Übungsblatt 5

Musterlösung

Aufgabe 1 Zugriffssicherheit (30 %)

Die Feldbeschreibung wird vom Konstruktor in einem konstanten Attribut gespeichert.

```
37      /**
38       * Die Feldbeschreibung. Jede zweite Spalte und Zeile enthält die
39       * eigentlichen Zellen. Dazwischen sind die Nachbarschaften
40       * vermerkt.
41       */
42      private final String[] field;
43
44      /**
45       * Erzeugt ein neues Feld.
46       * @param field Die Feldbeschreibung. Jede zweite Spalte und Zeile
47       *          enthält die eigentlichen Zellen. Dazwischen sind die
48       *          Nachbarschaften vermerkt.
49       */
50      Field(final String[] field)
51      {
52          this.field = field;
59      }
```

Um innerhalb der Feldbeschreibung zu liegen, müssen beide Koordinaten mindestens 0 sein. Außerdem muss die (vertikale) y-Koordinate kleiner als die Länge des Arrays sein und die (horizontale) x-Koordinate muss kleiner als die Länge der Zeichenkette in der durch y indizierten Zeile sein.[1] Dann wird das entsprechende Zeichen zurückgeliefert. Ansonsten wird das Leerzeichen zurückgegeben.

```
61      /**
62       * Liefert ein Zeichen der Feldbeschreibung.
63       * @param x Die horizontale Koordinate des Zeichens, das
64       *          zurückgeliefert wird.
65       * @param y Die vertikale Koordinate des Zeichens, das
66       *          zurückgeliefert wird.
67       * @return Das Zeichen an der entsprechenden Zelle oder ein
68       *          Leerzeichen, wenn die Koordinaten außerhalb der
69       *          Beschreibung liegen.
70       */
71      private char getCell(final int x, final int y)
72      {
73          if (x >= 0 && y >= 0 && y < field.length && x < field[y].length()) {
74              return field[y].charAt(x);
75          }
76          else {
77              return ' ';
78          }
79      }
```

[1] Achtung: Es ist wichtig, zuerst zu prüfen, ob y ein gültiger Index für das Array ist, bevor es verwendet wird, um auf das Array zuzugreifen. Schlägt nämlich einer der ersten Tests fehl, werden alle weiteren nicht mehr ausgewertet, was für den Array-Zugriff einen Bereichsfehler verhindert.

Aufgabe 2 Nachbarschaftshilfe (30 %)

Das Überprüfen der vier Nachbarn wird mit Hilfe eines Arrays erledigt, das die jeweiligen Versätze für die x- und y-Koordinaten enthält. Diese Koordinatenpaare werden durchlaufen, während gleichzeitig die Variable *bit* in jedem Durchlauf verdoppelt wird und dadurch immer den zu dem getesteten Nachbarn passenden Wert hat. Die eigentliche Signatur wird in der Variablen *neighborhood* aufsummiert.

```
81      /**
82       * Liefert die Nachbarschafts-Signatur einer Zelle der
83       * Feldbeschreibung zurück.
84       * @param x Die horizontale Koordinate der Zelle, deren
85       *          Nachbarschafts-Signatur zurückgeliefert wird.
86       * @param y Die vertikale Koordinate der Zelle, deren
87       *          Nachbarschafts-Signatur zurückgeliefert wird.
88       * @return Die Signatur als Summe der Zahlen 1 (x+1, y),
89       *         2 (x, y+1), 4 (x-1, y) und 8 (x, y-1), wenn in
90       *         der jeweilen Richtung eine Verbindung zum Nachbarn
91       *         besteht.
92       */
93      private int getNeighborhood(final int x, final int y)
94      {
95          // Die (x, y)-Versätze für die einzelnen Prüfrichtungen
96          final int[][] neighbors = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
97
98          // Anfangs ist die Nachbarschafts-Signatur leer.
99          int neighborhood = 0;
100
101         // Variable zum Hochzählen der Signatur-Komponenten
102         int bit = 1;
103
104         // Signatur berechnen und zurückgeben
105         for (final int[] offsets : neighbors) {
106             if (getCell(x + offsets[0], y + offsets[1]) != ' ') {
107                 neighborhood += bit;
108             }
109             bit *= 2;
110         }
111         return neighborhood;
112     }
```

Aufgabe 3 Feldkonstruktion (40 %)

Das eigentliche Feld wird im Konstruktor erzeugt. Die Feldbeschreibung wird vertikal (y) und horizontal (x) in Zweierschritten durchlaufen. Zu jeder Zelle wird die Signatur bestimmt und diese als Index in das Array der Grafiknamen verwendet. Mit Hilfe des passenden Grafiknamens wird dann das entsprechende *GameObject* erzeugt, wobei dessen Koordinaten jeweils halbiert werden, damit die Spielobjekte in der Anzeige direkt aneinander stoßen.

```
54          for (int y = 0; y < field.length; y += 2) {
55              for (int x = 0; x < field[y].length(); x += 2) {
56                  new GameObject(x / 2, y / 2, 0, neighborhoodToFilename[getNeighborhood(x,
                            y)]);
57              }
58          }
```

In Abbildung 1 ist das Ergebnis des Aufrufs der Methode *test()* zu sehen. Hierbei kommen alle 16 Gittervarianten vor. Werden in der Feldbeschreibung in den zwei ersten Strings alle Leerzeichen am Ende entfernt, fehlt bei einem erneuten Aufruf von *test()* die rechteste Zelle des Spielfeldes, aber ansonsten bleibt alles gleich. Es wird also korrekt mit verschieden langen Zeilen umgegangen.

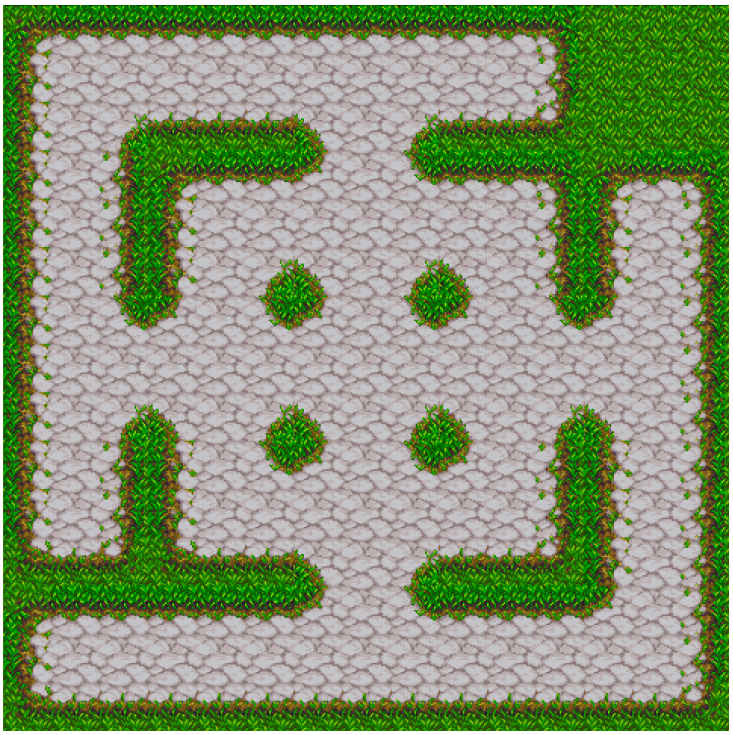

Abbildung 1: Das Ergebnis der Methode *test()*