Kolokvijski zadatak – notakto varijacija

*Izjavljujem da je rješenje zadatka 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.*

U ovom dokumentu objasnit će se kôd za pokretanje varijacije na igru križić-kružić, notakto. Što se tiče same povezanosti sa klasičnom verzijom igre križić-kružić, u ovoj varijaciji cilj nije postići potez koji označava „3 za redom“ (bilo po dijagonali, redu ili stupcu), nego obrnuto; onaj tko zadnji napravi potez koji dovršava 3 u nizu gubi. Naime, umjesto da igrači naizmjence koriste 'X', odnosno 'O', ovdje obje strane koriste 'X' za popunjavanje polja. S time je određeno da do izjednačenja ne može doći, tj. netko na kraju uvijek izgubi. Osim toga, ploča može biti više.

1. import math

2. import time

3.

4. # meowmeowmeowmeowmeowmeow

5.

6. class Notakto:

7. def \_\_init\_\_(self):

8. self.ploce = [[[' ' for red in range(3)] for stupac in range(3)] for br\_ploca in range(3)]

9. self.pune\_ploce = [False, False, False]

10. self.aktivne\_ploce = {0,1,2}

11.

12. def print\_ploce(self):

13. for i in self.aktivne\_ploce:

14. print(f"Board {i + 1}:")

15. print(" 1 2 3 ")

16. print(" " + "-" \* 13)

17. for red\_index, red in enumerate(self.ploce[i]):

18. print(f"{red\_index + 1} | " + " | ".join(red) + " |")

19. print(" " + "-" \* 13)

20. print("\n")

O vdje je specifično zadano da se igra odvija na tri ploče, ali lagano bi se podesilo da igra radi i za proizvoljan N broj ploča. Uglavnom, inicijalizacije klase te definiranja dimenzije ploče, stanja punih ploča (jesu li ili nisu pune), i koliko ploča je od ukupnog broja ispunjeno, morao bi se ažurirati br\_ploca da ide do n, pune\_ploce bi morali definirati kao [False] \* n, i postavio bi se range od n za indekse aktivnih ploča. Također bi se ispis doradio. S obizrom da je fokus ovog zadatka bio na algoritmu, tj. prikazivanja rada umjetne inteligencije, ovakvi dijelovi su se sveli na minimalno; utvrđeno je da je dovoljno dobro općenito prikazati ideju igre, a više se posvetiti samom radu algoritma.

1. # meowmeowmeowmeowmeowmeow

2.

3. def popunjena(self, ploca):

4. return all(cell != ' ' for red in ploca for cell in red)

5.

6. def gubitak(self, ploca):

7. for i in range(3):

8. if ploca[i][0] == ploca[i][1] == ploca[i][2] == 'X' or ploca[0][i] == ploca[1][i] == ploca[2][i] == 'X':

9. return True

10. if ploca[0][0] == ploca[1][1] == ploca[2][2] == 'X' or ploca[0][2] == ploca[1][1] == ploca[2][0] == 'X':

11. return True

12. return False

Dalje će se opisati funkcije 'popunjena' i 'gubitak'. Prva funkcija je jednostavna, i vraća koja je ploća popunjena na način da usporedjuje je li vrijednost polja jednaka ili različita od praznog znaka ' ', tako što provjerava red po red. Funkcija 'gubitak' provjerava je li se na nekom mjestu napravio niz od tri popunjena polja, bilo po redu, stupcu, ili dijagonali. Kasnije kako se prolazi po kodu, referencirat će se na ovu funkciju i osvrnut će se na njenu važnost.

1. # meowmeowmeowmeowmeowmeow

2.

3. def mogucnosti(self, ploca\_index):

4. potezi = []

5. for i in range(3):

6. for j in range(3):

7. if self.ploce[ploca\_index][i][j] == ' ':

8. potezi.append((i, j))

9. return potezi

10.

11. def potez(self, ploca\_index, red, stupac):

12. if self.ploce[ploca\_index][red][stupac] == ' ':

13. self.ploce[ploca\_index][red][stupac] = 'X'

14. return True

15. return False

16.

17. def undo(self, ploca\_index, red, stupac):

18. self.ploce[ploca\_index][red][stupac] = ' '

Funkcija koja sprema se moguće poteze, naziva se 'mogucnosti'. Ona isprva stvara praznu listu, zatim kroz dvostruku petlju provjerava koje je polje prazno i pohranjuje ga u listu poteza. Za postavljanje znaka na ploču, koristi se funkcija 'potez'. Provjerava je li vrijednost polja prazan znak i ukoliko je postavlja 'X' na isto. Funkcija 'undo' služi kako bi se na određeno vratila vrijednost praznog znaka ' '. Kasnije će se referencirati navedene funkcije da se bolje objasni u koju svrhu služe.

1. # meowmeowmeowmeowmeowmeow

2.

