Izveštaj - Faza 2

“Byte”

Filip Vidojković 18108

Anastasija Vasić 18096

Lucija Stojković 18434

1) **Funkcije za proveru valjanosti poteza na osnovu konkretnog poteza i trenutnog stanja problema (igre)**

Ova funkcija, nazvana *potez_validan*, ima zadatak da proveri validnost poteza na tabli na osnovu konkretnog poteza I trenutnog stanja igre.

Obuhvata:

- Provera da li se polje nalazi na tabli: Prvo se proverava da li su indeksi odredišnog polja (rowNext, colNext) unutar validnih granica velike table. Ukoliko nisu, funkcija ispisuje poruku o grešci i vraća False.

- Provera dozvoljenog pomeranja: Funkcija proverava da li pomeranje figure ima validne korake. Potezi su dozvoljeni ako se figura pomeri dijagonalno ili za jedno polje u horizontalnom ili vertikalnom smeru. Ako uslov nije zadovoljen, funkcija vraća False.

- Provera broja figura na polju: Funkcija računa trenutni broj figura na destinacionom polju i proverava da li će pomeranje izazvati prekoračenje maksimalnog broja figura na jednom polju (8 figura). Ako je prekoračenje moguće, funkcija ispisuje poruku o grešci i vraća False.

- Povratna vrednost True: Ukoliko su svi uslovi zadovoljeni, funkcija vraća True, označavajući da je potez validan prema pravilima igre.

Ova funkcija igra ključnu ulogu u osiguravanju ispravnosti poteza, pružajući informacije o tome da li je potez validan ili ne.

```
def potez_validan(rowNext, colNext, rowCurr, colCurr):
    if rowNext < 0 or rowNext >= len(velika_tabla) or colNext < 0 or colNext >= len(velika_tabla[0]):
        print("Izabrano polje se ne nalazi na tabli!")
        return False
    #  Da li se krecemo dijagonalno i da li se krecemo za samo jedno polje
    if abs(rowNext - rowCurr) != abs(colNext - colCurr) and abs(rowNext - rowCurr) != 1 and abs(colNext - colCurr) != 1:
        print("Izabrano polje nije u dometu!")
        return False
    # Da li je polje na koje zelimo da odemo crno
    if velika_tabla[rowNext][colNext][1][1] == ' ':
        print("Dozvoljeno je kretati se samo po crnim poljima table.")
        return False
    return True
```

Funkcija *ima_figura_u_polju(rowCurr, colCurr)* proverava da li postoje figure na određenom polju (rowCurr, colCurr) na tabli. Koristi se petlja koja prolazi kroz elemente male matrice i proverava da li se na bilo kojem polju male matrice nalazi figura, vraćajući True ukoliko pronađe figuru ili False ako ne pronađe nijednu figuru na tom polju.

```
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)

Ova funkcija prima poziciju i indeks figure, iterira kroz matricu figura, prikuplja figure na određenoj poziciji, premešta ih na drugu matricu i ažurira matricu odakle se premeštaju figure. Proverava uslove visine i dostupnog prostora za smeštanje figura, prebacujući ih samo ako su zadovoljeni uslovi. Ako su uslovi ispunjeni, funkcija vraća prikupljene figure i visinu matrice. U suprotnom, izbacuje odgovarajuće greške zavisno od situacije.

