Übungsblatt 10

Musterlösung

Aufgabe 1 Lässig Level laden (100 %)

Die Klasse *Level* implementiert etwas mehr als gefordert war. Sie erzeugt *Walker*-Instanzen mit drei verschiedenen Figuren und erzeugt neben dem Ziel-Symbol auch noch Brücken in zwei Ausrichtungen. Zudem akzeptiert sie ein 'W' (Wasser), um einen Bach zu erzeugen. Diese Funktionalität wurde allerdings der Klasse *Field* hinzugefügt, die dann einfach die Zeichenkette "path" durch "water" in Dateinamen ersetzt (bei der Bonusaufgabe abgedruckt). Damit lässt sich das ursprüngliche Spielfeld wieder erzeugen.

```
1  import java.io.BufferedReader;
2  import java.io.InputStream;
3  import java.io.InputStreamReader;
4  import java.io.FileNotFoundException;
5  import java.io.IOException;
6  import java.io.Reader;
7  import java.util.ArrayList;
8  import java.util.List;
9
10 /**
11  * Diese Klasse definiert den Level eines Spiels. Ein Level wird
12  * aus einer Beschreibung erzeugt, die in einer Datei gespeichert ist.
13  * Die Datei enthält Symbole für leere Zellen (' '), freie Gitterzellen
14  * ('O'), Wasserzellen ('W'), Zellverbindungen ('|', '-'), die Spielfigur
15  * ('p', 'P', 'q', 'Q' für die Rotationen 0-3), drei Sorten
16  * Spaziergänger:innen ('l', 'L', 'i', 'I' für Laila, 'c', 'C', 'd', 'D'
17  * für Claudius und 's', 'S', 'z', 'Z' für das Kind, jeweils für die
18  * Rotationen 0-3) sowie das Ziel und zwei Brückensymbole ('G', 'b', 'B').
19  * Beispiel:
20  * <pre>
21  * O-d-O-G b
22  *     |   |
23  * l-O-O-O W-
24  *     |
25  * O-O-O-z-O
26  *     |
27  * p-O-O W-W-
28  *       |
29  * </pre>
30  *
31  * @author Thomas Röfer
32  */
33 class Level
34 {
35     /** Die Gitterstruktur des Levels. */
36     private final Field field;
37
38     /** Die Liste aller Akteur:innen. */
39     private final List<Actor> actors = new ArrayList<>();

44     /**
45      * Erzeugt einen neuen Level aus einer Datei und stellt ihn dar.
46      * @param fileName Der Name der Datei mit der Level-Beschreibung.
47      * @throws IllegalArgumentException Die Datei kann nicht gefunden oder
48      *         gelesen werden oder ihr Inhalt ist ungültig.
49      */
50     Level(final String fileName)
51     {
52         // Alle Zeilen des Levels einlesen
53         final List<String> lines = new ArrayList<String>();
54         try (final InputStream input = Game.Jar.getInputStream(fileName);
55                 final Reader reader = new InputStreamReader(input);
```

```
56                  final BufferedReader stream = new BufferedReader(reader)) {
57              String line;
58              while ((line = stream.readLine()) != null) {
59                  lines.add(line);
60              }
61          }
62          catch (final FileNotFoundException e) {
63              throw new IllegalArgumentException("Level '" + fileName
64                      + "' wurde nicht gefunden.");
65          }
66          catch (final IOException e) {
67              throw new IllegalArgumentException("Fehler beim Lesen des Levels '"
68                      + fileName + "'.");
69          }
70
71          // Die Gitterstruktur konstruieren.
72          field = new Field(lines.toArray(new String[lines.size()]));
73
74          // Die Spielfigur vorab erzeugen, da sie die erste Akteur:in sein muss.
75          // Außerdem erwarten sie die Spaziergänger:innen als Parameter.
76          // Die x-Koordinate -1 wird als Markierung für eine noch nicht
77          // initialisierte Spielfigur verwendet. Außerdem sorgt sie dafür,
78          // dass die Spielfigur noch nicht zu sehen ist.
79          final Player player = new Player(-1, 0, 0, field);
80          actors.add(player);

83          // Alle Zellen des Feldes durchlaufen
84          for (int y = 0; y < lines.size(); y += 2) {
85              for (int x = 0; x < lines.get(y).length(); x += 2) {
86                  // Zelle mit erlaubten Symbolen vergleichen
87                  final int index = "pPqQlLiIcCdDsSzZGbBWO "
88                          .indexOf(lines.get(y).charAt(x));
89
90                  if (index == -1) {
91                      // Kein erlaubtes Symbol gefunden
92                      throw new IllegalArgumentException("Unbekanntes Symbol '"
93                              + lines.get(y).charAt(x) + "' in Level '" + fileName
94                              + "', Zeile " + (y + 1) + ", Spalte " + (x + 1)
95                              + " gefunden.");
96                  }
97                  else if (index < 4) {
98                      // Es darf nur eine Spielfigur geben.
99                      if (player.getX() != -1) {
100                         throw new IllegalArgumentException("Zweite Spielfigur in Level '"
101                                 + fileName + "', Zeile " + (y + 1) + ", Spalte "
102                                 + (x + 1) + " gefunden.");
103                     }
104
105                     // Existierende Spielfigur platzieren.
106                     player.setLocation(x / 2, y / 2);
107                     player.setRotation(index);
108                 }
109                 else if (index < 16) {
110                     // Spaziergänger:innen platzieren, sind nach Bild und Rotationen
111                     // geordnet.
112                     final String[] images = {"laila", "claudius", "child"};
113                     final Actor actor = new Walker(x / 2, y / 2, index % 4,
114                             images[index / 4 - 1], field, player);
115                     actors.add(actor);

117                 }
118                 else if (index < 19) {
119                     // Ziel und Brückensymbole einfügen
120                     final String[] images = {"goal", "bridge-0", "bridge-1"};
121                     new GameObject(x / 2, y / 2, 0, images[index - 16], field);
122                 }
123             }
124         }
125
126         // Wurde die Spielfigur nicht bewegt, wurde keine gefunden.
127         if (player.getX() == -1) {
128             throw new IllegalArgumentException("Keine Spielfigur in Level '" + fileName +
                        "' gefunden");
129         }
130     }
```

```
131
132     /**
133      * Liefere alle Akteur:innen dieses Levels.
134      * @return Die Akteur:innen des Levels. Die Spielfigur steht immer an erster
135      *         Stelle.
136      */
137     List<Actor> getActors()
138     {
139         return actors;
140     }
152 }
```

