SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Ivan Simić

KORIŠTENJE ALGORITMA MINIMAX ZA STVARANJE AI PROTIVNIKA U IGRI ČETIRI U NIZU

PROJEKT

UVOD U UMJETNU INTELIGENCIJU

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Simić

Studij: Informacijski i poslovni sustavi

KORIŠTENJE ALGORITMA MINIMAX ZA STVARANJE AI PROTIVNIKA U IGRI ČETIRI U NIZU

PROJEKT

Mentor:

dr. sc. Bogdan Okreša Đurić

Varaždin, siječanj 2024.

Ivan Simić

Izjava o izvornosti

Izjavljujem da je ovaj projekt izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI Radovi

Sadržaj

1. Uvod

Umjetna inteligencija jedno je od polja znanosti koje je trenutno u stanju brzog razvijanja. Ne radi se o novom obliku znanosti, ali ona nije usavršena. Stalno se otkrivaju novi i poboljšavaju prošli načini učenja umjetne inteligencije. Nekim ljudima je još uvijek teško zamisliti kako točno računalo može koristiti podatke za učenje. Kako program može usavršiti svoje znanje o nekoj domeni?

Ovaj projekt je kreiran kako bi se na pregledan način mogao prikazati proces učenja umjetne inteligencije na primjeru s kojim se većina ljudi može povezati, a to su stolne igre. Specifično na igri Četiri u nizu. Na ovaj način bi čak i ljudi sa malo znanja o samoj umjetnoj inteligenciji mogli lakše doći do zaključka o tome kako ona operira. Četiri u nizu je relativno jednostavna igra što se tiče pravila, ali zahtjeva duboku razinu razmišljanja i gledanja unaprijed kako bi se moglo doći do pobjede.

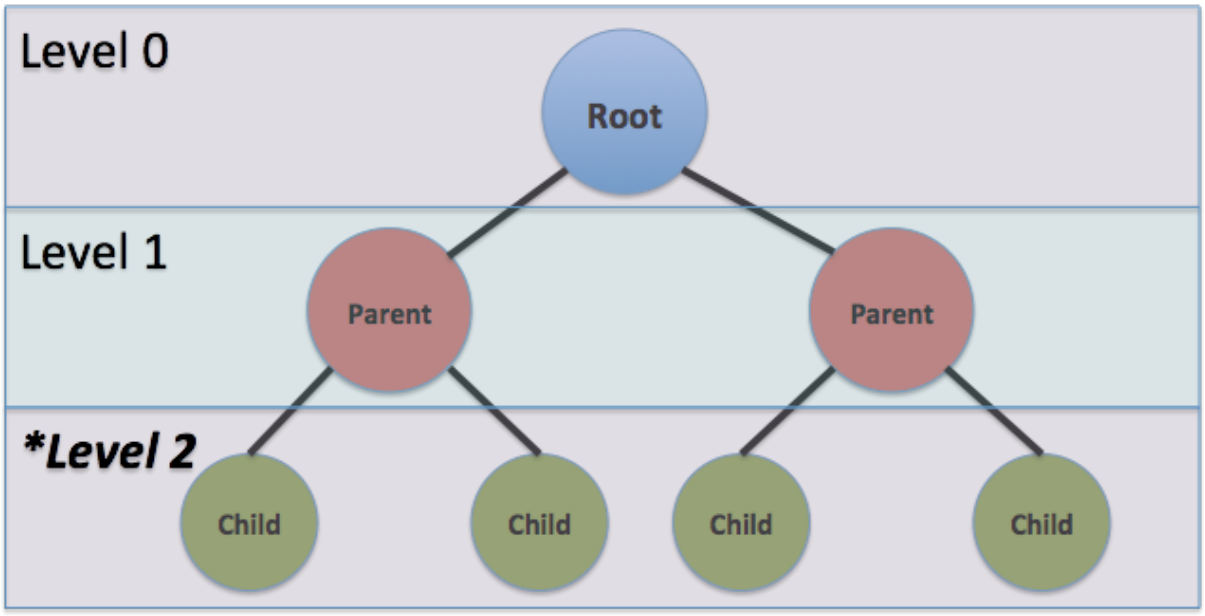
2. Minimax algoritam

Minimax je algoritam koji se koristi u donošenju odluka i u teoriji igara. Glavni princip prema kojem algoritam funkcionira je vraćanje unatrag (engl. *backtracking*) kako bi mogao analizirati stanje igre. Algoritam pronalazi optimalan potez za igrača, ali uz pretpostavku da njegov protivnik također igra optimalno. Koristi se u 1 na 1 igrama, kao što su šah, dama, ludo, othello i sl. Radi se o igrama nulte-sume (engl. *zero-sum games*) u kojima prednost jednog igrača označava gubitak prednosti za drugog igrača [1].

U algoritmu su dva igrača, onaj koji maksimizira i onaj koji minimizira. Jedan pokušava povećati svoj rezultat, dok ga drugi pokušava smanjiti. Svako stanje igre dobiva određen broj bodova ovisno o tome koji igrač pobjeđuje. Igrač koji maksimizira traži stanje u kojem je vrijednost što veća, a igrač koji minimizira traži što manju vrijednost. Algoritam je uvijek u stanju pronaći potreban rezultat, no sama brzina algoritma jako ovisi o tome koliko je mogućih poteza u trenutnom stanju igre, pošto algoritam mora proći kroz svaku mogućnost kako bi pronašao najoptimalniji potez [2].