```
245 def remove_figures_from_matrix(RedC,KolonaC, figure_index, brRedova, brKolona, trIgrac, poz0, poz1):
246     matrix = velika_tabla[RedC][KolonaC]
247
248     figure_index = figure_index - 1
249     if figure_index < 0 or figure_index > 7:
250         raise ValueError("Izabrati vrednost izmedju 1 i 8")
251     visina = 0
252     for i in range(brRedova):
253         for j in range(brKolona):
254             if (matrix[i][j] != '.'):
255                 visina = visina + 1
256     brojac = 0
257     arr = []
258     count = sum(1 for row in velika_tabla[poz0][poz1] for small_table_row in row for elem in small_table_row if
259                 elem != '.')
260     for i in range(brRedova):
261         for j in range(brKolona):
262             I = copy.deepcopy(i)
263             J = copy.deepcopy(j)
264             print(matrix)
265             if (brojac != figure_index):
266
267                 brojac = brojac + 1
268             else:
269                 #if (trIgrac == matrix[i][j]):
270                 for k in range(i, brRedova, 1):
271                         for m in range(j, brKolona, 1):
272                             if (matrix[k][m] != '.'):
273                                 arr.append(matrix[k][m])
274                             if (m == brKolona - 1):
275                                 j = 0
276
277                 okolina_polja = okolna_polja_prazna(RedC,KolonaC)
278                 if(okolina_polja == True):
279                     for k in range(I, brRedova):
280                         for m in range(J, brKolona):
281                             matrix[k][m] = '.'
282                     break
```

```
                else:

                    if (8 - count) >= len(arr):
                        if visina < (len(arr) + count):
                            for k in range(I, brRedova):
                                for m in range(J, brKolona):
                                    matrix[k][m] = '.'
                            break
                        else:
                            raise ValueError("Drugi stek će imati manji broj figura, mora da ih ima više")
                    else:
                        raise ValueError("Nema dovoljno mesta u steku")

        print(matrix)
        print(arr)

        return arr, visina

```

U promenljivu ‘niz’ se smeštaju elementi koje je potrebno prebaciti u odredišnu matricu. Takodje se vrši i provera da li je u odredišnoj matrici dostignuta visina steka 8. Ukoliko jeste, u zavisnosti od toga koja se boja (X ili O) nalazi na vrhu, stek se dodeljuje tom igracu.
Poziva se funkcija pobedio nakon svakog dodeljivanja steka kako bi se proverili rezultati.

```

    if (potez_validan(redNext, kolNext, redCurr, kolCurr) is True):
        niz = remove_figures_from_matrix(red_broj,kolona_broj, figura_mesto, brRedova: 3, brKolona: 3, trenutni_igrac, redNext, kolNext)
    else:
        raise ValueError("Pokusaj opet! Ne smes van table!")
    if(niz == False):
        raise ValueError("Ne valja niz")

    mala_matrica = velika_tabla[redNext][kolNext]
    if (potez_validan(redNext, kolNext, redCurr, kolCurr)):
        niz_izbacenih = niz[0]
        pred_visina = niz[1]

        k = 0
        for i in range(3):
            for j in range(3):
                if (mala_matrica[i][j] == '.'):
                    mala_matrica[i][j] = niz_izbacenih[k]
                    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)
                        if(trenutni_igrac == 'X'):
                            return 'O'
                        else:
                            return 'X'

                        #return 0

```

3) **Funkcije koje obezbeđuju odigravanje partije između dva igrača (dva čoveka, ne računara i čoveka)**

```
                reset_mala(mala_matrica)
            if(trenutni_igrac == 'X'):
                return 'O'
            else:
                return 'X'

            #return 0
```

Ovde se vrši naizmenična promena igrača

4) **Funkcije za operator promene stanja problema (igre) u opštem slučaju (proizvoljno stanje na tabli)**

- Određivanje svih mogućih poteza igrača na osnovu stanja problema (igre)

```
def potencijalni_potezi(row, col):
    print("Potencijalni potezi:", end=" ")
    if(ima_figura_u_polju(row - 1, col - 1) and row-1>=0 and col-1>=0):
        print("GL", end=" ")
    if(ima_figura_u_polju(row - 1, col + 1) and row-1>=0 and col+1<velicina_vece_table):
        print("GD", end=" ")
    if (ima_figura_u_polju(row + 1, col - 1)and row+1<velicina_vece_table and col-1>=0):
        print("DL", end=" ")
    if (ima_figura_u_polju(row + 1, col + 1) and row+1<velicina_vece_table and col+1<velicina_vece_table ):
        print("DD", end=" ")
```

Funkcija *potencijalni_potezi*() ima zadatak da identifikuje potencijalne poteze figure na tabli, pri čemu se razmatraju četiri osnovna pravca kretanja (gore-levo, gore-desno, dole-levo, dole-desno). Ova funkcija igra ključnu ulogu u određivanju mogućih poteza igrača u igri.

Kada igrač dođe na red i kada odabere polje sa kog želi da pomeri figuru, pre nego što definiše ciljno polje, pozivom ove funkcije se omogućava da se korisnik obavesti koji su to potencijalni potezi koje može odabrati, što dodatno korisniku olakšava samu igru.

