Izveštaj - Faza 3

“Byte”

Anastasija Vasić 18096

Lucija Stojković 18434

Filip Vidojković 18108

1) Funkcije za generisanje svih mogućih stanja

zaboravili smo da je ubacimo u dokumentaciju za prošlu fazu

Ova funkcija, nazvana *generisiSvaStanja* ima zadatak da generiše sva moguća stanja prilikom odigravanja određenih poteza. Pozivom funkcije *nova_stanja* dobijamo sve potencijalne poteze za igrače, a metodom i*grajZaStanja* dobijamo listu stanja za određene poteze.

```
def generisiSvaStanja(velicina_vece_table):
    poteziX=nova_stanja(velicina_vece_table, boja: "X")
    potezi0=nova_stanja(velicina_vece_table,  boja: "0")
    poteziX.update(potezi0)
    svaStanja=[]
    for key,value in poteziX.items():
        stanjce=igrajZaStanja(key,value)

    return svaStanja
```

```
def igrajZaStanja(kljuc,lista_poteza):
    trenutniRed, trenutnaKol, vrh, preostaloMesta = kljuc
    okolina_prazna = okolna_polja_prazna(velika_tabla, trenutniRed, trenutnaKol)
    lista_fush=[]
    for r in lista_poteza:
        fush_tabla = copy.deepcopy(velika_tabla)
        noviRed, novaKol, trenutnoMestoFigureNaSteku, mestaZaNove = r
        niz = remove_figures_from_matrix(fush_tabla, trenutniRed, trenutnaKol, trenutnoMestoFigureNaSteku, brRedova: 3, brKolona: 3, trenutni_igrac,
                                         noviRed, novaKol)
        mala_matrica = fush_tabla[noviRed][novaKol]
        if (potez_validan(noviRed, novaKol, trenutniRed, trenutnaKol)):
            niz_izbacenih = niz
            k = 0
            napusti_petlje = False
            for i in range(3):
                for j in range(3):
                    if mala_matrica[i][j] == '.':
                        mala_matrica[i][j] = niz_izbacenih[k]
                        k += 1

                        if k == len(niz_izbacenih):
                            # print(mala_matrica)
                            pr = provera_male_matrice(mala_matrica)

                            if pr == 8:
                                poslednji = mala_matrica[2][1]
                                pobedio(velicina_vece_table, poslednji)
```

```
                        reset_mala(mala_matrica)
                    napusti_petlje = True
                    break

            if napusti_petlje:
                break
        lista_fush.append(fush_tabla)
    return lista_fush
```

Funkcija *igrajZaStanja* vraća listu stanja

```
98      def ima_figura_u_polju( rowCurr, colCurr):
99          for row in velika_tabla[rowCurr][colCurr]:
100             for small_table_row in row:
101                 for elem in small_table_row:
102                     if elem != '.':
103                         return True
104         return False
```

2) Funkcije koje na osnovu konkretnog poteza menjaju stanje problema (igre)

```
def igra_je_zavrsena(tabla):
        n = len(tabla)
        ukupanBrojStekova = int((n - 2) * (n / 2) / 8)
        if (stekX > stek0 and (stekX >= (ukupanBrojStekova + 1) // 2)):
            return True
        elif (stekX < stek0 and (stek0 >= (ukupanBrojStekova // 2))):
            return True
        count = [sum(1 for row in matrix for elem in row if elem != '.') for matrix in velika_tabla]
        total_count = sum(count)
        if (total_count == 0):
           return True

        return False
```

Funkcija *igra_je_zavrsena* proverava da li je igra završena na osnovu trenutnog stanja igre predstavljenog matricom tabla. Uvodi se nekoliko uslova za završetak igre:

Prvi uslov proverava da li je igra završena na osnovu broja figura na stekovima igrača X i O. Ako je broj figura na steku igrača X veći od broja figura na steku igrača O i ako je taj broj veći ili jednak polovini ukupnog broja stekova plus jedan, ili ako je broj figura na steku igrača O veći od broja figura na steku igrača X i ako je taj broj veći ili jednak polovini ukupnog broja stekova, igra se smatra završenom.

Drugi uslov proverava da li je broj preostalih figura na tabli jednak nuli, što znači da su sva polja popunjena i da nema više mogućih poteza.