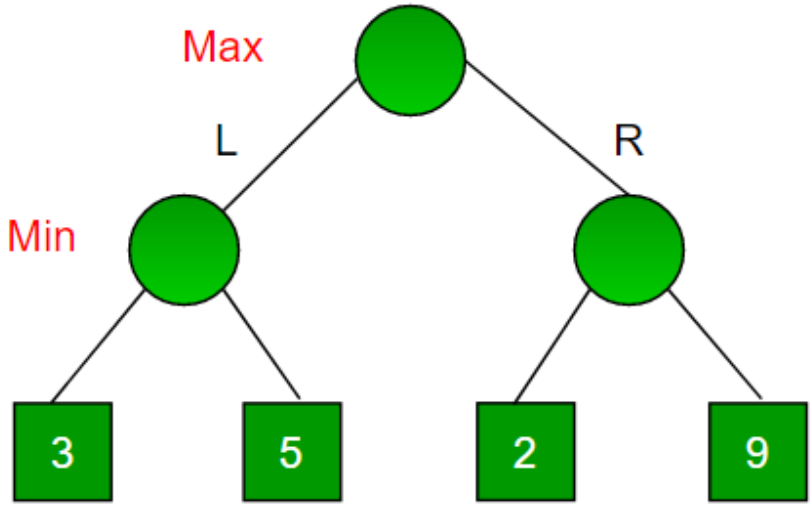
2.1. Dubinsko pretraživanje

Samo pretraživanje rezultata u algoritmu može se prikazati preko stabla. Algoritam prolazi kroz svaki čvor i bira onaj sa najboljim rezultatom. Korijen stabla nalazi se na razini nula i on prikazuje trenutno stanje ploče. Prva razina sadrži sve moguće poteze za sljedećeg igrača. Dakle, ako je prvi igrač napravio potez, korijen prikazuje stanje ploče nakon poteza prvog igrača, a čvorovi u prvoj razini prikazuju sve moguće poteze drugog igrača. Drugi čvor onda prikazuje sve poteze koje prvi igrač može izvesti nakon poteza drugog igrača, te se tako razine konstanto izmjenjuju.



Slika 1: Stablo pretraživanja; preuzeto iz [2]

Kada je stablo kreirano rade se pretpostavke o potezu igrača. Na primjer, potez igrača koji minimizira ovisi o tome na kojoj strani stabla se nalazi potez igrača koji maksimizira za kojeg se pretpostavlja da će izvesti. Prema tom potezu igrač koji minimizira uzima onaj potez na istom djelu stabla kao i potez igrača koji maksimizira i to potez, tj. čvor koji ima u sebi najmanju vrijednost. Algoritam može kreirati stablo proizvoljne dubine, ali vrijeme potrebno da se algoritam izvede se povećava sa dubinom.

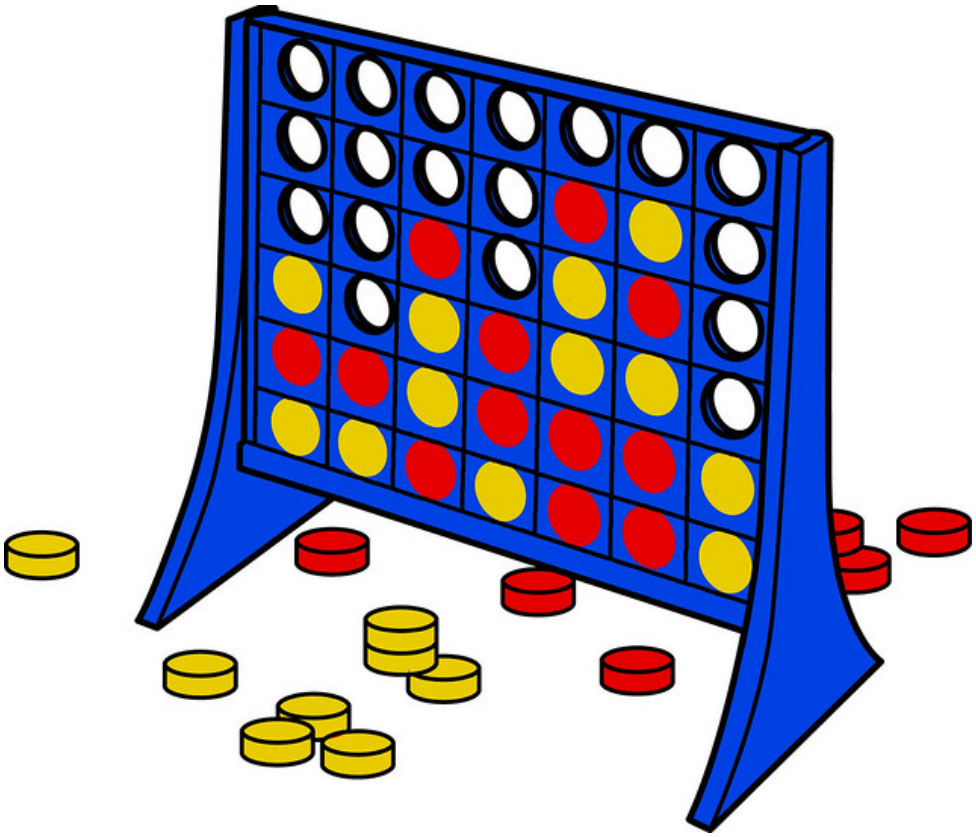


Slika 2: Stablo za primjer; preuzeto iz [3]

Uzmimo za primjer ovo malo stablo. Na početku, igrač koji maksimizira može ići desno ili lijevo na stablu. Ako se odluči za lijevo, slijedi igrač koji minimizira, koji bira manju vrijednost između tri i pet. Igrač će naravno uzeti manju vrijednost, tri. Ako se igrač koji maksimizira odluči za desno, igrač koji minimizira odabrati će vrijednost dva jer je definitivno manja od devet. Dakle, na lijevoj strani igrač koji minimizira dobiva vrijednost tri, a na desnoj dva. Za njega je očito bolja desna strana, jer tamo dobiva manju vrijednost. Zbog toga, igrač koji maksimizira će se odlučiti za lijevu stranu ne samo zato što je ta vrijednost bolja za njega, nego zato što je ta vrijednost također gora od one koju je njegov protivnik mogao postići na drugoj strani stabla.

