

# Veštačka Inteligencija

Izveštaj II faze projekta

## Domineering

Naziv tima : **MVP**

**Milić Aleksa 17774**

**Vulić Vladan 16511**

**Petrović Miloš 16285**

## Uvod:

Dominacija (Dominnering) je jednostavna matematička strateška igra za dva igrača sa nultom sumom. U Dominaciji dva igrača imaju kolekciju domina (veličine  $2 \times 1$ ) koje naizmenično postavljaju na tablu igre, prekrivajući kvadrate. Tabla igre može biti bilo kog oblika kvadrat ili pravougaonik, najčešće se igra na tabli veličine  $8 \times 8$ . Dva igrača su označena kao Vertikalni i Horizontalni igrači. U standardnoj igri Dominacije prvi igrač je Vertikalni, kome je dozvoljeno samo da svoje domine vertikalno postavlja na tablu a Horizontalni samo horizontalno. Naravno, domine ne smeju da se preklapaju i kao u većini igara u kombinatornoj teoriji igara, prvi igrač koji ne može da postavi dominu gubi igru.

## I faza projekta:

U prvoj fazi projekta treba definisati način predstavljanje stanje problema tj igre, osnovne funkcije igre i grafički korisnički interfejs.

### Predstavljanje stanje problema (igre):

Trenutno stanje igre se prati u klasi *Game*. U klasi *Game* atribut *matrix* prati stanje igre na tabeli, lista kolona i redova prikazuje slobodne pozicije i zauzete pozicije vertikalnih i horizontalnih domina. Imamo attribute *player1* i *player2* koji su tip klase *Player* kao i atribut *players\_turn* koji prati čiji je trenutni potez u igri. Klasa *Player* ima attribute *human\_or\_pc* koji prikazuje da li je igrač čovek ili računar što će biti relevantno u kasnijim fazama projekta. Takođe sadrži atribut *sign* koja prikazuje oznaku vertikalnog prvog igrača sa simbolom (X) i horizontalnog drugog igrača sa simbolom (O).

#### class Game:

```
matrix: []
players_turn: Player
player1: Player
player2: Player
```

#### class Player:

```
def __init__(self, sign, who_plays: bool):
    self.human_or_pc = who_plays # 0-pc, 1-human
    self.sign = sign
```

### Funkcija za postavljanje početnog stanja:

Funkciji za postavljanje početnog stanja se prosleđuju vrednosti kolone i vrste table kao i da li je prvi igrač računar ili čovek. Pre funkcije za postavljanje imamo funkcije za unos vrednosti kolona i vrsta table i tip prvog igrača. Postavljaju se vrednosti kolona i vrsta table, inicira se matrica početnog stanja sa praznim poljima. Kreiraju se dva igrača, prvom igraču se dodeljuje simbol X i vrednost da li se radi o računaru ili čoveku a drugom igraču se dodeljuje simbol O i vrednost igrača da se radi o čoveku. Dodelju se atributu *players\_turn* prvi igrač jer na početku svake igre prednost ima vertikalni igrač X. Zatim se štampa tabla igre sa početnim stanjima tj korisnički interfejs u konzoli.

```
def __init__(self, human_or_pc1, n: int, m: int):
    self.N = n
    self.M = m
    self.matrix = [{" " for i in range(0, M)] for j in range(0, N)]
    self.player1 = Player("X", human_or_pc1)
    self.player2 = Player("O", True) # 2.player is always human
    self.players_turn = self.player1
    self.print_table()
```

## Funkcija koja obezbeđuje prikaz proizvoljnog stanja igre:

Funkcija `print_table()` obezbeđuje prikaz proizvoljnog stanja igre tj table. Funkcija se poziva na kraju funkcije za postavljanje početnog stanja i na kraju funkcije za prelazak u novo stanje `play_a_turn()`. U funkciji se iscrtava matrica stanje igre sa trenutnim vrednostima postavljenih domina u formatiranoj tabeli sa gridom i simbolima vrste i kolona. Ceo prikaz se štampa u konzoli računara.

