju nove generacije lahko izpusti igralca, ki ima sicer zelo učinkovito strategijo, ampak, ker ni imel sreče pri igrah, ki jih je začel igrati, ni veljal za uspešnega. Zdaj pa, ko program upošteva le rezultat tistih iger, ki jih igralec ni začel, se to ne zgodi.

#### 3.2.2.3 Razdalja

Ostal nam je še genetski algoritem, kjer se za pogoj uporablja čim manjša razdalja do optimalne strategije. Tak algoritem nam prikaže graf, ki ga vidimo na **sliki 10**.




*Slika 10: Genetski turnir (kriterij = razdalja do optimalne strategije)*

Prav nič nas ne preseneča, da se igralci optimalni strategiji tokrat še bolj približajo. Je pa zanimivo, da igralci v zgornjem primeru ne postajajo podobni. Modra črta se najprej celo dviguje, nato pa obstane približno na vrednosti 0,8. Domnevamo, da zato, ker pri računanju razdalje do optimalne strategije ne upoštevamo mest, za katere optimalna strategija ne obstaja (3 × n + 1). V genetskem algoritmu zato evolucijski pritisk deluje le na tiste porazdelitve v strategiji, za katere optimalna strategija obstaja. Igralci tako vsak pri sebi pridejo do optimalne strategije, ampak porazdelitve za 3 × n + 1 števila žetonov pa ima vsak igralec drugačne. Genetski turnir se tako zaradi „ne podobnosti“ med igralci izvede do konca, a se po stoti generaciji v glavnem nič ne spremeni.

