Documentation Mitzi - Exercise 1

Christoph Hofer 0955139

Stefan Lew ???

19. November 2013

1 Implementation of chess pieces

The different chess pieces and the two are represented as an `enum` in `Piece.java` and `Side.java`. The class `Side` additionally provides methods for getting the opposite side and the sign of the color (+ for white and - for black), which is particularly important board evaluation and negamax algorithm.

2 Implementation of a simple chess board

The chess board is implemented via the class `GameState`. The class stores the position of the pieces, counts the `full_move_clock`, `halfe_move_clock` and saves the history of all played moves. A move can be performed via the method `doMove()` which additionally checks if the move is valid or not.

2.1 Implementation of the position class

The class `Position` contains the main information of the chess board. The class contains among others:

- `side_board`: An array of 65 `Sides`, representing the color (side) if the several pieces.
- `piece_board`: An array of 65 `Pieces`, representing the piece on a the different squares.
- `castling`: An array, which contains the square where the king can castle. It contains -1, if it is not possible.
- `en_passant_target`: The square, where the en-passant target is positioned.
- `active_color`: The side, which has to move.
- `analysis_result`: This class stores information of the value of the position and is of no interest for this exercise.

The arrays contains `null` if at a square is no pieces. The additional entry is reserved for illegal squares and is always set to `null`. Furthermore the class stores data, which is computed once and reuses it.

The class is able to:

- read and set Pieces on the board.
- reset the board to initial state.
- compute an copy of the board, where only the necessary members are copied.

- compute all valid moves for the active side and for each square.
- perform a given move
- check if a move is valid
- check if a move is a hit
- check if castling for a side is possible
- check if the current position is a check, mate or stale mate position.
- return a string representation of the position (FEN notation, seehttp://en.wikipedia.org/wiki/Forsyth-Edwards_Notation)

2.2 Representation of squares

We represent the squares as integer, however we do not use the usual notation 1,2,3, . . . , but we use the so called ICCF numeric notation (see http://en.wikipedia.org/wiki/ICCF_numeric_notation). The class `SquareHelper` provides methods to work with the notation:

- conversion: `int` ↔ `[row][column]`
- check if a square is black or white.
- check if a square is valid.
- conversation to string representation.
- receiving squares in a given `Direction`.

2.3 The enum Direction

The enum `Direction` contains the offset for the integer value of the square for each direction. Since the knight does not use the usual directions, it needs a separate offset. To simplify the code, an additional function was generated to return the capturing direction for a pawn for a certain side.

3 Implementation of chess moves

The class `Move` implements a move in a chess game. A move consists of

- the source square
- the destination square

- a `Piece` representing the promotion of the pawn.

4 The random chess player

The random chess player implemented in `RandyBrain` uses the function `search` to choose randomly a possible move.