```
def nadji_najblizeg_suseda_sa_figurama(row, col):
    distance = 0
    min_distance = 2 * velicina_vece_table
    closest_neighbors = []

    while True:
        distance += 1
        found_valid_neighbor = False
        current_neighbors = []

        for i in range(-distance, distance + 1):
            for j in range(-distance, distance + 1):
                if i != 0 or j!=0:
                    neighbor_row, neighbor_col = row + i, col + j
                    if ima_figura_u_polju(neighbor_row, neighbor_col):
                        current_neighbors.append((neighbor_row, neighbor_col))
                        found_valid_neighbor = True

        if found_valid_neighbor:
            if min_distance is None or distance < min_distance:
                min_distance = distance
                closest_neighbors = current_neighbors

        if min_distance and distance > min_distance:
            return closest_neighbors
```

Funkcija *nadji_najblizeg_suseda_sa_figurama()* ima za cilj identifikaciju najbližih suseda koji sadrže figure u odnosu na zadato polje tabli.

Algoritam funkcije koristi pristup pretrage po širini kako bi sistematski pretraživao sva susedna polja u sve većem opsegu od zadatog polja. U početku, postavljena je udaljenost na nulu, a zatim se iterativno povećava kako bi se proširila zona pretrage. Za svako polje u datom opsegu, proverava se da li sadrži figuru pozivom funkcije *ima_figura_u_polju().*

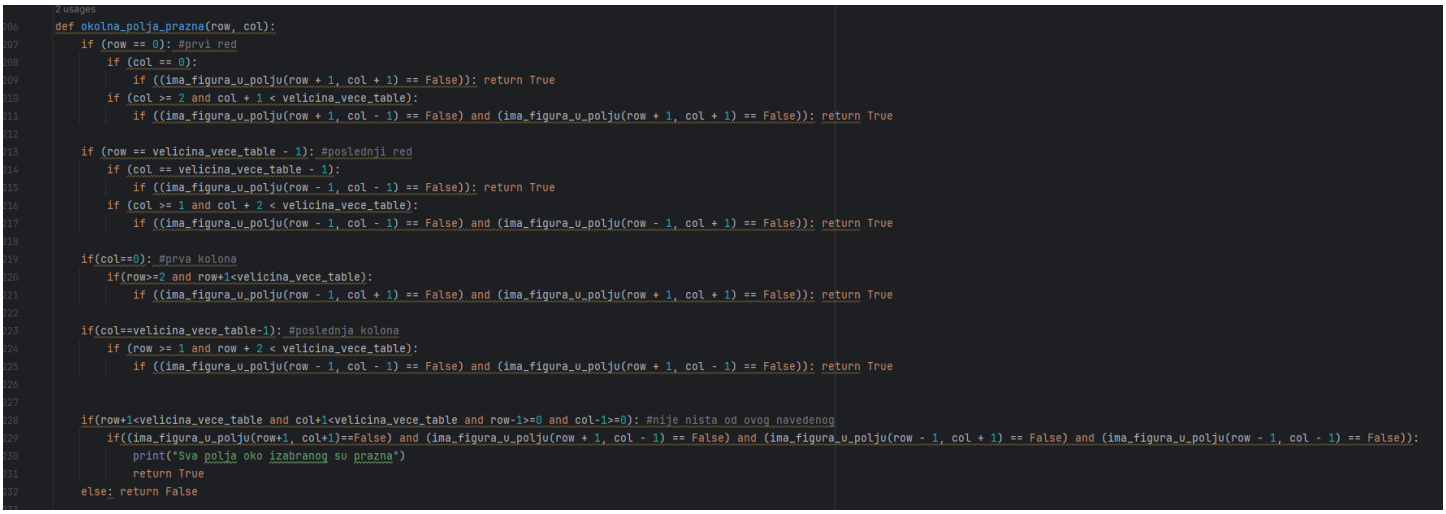
Ako se pronađe figura u susednom polju, informacije o toj poziciji se dodaju u listu *current_neighbors,* a promenljiva *found_valid_neighbor* se postavlja na *True*. Ovaj mehanizam obezbeđuje praćenje svakog suseda koji ispunjava uslove validnosti, tj. koji sadrži figuru.

Nakon svake iteracije, proverava se da li je trenutna udaljenost manja od minimalne dosadašnje udaljenosti (*min_distance*). Ako jeste, ažurira se minimalna udaljenost i lista *closest_neighbors* kako bi se zabeležile informacije o trenutno najbližim susedima sa figurama**.**

Funkcija se nastavlja izvršavati dok se ne dostigne zadnji red susednih polja koji se proveravaju u okviru odabrane udaljenosti. Konačno, vraća se lista *closest_neighbors* koja sadrži pozicije najbližih

suseda sa figurama u odnosu na zadato polje. Ovaj rezultat može biti od ključnog značaja za razne procese u okviru veštačke inteligencije, kao što su procene rizika, generisanje strategija i druge odluke koje se donose u igrama**.**

5) Funkcije koje proveravaju da li su susedna polja prazna



