# README for DIFFERO Game Implementation in Prolog

## Group Information
- **Group Designation:** Differo_1
- **Student Number (Contributions):**
  - Maria Eduarda Sousa Rabelo up202000130 (50%)
  - Vitor Alves Moreira up201900198 (50%)

## Installation and Execution
Other than SICStus Prolog 4.8 , no other software is required to execute the game. To run the program, you should follow these steps:

1. Open Sicstus.
2. Consult main.pl, located on the src folder.
3. Call the play predicate without any arguments.

## Description of the game
DIFFERO is a two-player board game played on a hexagonal board with 5 spaces on each side, and each player has 13 pieces. You win when one of your pieces reaches the goal or when the opponent doesn't have any more moves allowed.

Rules of DIFFERO:
1. The piece moves in a diagonal line.
2. The piece can jump over any number of pieces at once, whether they are yours or your opponent's.
3. You can take (number of your pieces) - (number of opponent's pieces) steps on the diagonal to be moved.
4. If this value is less than or equal to 0, the piece cannot move on the diagonal.
5. You cannot move the piece off the board or into a place already occupied by another piece.
6. You cannot move the piece into the opponent's goal.

Information was collected from https://boardgamegeek.com/boardgame/375056/differo.

## Game Logic
The DIFFERO game in Prolog has been implemented with the following key components and features:

### Internal Game State Representation

To keep track of the game's internal state, we make use of a GameState data structure that is composed by the following structure:

- Board: A list of lists, where each element of the main list represents a row of the board, and each element of a inner list represents a column of that row, containing atoms 'none,' 'white,' and 'black' for empty spaces, white pieces, and black pieces, respectively.
- Player: The current player (white or black).

The GameState data structure is initialized with the initial board and white player:

```
% Initialize the initial game board with a size of 5 on each side.
initial_board([
    [none, none, none, none, none],
    [none, black, black, black, black, none],
    [none, black, none, black, none, black, none],
    [none, black, black, black, black, black, black, none],
    [none, none, none, none, none, none, none, none, none],
    [none, white, white, white, white, white, white, none],
    [none,  white, none, white, none, white, none],
    [none, white, white, white, white, none],
    [none, none, none, none, none]
]).

% Initial game state
initial_state(Board, white) :- initial_board(Board).
```

### Game Loop

The `game_loop/2` predicate handles the main gameplay loop, allowing players to make moves and checks for game over conditions.

### Game State Visualization

The game has a hexagonal board that is printed before every player move.

- The `print_board/1` predicate displays the game board.
- The `display_rules/0` predicate displays the game rules.

User move is gotten with the get_move/2 predicate.

### Move Validation and Execution
- The `valid_move/2` checks if a move respects all rules listed below:
    - Piece in the original position does belong to current player.
    - Within bounds of the board.
    - Moving along a diagonal path.
    - Following the steps allowed based on the count of pieces on the diagonal path.
    - Not moving onto an already occupied space.
    - Not moving to your opponent's goal.

- The `move/3` predicate executes valid moves to produce a new game state.
- The `choose_move/4` predicate gets the move chosen by the player or by the bot.
- The `get_move` predicate reads the input from the user.

### List of Valid Moves
- The `valid_moves/3` predicate computes all valid moves for the current player.
- The `find_all_valid_moves_for_positions/4` predicate computes all valid moves from a position

### End of Game
- `game_over/2` identifies the game's end and the winner.

In order to verify that, we check if any piece of the current player reached its goal and if any player is out of moves.

### Computer Plays
- `choose_move/4` allows the computer to make random moves.
- `seed_random/0` seed the random number generator.

## Conclusions
This DIFFERO Prolog implementation offers a functional game with core features. Known issues include the different format of board that brought complexities to the game logic. Future improvements may involve including levels of difficulty to the bots.

## Bibliography
- https://boardgamegeek.com/boardgame/375056/differo
- https://sicstus.sics.se