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 Java

The task now is to develop a data model for the specific game Ploy within the already started class PloyGame, which can describe the state of the game. Using this data model you should implement the function ‘tryMove(String moveString, Player player)‘, which checks a move and executes it if necessary. Specifically, the following needs to be done:

- Check whether a user could attack or cheat the system by manipulating requests to the server. Check that the move input is in the format mentioned above. In addition, it should be checked whether the specified player is the one who is currently on the move.
- It should be checked if the move is valid according to the ploy rules given in this task.
- If the move is valid, it must be executed, i.e. the change should be reflected in the internal representation of the board so that the changed state of the game can be restored. It is then the other player's turn. If the game is won by one player, this information should be recorded in the attributes of the Game class. In addition, the game history must be maintained (as defined in the Game superclass).

Normally a new PloyGame starts with the initial state (according to the Ploy rules). In addition, two functions need to be implemented: one to set the game state (setBoard(String)) and another to get it (getBoard()). setBoard(), in conjunction with the existing setNextPlayer() function, allows us to start the game at any point in our and your test cases, making it possible to compactly test individual situations. As a test function, it does not need to validate inputs; all our inputs can occur in the game.

All the code you insert should be tested with JUnit tests. The aim is to achieve 100% branch coverage (at bytecode level). However, the object of the test is the specification. If functionality is missing, the test cases will not be complete. Only successfully executed test cases are counted in the test coverage measurement. Once the functionality and test cases have been implemented, appropriate refactoring may be required. The code you develop must meet the following metrics:

- Method Lines of Code : max. value 25

- McCabe cyclomatic complexity : max. value 10
- Nested Block Depth : max. value 4
- Number of parameters per function : max. value 5

6.1 Specification

We have provided an Eclipse Servlet project as a template, where all the client functionality (HTML files with JavaScript) and the Servlet are already included. The project also contains a README file that provides some information for orientation. In the existing classes, TODO comments mark the places where you should continue (functionality is missing in the PloyGame class). Of course, there are also classes that are missing. New classes can be created. Existing code should not be modified (unless explicitly stated).

- **startGame** The startGame function ensures that the game is started, making it easier to test with any state of the game (pieces on the board in the FEN string) and the selection of who should take their turn.
- **assertMove** The assertMove function provides the game with a move and the executing player, then checks if the result matches your expected result (tryMove returns true if the move was valid and executed).
- **assertGameState** The assertGameState function checks whether the overall state of the game (current board in FEN format, next player, game over or not, draw, who won) matches what you provide as an expectation.

7 Task Haskell WiSe 2019/2020

Your task is to complete the getMove and listMoves functions. Both functions take as arguments a game situation encoded as a string, consisting of the board (see notation) and the next player ('b' for black and 'w' for white). You can assume that you will only receive valid strings that actually encode possible game situations.

7.1 listMoves

The listMoves function should calculate a list of all possible moves in the given game situation and output them as a string according to the notation. You only get full points if all rule-compliant moves are correctly calculated in all tested game situations. Moves should not be repeated. Moves that produce the same change despite different rotation steps (e.g. Commander rotates 1 or 3 steps) count as different moves and must be listed in full.

7.2 getMove

The getMove function selects a move from the possible moves and returns only that move. The selection is based on your own strategy. This function is for implementing an executable bot for manual testing with the Java implementation and for the tournament.