Ako bilo koji od ovih uslova bude ispunjen, funkcija vraća vrednost True, što označava da je igra završena; u suprotnom, vraća vrednost False.

```
def nova_stanja(velicina_vece_table,boja):
    stanja = {}
    for i in range(velicina_vece_table):
        for j in range(velicina_vece_table):
            mala = velika_tabla[i][j]
            p = 0
            trenutni=None
            brojac=0
            for x in range(3):
                for y in range(3):
                    if(mala[x][y]=='.'):
                        break
                    brojac=brojac+1
                    trenutni=mala[x][y]
            if (i + j) % 2 == 0:
                brT = provera_male_matrice(velika_tabla[i][j])

                if brT == 0:
                    continue
                potezi = generisi_poteze(i, j, velicina_vece_table)
                if potezi.values() == 0:
                    continue
                P = list(potezi.values())

                stajeGL = 0
                stajeGD = 0
                stajeDL = 0
                stajeDD = 0
                if okolna_polja_prazna(velika_tabla,i,j):
                    result = nadji_najblizeg_suseda_sa_figurama(velika_tabla, i, j)
                    if result is not None:
                        putanja = []
                        for m in range(3):
                            for n in range(3):
                                if (mala[m][n] == '.'):
                                    break
```

```
                        k = p + 1
                        p = p + 1
                        prebacuje = brT + 1 - p
                for q in range(0, len(result)):
                    rezultati = result[q]
                    putanja.append(najkraca_putanja(i, j, rezultati[0], rezultati[1]))
                    for z in putanja:
                       # print(z[0])
                        skok = z[0]
                        kljucL = [i, j, trenutni, p, prebacuje]
                        kljuc = tuple(kljucL)
                        brooooj = provera_male_matrice(velika_tabla[skok[0]][skok[1]])
                        dozvoljenoooo = 8 - brooooj
                        l = [skok[0], skok[1], dozvoljenoooo]
                        stanja.setdefault(kljuc, []).append(l)
            else:

                if 'GL' in P[0]:
                    brGL = provera_male_matrice(velika_tabla[i-1][j-1])
                    dozvoljenoGL = 8 - brGL
                    stajeGL = brT - dozvoljenoGL

                    if brGL == 0:
                        stajeGL = 9

                    if stajeGL < 0:
                        stajeGL = 0

                if 'GD' in P[0]:

                    brGD = provera_male_matrice(velika_tabla[i - 1][j + 1])
                    dozvoljenoGD = 8 - brGD
                    stajeGD = brT - dozvoljenoGD

                    if brGD == 0:
                        stajeGD = 9