3. Kritički osvrt

Većina stolnih igara zahtjeva od igrača da gledaju unaprijed kako bi mogli predvidjeti pokrete svojeg protivnika. Čovjeku je teško u malom vremenu vizualizirati sve poteze svojeg protivnika u igrama koje nisu nužno kompleksne, ali imaju veliki broj mogućih poteza. Umjetna inteligencija može korištenjem algoritama puno bolje upravljati svim situacijama koje nastaju tijekom igre.



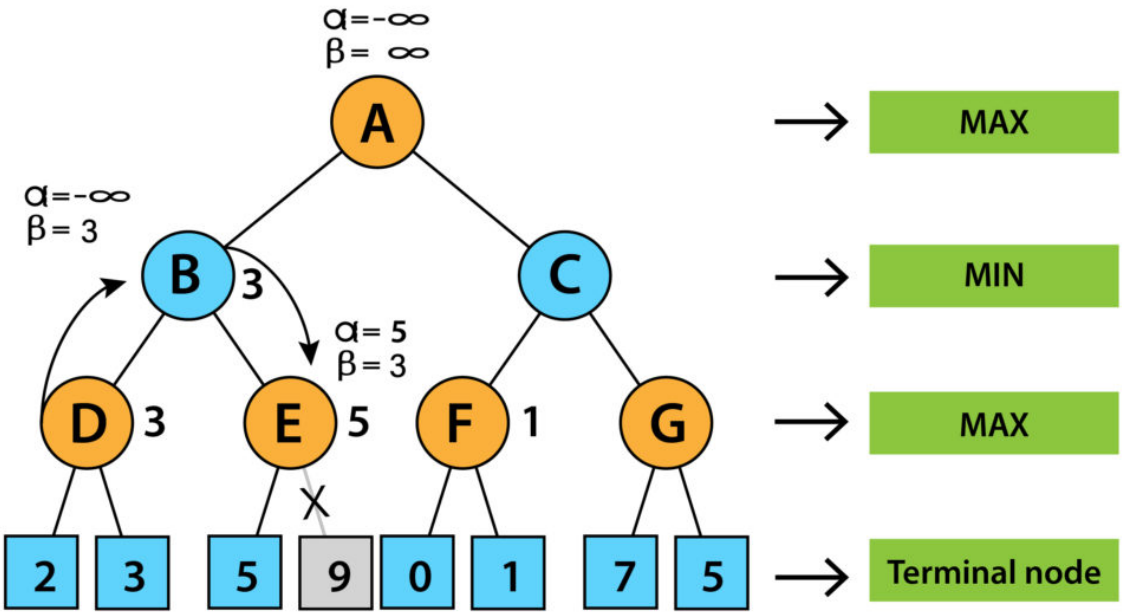
Slika 3: Igra četiri u nizu; preuzeto iz [4]

Četiri u nizu je stolna igra čija se ploča sastoji od matrice sa šest redova i sedam stupaca. Dva igrača ubacuju svoje žetone u ploču sve dok jedan od igrača ne stvori red od četiri ista žetona horizontalno, vertikalno ili dijagonalno. Postoji otprilike 4,5 trilijuna mogućih stanja ploče [5]. Pošto se u ovoj igri nalazi veliki broj mogućih poteza, potrebno je skratiti količinu čvorova koji se provjeravaju. Ovo se u minimax algoritmu može izvesti uz alfa-beta orezivanje.

3.1. Alfa-beta orezivanje

Orezivanje nije poseban algoritam, već nadogradnja za postojeći minimax algoritam. Pomoću njega može se smanjiti vrijeme potrebno da algoritam izvrši odabir time što se smanji broj čvorova kroz koji algoritam prolazi. Dodaju se dva nova parametra, alfa i beta. Oba parametra su postavljena na najmanju moguću vrijednost za oba igrača, alfa na minus beskonačno i beta na beskonačno. Dakle, alfa je za igrača koji maksimizira, a beta za igrača koji minimizira.

Njihova vrijednost uvijek će biti trenutni najbolji mogući rezultat za njihovog igrača. Na svakom čvoru u stablu igrač je svijestan koja je vrijednost najveća (alfa) ili najmanja (beta). Ako se na grani odabere čvor zbog kojeg najmanja moguća vrijednost za igrača koji minimizira (beta) postane manja ili jednaka najvećoj mogućoj vrijednosti za igrača koji maksimizira (alfa), tada algoritam zna da će, ako odabere taj čvor, vrijednost postati gora. U ovom slučaju algoritam će u potpunosti ignorirati ostale grane tog čvora [3].



Slika 4: Primjer orezivanja; preuzeto iz [6]

Uzmimo ovaj primjer. U korijenu A alfa i beta postavljaju se na svoju najgoru vrijednost (minus beskonačno i beskonačno). Tu je beta veća od alfe pa se orezivanje ne izvodi. Čvor B još nema zapisane vrijednosti, pa stanje ostaje isto. U čvoru D, koji je čvor za igrača koji maksimizira, će se mijenjati vrijednost alfe na najveću moguću, u ovom slučaju to je tri. Kada se vratimo u čvor B, mijenjat će se samo vrijednost bete, pošto je to čvor za igrača koji minimizira. Tu alfa ostaje minus beskonačno, a beta postaje tri. Sada će se vrijednost bete prebaciti u čvor E jer je taj čvor dijete čvora B. Čvor E će isto imati alfu na minus beskonačno, a betu na tri. Prva moguća vrijednost koju E može poprimiti je pet i ona se zapisuje u alfu jer je ta vrijednost bolja od prethodne za igrača koji maksimizira. U ovom slučaju vrijednost alfe je postala veća od vrijednosti bete, dakle zahtjev za izvođenje orezivanja je ispunjen. Preostale vrijednosti čvora se ignoriraju. Ovako se ubrzava proces traženja najbolje vrijednosti za igrača.

