Veštačka inteligencija - izveštaj II faza

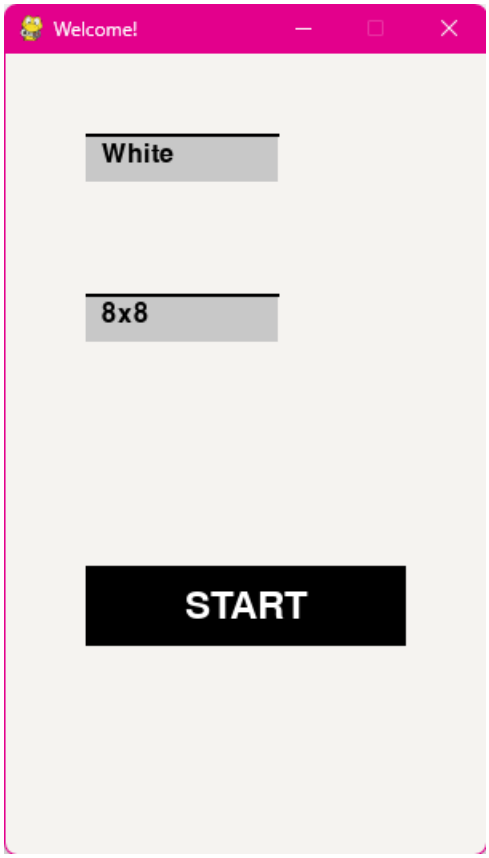
PinkTeam

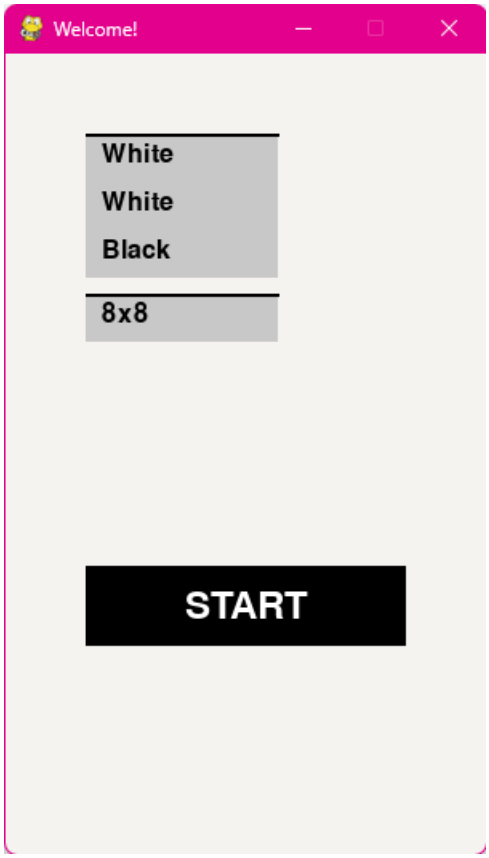
Mijajlović Anđelija 18247

Joksimović Kristina 18203

1. **Odigravanje partije između dva igrača naizmeničnim potezima:**

Na početku je prikazan prozor sa izborom boje, pri čemu uvek beli igrač igra prvi. Na osnovu toga se pokreće igra, a potezi su naizmenično dozvoljeni igračima odgovarajuće boje.





Na osnovu sledećeg uslova se brani/ dozvoljava potez igraču:

```
if(self.currentPlayer == 1 and bit == 0 or self.currentPlayer == 0 and bit == 1):
    return None
```

Sve dok igrač unosi netačne poteze ostaće njegov red da igra. Nakon svakog poteza se ažuriraju poeni (ukoliko je formiran stek) i proverava da li je kraj igre pomoću ove dve funkcije:

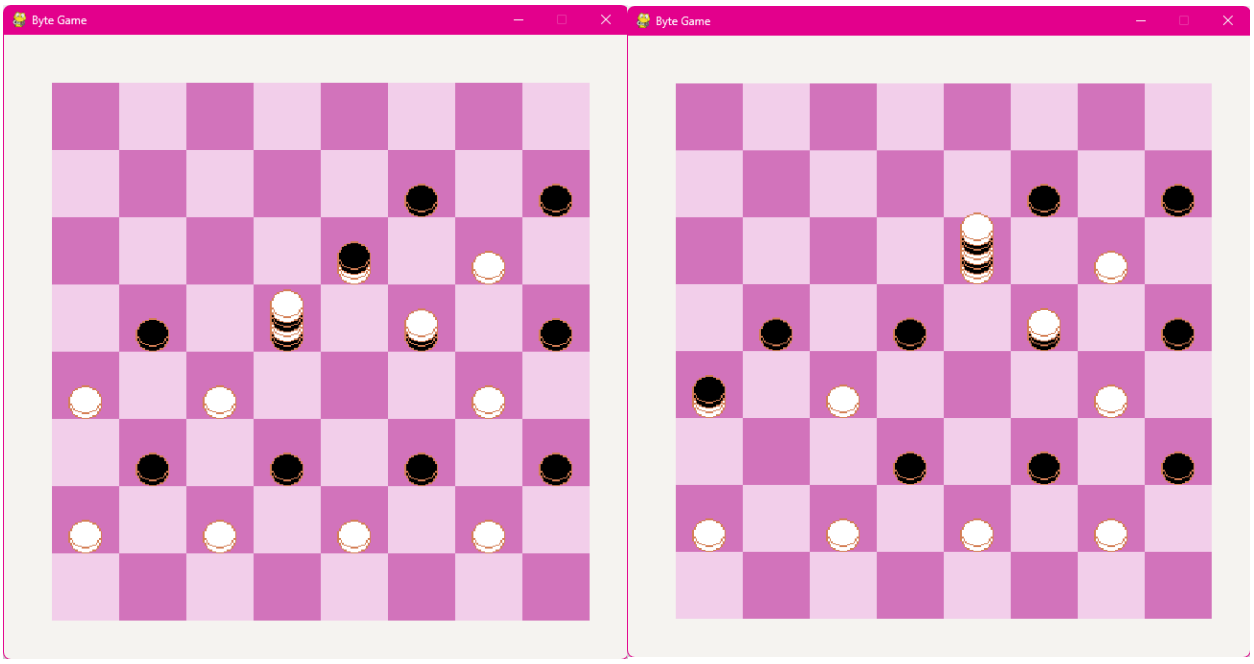
```
def updateScore(self, row, col):
    if(self.board[row][col][1] == 8):
        resColor = self.readBit(row, col, 7)
        for u in self.users:
            if(u.color == resColor):
                u.score += 1


def isOver(self):
    if(self.users[self.currentPlayer].score >= self.maxStacks/2):
        return True
    return False
```

2. Provera valjanosti poteza:

```
def valid_move(self, row1, col1, row2, col2, positionFrom, bit):
    if(row1 == row2 or col1 == col2):
        return None
    if(row1 < 0 or row1 >= self.dim or col1 < 0 or col1 >= self.dim):
        return None
    if(row2 < 0 or row2 >= self.dim or col2 < 0 or col2 >= self.dim):
        return None
    if(self.board[row1][col1][1] == 0):
        return None
    if(self.board[row2][col2][1] == 8):
        return None
    if(self.currentPlayer == 1 and bit == 0 or self.currentPlayer == 0 and bit == 1):
        return None
if(positionFrom < 0 or positionFrom >= self.board[row1][col1][1]):
    return None

if not self.stackRules(row1, col1, row2, col2, positionFrom):
    return None


    # Provera da li je potez dijagonalan
    diag = self.diagonal(row1, col1, row2, col2)
    return diag if diag is not None else None
```

```python
def stackRules(self, row1, col1, row2, col2, positionFrom):
    #broj ukupnih bitova na novom steku je manji od 8
    if(self.board[row2][col2][1] + self.board[row1][col1][1] - positionFrom > 8):
        return False

    #bitovi se pomeraju na visu ili jednaku poziciju
    #to ne vazi kad su sva polja okolo prazna
    if(positionFrom > self.board[row2][col2][1]):
        return False

    if(positionFrom == self.board[row2][col2][1]):
        if(self.areDiagonalEmpty(row1, col1)):
            #naci najblizi stek i proveriti da li je u pravcu
            #ako jeste, onda je dozvoljeno
            (nzrow, nzcol)= self.find_nearest_nonzero(row1, col1)
            print(nzrow,nzcol)
            if nzrow is not None and self.is_in_direction(row1, col1, nzrow, nzcol, (row2, col2)):
                return True

            return False

    return True
```

Po pravilima vezanim za stekove (deo steka se može pomeriti ukoliko se pomera na veću visinu):



