Veštačka Inteligenčija

Izveštaj II faze projekta

Slaganje (Byte)

Naziv tima : DavidDusanVeljko
Dušan Stojanovic 17450
Veljko Marković 17745
David Stanisavljević 17943

Provera valjanosti poteza

```
    def is_valid_move(self, start_row, start_col, stack_pos, direction):
        row_index = ord(start_row) - ord('A')
        col_index = start_col - 1
        start_field = self.board.fields[row_index][col_index]

        if not (0 <= row_index < self.board.num_of_fields and 0 <= col_index <
self.board.num_of_fields):
            return False, "Move is outside the board boundaries."

        if (row_index + col_index) % 2 != 0:
            return False, "Can only move on dark squares."

        start_field = self.board.fields[row_index][col_index]
        if len(start_field.stack) == 0:
            return False, "No stack to move."

        if start_field.stack[stack_pos] != self.current_player.checker_color:
            return False, "You do not own the checker you want to move."

        dir_offsets = {"GL": (-1, -1), "GD": (-1, 1), "DL": (1, -1), "DD": (1, 1)}
        if direction not in dir_offsets:
            return False, "Invalid direction."

        delta_row, delta_col = dir_offsets[direction]
        target_row_index = row_index + delta_row
        target_col_index = col_index + delta_col
        target_field = self.bounds_check_and_get_field(target_row_index,target_col_index)

        if not target_field:
            return False, "Out of bounds"

        if not (0 <= target_row_index < self.board.num_of_fields and 0 <= target_col_index <
self.board.num_of_fields):
            return False, "Target position is outside the board boundaries."

        if (target_row_index + target_col_index) % 2 != 0:
            return False, "Can only move to dark squares."

        if len(target_field.stack) > 0:
            if len(start_field.stack) - stack_pos + len(target_field.stack) >= 9:
                return False, "Cannot form a stack of nine or more."

        if not target_field.is_empty():
            if len(target_field.stack) > 0:
                if (
                    len(start_field.stack) > stack_pos
                ):
                    if stack_pos > 0 and target_field.stack[-1] >= start_field.stack[stack_pos]:
                        return False, "Invalid move: The moving checker must be on top of the stack."
                    num_checkers_to_move = len(start_field.stack[stack_pos:])

                    resulting_stack_size = len(target_field.stack) + num_checkers_to_move

                    if resulting_stack_size <= 8:
                        return True, "Valid move."
                    else:
                        return False, "Cannot move the stack as it exceeds the maximum size."
                else:
                    return False, "Index out of bounds."
        else:
            possible_moves = {"GL": (-1, -1), "GD": (-1, 1), "DL": (1, -1), "DD": (1, 1)}

            temp=possible_moves.pop(direction)

            for _, (move_row, move_col) in possible_moves.items():
                new_row, new_col = row_index + move_row, col_index + move_col

                if 0 <= new_row < self.board.num_of_fields and 0 <= new_col < self.board.num_of_fields:
                    move_field = self.board.fields[new_row][new_col]
                    if not move_field.is_empty():
                        return False, "There is a non-empty, adjacent field."
                else:
                    return False, "Move is outside the board boundaries."

            possible_moves[direction]=temp

            all_paths=self.bfs(row_index,col_index,possible_moves)

            shortest_paths=sorted(all_paths,key=lambda x:len(x))

            shortest_path=shortest_paths[0]

            paths = [path for path in shortest_paths if len(path) == len(shortest_path)]

            move_direction=possible_moves[direction]

            for path in paths:
                new_row,new_col,_=path[1]
                test_row=row_index+move_direction[0]
                test_col=col_index+move_direction[1]
                if (test_row,test_col)==(new_row,new_col):
                    return True, "Valid move."

            return False, "There is no valid move."
```

BFS iz prethodne funkcije

```
    def bfs(self, start_row, start_col, directions):
        queue = Queue()
        visited = set()
        paths = []

        queue.put([(start_row, start_col, 0)])

        while not queue.empty():
            path = queue.get()
            curr_row, curr_col, distance = path[-1]

            if not self.board.fields[curr_row][curr_col].is_empty() and distance > 0:
                paths.append(path)
                continue

            if (curr_row, curr_col) not in visited:
                visited.add((curr_row, curr_col))

                for _, (dr, dc) in directions.items():
                    new_row, new_col = curr_row + dr, curr_col + dc

                    if 0 <= new_row < self.board.num_of_fields and 0 <= new_col <
self.board.num_of_fields:
                        queue.put(path + [(new_row, new_col, distance + 1)])

        valid_paths = [path for path in paths if not self.board.fields[path[-1][0]][path[-1]
[1]].is_empty()]

        return valid_paths
```

Funkcije za promenu stanja

```
    def move(self, row, col, stack_pos, direction):
        row_index = ord(row) - ord('A')
        col_index = col - 1
        start_field = self.board.fields[row_index][col_index]

        stack_to_move = start_field.stack[stack_pos:]
        start_field.stack = start_field.stack[:stack_pos]

        target_row_index, target_col_index = self.calculate_target_position(row_index, col_index,
direction)

        target_field = self.board.fields[target_row_index][target_col_index]

        exceeded,last_checker=target_field.add_checker(stack_to_move)
        if exceeded:
            return True, last_checker
        return False,None

    def execute_move(self,move):
        row, col, stack_pos, direction = move
        valid_move,message= self.is_valid_move(row, col, stack_pos, direction)
        if valid_move:
            exceeded, last_checker=self.move(row, col, stack_pos, direction)
            if exceeded:
                self.add_point(last_checker)
        else:
            print(message)
        return valid_move
```

Provera da li je gotova igra

```
def is_over(self):
    if self.board.empty():
        return False
    if not self.won():
        return False
    return True

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

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

Funkcija koja obezbedjuje igranje

```
def main():
    game = Game()
    game.start()

    while not game.is_over():
        print(game)
        move=game.get_move()
        game.execute_move(move)
        game.switch_player()
    print(game)
    if game.won():
        print(f"{game.winner} has won!")
```

Funkcije za generisanje poteza I za kopiranje trenutnog stanja(radi izvrsavanja poteza nad novim stanjem)

```python
def generate_moves_from_field(self, row, col):
        moves_from_field = []

        field = self.bounds_check_and_get_field(row, col)
        if not field or field.is_empty():
            return moves_from_field

        for stack_pos in range(len(field.stack)):
            if field.stack[stack_pos] == self.current_player.checker_color:
                for direction in ["GL", "GD", "DL", "DD"]:
                    valid_move, _ = self.is_valid_move(chr(65 + row), col + 1, stack_pos, direction)
                    if valid_move:
                        moves_from_field.append((chr(65 + row), col + 1, stack_pos, direction))

        return moves_from_field

def generate_all_moves(self):
    all_moves = []

    for row in range(self.board.num_of_fields):
        for col in range(self.board.num_of_fields):
            if self.board.fields[row][col].field_type==field_black:
                field=self.bounds_check_and_get_field(row,col)
                if not field:
                   continue
                if not field.is_empty() and self.current_player.checker_color in field.stack:
                    moves_from_field = self.generate_moves_from_field(row, col)
                    all_moves.extend(moves_from_field)
    return all_moves

def copy_game(game):
    new_game = Game()
    new_game.board = copy.deepcopy(game.board)
    new_game.current_player = copy.deepcopy(game.current_player)
    new_game.player1 = copy.deepcopy(game.player1)
    new_game.player2= copy.deepcopy(game.player2)
    new_game.winner = game.winner
    return new_game
```