4. Opis implementacije

Algoritam je implementiran u pythonu uz pomoć biblioteke pygame kako bi se tijek igre mogao bolje prikazati. Za početak je potrebno preuzeti pygame biblioteku kako bi se mogle koristiti njene funkcije.

```
pip install pygame
```

Isječak koda 1: Preuzimanje pygame biblioteke

Sada možemo uvesti sve potrebne funkcije u program.

```
import pygame
import random
import numpy as np
import math

#Konstante
REDOVI = 6
STUPCI = 7
VISINA = 600
SIRINA = 700
CHELIJA = 100
HI = 1
AI = 2
DUBINA = 4

#Boje
ZUTA = (230, 191, 0)
CRVENA = (230, 0, 0)
PLAVA = (0, 0, 179)
CRNA = (0, 0, 0)
```

Isječak koda 2: Početak programa

Uvedene su konstante kako bi kod bio pregledniji [7]. Redovi i stupci grade ploču za igranje. Visina i širina određuju veličinu prozora u kojem se igra izvršava. "CHELLIJA" označava veličinu jedne čelije u prozoru. "HI" ("Human inteligence") služi za prikaz poteza ili žetona prvog igrača, a "AI" ("Artificial inteligence") za prikaz drugog igrača. Od sada će se u projektu prvi igrač nazivati "čovjek", a drugi "robot". Dubina označava dubinu pretraživanja za algoritam. Varijabla je proizvoljna, ali sa trenutnom varijablom je uspostavljena ravnoteža između uspješnosti algoritma i brzine potrebne da se pronađe najbolji potez. Boje služe samo za bolji prikaz ploče u prozoru. Korištenje ovih boja biti će vidljivo u petom poglavlju.

4.1. Funkcionalnost algoritma

Algoritam je podjeljen na dvije funkcije u programu, jedna za izvođenje poteza i jedna za traženje poteza.

Funkcija "ai potez" prvo traži stupce koji nisu popunjeni do vrha uz pomoć "if plocha[0][stup] == 0", dakle traže se stupci kojima je gornji red prazan (ako je red = 0 radi se o redu na vrhu ploče, a ako je stupac = 0 radi se o stupcu na lijevoj strani ploče). Ako takav stupac postoji algoritam može započeti. Najbolji stupac i najbolja vrijednost stavljaju se na najmanji mogući iznos tako da svaki pronađeni iznos bude bolji od zadanog. Algoritam prolazi kroz svaki pronađeni stupac. Za dobiveni stupac traži se prvi red na dnu ploče koji u tom stupcu nema već postavljen žeton. Tu se koristi funkcija prikazana u isječku koda 3 koja prolazi kroz sve redove odozdola prema gore u određenom stupcu i vraća prvi prazni red. U odabranu ćeliju postavlja se žeton robota. Sada se pokreće simulacija igre preko funkcije za izvođenje minimax algoritma. Funkcija prima trenutno stanje ploče nakon postavljenog žetona, dubinu pretraživanja, vrijednosti za alfu i betu, te je "maksimiziranje" postavljeno na "False", pošto se nakon poteza robota gleda potez čovjeka, a čovjek je u ovoj implementaciji igrač koji minimizira. Nakon izvršenog pretraživanja postavljeni žeton se briše. Dobiveni bodovi se uspoređuju sa trenutnom vrijednošću i vrijednost se mijenja samo ako je pronađena vrijednost sa više bodova. Stupac se također mijenja. Kada algoritam prođe kroz sve moguće stupce funkcija će vratiti onaj stupac za koji je algoritam pronašao vrijednost sa najviše bodova.

```
1 #Funkcija koja traži prvi prazan red
2 def prazan_red(stup):
3     for red in range(REDOVI - 1, -1, -1):
4         if plocha[red, stup] == 0:
5             return red
6     return None
```

Isječak koda 3: Traženje praznih redova za određeni stupac

Funkcija "minimax" izvršava se ovisno o igraču i stanju ploče. Ako je na redu robot, vrijednost se maksimizira. Prolazi se kroz sve moguće ćelije i postavlja se žeton robota. Stanje ploče nakon postavljanja, kao i trenutna vrijednost parametara alfa i beta šalju se opet u funkciju, ali ovaj puta za igrača koji minimizira, dakle za čovjeka. Rekurzije se odvijaju sve dok se ne dođe do kraja stabla (za ovo služi parametar "dubina" koji se smanjuje u svakoj rekurziji) ili dok u jednom pozivu ne dođe do pobjede. Tada se poziva funkcija koja može dodjeliti bodove za stanje na ploči. Nakon toga brišu se izvedeni potezi i traže se najbolje vrijednosti prema dobivenim bodovima. Određuju se alfa i beta, te se prekida izvođenje ako dođe do orezivanja.

```python
#Funkcija za AI potez i pozivanje algoritma
def ai_potez():
    dostupno = [stup for stup in range(STUPCI) if plocha[0, stup] == 0]
    if dostupno:
        m_stup = -1
        m_bodovi = -math.inf
        for stup in dostupno:
            red = prazan_red(stup)
            plocha[red, stup] = AI
            bodovi = minimax(plocha, DUBINA, -math.inf, math.inf, False)
            plocha[red, stup] = 0
            if bodovi > m_bodovi:
                m_bodovi = bodovi
                m_stup = stup
        return m_stup
    return None

#Funkcija minimax algoritma sa alfa-beta orezivanjem
def minimax(plocha, dubina, alfa, beta, maksimiziranje):
    if dubina == 0 or pobjeda(HI) or pobjeda(AI):
        return pozicija_bodovanje(plocha, AI)
    if maksimiziranje:
        maksimum = -math.inf
        for stup in range(STUPCI):
            if plocha[0, stup] == 0:
                red = prazan_red(stup)
                plocha[red, stup] = AI
                eval = minimax(plocha, dubina - 1, alfa, beta, False)
                plocha[red, stup] = 0
                maksimum = max(maksimum, eval)
                alfa = max(alfa, eval)
                if beta <= alfa:
                    break
        return maksimum
    else:
        minimum = math.inf
        for stup in range(STUPCI):
            if plocha[0, stup] == 0:
                red = prazan_red(stup)
                plocha[red, stup] = HI
                eval = minimax(plocha, dubina - 1, alfa, beta, True)
                plocha[red, stup] = 0
                minimum = min(minimum, eval)
                beta = min(beta, eval)
                if beta <= alfa:
                    break
        return minimum
```