	A	B	C	D	E	F	
	=	=	=	=	=	=	
6							6
	---	---	---	---	---	---	
5				O	O		5
	---	---	---	---	---	---	
4			X				4
	---	---	---	---	---	---	
3			X				3
	---	---	---	---	---	---	
2							2
	---	---	---	---	---	---	
1							1
	---	---	---	---	---	---	

```
def print_table(self):
    letter = 65 # A
    # vrh table
    print(" ", end=" ") # corner
    for i in range(0, M):
        print(f" {chr(letter + i)}", end=" ")

    print("")
    print(" ", end=" ")
    for i in range(0, M):
        print(" =", end=" ")

    print("")
    # matrix
    for i in range(0, N):
        print(f"{N - i}||", end=" ")
        for j in range(0, M):
            print(f" {self.matrix[i][j]} |", end=" ")
        print(f"{N - i}")
        print(" ", end=" ")
        for _ in range(0, M):
            print(" ---", end=" ")
        print(" ")
```

## Funkcija za unos novog stanja:

Funkcija `play_a_turn()` služi za unos novog stanja u igri. Prosleđuje joj se trenutno stanje igre i igrač unosi koordinate table gde želi da postavi dominu. Ukoliko je potez valjan, ažurira se stanje table igre u matrici `matrix` na osnovu čiji je trenutni red igrača `players_turn`. Ukoliko je prvi igrač upisuje se simbol X na zadatu poziciju i poziciju iznad nje ako je drugi igrač upisuje se simbol O na zadatu poziciju i na poziciju desno od nje. Zatim se poziva funkcija `print_table()` za prikaz stanja igre.

```
def play_a_turn(self):
    while True:
        try:
            row = int(input("Unesite vrstu polja: "))
            column = input("Unesite kolonu polja [A-Z]: ")
        except ValueError:
            return False
        else:
            break
    if self.move_valid(row, column):
        m = ord(column) - 65
        n = N - row
        if self.players_turn is self.player1:
            self.matrix[n][m] = 'X'
            self.matrix[n - 1][m] = 'X'
            self.players_turn = self.player2
        else:
            self.matrix[n][m] = 'O'
            self.matrix[n][m + 1] = 'O'
            self.players_turn = self.player1
        self.print_table()
        return True
    else:
        return False
```

## Funkcija za proveru da li je potez valjan:

Pre unosa novog stanja prosleđujemo funkciji `move_valid()` stanje igre, poziciju novog unosa igrača na proveru poteza. Na osnovu tipa igrača proveravamo po definisanim pravilima da li je potez van tabele, na ivici tabele ili da li je zadata pozicija zauzeta i vraćamo bool vrednost u odnosu na to da li može da se odigra odgovarajući potez.

```
def move_valid(self, row, column):
    m = ord(column) - 65 # A -> 1
    n = self.N - row # inverted rows
    if self.players_turn is self.player1: # checking if vertical one
        can be placed
        if row < 0 or row >= N or m < 0 or m > M:
            return False
        if self.matrix[n][m] == '' and self.matrix[n - 1][m] == '':
            return True
        else:
            return False
    else: # checking horizontal one
        if row < 0 or row > N or m < 0 or m >= M - 1:
            return False
        if self.matrix[n][m] == '' and self.matrix[n][m + 1] == '':
            return True
        else:
            return False
```

## Funkcija za proveru kraj igre:

Na početku svakog poteza funkcija `is_game_over()` proverava da li je igra gotova. Njoj se prosleđuje stanje igre i na osnovu tipa igrača čiji je potez, proverava se stanje da li ima slobodnih vertikalnih ili horizontalnih poteza. Ukoliko ima slobodnih poteza funkcija vraća bool `False` i igra se nastavlja.

```
def is_game_over(self):
    if self.players_turn is self.player1: # any two empty vertical spaces?
        for i in range(0, self.N - 1):
            for j in range(0, self.M):
                if self.matrix[i][j] == '' and self.matrix[i + 1][j] == '':
                    return False
    else: # any two empty horizontal spaces?
        for i in range(0, self.N):
            for j in range(0, self.M - 1):
                if self.matrix[i][j] == '' and self.matrix[i][j + 1] == '':
                    return False
    return True
```

## II faza projekta:

U II fazi projekta treba da definišemo operator promena stanja igre. Funkcija koja na osnovu konkretnog poteza menja stanje igre (table) je definisana u I fazi projekta funkcijom za unos novog stanja pomoću funkcije *play\_a\_turn()*.

