Veštačka Inteligencija

Izveštaj I faze projekta

Slaganje (Byte)

Naziv tima : DavidDusanVeljko

Dušan Stojanovic 17450
Veljko Marković 17745
David Stanisavljević 17943

Uvod

Slaganje (Byte) je strateška igra gomilanja figura unapred postavljenih na tabli. U nastavku mozete pročitati kratak opis igre .

1. ***Korisnički interfejs i Izgled igre*:** Table mogu biti različitih veličina($n \times n$, $n\%2 = 0$), od preporučene veličine *8x8* do maksimalne veličine *16x16*. Figure se nalaze na crnim poljima table tako da se figure jednog igrača nalaze u parnim, a drugog u neparnim redovima, pri čemu su prvi i poslednji red prazni.

2. ***Mehanika igre:*** Igrači (označeni kao X i O, ili crni i beli) naizmenično naizmenično odigravaju po jedan potez, u kome figure mogu biti pomaknute samo dijagonalno za jedno polje.

3. ***Cilj igre*:** Cilj igre je složiti što više "stekova" od 8 figura, sa figurom svoje boje na vrhu. Pobednik je igrač koji uspe da složi više takvih stekova.

4. ***Modovi igre*:** Igra podržava igru čovek protiv računara, pri čemu igrač može izabrati da li će prvi igrati on ili računar.

I faza projekta:

U prvoj fazi projekta treba definišati način predštavljanje štanje problema tj igre ,ošnovne funkčije igre i grafički korišnički interfejš.

Predstavljanje stanje problema (igre):

FieldColor i CheckerColor: Ove dve klase su Enum klase koje se koriste za definisanje boja za polja i figure u igri. One pružaju skup simboličkih imena (BLACK, WHITE, X i O) vezanih za jedinstvene konstantne vrednosti ["Black", "White", "X", "O"]. Ove vrednosti se koriste za predstavljanje stanja polja ili figure u igri.

```
class FieldColor(Enum):
    BLACK = "Black"
    WHITE = "White"

field_black=FieldColor.BLACK
field_white=FieldColor.WHITE

class CheckerColor(Enum):
    X="X"
    O="O"

checker_black=CheckerColor.X
checker_white=CheckerColor.O
```

Klasa Field predstavlja jedno polje na igračkoj tabli. Ona ima tip polja (crno ili belo), i stek koji može da sadrži figure. Metoda `add_checker` se koristi za dodavanje figure na stek, a metoda `__str__` se koristi za kreiranje string reprezentacije polja za prikaz igračke table.

Dok klasa Board predstavlja igračku tablu. Ona sadrži listu `Field` objekata i ima metode za kreiranje igračke table (`__init__`), i za kreiranje string reprezentacija igračke table (`str` i `__str__`).

```python
class Field():
    def __init__(self,field_type,checker):
        self.field_type=field_type
        self.stack=[]
        if checker is not None:
            self.stack.append(checker)

    def add_checker(self,checker)->Optional[CheckerColor]:
        length=len(self.stack)
        if length+1<8:
            self.stack.append(checker)
            return None
        else:
            self.stack=[]
            return checker

    def empty(self)->bool:
        if len(self.stack)>0:
            return False
        return True

    def __str__(self):
        character='.' if self.field_type==field_black else ' '
        matrix = [[character for _ in range(3)] for _ in range(3)]
        for i, checker in enumerate(self.stack):
            if i < 9:
                row, col = divmod(i, 3)
                matrix[row][col] = checker.value
        return "\n".join(''.join(row) for row in matrix) + "\n"
```

```python
class Board():
    def __init__(self, num_of_fields):
        self.num_of_fields = num_of_fields
        self.fields = []

        for row in range(num_of_fields):
            row_fields = []
            for col in range(num_of_fields):
                is_black_field = (row + col) % 2 == 0
                field = Field(field_black if is_black_field else field_white, None)

                if is_black_field and 0 < row < num_of_fields - 1:
                    checker = checker_black if row % 2 == 1 else checker_white
                    field.stack.append(checker)

                row_fields.append(field)
            self.fields.append(row_fields)

    def empty(self):
        for field in self.fields:
            if not field.empty():
                return False
        return True

    def __str__(self):
        column_labels = "  " + "   ".join(str(i + 1) for i in range(self.num_of_fields))
        board_str = column_labels + "\n"

        for i, row in enumerate(self.fields):
            row_label = chr(65 + i)
            row_str_lines = ['' for _ in range(3)]
            for field in row:
                field_lines = field.__str__().split('\n')
                for j in range(3):
                    row_str_lines[j] += field_lines[j] + " "

            board_str += f"{row_label} " + "\n  ".join(row_str_lines) + "\n"

        return board_str
```

