P. Thiemann, L. Fennell Sommersemester 2017

Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2017/

simple-chess

Vereinfachtes Schach

Woche 12 Aufgabe 1/3

Herausgabe: 2017-07-20
Abgabe: 2017-08-08

Achtung: beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.

Project `simple-chess`
Package `simplechess`
Klassen

Main
`public static boolean winnable(IBoard board, int turns)`

IBoard
`boolean hasQueen(Player player)`
`List<IBoard> nextBoards(Player turn)`

enum Player
`WHITE,`
`BLACK;`
`public Player next()`

enum PieceKind
`QUEEN,`
`KNIGHT,`
`BISHOP,`
`ROOK`

Position
`public Position(int row, int column)`
`public int getRow()`
`public int getColumn()`

PositionPiece
`public PositionPiece(PieceKind pieceKind, Position position)`
`public PieceKind getPieceKind()`
`public Position getPosition()`

In dieser Aufgabe sollen Sie „Simple-Chess", eine vereinfachte Version von Schach, modellieren. Die wesentlichen Spielregeln von Simple-Chess sind die folgenden:

- Es wird auf einem quadratischen Schachbrett der Seitenlänge $n \geq 4$ gespielt. Wie gewohnt gibt zwei Spieler, *Schwarz* und *Weiß*.
- Die Spielfiguren sind die Königin (*Queen*), der Turm (*Rook*), der Läufer (*Bishop*) und der Springer (*Knight*). Es gibt keine Könige oder Bauern bei Simple-Chess.
- Die Bewegungsmöglichkeiten der Figuren sind wie bei gewöhnlichem Schach. Siehe dazu auch:

  `https://en.wikipedia.org/wiki/Chess#Movement`

- Jeder Spieler hat genau eine Königin. Weiß beginnt das Spiel. Der Spieler, dessen Königin zuerst geschlagen wird, hat verloren.

Im Skelett dieser Aufgabe finden Sie die Funktion `Main.winnable`, die (auf naive Weise) bestimmt, ob ein gegebenes Spiel in einer bestimmten Anzahl von Schritten für Weiß zu gewinnen ist, egal wie sich Schwarz verhält. Sie stützt sich dabei auf eine Implementierung des Interfaces `IBoard`

```
1  package simplechess;
2
3  import java.util.List;
4
5  /**
6   * A simple-chess board.
7   */
8  public interface IBoard {
9
10     /**
11      * Returns true iff the board has a queen for player "player".
12      */
13     boolean hasQueen(Player player);
14
15     /**
16      * Returns the list of boards for the possible moves that Player "turn" can make.
17      * This IBoard is not modified by "nextBoards".
18      */
19     List<IBoard> nextBoards(Player turn);
20 }
```

Das Enum `Player` besteht dabei aus den Werten `Player.WHITE` und `Player.BLACK`. **Ihre Aufgabe ist es nun,** das Interface `IBoard` mit einer Klasse `Board` zu implementieren. Außerdem sollten Sie eine Fabrikmethode `Boards.fromPositionPieces` implementieren, die ein `IBoard` aus einer Liste von `PositionPieces` erstellt.

Beispieltests

```
1  package simplechess;
2
```

```
3  import org.junit.Test;
4
5  import java.util.Arrays;
6  import java.util.Collections;
7  import java.util.List;
8
9  import static org.junit.Assert.*;
10
11 public class ExampleTests {
12
13
14     @Test
15     public void test1() {
16         List<PositionPiece> whitePieces = Arrays.asList(
17                 new PositionPiece(PieceKind.KNIGHT,
18                                   new Position(1, 1)),
19                 new PositionPiece(PieceKind.QUEEN,
20                                   new Position(0, 1)));
21         List<PositionPiece> blackPieces = Collections.singletonList(
22                 new PositionPiece(PieceKind.QUEEN, new Position(3, 0))
23         );
24
25         assertTrue(Main.winnable(Boards.fromPositionPieces(whitePieces,
26                                                            blackPieces, 4)
27                 , 1));
28     }
29
30     @Test
31     public void test2() {
32         List<PositionPiece> whitePieces = Arrays.asList(
33                 new PositionPiece(PieceKind.QUEEN,
34                                   new Position(0, 3)));
35         List<PositionPiece> blackPieces = Collections.singletonList(
36                 new PositionPiece(PieceKind.QUEEN, new Position(3, 0))
37         );
38
39         assertTrue(Main.winnable(Boards.fromPositionPieces(whitePieces,
40                                                            blackPieces, 4)
41                 , 1));
42     }
43
44 }
```