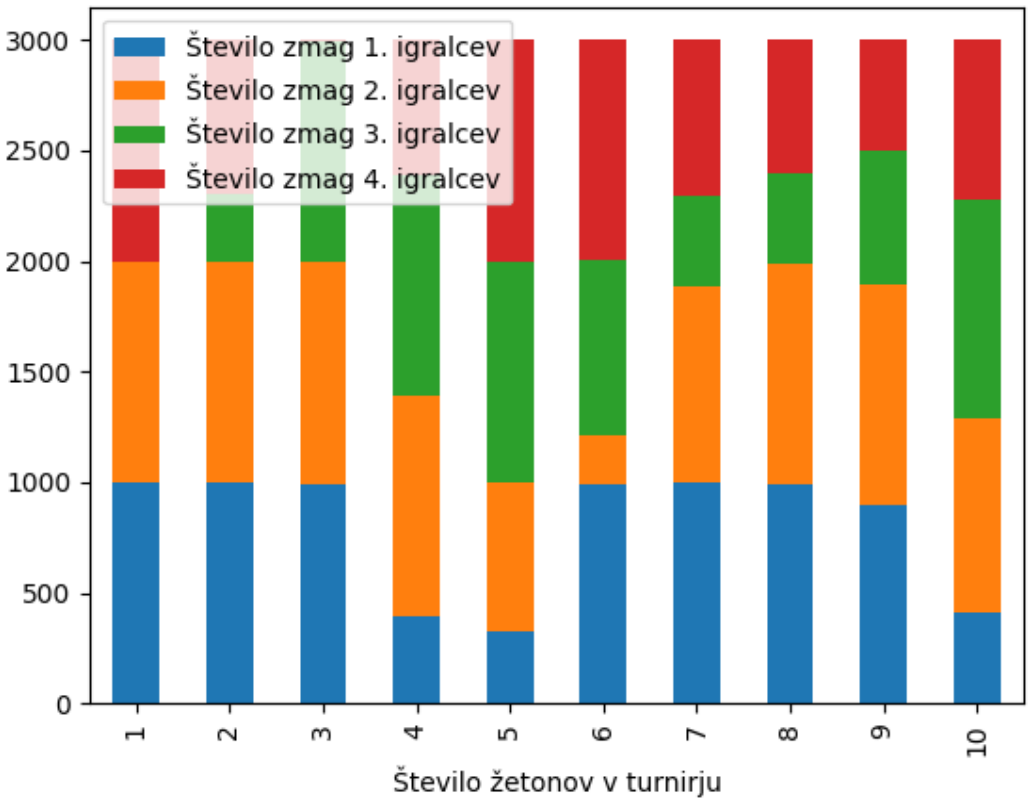
### 3.2.3 Primerjava

Izkazalo se je, da je za iskanje optimalne strategije spodbujevano učenje veliko bolj primerno — vse opisane grafe, ki prikazujejo spodbujevano učenje skupaj, je program izdelal hitreje kot en sam graf, ki prikazuje genetski algoritem. Poleg tega pri spodbujevanem učenju Učljivi igralec izpopolni svojo strategijo do te mere, da je razdalja med njegovo in optimalno strategijo 0, medtem ko pri genetskem algoritmu tega ne dosežemo pri nobenem od opisanih primerov. To nas niti ne bi smelo presenečati — že pri opisovanju genetskega algoritma namreč izvemo, da njegov namen ni iskanje optimalne strategije, pač pa, da se le tej čim bolj približa. Zato se navadno uporablja v tistih primerih, kjer optimalne strategije z matematiko ne moremo najti.

Take situacije imamo tudi pri igri Nim, kadar sta v igri več kot dva igralca. V takem primeru strategija ni odvisna le od števila žetonov, pač pa tudi od tega, kako igralci sodelujejo med seboj. Zato optimalna strategija pri igri Nim za več igralcev ne obstaja v takšni obliki, kot smo vajeni pri igri z dvema igralcema. Kljub temu lahko v programu nastavimo, da igro igra več igralcev in opazujemo rezultate. Morda pa se genetski algoritem tu izkaže za boljšega.

#### 3.2.3.1 Spodbujevano učenje za več igralcev

Ker pri igri za več igralcev ne moremo računati razdalje do optimalne strategije, program tudi ne more narediti grafov, ki bi jo prikazovali. Zato naredimo tu nov diagram - prikaz deleža zmag glede na število žetonov, ki ga lahko vidimo na **sliki 11**.