```

```
654
655                     if stajeGD < 0:
656                         stajeGD = 0
657
658                 if 'DL' in P[0]:
659
660                     brDL = provera_male_matrice(velika_tabla[i + 1][j - 1])
661                     dozvoljenoDL = 8 - brDL
662                     stajeDL = brT - dozvoljenoDL
663
664                     if brDL == 0:
665                         stajeDL = 9
666
667                     if stajeDL < 0:
668                         stajeDL = 0
669
670                 if 'DD' in P[0]:
671
672                     brDD = provera_male_matrice(velika_tabla[i + 1][j + 1])
673                     dozvoljenoDD = 8 - brDD
674                     stajeDD = brT - dozvoljenoDD
675
676                     if brDD == 0:
677                         stajeDD = 9
678
679                     if stajeDD < 0:
680                         stajeDD = 0
681
682             for m in range(3):
683                 for n in range(3):
684                     if(mala[m][n] == '.' ):
685                         break
686
687                     k = p + 1
688                     p = p + 1
689                     prebacuje = brT + 1 - p
690
```

```
                if 'GL' in P[0] and k > stajeGL and (brT < prebacuje + brGL) and boja == mala[m][n]:
                    kljucL = [i, j, trenutni, p, prebacuje]
                    kljuc = tuple(kljucL)

                    l = [i - 1, j - 1, dozvoljenoGL]
                    stanja.setdefault(kljuc, []).append(l)
                if 'DL' in P[0] and k > stajeDL and (brT < prebacuje + brDL) and boja == mala[m][n]:
                    kljucL = [i, j, trenutni, p, prebacuje]
                    kljuc = tuple(kljucL)

                    l = [i + 1, j - 1, dozvoljenoDL]
                    stanja.setdefault(kljuc, []).append(l)
                if 'DD' in P[0] and k > stajeDD and (brT < prebacuje + brDD) and boja == mala[m][n]:
                    kljucL = [i, j, trenutni, p, prebacuje]
                    kljuc = tuple(kljucL)
                    l = [i + 1, j + 1, dozvoljenoDD]
                    stanja.setdefault(kljuc, []).append(l)
                if 'GD' in P[0] and k > stajeGD and (brT < prebacuje + brGD) and boja == mala[m][n]:
                    kljucL = [i, j, trenutni, p, prebacuje]
                    kljuc = tuple(kljucL)
                    l = [i - 1, j + 1, dozvoljenoGD]
                    stanja.setdefault(kljuc, []).append(l)

        return stanja
```

```
def generisi_poteze(row,col,velicina_vece_table):
    kljucL = [row,col]
    kljuc = tuple(kljucL)
    moguci_potezi = {}


    if row - 1 >= 0 and col - 1 >= 0 and (provera_male_matrice(velika_tabla[row - 1][col - 1]) != 0):
        moguci_potezi.setdefault(kljuc, []).append("GL")
    if row + 1 < velicina_vece_table and col - 1 >= 0 and (provera_male_matrice(velika_tabla[row + 1][col - 1]) != 0):
        moguci_potezi.setdefault(kljuc, []).append("DL")
    if row - 1 >= 0 and col + 1 < velicina_vece_table and (provera_male_matrice(velika_tabla[row - 1][col + 1]) != 0):
        moguci_potezi.setdefault(kljuc, []).append("GD")
    if row + 1 < velicina_vece_table and col + 1 < velicina_vece_table and (provera_male_matrice(velika_tabla[row + 1][col + 1]) != 0):
        moguci_potezi.setdefault(kljuc, []).append("DD")
    if moguci_potezi == {}:
        moguci_potezi.setdefault(kljuc, []).append("Prazno")


    return moguci_potezi
```

Funkcija "nova_stanja" generiše moguća nova stanja na tabli datoj veličine "velicina_vece_table" u skladu sa određenom "bojom". Iterira kroz svako polje na tabli veće veličine, analizira male matrice unutar svakog polja, i na osnovu parnosti pozicije i provere matrice poziva funkcije "generisi_poteze", "provera_male_matrice", "okolna_polja_prazna", "nadji_najblizeg_suseda_sa_figurama" i "najkraca_putanja". Generiše moguće nove poteze za datu poziciju, uzimajući u obzir različite scenarije (prazna okolna polja, prisustvo figura, dostupni potezi) i ograničenja kretanja figura u skladu sa pravilima igre, dodajući dobijena stanja u rezultujući rečnik stanja "stanja".

Funkcija "generisi_poteze" prima poziciju na tabli, veličinu veće tabele i proverava moguće poteze (gore-levo, dole-levo, gore-desno, dole-desno) unutar veće table, koristeći funkciju "provera_male_matrice". Ako nema mogućih poteza, vraća "Prazno" za tu poziciju kao jedini potez.

3) MinMax algoritam sa odsecanjem

```
def minmax_alpha_beta(igrac, stanje, dubina, alfa, beta):
    if dubina == 0 or igra_je_zavrsena(len(velika_tabla)):
        return proceni_stanje(igrac, stanje)
    moguci_potezi = nova_stanja(len(velika_tabla),igrac)

    if igrac == 'X': #to nam je MAX_IGRAC
        v = float('-inf')
        for i, j, poslednji, prostor in moguci_potezi:
            v = max(v, minmax_alpha_beta(promeni_igraca(igrac), j, dubina - 1, alfa, beta))
            alfa = max(alfa, v)
            if beta <= alfa:
                break
        return v
    else:
        v = float('inf')
        for i, j, poslednji, prostor in moguci_potezi:
            v = min(v, minmax_alpha_beta(promeni_igraca(igrac), j, dubina - 1, alfa, beta))
            beta = min(beta, v)
            if beta <= alfa:
                break
        return v
```

Funkcija *minmax_alpha_beta* predstavlja implementaciju Min-Max algoritma sa alfa-beta odsecanjem za donošenje odluka u igri Slaganje (Bytes). Ova funkcija izračunava optimalnu vrednost za trenutnog igrača u datom stanju koristeći Min-Max pristup, gde se ocenjuje trenutno stanje igre, a zatim se primenjuje alfa-beta odsecanje kako bi se smanjilo vreme pretrage stabla igre. Parametri funkcije uključuju trenutnog igrača, indeks trenutnog stanja na tabli, dubinu pretrage, te vrednosti alfa i beta za praćenje najboljih vrednosti tokom pretrage. Povratna vrednost funkcije predstavlja optimalnu ocenu za trenutnog igrača.