Unos početnih parametra igre realizujemo preko sledećeg koda gde unosimo broj vrsta i kolona tabli kao i da li zelimo da igramo protiv računara ili čoveka.

```
while True:
    try:
        N = int(input("Unesite broj vrsta table: "))
        M = int(input("Unesite broj kolona table: "))
    except ValueError:
        print("Nevalidan unos, pokušajte ponovo!")
        continue
    else:
        break
human_or_pc = bool(input("Igrac 1 je X...\nDa li je on covek ili racunar? (0-racunar, 1-covek): "))
```

Unos novog poteza sve dok on nije validan proveravamo ,nakon provere da li je završena igra , sledećim kodom gde u funkciji *play\_a\_turn()* pozivamo funkciju *move\_valid()* koji proverava da li je potez valjan.

```
placed_correctly: bool = game.play_a_turn()
while not placed_correctly:
    print("Nevalidan potez, pokušajte ponovo!")
    placed_correctly: bool = game.play_a_turn()
```

Ukoliko potez nije valjan posleđujemo bool *False* da je nevalidan potez i pozivamo funkciju *play\_a\_turn()* sve dok ne odigramo validan potez. Nakon sto odigramo validan potez stanje table se menja i šampamo novonastalo stanje table. U sledećem potezu pozivamo prvo funkciju za proveru kraja igre i ukoliko funkcija vrati bool *True* imamo kraj igre i šampamo ko je pobednik igre na osnovu čiji je potez. Ako nije kraj igre nastavljamo igru.

```
if game.is_game_over():
    print("#####\nKraj igre!")
    print("Pobedio je 2. igrac - O!") if game.players_turn.sign == "X" else print("Pobedio je 1. igrac - X!")
    print("#####")
    break
print("Igrac X je na potezu") if igrac1_na_potezu is True else print("Igrac O je na potezu")
```

## Funkcija koja formira novo stanje igre

U klasi *Game* dodali smo novi atribut *matrix\_states* koja će da pamti sva stanja igre, od početka do kraja igre. Realizovali smo funkciju *add\_new\_state()* koja pravi trenutnu kopiju stanja igre tj atributa *matrix* i dodaje u niz stanja *matrix\_states* koji se pamte do kraja igre. Funkciju *add\_new\_state()* pozivamo u okviru funkcije *play\_a\_turn()* nakon sto promenimo stanje igre.

```
def add_new_state(self):
    #Kopira trenutno stanje table i dodaje u listu stanja
    self.matrix_states.append(copy.deepcopy(self.matrix))
```

## Funkcija za formiranje svih mogućih stanja igre

Realizovali smo funkciju `find_all_available_states()` i dodali smo novi atribut `all_available_states`, tipa lista, klasi `Player`. Funkciju pozivamo u okviru funkcije `play_a_turn()` pre zahteva za unos željenog poteza. Funkcija `find_all_available_states()` pre svega prazni listu `all_available_states` ukoliko ima prethodno ubačena stanja nakon toga na osnovnu tipa igrača, parametra i stanja trenutne table proverava svako slobodno mesto i dodaje na to mesto odgovarajuću dominu i snima kopiju stanja table i ubacuje u listu `all_available_states`. Na završetku funkcije generisali smo svako moguće stanje igrača koju on može da odigra i sačuvali smo u njegovoj listi.

```
def find_all_available_states(self):
    self.players_turn.all_available_states.clear()
    if self.players_turn is self.player1: # any two empty vertical spaces?
        for i in range(0, self.N - 1):
            for j in range(0, self.M):
                if self.matrix[i][j] == '' and self.matrix[i + 1][j] == '':
                    self.matrix[i][j] = 'X'
                    self.matrix[i + 1][j] = 'X'
                    self.player1.all_available_states.append(copy.deepcopy(self.matrix))
                    self.matrix[i][j] = ''
                    self.matrix[i + 1][j] = ''
    else:
        for i in range(0, self.N):
            for j in range(0, self.M - 1):
                if self.matrix[i][j] == '' and self.matrix[i][j + 1] == '':
                    self.matrix[i][j] = 'O'
                    self.matrix[i][j + 1] = 'O'
                    self.player2.all_available_states.append(copy.deepcopy(self.matrix))
                    self.matrix[i][j] = ''
                    self.matrix[i][j + 1] = ''
```