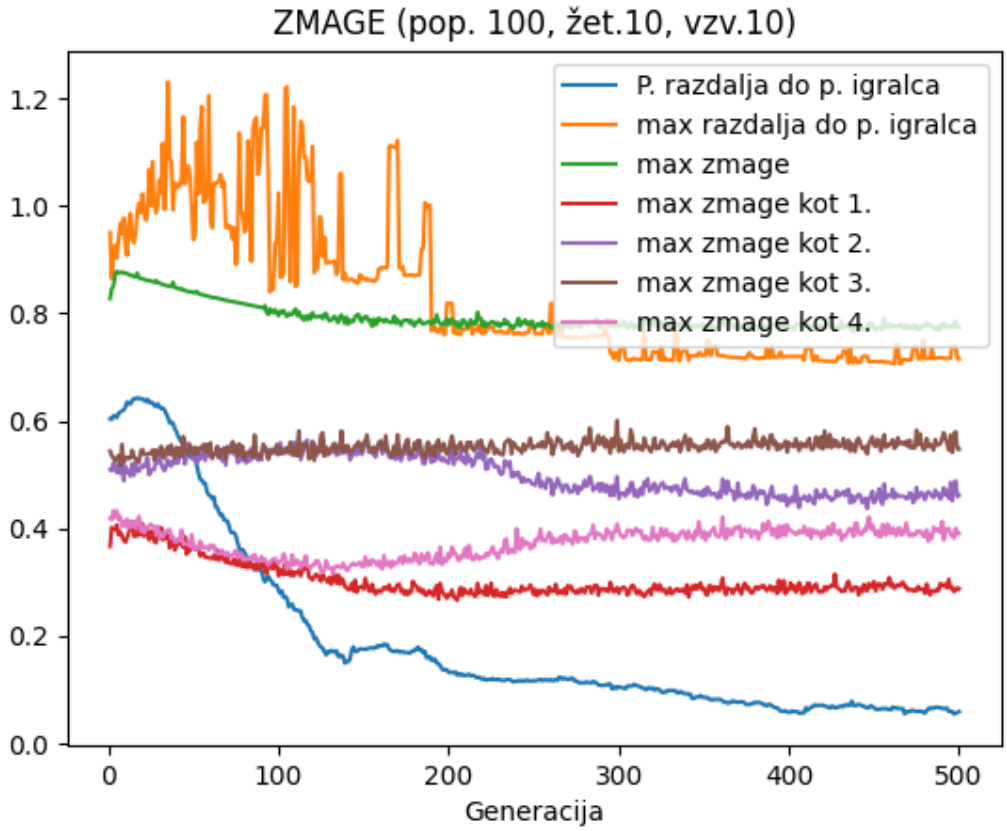

*Slika 11: Prikaz deleža zmag glede na število žetonov (4 Učljivi igralci).*

Ta diagram je enakovreden grafu povprečnega učenja - oba namreč nastaneta v eni *primerjavi uspešnosti* in z njiju lahko razberemo, kako število žetonov vpliva na igro. Ker genetski algoritem izvajamo pri desetih žetonih, nas najbolj zanima zadnji stolpec. Na njem vidimo, da igralci najmanjkrat zmagajo, kadar igro začnejo, nekoliko večkrat, kadar so na vrsti zadnji (četrti), še večkrat, kadar na potezo pridejo drugi, največkrat pa, kadar so na vrsti tretji po vrsti.

Ko se štirje igralci nehajo učiti, program izpiše strategijo tistega igralca, ki je največkrat zmagal. Dobljeno strategijo prikažemo na **tabeli 9**, kjer jo tudi primerjamo s strategijo, ki nastane po genetskem algoritmu.

#### 3.2.3.2 Genetski algoritem za več igralcev

Tudi pri genetskem algoritmu ne moremo računati razdalje do optimalne strategije, ker je le-ta odvisna od soigralcev in zato v splošnem ne obstaja. Zato program nastavimo tako, da pri oblikovanju nove generacije upošteva delež vseh zmag.




*Slika 12: Genetski turnir za 4 igralce.*

Ko izvedemo genetski turnir za 4 igralce, nastane graf, ki ga vidimo na **sliki 12**. Tokrat nimamo krivulje, ki bi prikazovala razdaljo do optimalne strategije, a vseeno lahko z grafa nekaj razberemo.

Igralci tudi tokrat od generacije do generacije počasi postajajo podobni, a ne do te mere, da bi bili vsi enaki in bi se genetski algoritem predčasno končal. Opazimo tudi, da je delež zmag različen glede na to, kdaj igralec prvič pride na vrsto. Najmanjši delež zmag je dosežen takrat, ko je igralec prvi na potezi, malo večji je, ko je igralec četrti na potezi,

drugi največji takrat, ko je igralec drugi na vrsti, največji delež zmag pa takrat, ko je igralec tretji na vrsti. Enako se je zgodilo tudi pri spodbujevanem učenju, torej v obeh primerih igralci naletijo na podobne težave, kar je znak, da program pravilno deluje. Tudi pri genetskem algoritmu na koncu poiščemo najuspešnejšega igralca. Njegovo strategijo na **tabeli 5** primerjamo s strategijo, pridobljeno s spodbujevanim učenjem.

*Tabela 5: Genetski turnir za 4 igralce.*

| Žetoni | Spodbujevano učenje | Genetski algoritem | Primerjava |
|---|---|---|---|
| 1 | [1.0, 0.0] | [1.0, 0.0] | Ko je na polju le še 1 žeton, igralca ne moreta vzeti 2. |
| 2 | [0.55, 0.45] | [0.5546875, 0.4453125] | Oba igralca bosta raje vzela 1 žeton, vendar v obeh primerih verjetnost za to ni presenetljivo velika. Očitno se igralca ne srečata pogosto s to situacijo in za to ni tako pomembna. |
| 3 | [0.0, 1.0] | [0.000244140625, 0.999755859375] | Igralec s strategijo spodbujevanega učenja bo tu zagotovo vzel 2 žetona, tisti s strategijo genetskega algoritma pa najverjetneje tudi. |
| 4 | [0.05, 0.95] | [0.0009765625, 0.9990234375] | Oba igralca bosta skoraj gotovo vzela dva žetona. V tem primeru je za to bolj odločna strategija genetskega algoritma. |
| 5 | [0.994371239941257, 0.005628760058743021] | [0.890625, 0.109375] | Obe porazdelitvi kažeta na to, da bosta igralca vzela 1 žeton. Tokrat je to verjetneje pri spodbujevanem učenju. |
| 6 | [1.0, 0.0] | [0.999969482421875, 3.0517578125e-05] | Strategija spodbujevanega učenja pri 6 žetonih z vso gotovostjo vzame 1 žeton, pri strategiji genetskega algoritma pa je tudi skoraj nemogoče, da bi igralec storil drugače. |
| 7 | [0.9049964309030956, 0.09500356909690444] | [0.125, 0.875] | Tu je razlika med porazdelitvama največja. Strategija spodbujevanega učenja tukaj skoraj gotovo vzame 1 žeton, strategija genetskega algoritma pa obratno. |
| 8 | [0.0, 1.0] | [0.00048828125, 0.99951171875] | Pri 8 žetonih je stvar spet bolj podobna. Obe strategiji želita, da igralec vzame 2 žetona, pri strategiji spodbujevanega učenja z vso verjetnostjo (1). |
| 9 | [0.44, 0.56] | [0.984375, 0.015625] | Porazdelitvi pri 9 žetonih se spet nekoliko bolj razlikujeta. Pri spodbujevanem učenju sta obe verjetnosti dokaj podobni, nekoliko večja je verjetnost, da igralec vzame 2 žetona, pri strategiji genetskega algoritma pa igralec skoraj gotovo vzame 1 žeton. |
| 10 | [0.0, 1.0] | [6.103515625e-05, 0.99993896484375] | Porazdelitvi pri 10 žetonih sta dokaj podobni porazdelitvama pri 8 žetonih. Spodbujevano učenje zagotovo vzame 2 žetona, genetski algoritem pa skoraj gotovo 1 žeton. |