Funkcija za postavljanje početnog stanja:

Za Klasu *Game* možemo reći da predstavlja samu igru. Ova klasa sadrži objekat Board, igrače, i pobednika,pored ovih metoda sadržo I metode za početak igre (start), inicijalizaciju igre (init), dobijanje veličine table (get_board_size), dobijanje prvog igrača (get_first_player), unosenje poteza (input_move), i proveru da li je potez validan (is_valid_move). Metoda __str__ se koristi za kreiranje string reprezentacije igre.Kada pokrenemo nasu program naipre se pokrene metoda da proveri kolku dimenziju table zelimo. Kada korisnik unese velicinu I vrednost se proveri da li je parna(uslov igre). Pokrenuce se funkcija koja ce korisnika upitati da li igra prvo korisnik ili racunar , posle izabira moda table zeljene veličine ce se iscrtati.

```python
class Game():
    def __init__(self):
        self.board=None
        self.winner=None

    def start(self):
        self.get_board_size()
        player1=Player(checker_white)
        player2=Player(checker_black)
        if(1==self.get_first_player()):
            self.init(player1,player2)
        else:
            self.init(player2,player1)
        print(self)

    def init(self,player1,player2):
        self.board=Board(self.board_size)
        self.current_player=player1
        self.player1=player1
        self.player2=player2

    def get_board_size(self):
        self.board_size = int(input("Board size: "))

    def get_first_player(self):
        print("Choose who goes first: ")
        print("1.O")
        print("2.X")
        return int(input())
```

```
Board size: 8
Choose who goes first:
1.O
2.X
```

```
  1   2   3   4   5   6   7   8
A ...     ...     ...     ...
  ...     ...     ...     ...
  ...     ...     ...     ...
B     X..     X..     X..     X..
      ...     ...     ...     ...
      ...     ...     ...     ...
C O..     O..     O..     O..
  ...     ...     ...     ...
  ...     ...     ...     ...
D     X..     X..     X..     X..
      ...     ...     ...     ...
      ...     ...     ...     ...
E O..     O..     O..     O..
  ...     ...     ...     ...
  ...     ...     ...     ...
F     X..     X..     X..     X..
      ...     ...     ...     ...
      ...     ...     ...     ...
G O..     O..     O..     O..
  ...     ...     ...     ...
  ...     ...     ...     ...
H     ...     ...     ...     ...
      ...     ...     ...     ...
      ...     ...     ...     ...
```

Funkcija za proveru kraj igre:

U klasi Game takodje se nalaze metoda won I is_over koje određuju pobednika partije zavisno od od razloga koji je doveo do zavrsetka partije bilo to da je tabla prazna ili jedan od igrača poseduje više od polovine mogućih stekova.

```
def won(self):
    num_of_checkers=((self.board_size-2)*self.board_size/2)
    max_score=num_of_checkers/8
    win_score=(2*max_score)//3

    if(self.player1.score>win_score):
        self.winner=self.player1
        return True

    elif(self.player2.score>win_score):
        self.winner=self.player2
        return True

    return False

def is_over(self):
    if self.board.empty():
        return False, "Board is empty."
    if not self.won():
        return False, "No winner, yet."
    return True, f"{self.winner} has won!"

def __str__(self):
    return f"{self.board}\n" \
           f"Player 1: {self.player1}\n" \
           f"Player 2: {self.player2}\n\n"
```