Napisali smo i funkciju koja prikazuje sva moguća stanja koja su odigrana u igri ili sva moguća stanja koje igrač može da odigra u njegovom potezu poroslađivanjem odgovarajuće liste stanja.

```
def print_states(self, matrica):
    #Stampa sva dosadasnja stanja u terminalu
    for k in range(0, len(matrica)):
        if(matrica == self.matrix_states):
            print(f"State number {k}")
        else:
            print(f"Available move num. {k}")
    letter = 65 # A
    # vrh table
    print(" ", end=" ") # corner
    for i in range(0, M):
        print(f" {chr(letter + i)}", end=" ")
    print("")
    print(" ", end=" ")
    for i in range(0, M):
        print(" =", end=" ")
    print("")

    for i in range(0, N):
        print(f"{{N - i}}", end=" ")
        for j in range(a0, M): # counting backwards
            print(f"{{matrica[k][i][j]}} |", end=" ")
        print(f"{{N - i}}")
        print(" ", end=" ")
        for _ in range(0, M):
            print(" ---", end=" ")
        print(" ")
```

```

### Game Domineering ###
Unesite broj vrsta table: 4
Unesite broj kolona table: 4
Igrac 1 je X...
Da li je on covek ili racunar? (0-racunar, 1-covek): 1
Igrac 2 je O
  A  B  C  D
  =  =  =  =
4||  |  |  |  ||4
  ---
3||  |  |  |  ||3
  ---
2||  |  |  |  ||2
  ---
1||  |  |  |  ||1
  ---
Igrac X je na potezu

```

Primer 1: Početak Igre

```

Unesite vrstu polja: 1
Unesite kolonu polja [A-Z]: A
  A  B  C  D
  =  =  =  =
4||  |  |  |  ||4
  ---
3||  |  |  |  ||3
  ---
2|| X |  |  |  ||2
  ---
1|| X |  |  |  ||1
  ---

```

Primer 3: Unos novog stanja

```

Available move num. 0
  A  B  C  D
  =  =  =  =
4|| X |  |  |  ||4
  ---
3|| X |  |  |  ||3
  ---
2||  |  |  |  ||2
  ---
1||  |  |  |  ||1
  ---
Available move num. 1
  A  B  C  D
  =  =  =  =
4||  | X |  |  ||4
  ---
3||  | X |  |  ||3
  ---
2||  |  |  |  ||2
  ---
1||  |  |  |  ||1
  ---
Available move num. 2
  A  B  C  D
  =  =  =  =
4||  |  | X |  ||4
  ---
3||  |  | X |  ||3
  ---
2||  |  |  |  ||2
  ---
1||  |  |  |  ||1
  ---

```

Primer 2: Prikaz  
prvi 3 moguća  
poteza

```

Unesite vrstu polja: 3
Unesite kolonu polja [A-Z]: A
  A  B  C  D
  =  =  =  =
4|| X | O | O | X ||4
  ---
3|| X | X |  | X ||3
  ---
2|| X | X | O | O ||2
  ---
1|| X |  | O | O ||1
  ---

#####
Kraj igre!
Pobedio je 1. igrac - X!
#####

```

Primer 4: Kraj igre