Isječak koda 4: Izvođenje minimax algoritma

4.2. Bodovanje

Slijede funkcije pomoću kojih minimax algoritam može dodjeliti bodove za odrađene poteze.

Funkcija "pozicija bodovanje" služi kako bi se bodovi mogli dodjeliti za svaki način postavljanja žetona. Prvo funkcija dodjeljuje dodatne bodove ako su žetoni postavljeni u centar ploče, pošto u igri Četiri u nizu centar omogućava najbolje prilike za pobjedu [7]. Za svaki žeton postavljen u centru bodovi će se uvećati pet puta. Za svaki način postavljanja žetona (horizontalno, vertikalno, dijagonalno) poziva se posebna funkcija koja dodaje bodove ovisno o broju žetona postavljenih u redoslijedu. Dobiveni bodovi se na kraju vraćaju u minimax funkciju.

```
#Funkcije za dodavanje bodova
def pozicija_bodovanje(plocha, zeton):
    bodovi = 0
    #Bodovanje centra
    centar = [int(i) for i in list(plocha[:, STUPCI // 2])]
    bodovi += centar.count(zeton) * 5
    #Horizontalno bodovanje
    for red in range(REDOVI):
        for stup in range(STUPCI - 3):
            bodovi += plocha_bodovanje([plocha[red, stup + i] for i in range(4)],
            ↪  zeton)
    #Vertikalno bodovanje
    for red in range(REDOVI - 3):
        for stup in range(STUPCI):
            bodovi += plocha_bodovanje([plocha[red + i, stup] for i in range(4)],
            ↪  zeton)
    #Dijagonalno bodovanje (gore-lijevo -> dolje-desno)
    for red in range(REDOVI - 3):
        for stup in range(STUPCI - 3):
            bodovi += plocha_bodovanje([plocha[red + i, stup + i] for i in
            ↪  range(4)], zeton)
    #Dijagonalno bodovanje (dolje-lijevo -> gore-desno)
    for red in range(3, REDOVI):
        for stup in range(STUPCI - 3):
            bodovi += plocha_bodovanje([plocha[red - i, stup + i] for i in
            ↪  range(4)], zeton)
    return bodovi
```

Isječak koda 5: Dobivanje bodova za svaki mogući smjer

Funkcija "plocha bodovanje" je pravi izvor bodova u programu [8]. Robot dobiva bodove za broj žetona koje ima u redoslijedu. Ako uspije postaviti četiri žetona za redom dobiva najveći mogući broj bodova. Ako postavi tri žetona za redom sa dostupnim praznim poljem dobiva četiri boda. Ako postavi dva žetona sa dva dostupna polja dobiva dva boda. Ako čovjek uspije postaviti tri žetona za redom sa dostupnim praznim poljem robot gubi šest bodova. Dakle, što više žetona za redom robot uspije postaviti, to je veća vrijednost. Za četiri žetona robot će dobiti najveću vrijednost da se naglasi važnost pobjede. Robot će blokirati čovjekove poteze ako pronađe situaciju gdje je čovjeku potreban još jedan žeton za pobjedu. Vrijednosti bodova su određene preko testiranja programa. Naravno, vrijednosti su proizvoljne, ali promjene imaju veliki utjecaj na uspješnost algoritma.

```
1  def plocha_bodovanje(plocha, zeton):
2      bodovi = 0
3      protivnik = HI if zeton == AI else AI
4      if plocha.count(zeton) == 4:
5          bodovi += 100
6      elif plocha.count(zeton) == 3 and plocha.count(0) == 1:
7          bodovi += 4
8      elif plocha.count(zeton) == 2 and plocha.count(0) == 2:
9          bodovi += 2
10     if plocha.count(protivnik) == 3 and plocha.count(0) == 1:
11         bodovi -= 6
12     return bodovi
```

Isječak koda 6: Dobivanje bodova za postavljanje žetona u redoslijedu

4.3. Izvođenje igre

Funkcija "plocha nacrtaj" služi za crtanje ploče za igranje. Prvo se crta plava pozadina koja pokriva cijeli prozor. Zatim se crtaju horizontalne i vertikalne linije preko pozadine koje čine rešetku. Rešetka prikazuje polje za svaku ćeliju u koju je moguće postaviti žeton. Čovjekovi žetoni su prikazani žutom bojom, a žetoni od robota crvenom [9].

```
#Funkcija crtanja ploče i žetona
def plocha_nacrtaj():
    for red in range(REDOVI):
        for stup in range(STUPCI):
            pygame.draw.rect(prozor, PLAVA, (stup * CHELIJA, red * CHELIJA, CHELIJA,
            ↪  CHELIJA))
            pygame.draw.line(prozor, CRNA, (stup * CHELIJA, 0), (stup * CHELIJA,
            ↪  VISINA), 2)
            pygame.draw.line(prozor, CRNA, (0, red * CHELIJA), (SIRINA, red *
            ↪  CHELIJA), 2)
            if plocha[red, stup] == HI:
                pygame.draw.circle(prozor, ZUTA, (stup * CHELIJA + 50, red * CHELIJA
                ↪  + 50), 40)
            elif plocha[red, stup] == AI:
                pygame.draw.circle(prozor, CRVENA, (stup * CHELIJA + 50, red *
                ↪  CHELIJA + 50), 40)
```