Zdaj je čas, da ugotovimo, kateri igralec je boljši. Strategiji obeh najboljših igralcev shranimo in naredimo štiri Naključne utežene igralce. Dvema dodelimo strategijo najboljšega igralca, ki nastane po spodbujevanem učenju, dvema pa tisto, ki nastane po genetskem algoritmu. Dobljene rezultate prikazuje **tabela 6** (črki **G** in **S** ponazarjata igralce, ki imajo strategijo genetskega algoritma (G) oz. spodbujevanega učenja (S) in to, v kakšnem vrstnem redu igrajo).

*Tabela 6: Rezultati primerjave pri več igralcih.*

| G G S S | | G S G S | | S G S G | | S S G G | |
|---|---|---|---|---|---|---|---|
| **Igralec** | **Zmage** | **Igralec** | **Zmage** | **Igralec** | **Zmage** | **Igralec** | **Zmage** |
| 1 (G) | **390**[1] | 1 (G) | 358 | 1 (S) | 358 | 1 (S) | *373* |
| 2 (G) | 377[2] | 2 (S) | 386 | 2 (G) | **390** | 2 (S) | 381 |
| 3 (S) | *374*[3] | 3 (G) | **391** | 3 (S) | 381 | 3 (G) | **391** |
| 4 (S) | 359[4] | 4 (S) | *365* | 4 (G) | *371* | 4 (G) | 355 |

Opazimo, da se še zmeraj nekoliko pozna prednost tretjega igralca. A kljub temu se igralca, ki uporabljata strategijo, pridobljeno z genetskim algoritmom, v svojih položajih izkažeta bolje, kot igralca, ki uporabljata strategijo, pridobljeno s spodbujevanim učenjem, ko sta v istih situacijah. Domnevamo, da bi bila razlika v uspešnosti, če bi bilo v igri še več igralcev in več žetonov, še večja. Tako ugotovimo, da ima tudi genetski algoritem svoje prednosti, a seveda le, kadar ga uporabljamo za to, za kar je namenjen.

[1] Najboljši rezultati so odebeljeni.
[2] Drugi najboljši rezultati so podčrtani.
[3] Tretji najboljši rezultati so napisani poševno.
[4] Najslabši rezultati so zapisani z navadno pisavo.