Die Klasse wird in der Methode *main()* der Klasse *PI1Game* verwendet:

```
15          // Den Level erzeugen
16          final Level level = new Level("levels/1.lvl");
17
18          // Die Hauptschleife des Spiels
19          while (level.getActors().get(0).isVisible()) {
20              for (final Actor actor : level.getActors()) {
```

Aufgabe 2 Bonusaufgabe: Ich bin dann mal weg (10 %)

Die Klasse *Field* enthält nun eine Liste aller erzeugten Spielobjekte, die im Konstruktor befüllt und in der Methode *hide()* verwendet wird, um die Objekte wieder verschwinden zu lassen:

```
47      /** Die Liste aller erzeugten Spielobjekte. */
48      private final List<GameObject> gameObjects = new ArrayList<>();

62                  gameObjects.add(new GameObject(x / 2, y / 2, 0,
63                          neighborhoodToFilename[getNeighborhood(x, y)]
64                          .replace("path", getCell(x, y) == 'W' ? "water" : "path")));

135     /**
136      * Lässt das Feld wieder vom Bildschirm verschwinden.
137      */
138     void hide()
139     {
140         for (final GameObject gameObject : gameObjects) {
141             gameObject.setVisible(false);
142         }
143     }
```

Die Klasse *Level* enthält ebenso eine Liste, die ebenfalls mit den erzeugten Spielobjekten befüllt und in der Methode *hide()* verwendet wird, um die Objekte wieder verschwinden zu lassen. Außerdem wird dort die *hide()*-Methode von *Field* aufgerufen.

```
41      /** Die Liste aller erzeugten Spielobjekte. */
42      private final List<GameObject> gameObjects = new ArrayList<>();

81          gameObjects.add(player);

116                     gameObjects.add(actor);

121                     gameObjects.add(new GameObject(x / 2, y / 2, 0, images[index - 16]));

142     /**
143      * Lässt den Level wieder vom Bildschirm verschwinden.
144      */
145     void hide()
146     {
147         for (final GameObject gameObject : gameObjects) {
148             gameObject.setVisible(false);
149         }
150         field.hide();
151     }
```

Am Ende der Methode *main()* der Klasse *PI1Game* wird diese Methode aufgerufen, so dass der Level wieder verschwindet, was auch funktioniert.

```
25          level.hide();
```