Isječak koda 7: Crtanje ploče u pygame-u

Funkcije "pobjeda poruka" i "izjednacenje poruka" ispisuju poruku pri vrhu prozora kada dođe do kraja igre. Ako dođe do pobjede, u poruci se vidi pobjednik, te je tekst obojan u boji žetona od igrača koji je pobjedio. Iza teksta postavljena je crna pozadina za bolju vidljivost teksta [9]. Nakon ispisa ćeka se dvije sekunde prije nego što se nastavi sa izvođenjem programa.

```
#Funkcija za ispis poruke pobjede
def pobjeda_poruka(zeton):
    font = pygame.font.SysFont("Arialblack", 30)
    tekst = font.render(f"Igrač {zeton} je pobjedio!", True, ZUTA if zeton == HI
    ↪  else CRVENA)
    tekst_oblik = tekst.get_rect(center=(width // 2, 20))
    pygame.draw.rect(prozor, CRNA, (tekst_oblik.x - 5, tekst_oblik.y - 5,
    ↪  tekst_oblik.width + 10, tekst_oblik.height + 10))
    prozor.blit(tekst, tekst_oblik)
    pygame.display.update()
    pygame.time.delay(2000)

#Funkcija za ispis poruke izjednacenja
def izjednacenje_poruka():
    font = pygame.font.SysFont("Arialblack", 30)
    tekst = font.render(f"Izjednačenje!", True, PLAVA)
    tekst_oblik = tekst.get_rect(center=(width // 2, 20))
    pygame.draw.rect(prozor, CRNA, (tekst_oblik.x - 5, tekst_oblik.y - 5,
    ↪  tekst_oblik.width + 10, tekst_oblik.height + 10))
    prozor.blit(tekst, tekst_oblik)
    pygame.display.update()
    pygame.time.delay(2000)
```

Isječak koda 8: Ispis poruke nakon završetka igre

Funkcija "pobjeda" provjera je li došlo do pobjede na trenutnoj ploči. Pobjeda se provjerava u svim mogućim smjerovima: horizontalno, vertikalno i dijagonalno. Ako nema pobjede funkcija vraća False. Pozivi (STUPCI - 3), (REDOVI - 3) i (3, REDOVI) osiguravaju da se provjere neće odvijati na onim djelovima ploče gdje je nemoguće dobiti pobjedu u smjeru postavljanja žetona koji se provjerava [7].

```
#Funkcija koja provjerava pobjedu
def pobjeda(zeton):
    #Horizontalna provjera
    for red in range(REDOVI):
        for stup in range(STUPCI - 3):
            if plocha[red, stup] == zeton and plocha[red, stup + 1] == zeton and \
               plocha[red, stup + 2] == zeton and plocha[red, stup + 3] == zeton:
                return True
    #Vertikalna provjera
    for red in range(REDOVI - 3):
        for stup in range(STUPCI):
            if plocha[red, stup] == zeton and plocha[red + 1, stup] == zeton and \
               plocha[red + 2, stup] == zeton and plocha[red + 3, stup] == zeton:
                return True
    #Dijagonalna provjera (gore-lijevo -> dolje-desno)
    for red in range(REDOVI - 3):
        for stup in range(STUPCI - 3):
            if plocha[red, stup] == zeton and plocha[red + 1, stup + 1] == zeton and
            ↪  \
               plocha[red + 2, stup + 2] == zeton and plocha[red + 3, stup + 3] ==
               ↪  zeton:
                return True
    #Dijagonalna provjera (dolje-lijevo -> gore-desno)
    for red in range(3, REDOVI):
        for stup in range(STUPCI - 3):
            if plocha[red, stup] == zeton and plocha[red - 1, stup + 1] == zeton and
            ↪  \
               plocha[red - 2, stup + 2] == zeton and plocha[red - 3, stup + 3] ==
               ↪  zeton:
                return True
    return False
```

Isječak koda 9: Provjera pobjede

Prije samog izvođenja igre potrebno je inicijalizirati ploču i prozor. Ime prozora postavlja se na "Connect Four". Varijable "pokrenuto" i "na redu" služe sa upravljanje stanja igre. Zakomentirane linije služe za biranje igrača koji ide prvi. Trenutno čovjek igra prvi, ali moguće je promjeniti da ili robot igra prvi ili da se na početku nasumično odabere koji će igrač biti prvi.

```
#Inicijaliziranje igre
plocha = np.zeros((REDOVI, STUPCI))
pygame.init()
width, height = SIRINA, VISINA
prozor = pygame.display.set_mode((width, height))
pygame.display.set_caption("Connect Four")

#Početak igre
pokrenuto = True
na_redu = HI
#na_redu = AI
#na_redu = random.choice([HI, AI])
```

Isječak koda 10: Početak igre

Sada slijedi prikaz tijeka igre. Crta se ploča za igranje. Ako je na redu čovjek, program provjerava dva moguća događaja (engl. *event*) koje čovjek može unjeti dok igra traje. Čovjek može pritisnuti X na prozoru kako bi se igra prekinula. Ako lijevim klikom miša pritisne negdje na ploču program će pokušati pronaći stupac koji je pritisnut. Ako pritisnuti stupac nije popunjen do vrha traži se prvi slobodni red od dna tog stupca. U pronađeni redak postavlja se čovjekov žeton. Ukoliko čovjek sa postavljenim žetonom postiže pobjedu program će prikazati poruku pobjede i igra se gasi. Ako nije došlo ni do pobjede ni do izjednačenja, red je na drugog igrača. Robotov potez odvija se na isti način, samo što se stupac ne određuje prema pritisnutom stupcu, nego prema stupcu koji minimax algoritam pronađe kao optimalan stupac. Ako robot igra prvi, njegov prvi žeton će uvijek biti postavljen u sredini jer je u funkciji "pozicija bodovanje" u isječku koda 5 postavljeno da se za postavljanje žetona u sredinu dobiva više bodova.

