Veštačka inteligencija

Projekat – Dominacija (Domineering)

Faza I – Formulacija problema i interfejsa

Stefan Stojadinović 17975, Andrija Tošić 18015

Deklaracije i promenljive koje se koriste u daljoj izradi

Za čuvanje redosleda poteza

```
class Turn(Enum):
    VERTICAL = True,
    HORIZONTAL = False
```

Za prikaz stanja table

```
class Square(Enum):
    EMPTY = '[ ]'
    VERTICAL = Fore.BLUE + '[x]' + Style.RESET_ALL
    HORIZONTAL = Fore.RED + '[*]' + Style.RESET_ALL
```

Za čuvanje tipa igrača

```
class Player(Enum):
    HUMAN = True
    AI = False
```

Type alias za potez

```
Move = tuple[int, int]
```

Regularni izraz za izdvajanje koordinata iz unosa poteza

```
move_matcher = re.compile(r'\[?(?P<xcord>\d+)(?:[, ])*(?P<ycord>[a-zA-Z])\]?')
```

Način za predstavljanje stanja problema (igre)

Stanje na tabli jednoznačno je određeno članicama klase `State` (imenovani tuple): `n` i `m` koje definišu dimenzije table, članicom `to_move` koja ukazuje na igrača koji je na potezu i skupovima koji sadrže moguće i odigrane poteze za vertikalnog i horizontalnog igrača: `v_possible_moves`, `v_played_moves`, `h_possible_moves`, `h_played_moves` respektivno.

Funkcija koja obezbeđuje unos početnih parametara igre

```
def input_board_dimensions() -> tuple[int, int]
```

Funkcija za postavljanje početnog stanja table

Funkcija na osnovu prosleđenih dimenzija table i zadatog igrača koji je prvi na potezu kreira inicijalnu reprezentaciju stanja koja u skupovima mogućih poteza za vertikalnog i horizontalnog igrača sadrži sve dostupne poteze za zadate dimenzije table, dok promenljive za odgovarajuće odigrane poteze ukazuju na prazne skupove.

```
def create_initial_state(
      n: int = 8,
      m: int = 8,
      initial_to_move = Player.VERTICAL
) -> State
```

Funkcija za proveru kraja igre

```
def is_game_over(state: State) -> bool
```

Funkcija koja obezbeđuje prikaz proizvoljnog stanja problema (igre)

```
def print_state(state: State) -> None
```

```
   A  B  C  D  E  F  G  H  I  J
10 [ ][x][ ][ ][ ][ ][ ][ ][x][ ] 10
 9 [ ][x][ ][ ][ ][ ][ ][ ][x][ ] 9
 8 [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ] 8
 7 [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ] 7
 6 [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ] 6
 5 [*][*][ ][ ][ ][ ][ ][ ][ ][ ] 5
 4 [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ] 4
 3 [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ] 3
 2 [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ] 2
 1 [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ] 1
   A  B  C  D  E  F  G  H  I  J
HORIZONTAL to move
```

Funkcija za unos poteza

Uspešno prepoznavanje poteza kao rezultat vraća tuple (x, y) gde x i y određuju vrstu i kolonu gornjeg ili levog kvadrata vertikalne i horizontalne pločice respektivno. U slučaju nevaljanog unosa vraća se None.

```
def input_move(to_move: Player) -> Move | None
def parse_move(move: str) ->  Move | None
```

Funkcija koja proverava da li je potez valjan

Provera valjanosti poteza omogućena je pomoću ažurnosti skupova koji sadrže moguće poteze za odgovarajućeg igrača.

```
def is_valid_move(state: State, move: Move) -> bool
```

Izbor igrača koji je prvi na potezu

```
def input_move_order() -> Turn
```

Izbor računara ili čoveka za prvog igrača (izbor i jednog i drugog igrača)

```
def input_player_type(prompt: str = "") -> Player
```