3. Provera da li su sva susedna polja prazna

```
def areDiagonalEmpty(self, row, col):
    ll = 0
    if (row - 1 >= 0 and col - 1 >= 0 and row - 1 < len(self.board) and col - 1 < len(self.board[0])):
        ll = self.board[row - 1][col - 1][1]

    # Provera donje desne dijagonale
    lr = 0
    if (row - 1 >= 0 and col + 1 >= 0 and row - 1 < len(self.board) and col + 1 < len(self.board[0])):
        lr = self.board[row - 1][col + 1][1]

    # Provera gornje leve dijagonale
    ul = 0
    if (row + 1 >= 0 and col - 1 >= 0 and row + 1 < len(self.board) and col - 1 < len(self.board[0])):
        ul = self.board[row + 1][col - 1][1]

    # Provera gornje desne dijagonale
    ur = 0
    if (row + 1 >= 0 and col + 1 >= 0 and row + 1 < len(self.board) and col + 1 < len(self.board[0])):
        ur = self.board[row + 1][col + 1][1]

    # Provera da li su sva dijagonalna polja prazna
    if ll == 0 and lr == 0 and ul == 0 and ur == 0:
        return True

    return False
```

4. **Funkcija koja na osnovu poteza i trenutnog stanja igre menja stanje igre**

Movement sadrži koordinate klika na početno polje i koordinate klika odredišnog polja.

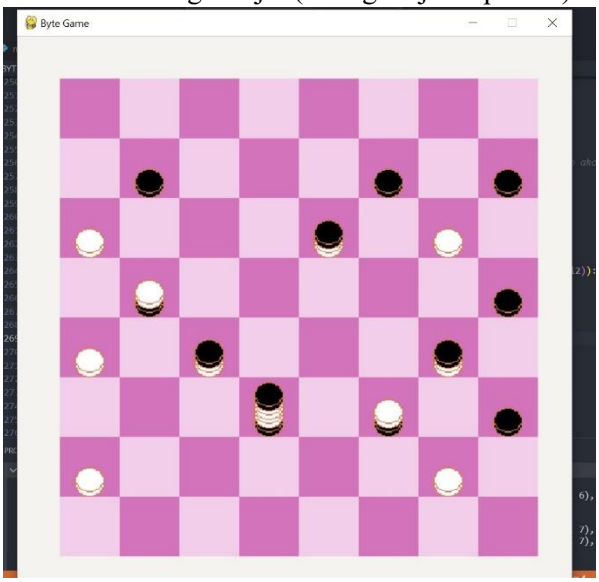
```
def move(self, movement):

    x1, y1 = self.get_field_start(movement[0], movement[1])
    x2, y2 = self.get_field_start(movement[2], movement[3])

    row1 = int(y1 / self.squareSize)
    col1 = int(x1 / self.squareSize)
    row2 = int(y2 / self.squareSize)
    col2 = int(x2 / self.squareSize)

    ########
    clicked_bit = int(((row1 + 1) * self.squareSize) - y1 ) / self.bitHeight

    positionFrom = 0
    if(clicked_bit < 0):
        return None
    if(clicked_bit > self.board[row1][col1][1]):
        positionFrom = self.board[row1][col1][1] - 1
    else:
        positionFrom = int(clicked_bit)

    #citanje
    bits = []

    numOfBits = self.board[row1][col1][1] - positionFrom
    for i in range(numOfBits):
        bits.append(self.readBit(row1, col1, positionFrom + i))

    isValid = self.valid_move(row1, col1, row2, col2, positionFrom, bits[0])
    if( isValid == None):
        return

    #brisanje
    self.writeBits(row1, col1, [0 for _ in range(numOfBits)], numOfBits, True)

    #upis
    self.writeBits(row2, col2, bits, numOfBits, False)

    self.currentPlayer = 0 if self.currentPlayer == 1 else 1

    #provera da li je neki stek popunjen
    self.updateScore(row2, col2)


    #provera da li je gotova igra
    if(self.isOver):
        return True
```

Nakon ove funkcije je stanje problema izmenjeno i ponovo se iscrtava tabla, izračunati su poeni i ukoliko je došlo do kraja igre to je ispisano na ekranu.