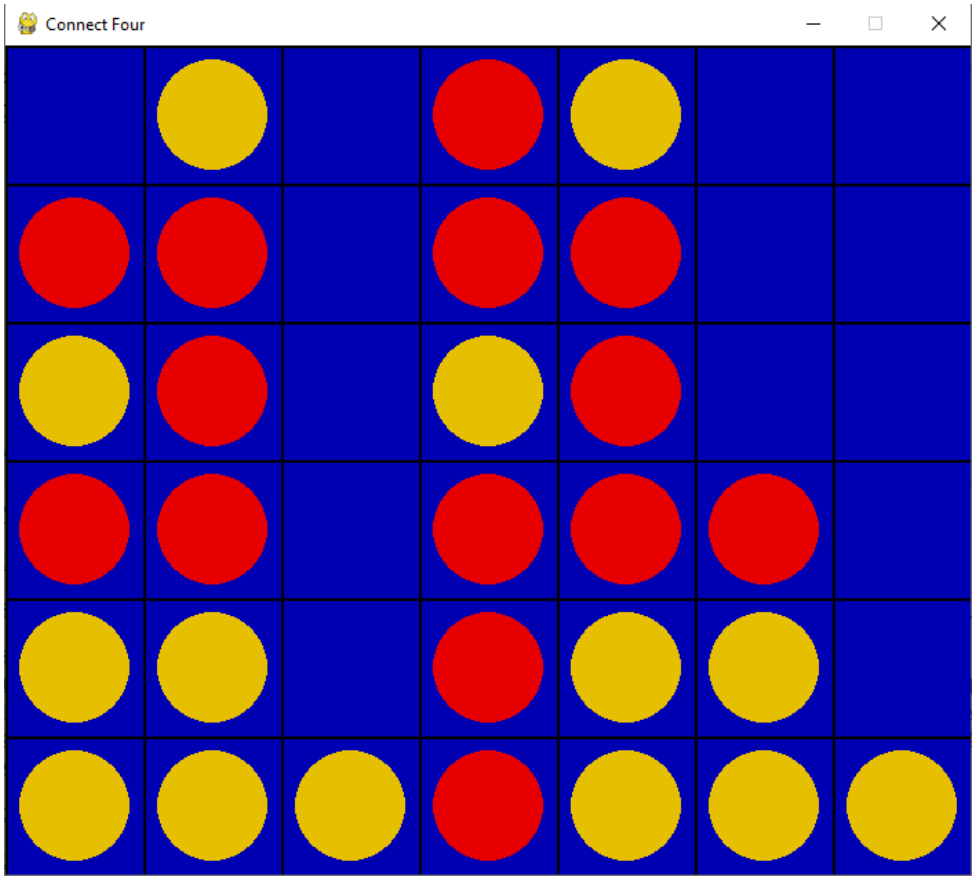
```
plocha_nacrtaj()
pygame.display.update()
while pokrenuto:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pokrenuto = False
        elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
            stup = event.pos[0] // CHELIJA
            if plocha[0, stup] == 0:
                red = prazan_red(stup)
                if red is not None:
                    plocha[red, stup] = na_redu
                    if pobjeda(na_redu):
                        plocha_nacrtaj()
                        pygame.display.update()
                        pobjeda_poruka(na_redu)
                        pokrenuto = False
                        break
                    elif np.all(plocha != 0):
                        plocha_nacrtaj()
                        pygame.display.update()
                        izjednacenje_poruka()
                        pokrenuto = False
                        break
                    plocha_nacrtaj()
                    pygame.display.update()
                    na_redu = 3 - na_redu

    if na_redu == AI and pokrenuto:
        stup = ai_potez()
        if stup is not None:
            red = prazan_red(stup)
            if red is not None:
                plocha[red, stup] = na_redu
                if pobjeda(na_redu):
                    plocha_nacrtaj()
                    pygame.display.update()
                    pobjeda_poruka(na_redu)
                    pokrenuto = False
                    break
                elif np.all(plocha != 0):
                    plocha_nacrtaj()
                    pygame.display.update()
                    izjednacenje_poruka()
                    pokrenuto = False
                    break
                plocha_nacrtaj()
                pygame.display.update()
                na_redu = 3 - na_redu
pygame.quit()
```