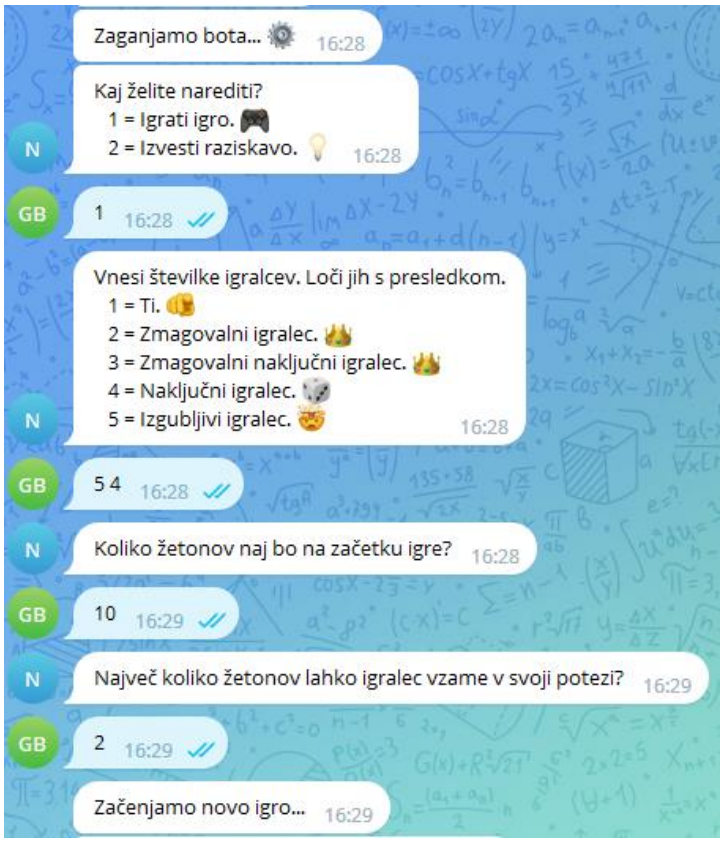
## 3.3 Tehnologija - Telegram bot za komunikacijo

Do zdaj smo opisali nekaj zanimivih rezultatov, ki jih lahko pridobimo z našim programom za simulacijo igre Nim. Da bomo zanimivo vsebino naredili dostopno širokemu krogu bralcev naredimo tudi Telegram bota, preko katerega lahko do našega programa dostopajo tudi drugi in raziskujejo naprej.

Telegram je aplikacija za pošiljanje sporočil. Zelo je podobna aplikaciji Whatsapp, prednost Telegrama pa je v tem, da lahko zanj dokaj enostavno sprogramiramo bota. Telegram bot je program (v našem primeru spet napisan v Pythonu), ki prejme informacije o sporočilu, ki mu je bilo poslano in nato glede na to, kako ga sprogramiramo, odgovori nanje.

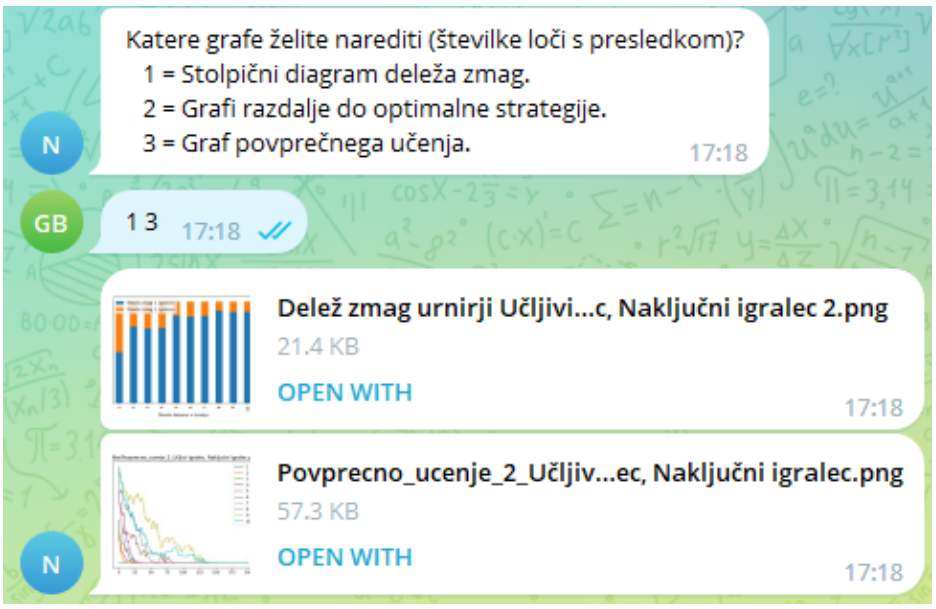
Naš bot je dostopen na povezavi https://t.me/nim_learning_bot. Pred uporabo je potrebno na telefon namestiti aplikacijo Telegram (prenos je brezplačen). Če želimo, lahko potem naložimo tudi Telegram desktop, ki je namenjen za uporabo na računalniku, a pred tem je potrebno imeti Telegram na telefonu, saj je profil povezan s telefonsko številko. Ko je aplikacija naložena, lahko odpremo zgornjo povezavo in začnemo z uporabo bota (krajša navodila so opisana kasneje v razdelku *Navodila za uporabo bota*).