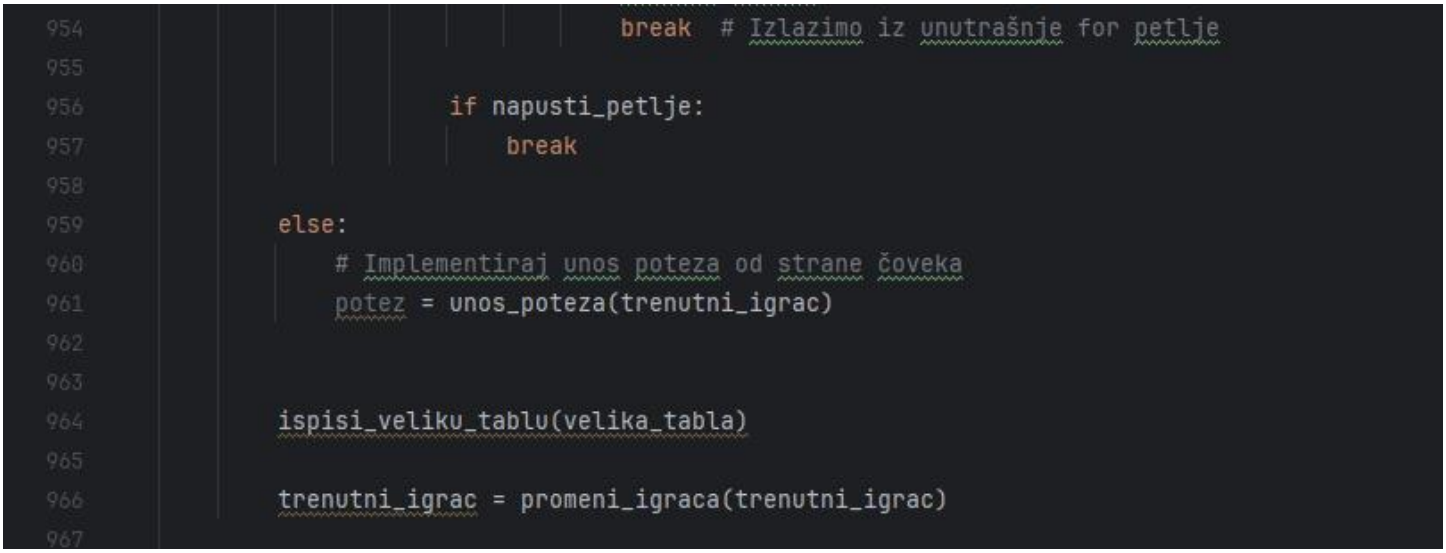
4) Funkcija koja odredjuje najbolji potez

```python
def najbolji_potez(igrac, stanje, dubina):
    moguci_potezi = nova_stanja(len(velika_tabla), igrac)
    najbolji_potez = None
    najbolja_vrednost = float('-inf') if igrac == 'X' else float('inf')

    for potez, lista_poteza in moguci_potezi.items():
        for pojedinacni_potez in lista_poteza:
            vrednost_poteza = minmax_alpha_beta(promeni_igraca(igrac), pojedinacni_potez, dubina - 1, float('-inf'), float('inf'))

            if (igrac == 'X' and vrednost_poteza > najbolja_vrednost) or (igrac == 'O' and vrednost_poteza < najbolja_vrednost):
                najbolja_vrednost = vrednost_poteza
                najbolji_potez = (potez, pojedinacni_potez)

    return najbolji_potez
```

Funkcija *najbolji_potez* predstavlja metodu koja koristi Min-Max algoritam sa alfa-beta odsecanjem kako bi odabrala najbolji potez za trenutnog igrača u igri Slaganje (Bytes). Ova funkcija prolazi kroz sve moguće poteze koje trenutni igrač može izvršiti i koristi Min-Max algoritam za procenu optimalne vrednosti svakog poteza na osnovu dubine pretrage `dubina`. Funkcija prolazi kroz sve moguće poteze igrača, koristeći *moguci_potezi* koji je generisan funkcijom *nova_stanja*. Za svaki potez, poziva Min-Max algoritam sa alfa-beta odsecanjem (implementiran u funkciji *minmax_alfa_beta*) kako bi procenila njegovu vrednost. Zatim, na osnovu vrednosti, bira najbolji potez i vraća ga kao rezultat.

