1 Ploy

Ploy is a chess-like board game in which each player has a set of pieces with different ranges of movement that can be used to capture the opponent's pieces. The goal is to capture either the commander or all of your opponent's pieces. The game is available for two or four players, although only the two-player version is part of the task.

2 Regulations

The game is played on a two-dimensional board of 9x9 squares - black starts. At the start of the game, each player has 15 pieces, with lines indicating the direction in which they can move. On each move, one piece can be moved and/or rotated, with each piece having different abilities. There are the following pieces:

- **Shield** Has one line, so one way to move and can go ≤ 1 square and/or one turn (afterwards).
- **Probe** Has two lines, so two ways to move, and can either move up to 2 squares or turn.
- **Lance** Has three lines, so three ways to move, and can either move up to 3 squares or turn.
- **Commander** Has 4 lines, so four ways to move, and can either move ≤ 1 square or turn.

Each move must be a movement and/or rotation that causes a change of state. A piece can be moved horizontally, vertically and diagonally along the directional lines shown on the piece. Pieces cannot be jumped over. The target square can either be empty or occupied by an opposing piece, in which case the opposing piece is removed from the board. The game ends when one of the two players either has no Commander left or only the Commander. The player who made the last move wins.

3 Notation

To describe the game situation and the moves, we use a modified FEN notation (known from chess). This means that the columns are labelled with small letters from a to i and the rows with numbers from 1 to 9. To

uniquely identify a square, the column is always given first, followed by the row. For example, a1 stands for the lower left corner and i9 for the upper right corner. In the beginning, the black pieces are on the 'lower' squares in rows 1-3, while the white pieces are on the 'upper' squares in rows 7-9.

4 Game board

The board is described by a string structured as follows: Rows are separated by "/" and squares within rows are separated by ",". There are no "/" characters at the beginning and end of the string, and there are no "," characters at the beginning and end of each row. The squares are listed from a9 to i1. Within each square, a piece can be encoded as follows: The first character represents the colour, "b" for black and "w" for white. The following positions represent a decimal number between 1 and 255. Viewed as an unsigned 8-bit binary number, each bit corresponds to a direction in which the piece can move (represented as lines in the GUI). The least significant bit (LSB) represents upwards (or north, or along a column towards the row with the higher number), and the higher bits represent the other directions, clockwise from north. If there is no figure on the board, the space in the string remains empty. So the board at the start is described as follows:

,w84,w41,w56,w170,w56,w41,w84,/ , ,w24,w40,w17,w40,w48, ,/ , , , w16 , w16 , w16 , , , / , , , , , , , , / , , , , , , , , / , , , , , , , , / , , , b1 , b1 , b1 , , , / , , b3 , b130 , b17 , b130 , b129 , , / , b69 , b146 , b131 , b170 , b131 , b146 , b69 ,

5 Moves

Moves are described by a string in the format <start>-<destination>-<rotation>, where <rotation> is the number of clockwise rotation steps. A rotation that causes a line to point southeast instead of north would be described by <rotation>=3. A counterclockwise step to the northwest is described by <rotation>=7. Rotations of more than seven steps are not allowed. Therefore, a move always consists of seven characters, based on the available options.

6 Task Haskell WiSe 2022/2023

To implement the Ploy bot, you have 7 interface specifications provided in the form of function signatures. You should not modify the supplied code except for module imports, implementation of interfaces, and instantiations of the Show type class instead of the supplied derived Show. You may only import modules and functions from the base and split packages. You should not modify the stack configuration files. Network access and embedding of other programming languages is not allowed.

6.1 validateFEN

The function (validateFEN :: String − > Bool) takes a string and validates it according to the FEN format described for Ploy. The function returns True only if the input is a valid FEN string. The validity of the game situation does not need to be checked.

6.2 buildBoard

The function (buildBoard :: String − > Board) takes a valid FEN string and creates a board of the given type Board = [[Cell]] from it. The element of the state (board!!0)!!0 corresponds to the upper left corner (cell a9) and the element (board!!0)!!8 corresponds to the upper right corner (cell i9).
Note: The !! operator is used to access the index of a list.

6.3 line

The function (line :: Pos − > Pos − > [Pos]) takes two arguments (1: start position, 2: end position). It returns a list of positions representing the line from the starting cell to the target cell. Positions of the specified Pos type correspond to the identifiers on the board (a1 to i9). The return value is sorted and contains the start position as the first element and the destination position as the last element. It is assumed that the given positions are on the board and that the start and end positions are on a horizontal, vertical or diagonal line.

6.4 gameFinished

The function (gameFinished :: Board − > Bool) checks if the game is finished in the given state. The state (board) corresponds to a valid game situation. The function returns True only if no more moves are allowed.

6.5 isValidMove

The function (isValidMove :: Board − > Move − > Bool) takes a game state (board) and a move of the specified move type. The function returns True if and only if the move can be made in the current state. It can be assumed that a valid state is provided. The move contains a possible movement (rotation and steps) of the piece from the starting square of the state.

6.6 possibleMoves

The function (possibleMoves :: Pos − > Cell − > [Move]) returns all possible moves for the given cell at the given position. No explicit game state is taken into account. The parameters Pos and Cell are valid values. Moves must result in a change of the game state and may not occur more than once. Moves with different rotations but the same state change (e.g. Commander rotates 1 or 3 steps) are counted as different moves.

6.7 listMoves

This function (listMoves :: Board − > Player − > [Move]) returns all possible moves that can be made by a player in a valid game state. As with possibleMoves, moves must result in a change of state and may not occur more than once. Moves with different rotations but the same state change (e.g. rotate 1 or 3 steps) are counted as different moves.