5. **Određivanje svih mogućih poteza igrača.**

```
def calculate_all_possible_moves(self):
    possible_moves = []
    for row in range(self.dim):
        for col in range(self.dim):
            # proverava da li stek pripada trenutnom igracu
            if self.board[row][col][1] > 0 and self.readBit(row, col, 0) == self.currentPlayer:
                # dijagonalni susedi
                for dr, dc in [(-1, -1), (-1, 1), (1, -1), (1, 1)]:
                    new_row, new_col = row + dr, col + dc

                    if 0 <= new_row < self.dim and 0 <= new_col < self.dim:
                        if self.valid_move(row, col, new_row, new_col, 0, self.currentPlayer):
                            possible_moves.append((row, col, new_row, new_col))
    return possible_moves
```

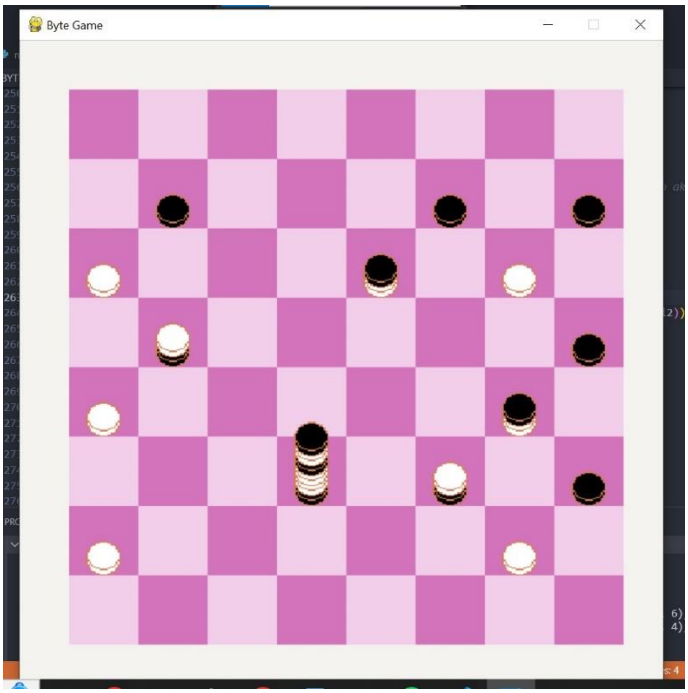
Na osnovu datog stanja: (beli igrač je na potezu)



Mogući su sledeći potezi:

```
[(2, 0, 1, 1), (2, 0, 3, 1), (2, 4, 1, 3), (2, 4, 1, 5), (2, 4, 3, 3), (2, 4, 3, 5), (2, 6, 1, 5), (2, 6, 1, 7), (2, 6, 3, 5), (2, 6, 3, 7),
 (4, 0, 3, 1), (4, 0, 5, 1), (4, 2, 3, 1), (4, 2, 3, 3), (4, 2, 5, 1), (4, 2, 5, 3), (4, 6, 3, 5), (4, 6, 3, 7), (4, 6, 5, 5), (4, 6, 5, 7),
 (6, 0, 5, 1), (6, 6, 5, 5), (6, 6, 5, 7), (6, 6, 7, 5), (6, 6, 7, 7)]
```

Nakon pomeranja figure:




```
[(1, 1, 0, 0), (1, 1, 0, 2), (1, 1, 2, 0), (1, 1, 2, 2), (1, 5, 0, 4), (1, 5, 0, 6), (1, 5, 2, 4), (1, 5, 2, 6), (1, 7, 0, 6), (1, 7, 2, 6),
 (3, 1, 2, 0), (3, 1, 2, 2), (3, 1, 4, 0), (3, 1, 4, 2), (3, 7, 2, 6), (3, 7, 4, 6), (5, 3, 4, 2), (5, 3, 6, 4), (5, 5, 4, 4), (5, 5, 4, 6),
 (5, 5, 6, 4), (5, 5, 6, 6), (5, 7, 4, 6), (5, 7, 6, 6)]
```

6. **Provera da li potez vodi ka jednom od najbližih figura.**

Pomoću modifikovanog BFS algoritma.

```
def find_nearest_nonzero(self, start_row, start_col):
    directions = [(-1, -1), (-1, 1), (1, -1), (1, 1)]
    visited = [[False for _ in range(self.dim)] for _ in range(self.dim)]
    queue = deque([(start_row + dr, start_col + dc)
                   for dr, dc in directions
                   if 0 < start_row + dr < len(self.board) and 0 < start_col + dc < len(self.board[0])
                   ])

    while queue:
        current_row, current_col= queue.popleft()
        visited[current_row][current_col] = True

        if self.board[current_row][current_col][1] > 0 and current_col != start_col or current_row != start_row:
            return (current_row, current_col)

        for dr, dc in directions:
            new_row, new_col = current_row + dr, current_col + dc
            if (0 <= new_row < self.dim and 0 <= new_col < self.dim and not visited[new_row][new_col]
                and (new_row != start_row or new_col != start_col)):
                queue.append((new_row, new_col))

    return (None, None)
```