5) Funkcija za odigravanje partije izmdju čoveka i računara

```python
def igraj_partiju():
    trenutni_igrac = 'X'
    stanje = pocetnoStanje()  #inicijalno postavljanje igre

    while not igra_je_zavrsena(velika_tabla):
        if trenutni_igrac == 'X':
            potez = najbolji_potez(trenutni_igrac, stanje, dubina=3)  # Prilagodi dubinu pretrage
            print("OVO JE POTEZ")
            kljuc, lista_poteza = potez
            trenutniRed, trenutnaKol, vrh, poRedu, preostaloMesta = kljuc
            noviRed, novaKol,  mestaZaNove = lista_poteza
            #OVDE REMOVEEE
            niz = remove_figures_from_matrix(velika_tabla, trenutniRed, trenutnaKol, poRedu, brRedova: 3, brKolona: 3, trenutni_igrac, noviRed, novaKol)
            #E SAD IH PREBACIMO
            mala_matrica = velika_tabla[noviRed][novaKol]
            if (potez_validan(noviRed, novaKol, trenutniRed, trenutnaKol)):
                niz_izbacenih = niz
                k = 0
                napusti_petlje = False

                for i in range(3):
                    for j in range(3):
                        if mala_matrica[i][j] == '.':
                            mala_matrica[i][j] = niz_izbacenih[k]
                            k += 1

                            if k == len(niz_izbacenih):
                                #print(mala_matrica)
                                pr = provera_male_matrice(mala_matrica)

                                if pr == 8:
                                    poslednji = mala_matrica[2][1]
                                    pobedio(velicina_vece_table, poslednji)
                                    reset_mala(mala_matrica)

                                napusti_petlje = True
                                break  # Izlazimo iz unutrašnje for petlje
```



```
                                    break  # Izlazimo iz unutrašnje for petlje

                        if napusti_petlje:
                            break

            else:
                # Implementiraj unos poteza od strane čoveka
                potez = unos_poteza(trenutni_igrac)


            ispisi_veliku_tablu(velika_tabla)

            trenutni_igrac = promeni_igraca(trenutni_igrac)
```

Funkcija *igraj_partiju* predstavlja proceduru za izvršavanje partije igre Slaganje (Bytes) između igrača X (računar) i igrača O (čovek). U svakom potezu, funkcija proverava trenutnog igrača, a zatim donosi potez na osnovu Min-Max algoritma za igrača X, ili prima unos poteza od strane čoveka za igrača O. Potezi se primenjuju na veliku tablu, a rezultat igre se prikazuje nakon završetka partije. Petlja se izvršava dok igra nije završena, odnosno dok nije ispunjen uslov za završetak igre.

1. Za svaki potez igrača X, program koristi Min-Max algoritam sa alfa-beta odsecanjem kako bi odabrao najbolji potez u datom trenutnom stanju igre.
2. Potezi se primenjuju na veliku tablu, a rezultat igre se ažurira nakon svakog poteza.
3. Igrači se smenjuju između igrača X (računar) i igrača O (čovek) sve do završetka partije.
4. Na kraju partije, rezultat se prikazuje, pri čemu se proverava da li je pobedio igrač X, igrač O ili je došlo do nerešenog rezultata.

Funkcija koristi niz pomoćnih funkcija za manipulaciju matricama, evaluaciju stanja igre, unos poteza od strane čoveka, i druge aspekte igre. Prilagodite ove funkcije prema specifičnostima vaše implementacije igre Slaganje (Bytes) i njenim pravilima. Sveukupno, *igraj_partiju* predstavlja srce logike igre, usmeravajući njen tok i pružajući interakciju između igrača i računara tokom partije.