```
2 usages
def okolna_polja_prazna(row, col):
    if (row == 0): #prvi red
        if (col == 0):
            if ((ima_figura_u_polju(row + 1, col + 1) == False)): return True
        if (col >= 2 and col + 1 < velicina_vece_table):
            if ((ima_figura_u_polju(row + 1, col - 1) == False) and (ima_figura_u_polju(row + 1, col + 1) == False)): return True

    if (row == velicina_vece_table - 1): #poslednji red
        if (col == velicina_vece_table - 1):
            if ((ima_figura_u_polju(row - 1, col - 1) == False)): return True
        if (col >= 1 and col + 2 < velicina_vece_table):
            if ((ima_figura_u_polju(row - 1, col - 1) == False) and (ima_figura_u_polju(row - 1, col + 1) == False)): return True

    if(col==0): #prva kolona
        if(row>=2 and row+1<velicina_vece_table):
            if ((ima_figura_u_polju(row - 1, col + 1) == False) and (ima_figura_u_polju(row + 1, col + 1) == False)): return True

    if(col==velicina_vece_table-1): #poslednja kolona
        if (row >= 1 and row + 2 < velicina_vece_table):
            if ((ima_figura_u_polju(row - 1, col - 1) == False) and (ima_figura_u_polju(row + 1, col - 1) == False)): return True


    if(row+1<velicina_vece_table and col+1<velicina_vece_table and row-1>=0 and col-1>=0): #nije nista od ovog navedenog
        if((ima_figura_u_polju(row+1, col+1)==False) and (ima_figura_u_polju(row + 1, col - 1) == False) and (ima_figura_u_polju(row - 1, col + 1) == False) and (ima_figura_u_polju(row - 1, col - 1) == False)):
            print("Sva polja oko izabranog su prazna")
            return True
    else: return False
```

Funkcija *okolna_polja_prazna*() ima zadatak da proveri da li su sva susedna polja određenog polja prazna.
If blok proverava da li sva susedna polja (gore-levo, gore-desno, dole-levo, dole-desno) nisu zauzeta figurama, i da li su nove pozicije unutar granica table.
Ako su sva susedna polja prazna, ispisuje se odgovarajuća poruka, a funkcija vraća True.
Ako uslov nije ispunjen, vraća se False, što znači da bar jedno od susednih polja sadrži figuru ili da nova pozicija nije unutar granica table.

Ova funkcija je korisna u situacijama kada je potrebno proveriti da li je neka figura okružena praznim poljima, što ima uticaj na strategiju igre ili pravila kretanja pojedinih figura.

Konkretno, ukoliko su sva okolna polja prazna, pozivamo metodu koja će da nam vrati najbliže stekove I putanje kako bi igrač predvideo u kom pravcu treba da se kreće.

6) **Funkcije koje na osnovu konkretnog poteza i stanje igre proveravaju da li on vodi ka jednom od najbližih stekova (figura)**

Ukoliko su okolna polja prazna ova funkcija omogućava da se pronadju najbliži stekovi do kojih je potrebno otići da bi se igra nastavila. Prikazuju se najblizi stekovi i putanje do njih. Ukoliko igrac ne ispostuje neku od putanja, već krene u nekom drugom smeru, potez nece biti važeći. Dobija obaveštenje o tome i vraća se na ponovno unošenje poteza.