3. def minimax(self, ploca\_index, maksimiziranje):

4. if self.gubitak(self.ploce[ploca\_index]):

5. return 1 if maksimiziranje else -1

8.

9. if maksimiziranje:

10. najbolja\_vr = -math.inf

11. for (red, stupac) in self.mogucnosti(ploca\_index):

12. self.potez(ploca\_index, red, stupac)

13. vr = self.minimax(ploca\_index, False)

14. self.undo(ploca\_index, red, stupac)

15. najbolja\_vr = max(vr, najbolja\_vr)

16. return najbolja\_vr

17. else:

18. najbolja\_vr = math.inf

19. for (red, stupac) in self.mogucnosti(ploca\_index):

20. self.potez(ploca\_index, red, stupac)

21. vr = self.minimax(ploca\_index, True)

22. self.undo(ploca\_index, red, stupac)

23. najbolja\_vr = min(vr, najbolja\_vr)

24. return najbolja\_vr

25.

26. def najbolji\_potez(self):

27. najbolja\_vr = math.inf

28. najbolji\_potez = None

29. best\_ploca = None

30.

31. for ploca\_index in range(3):

32. if self.pune\_ploce[ploca\_index]:

33. continue

34. for (red, stupac) in self.mogucnosti(ploca\_index):

35. self.potez(ploca\_index, red, stupac)

36. vr = self.minimax(ploca\_index, True)

37. self.undo(ploca\_index, red, stupac)

38. if vr < najbolja\_vr:

39. najbolja\_vr = vr

40. najbolji\_potez = (red, stupac)

41. best\_ploca = ploca\_index

42. return best\_ploca, najbolji\_potez

1. # meowmeowmeowmeowmeowmeow

2.

3. def azuriraj\_aktivne\_ploce(self, ploca\_index):

4. if self.gubitak(self.ploce[ploca\_index]):

5. self.aktivne\_ploce.remove(ploca\_index)

6. self.pune\_ploce[ploca\_index] = True

7. if not all(self.pune\_ploce):

8. print(f"Ploca {ploca\_index + 1} je mrtva i nece se vise prikazivati.\n")

9. time.sleep(2)

Za pomoc prilikom stvaranja minmax algoritma (funkcije minimax i najbolji\_potez) koji će koristiti UI, koristio se chatgpt kako bi dao inicijalnu ideju/pristup. Link za pristup razgovoru dostupan je ovdje: [chatgpt prompt](https://chatgpt.com/share/67309285-e754-8000-b1cc-bf4047fd9cdb). Objašnjeno je da je potreban minmax algoritam i kontekst korištenja te su definirani slučajevi za minimiziranje i maksimiziranje, odnosno kada je koja vrijednost poželjna. Kôd koji je chatgpt generirao nije direktno prepisan ili „izmijenjen“, a generirani odgovor je poslužio kao ideja za stvaranjem funkcije koja će „isprobavati“ svaki potez i simulirati kakav bi rezultat bio za protivnika te funkcije za vraćanje poteza, što omogućuje da se izvede i pohrani vrijednost ili korisnost i rizik svake mogućnosti, tj. stanja ploče dok se ne dođe do najbolje opcije.

Uglavnom, poanta gore navedenog dijela kôda koji je bio prikazan na prvom isječku ('minimax' i 'najbolji\_potez' funkcije) je da se traži najmanja vrijednost protivnika, tj. najniža 'najbolja\_vr' vrijednost. UI „isprobava“ iterativno svaki mogući potez i preko prijašnje spomenute funkcije 'undo' vraća se unazad. Zatim se sprema ploča (best\_ploca) i potez (najbolji\_potez) s najmanjim 'vr' iznosom, što minimizira šanse poraza.

U drugom isječu kôda prikazana je funkcija za ažuriranje aktivnih ploča. Ova funkcija ne služi ubrzanju programa niti je potrebna za pravilan rad programa, ali je načinjena radi bolje preglednosti za vrijeme igre. Ploča se briše iz ispita kad god neka od ploča „umre“, tj. kada netko od igrača na ploči napravi tri u nizu. Na taj način, svaki put kada se traži da igrač odabere koje polje želi ispuniti, bit će vidljive samo one ploče koje su do tog trena aktivne i na kojima je uopće moguće napraviti potez. Funkcija se poziva kad god igrač ili UI naprave potez.

1. # meowmeowmeowmeowmeowmeow

2.

3. def zaigrajmo(self):

4. print("Bokic!! ovo je Notakto, varijacija na igru krizic-kruzic~")

5. print("Oba igraca koriste 'X', a cilj je \*izbjeci\* da se popuni 3 za redom.")

6. print("Dakle, ako na ploci dobijete 3 za redom, gubite!")

7. print("Takodjer se igra preko tri ploce, a ne samo na jednoj :)\n")

8. time.sleep(2)

9.