Bistvo programa za Telegram bota je funkcija *echo(update, context)*. Echo po latinsko in po angleško pomeni odmev, kar nam pove, da ta funkcija skrbi za to, da program prejme sporočilo, in nanj odgovori z novim sporočilom. Ta funkcija dobi parameter *update*, v kateri so vsi podatki o sporočilu (ime, priimek in ID pošiljatelja, ID klepeta, vsebina sporočila itd.). To funkcijo nato sprogramiramo tako, kot si želimo. Mi naredimo tako, da lahko preko nje uporabnik dostopa do našega programa. Nekaj primerov uporabe prikazujejo naslednje slike:

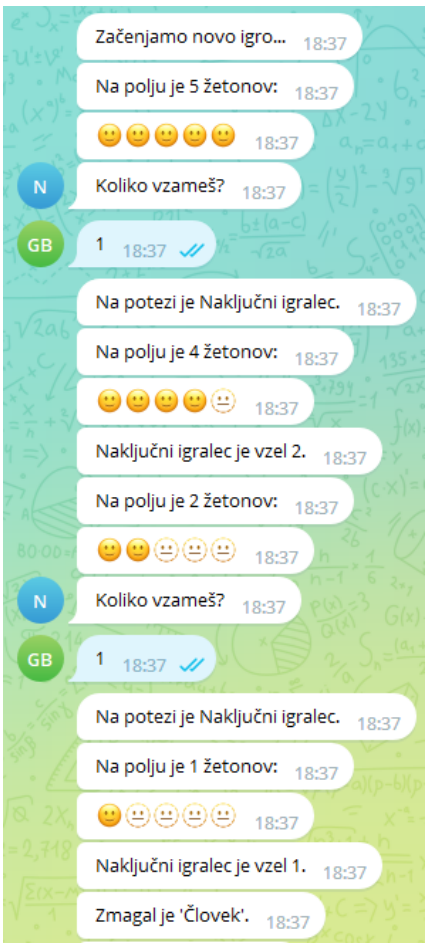



*Slika 13: Nastavitve za igro.*


*Slika 14: Nastavitve za raziskavo.*




*Slika 15: Rezultati raziskave.*




*Slika 16: Igra.*

Naš bot se, tako kot vsi preostali Telegram boti, zažene tako, da mu uporabnik pošlje sporočilo */start*. Nato sledijo nastavitve, preko katerih uporabnik določi, kaj želi, da program stori. Nastavitve prikazuje spodnji diagram:


*Slika 17: Prikaz nastavitev za Telegram bota.*

Ta navodila za uporabo so nekoliko krajša, je pa v pripravi podrobnejši opis za učence, ki jih zanima. Ko se uporabnik prebije čez nastavitve, se izvede bistvo programa. Če želi uporabnik igrati igro, bo bot s svojimi sporočili obveščal uporabnika o dogajanju na igralnem polju in ga, ko bo na potezi, vprašal po številu vzetih žetonov. Če pa uporabnik želi z botom izvesti raziskavo, se bo izvedel tisti del našega programa, ki ga nekoliko že poznamo, saj smo z njim pridobili rezultate za raziskovalno nalogo. Torej se pod pogoji, ki jih uporabnik določi v nastavitvah, izvede raziskava, na koncu pa bot pošlje grafe, ki prikazujejo rezultate.

# 4 Zaključki

V raziskovalni nalogi *Primerjava genetskega algoritma in spodbujevanega učenja pri iskanju strategije igre Nim* raziskujemo, kako računalnik s spodbujevanim učenjem in genetskim algoritmom išče čim boljšo strategijo za igro Nim.

Za simulacijo igranja igre in izvajanja poskusov naredimo Python program, ki ga skupaj z osnovami umetne inteligence in evolucijskih algoritmov opišemo v teoretičnem delu. S tem programom računalnik simulira igranje iger in iskanje strategije z obema obravnavanima načinoma.

