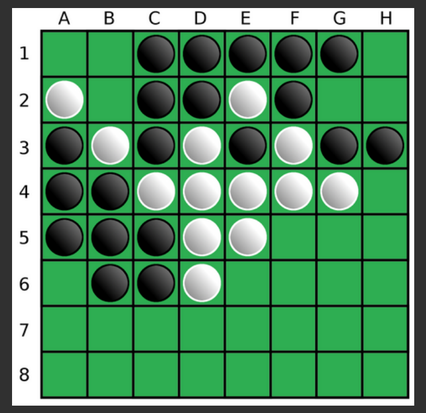
STRATEŠKA IGRA REVERSI

Nikola Sovilj SW75/2019

Motivacija

Reversi ili *Othello* je strateška igra za dva igrača koja se igra na tabli dimenzije 8x8. Svaki igrač poseduje figure (crne ili bele) i pobednik je onaj koji pozicioniranjem svojih figura promeni boju svih protivničkih figura u boju svojih figura. U ovom radu, igra je kreirana u režimu čovek VS bot.



Slika 1 - Tabla za igru Reversi

Promena boje figura

Figure menjaju boju tako što se protivničke figure postave na takav način da ih okruže sa dve strane u jednom pravcu. Mogući pravci su: sever, jug, zapad, istok, severozapad, severoistok, jugozapad i jugoistok. Obratiti pažnju da je moguće jednim potezom eliminisati i više protivničkih figura, kao što je prikazano na *slici 2*. Simbolom $ prikazane su moguće pozicije na koje čovek može postaviti figuru.

Slika 2 - Postupak promene boje figura sa simbolom "W"

Problem 1 - Pronalazak svih mogućih poteza

Prvi problem koji treba rešiti jeste pronalazak svih mogućih poteza koje bot može da odigra, odnosno sve moguće pozicije na koje može postaviti svoju figuru. Takve pozicije su prikazane simbolom $.

Slika 3 - Prikaz mogućih pozicija za sledeći potez

Kako bi se odredilo na koje koordinate treba postaviti simbol mogućeg poteza, potrebno je za svaku od figura:

- Iterirati u svim pravcima
- Prepoznati niz protivničkih figura
- Na prvoj sledećoj slobodnoj poziciji, posle niza protivničkih figura, postaviti $ simbol

Konkretno u kodu, izvršavanje drugog i trećeg koraka je implementirano u funkcijama *iterate_north(), iterate_south() ...* Funkcija *mark_allowed_fields()* poziva sve prethodno navedene funkcije, gde je svaka zadužena za iteriranje u jednom od pravaca.

```
def mark_allowed_fields(self, is_disc_changing):
    for i in range(self._row_length):
        for j in range(self._column_length):
            if self._game_table[i][j] == 'B':
                north_x, north_y = self.iterate_north(i, j, is_disc_changing)

                if north_x != None and north_y != None and self._game_table[north_x][north_y] != 'B':

                    self._game_table[north_x][north_y] = '$'

                south_x, south_y = self.iterate_south(i, j, is_disc_changing)

                if south_x != None and south_y != None and self._game_table[south_x][south_y] != 'B':
```

Slika 4 - Snippet koda za prethodne korake

Problem 2 - Pronalazak najpovoljnijeg poteza

Kako bi bot bio što efikasniji, potrebno je implementirati algoritam koji bi kao rezultat predstavio najbolji potez od svih mogućih. Najbolji potez zapravo predstavlja izbor onog poteza koji bi uspeo da promeni boju što većem broju protivničkih igrača. Stoga, heuristika za ovaj slučaj predstavlja broj promenjenih protivničkih figura, prilikom igranja poteza od strane bota. Algoritam koji se u ovom radu koristio je modifikovana verzija Minimax algoritma, koja na brz način određuje najbolji sledeći potez.

```
def minimax(self, allowed_white_moves):
    max_changed_discs_move = [0, 0, -1]
    for move in allowed_white_moves:
        x_coord = move[0]
        y_coord = move[1]
        num_of_changed_discs = move[2]
        if num_of_changed_discs > max_changed_discs_move[2]:
            max_changed_discs_move[0] = x_coord
            max_changed_discs_move[1] = y_coord
            max_changed_discs_move[2] = num_of_changed_discs

    return max_changed_discs_move
```

Slika 5 - Snippet koda izmenjenog minimax algoritma

Analiza rezultata

- Bot nije nepobediv, međutim, predstavlja dobrog saigrača za početnike
- Vremenski intervali odabira sledećeg poteza od strane bota:
  - 0,000996 sekundi
  - 0,001001 sekundi
  - 0,001000 sekundi
  - 0,002063 sekundi
  - 0,001000 sekundi

PROSEK: **0,001212 sekundi**