10. while not all(self.pune\_ploce):

11. self.print\_ploce()

12. igrac\_na\_potezu = 'igrac'

13.

14. try:

15. ploca\_index = int(input("Ploca na kojoj zelite napraviti potez (1-3): ")) - 1

16. if not (0 <= ploca\_index < len(self.pune\_ploce)):

17. print("Ploca ne postoji. Molimo unesite broj između 1 i 3.")

18. time.sleep(2)

19. continue

20. if self.pune\_ploce[ploca\_index]:

21. print("Ta ploca je vec ispunjena, biraj neku drugu!")

22. time.sleep(2)

23. continue

24.

25. red = int(input("Zeljeni red (1-3): ")) - 1

26. stupac = int(input("Zeljeni stupac(1-3): ")) - 1

27. if not (0 <= red < 3) or not (0 <= stupac < 3):

28. print("Uneseni red ili stupac su izvan opsega. Molimo unesite broj između 1 i 3.")

29. time.sleep(2)

30. continue

31.

32. if not self.potez(ploca\_index, red, stupac):

33. print("To polje je vec ispunjeno :( biraj ponovno!!!")

34. time.sleep(2)

35. continue

36.

37. except ValueError:

38. print("Neispravan unos.")

39. time.sleep(2)

40. continue

41.

Preostala je još samo glavna funkcija. Pri pokretanju programa ispisuju se pravila i preko importane 'time' bibilioteke program se stavlja u sleep mode radi bolje preglednosti i intuitivnijeg načina rada programa. Sljedeća je petlja koja se vrti sve dok sve ploče nisu pune, odnosno mrtve. Za početak se ispisuju sve ploče kako bi igrač mogao vidjeti koje su mu sve opcije. S obzirom na to da su ploče, redovi i stupci indeksirani, lakše je snalaziti se. Pokreću se promptovi za željenu ploču, red, i stupac, sve dok su zadovoljeni uvjeti. Ukoliko je odabrana ploča puna ili ukoliko odabrana ploča ne postoji („out of bounds“), ispisat će se primjerena poruka. Isto tako i za biranje reda i stupca. Može se ispisati poruka da je odabrana vrijednost van opsega, a ako postoji odabrana vrijednost, ispisat će se poruka da je polje već ispunjeno ako je utvrđeno da potez nije moguće napraviti.

1. if self.gubitak(self.ploce[ploca\_index]):

2. self.print\_ploce()

3. print(f"Izgubili ste na ploci {ploca\_index + 1}! Napravili ste 3 za redom!\n")

4. self.azuriraj\_aktivne\_ploce(ploca\_index)

5. time.sleep(2)

6.

7. ai\_ploca, ai\_potez = self.najbolji\_potez()

8. if ai\_potez:

9. print("\nSada je AI na potezu...")

10. igrac\_na\_potezu = 'ai'

11. time.sleep(1)

12. self.potez(ai\_ploca, ai\_potez[0], ai\_potez[1])

13. print(f"AI bira plocu {ai\_ploca + 1}, i postavlja X na polje ({ai\_potez[0]}, {ai\_potez[1]})\n")

14.

15. if self.gubitak(self.ploce[ai\_ploca]):

16. self.print\_ploce()

17. print(f"Yay! AI je izgubio na ploci {ai\_ploca + 1}!")

18. self.azuriraj\_aktivne\_ploce(ai\_ploca)

19. time.sleep(2)

20.

21. print("Kraj! Sve ploce su popunjene.")

22. if igrac\_na\_potezu == 'igrac':

23. print("Igrac je napravio zadnji potez - AI pobjedjuje ! >:)\n")

24. else:

25. print("AI je napravio zadnji potez - Igrac pobjedjuje! :)\n")

26.

27. Notakto().zaigrajmo()

U gornjem isječku vidljiv je ostatak prijašnje funkcije. Nakon unosa željenog polja, provjerava se uzrokuje li potez gubitak na odabranoj ploči. Ako da, ispisuje se adekvatna poruka. Kao što je već bilo spomenuto, nakon poteza ažurira se stanje aktivnih ploča. Zatim slijedi dio koda koji služi za određivanje najboljeg poteza za umjetnu inteligenciju. Ispisuje se poruka o odabranom potezu, a zatim se ponovno provjerava je li potez uzrokovao gubitak na odabranoj ploči.

Nadalje, jednom kada se izađe iz prijašnje utvrđene while petlje, tj. kada sve ploče postanu popunjene, ispisat će se da je došao kraj igre. Ovisno o vrijednosti varijable 'igrac\_na\_potezu', čija se vrijednost mijenja ovisno o tome tko je na potezu, ispisat će se adekvatna poruka o finalnom stanju igre; ukoliko je zadnji na potezu bio igrač (osoba), UI pobjeđuje, i obrnuto. Na posljetku, zadnja linija nam služi nam pokretanje igre, tj. pozivanje glavne funkcije.