V eksperimentalnem delu si ogledamo in obrazložimo grafe, ki jih program naredi za prikaz dogajanja. Ugotovimo, da je spodbujevano učenje za iskanje optimalne strategije pri igri z dvema igralcema veliko boljše od genetskega algoritma, saj Učljivi igralec veliko hitreje najde veliko boljšo strategijo.

Na to, kako hitro bo Učljivi igralec našel optimalno strategijo, vpliva strategija, s katero igra njegov soigralec. Kadar igra z Naključnim igralcem, do strategije vedno pride, a bolj počasi. Ko pa igra z Zmagovalnim igralcem, do optimalne strategije ne pride praktično nikoli, ker se Učljivi igralec z določenimi situacijami nikoli ne sreča. Pri igri z Zmagovalnim naključnim igralcem pa do optimalne strategije pride dokaj hitro, a le, kadar se igra začne s številom žetonov, ki ga ne moremo opisati s formulo $3 \times n + 1$ za neko naravno število $n$.

Kljub temu, da se genetski algoritem pri igri z dvema igralcema ni izkazal za najboljšega, se naučimo, da na rezultat vpliva kriterij, s katerim program odloči, katere igralce bo predal v naslednjo generacijo.

Povsem drugače pa je, kadar je v igro vpletenih več igralcev. V tem primeru se spodbujevano učenje ne obnese najbolje, saj se se Učljivi igralci učijo eden od drugega in se tudi prilagodijo eden na drugega. Ko igralec, ki je bil od teh najboljši, tekmuje z najboljšim genetskim igralcem, se genetski igralec izkaže za uspešnejšega. Pri njem namreč strategija ni izpopolnjena le za igro proti trem, vedno istim soigralcem, pač pa se lahko kosa s celotno populacijo, ki se od generacije do generacije spreminja in skozi ta razvoj vključuje vedno več izkušenj.

Genetski algoritem torej pri igri z več igralci najde boljšo strategijo. To je povsem razumljivo, saj je cilj genetskega algoritma najti čim boljšo rešitev v tistih primerih, kjer z matematiko ne moremo najti optimalnih rešitev. S tem se naučimo tudi, da ima vsaka stvar svoj namen in da bomo dober rezultat dobili le, če jo bomo prav uporabili.

Tako v raziskovalni nalogi odkrijemo nekaj zanimivih stvari pri igri Nim. Toda to še zdaleč ni vse! Z našim Telegram botom, dostopnim na https://t.me/nim_learning_bot, predajamo naš program v javnost, tako da ima vsak možnost, da odkrije kaj novega.

# Literatura


Artificial intelligence (AI)  (2023-2-16), Evropska komisija. URL: https://research-and-innovation.ec.europa.eu/research-area/industrial-research-and-innovation/key-enabling-technologies/artificial-intelligence-ai_sl

BOKAL, Gregor (2023-2-1): Nim_learning_bot URL: https://t.me/nim_learning_bot

BOKAL, Gregor. 2022. Igra Nim skozi matematiko in spodbujevano učenje. Ljubljana.

Class diagrams (2023-1-29), Wikipedia. URL: https://en.wikipedia.org/wiki/Class_diagram

GUID, Nikola, in STRNAD, Damjan. 2007. Umetna inteligenca. Maribor: Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko.

HarshaVardhanBabu (2023-1-29): Instructions_UML_Python. URL: https://gist.github.com/HarshaVardhanBabu/9a47db9e33cf06e9e1e917520bb54056

JAMNIK, Rajko. 1973. Teorija iger. Ljubljana: Državna založba Slovenije.

KAVRAN, Narsej. 2008. Genetski algoritem za razvoj borznih trgovalnih strategij. Maribor: Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko.

KREBELJ, Peter. 2016. Spoznavamo programski jezik python. Ljubljana: Doria.

SLOČAJ, Danijel (2023-2-16): Porazdeljene inteligentne programske tehnologije, Inteligentni agenti URL: https://studentski.net/gradivo/ulj_fri_ri1_pos_sno_inteligentni_agenti_01

Spodbujevano učenje (2023-1-31), Wikipedia. URL: https://en.wikipedia.org/wiki/Class_diagram

What Is Reinforcement Learning (2023-1-31), Mathworks. URL: https://www.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html