```
def najkraca_putanja(trenutni_red, trenutni_kol, ciljni_red, ciljni_kol):
    redovi = len(velika_tabla)
    kolone = len(velika_tabla[0])
    def validno_polje(red, kol):
        return 0 <= red < redovi and 0 <= kol < kolone and velika_tabla[red][kol][1][1] == '.'
    # Inicijalizacija reda za BFS
    red = deque([(trenutni_red, trenutni_kol, [])])

    # Set za praćenje posećenih polja
    posecena_polja = set()

    while red:
        trenutno_polje = red.popleft()
        trenutni_red, trenutni_kol, putanja = trenutno_polje

        if (trenutni_red, trenutni_kol) == (ciljni_red, ciljni_kol):
            # Ako smo stigli do ciljnog polja, vraćamo putanju
            return putanja

        if (trenutni_red, trenutni_kol) not in posecena_polja:
            posecena_polja.add((trenutni_red, trenutni_kol))
```

```
        # Susedna polja
        susedi = [
            (trenutni_red - 1, trenutni_kol - 1),  # gore-levo
            (trenutni_red - 1, trenutni_kol + 1),  # gore-desno
            (trenutni_red + 1, trenutni_kol - 1),  # dole-levo
            (trenutni_red + 1, trenutni_kol + 1),  # dole-desno
        ]

        for sused in susedi:
            red_sused, kol_sused = sused
            if validno_polje(red_sused, kol_sused):
                red.append((red_sused, kol_sused, putanja + [(red_sused, kol_sused)]))


return []
```

```
    figura_mesto=None
    redNext = None
    kolNext = None
    flag = 0
    #provera da nisu sva okolna polja oko naseg polja prazna
    okolina_prazna=okolna_polja_prazna(red_broj, kolona_broj)
    #ako je okolina prazna da se pravi funkcija koja proverava najkraci put do sledeceg polja koje nije prazno
    if(okolina_prazna):
        flag = 1
        result = nadji_najblizeg_suseda_sa_figurama(red_broj, kolona_broj)
        if result is not None:
            putanja = []
            for i in range(0, len(result)):
                rezultati = result[i]
                putanja.append(najkraca_putanja(red_broj, kolona_broj, rezultati[0], rezultati[1]))
            print("Putanje do najblizih stekova: ")
            print(putanja)
            rn, cn = result  #ovo su red next i col next iz putanje

            print("Najblizi stekovi su na pozicijama:", rn, cn)

        else:
            print("Result je nun")
            return
```

```python
#PROVERA DA LI SE KRECEMO DOBROM PUTANJOM
if(flag == 1):
    def trenutna_pozicija_u_putanjama(trenutna_pozicija, lista_putanja):
        for putanja in lista_putanja:
            if trenutna_pozicija in putanja:
                return True
        return False
    trenutna_pozicija = (redNext, kolNext)

    if trenutna_pozicija_u_putanjama(trenutna_pozicija, putanja):
        print(f"Trenutna pozicija {trenutna_pozicija} se krece ka jednom od najblizih stekova.")
    else:
        raise ValueError(f"Trenutna pozicija {trenutna_pozicija} se ne krece ka jednom od najblizih stekova.")
```

----opisss

Plus napisi za proveru da li idemo u jednom ka najblizih stekova