Isječak koda 11: Tijek igre

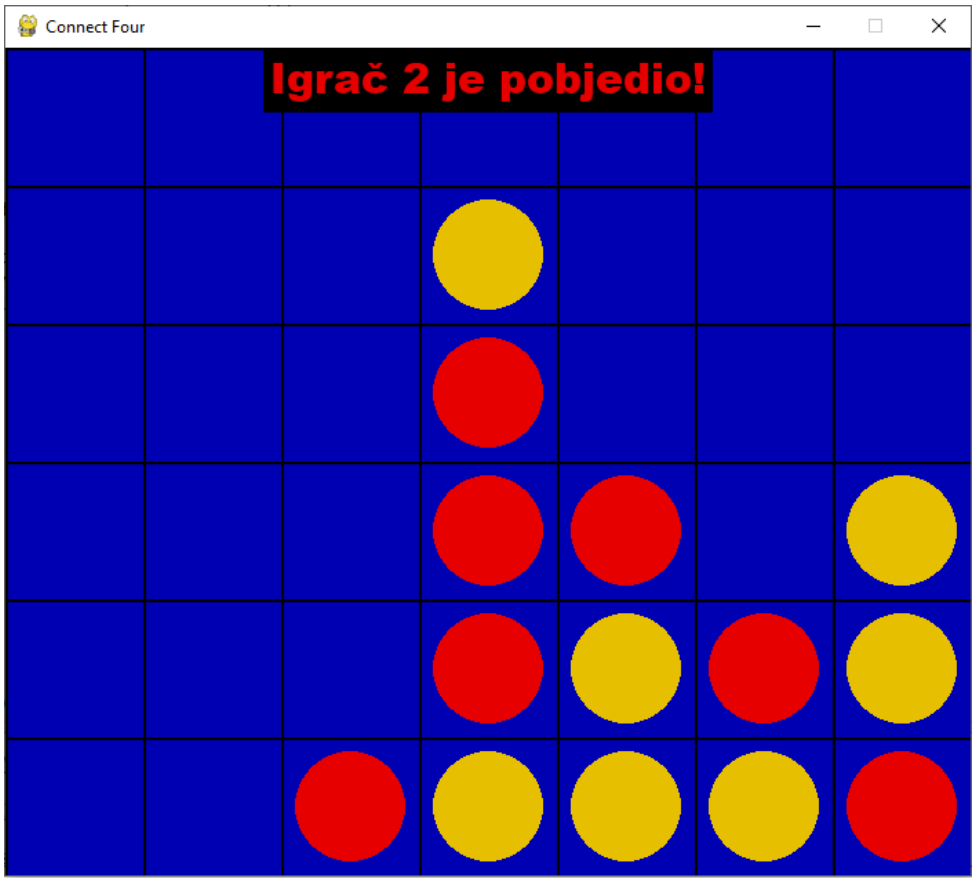
5. Prikaz rada aplikacije

Prilikom pokretanja programa otvara se prozor u kojem je prikazana ploča za igranje. Pritiskom na ploču postavlja se žuti žeton na dnu stupca gdje je ploča pritisnuta mišem. Nakon izvedenog poteza slijedi robot koji odmah nakon traži najbolje mjesto za postaviti svoj žeton. Čovjek može postaviti svoj žeton tek nakon što robot postavi svoj crveni žeton.



Slika 5: Prikaz ploče za igranje

Nakon pobjede prikazuje se poruka pobjednika, te se automatski izlazi iz igre nakon dvije sekunde. Nažalost, nisam u stanju pobjediti umjetnu inteligenciju koju sam kreirao, pa ne postoji slika gdje se prikazuje žuta pobjeda, čak ni izjednačenje.



Slika 6: Prikaz pobjede

6. Zaključak

Ovim projektom prikazano je kako je moguće algoritam pretraživanja primjeniti u korist kreiranja umjetne inteligencije koja je u stanju igrati stolnu igru jako optimalno, do te razine da je prilično teško za prosječnog ljudskog igrača da umjetnu inteligenciju nadmudri i pobjedi. Naravno, kako je već bilo gore opisano, ovaj algoritam nije savršen. On je u stanju pronaći najbolje moguće rješenje, ali to mu uspjeva samo ako je bodovanje za igru pravilno implementirano i samo ako protivnik umjetne inteligencije igra optimalno. Također, ovaj algoritam ovisi o tome koliko mu je dopušteno da traži svoje optimalno rješenje. Najbolje bi bilo omogućiti algoritmu da pretražuje beskonačno mnogo čvorova u stablu, ali danas vjerojatno ne postoji računalo na ovom svijetu koje je u stanju održavati takav algoritam.

Nadam se da je zahtjev ovog projekta koji je bio naveden u uvodu, dakle da se omogući ljudima koji imaju malo iskustva sa umjetnom inteligencijom da steknu novo znanje i razumijevanje vezano uz programiranje umjetne inteligencije, ispunjen do neke razine uspješnosti. Neki ljudi bi možda poprimili veći strah umjetne inteligencije nakon ovog projekta, pošto je ovaj mali python program bio u stanju kreirati mašinu koja u većini stanja igre može pobjediti svojeg ljudskog protivnika.

Popis literature

[1] M. Schatten, „Uvod u umjetnu inteligenciju – Rješavanje problema –," *Sveučilište u Zagrebu, Fakultet organizacije i informatike*, str. 25–26, 2023.

[2] P. Vadapalli. „Min Max Algorithm in AI: Components, Properties, Advantages and Limitations," upGrad. (2020.), adresa: https://www.upgrad.com/blog/min-max-algorithm-in-ai/ (pogledano 5. 1. 2024.).

[3] R. Nasa, R. Didwania, S. Maji i V. Kumar, „Alpha-beta pruning in mini-max algorithm–an optimized approach for a connect-4 game," *Int. Res. J. Eng. Technol*, str. 1637–1641, 2018.

[4] C. Communication. „Connect 4 game," Flickr. (2013.), adresa: https://flickr.com/photos/easy-pics/ (pogledano 6. 1. 2024.).

[5] K. Sheoran, G. DHAND, M. DABASZS, N. DAHIYA i P. PUSHPARAJ, „Solving Connect 4 Using Optimized Minimax and Monte Carlo Tree Search," *Mili Publications. Advances and Applications in Mathematical Sciences*, sv. 21, str. 3303–3313, 2022.

[6] G. L. Team. „Alpha Beta Pruning in AI," Great Learning. (2023.), adresa: https://www.mygreatlearning.com/blog/alpha-beta-pruning-in-ai/ (pogledano 5. 1. 2024.).

[7] K. Galli. „Connect4-Python," GitHub. (2017.), adresa: https://github.com/KeithGalli/Connect4-Python (pogledano 6. 1. 2024.).

[8] B. Pitt. „Connect 4 Algorithm," Sphinx. (2020.), adresa: https://roboticsproject.readthedocs.io/en/latest/ConnectFourAlgorithm.html (pogledano 6. 1. 2024.).

[9] Pygame. „Pygame documentation," Pygame. (2023.), adresa: https://www.pygame.org/docs/ (pogledano 6. 1. 2024.).

Popis slika

Popis isječaka koda