Praktische Informatik I
Tutor:in: Leander Staack

WiSe 2021/22
Bearbeiter:in: Julius Walczynski [6113829]

Übungsblatt 8

Lösungsvorschlag

Aufgabe 1 Nachbarschaften

```
81      public boolean hasNeighbor(int x, int y, int direction){
82          return ((getNeighborhood(x*2, y*2)>>direction)&1)==1;
83      }
```

Die Methode *hasNeighbor* bekommt drei Parameter übergeben:

x	Die X-Koordinate des Feldes von dem aus getestet werden soll.
y	Die Y-Koordinate des Feldes von dem aus getestet werden soll.
direction	Eine Richtung kodiert in 2Bits. Dargestellt durch die Nummern 0-3.

Im die Spielfeld-Koordinaten zu den Koordinaten des zu Grunde liegenden String-Arrays *map* umzurechnen, müssen diese zuerst mit 2 multipliziert werden.
Mit Hilfe dieser Koordinaten, kann die Methode *getNeighborhood* eine 4-Bit kodierte Nachbarschaftssignatur errechnen, wobei das n-te Bit jeweils die Existent eines validen Nachbarfeld in der jeweilig kodierten Richtung darstellt (0: wenn bei Nichtexistienz, 1: bei Existenz).

Um die gewünschte Richtung/die Information des Bits für die entsprechende Richtung besser überprüfen zu können, wird es mit Hilfe des Bit-Shift Operators auf das niedrigste Bit gebracht. Da das gewünschte Bit an der Position der kodierten Richtung sitzt, kann es mit Hilfe von *getNeighborhood(...)»direction* an die erste Stelle geschoben werden.

Um alle anderen Bits mit unnötigen Informationen loszuwerden, wird der „Und“-Operator auf das Ergebnis und eine Bitmaske von *1=0b0001* angewendet. Das resultierende Ergebnis besteht nun nur noch aus einem einzigen informationstargenden Bit (dem ersten Bit).

Sollte die Nachbarschaftssignatur für die gefragte Richtung wahr gewesen sein (ein Bit mit dem Wert 1 an der entsprechenden Stelle enthalten haben), ist das Ergebnis = 1; ansonsten = 0.

Nun kann einfach durch *==1* überprüft werden, ob ein entsprechender Nachbar existiert.
Das logische Ergebnis wird danach als Rückgabewert zurück gegeben.

Aufgabe 2 Nachbarschaftstest

Der Unit-Test für die *Field*-Klasse besteht aus 4 Tests, die alle auf eine grundlegende Test-Methode (*testField*) zurückgreifen.

```
76      public void testField(String[] map) {
77          Field field = new Field(map);
78          for(int y = -1; y <= field.SIZE_Y; y++){
79              for(int x = -1; x <= field.SIZE_X; x++){
80                  if(field.hasNeighbor(x,y,0)) assertEquals(map[y*2].charAt(x*2+1),'-');
81                  if(field.hasNeighbor(x,y,1)) assertEquals(map[y*2+1].charAt(x*2),'|');
82                  if(field.hasNeighbor(x,y,2)) assertEquals(map[y*2].charAt(x*2-1),'-');
83                  if(field.hasNeighbor(x,y,3)) assertEquals(map[y*2-1].charAt(x*2),'|');
84              }
85          }
86      }
```

testField loopt über jede mögliche Koordinate innerhalb der gegebenen *map*, sowie jeweils über ein Feld außerhalb der *map*, um zu überprüfen, ob *OutOfBoundsExceptions* richtig gehandelt werden. Die realen Feld-Koordinaten werden in die Koordinaten der *map* umgerechnet. Wenn *hasNeighbor* für eine Richtung wahr ist, so muss in der *map* in dieser Richtung eine Verbindung zu einer Nachbarzelle gegeben sein. Dies wird in 4 *if-statements* für alle 4 möglichen Richtungen überprüft.

Die 4 Testfälle *testMap1, testMap2, testMap3, testMap4* benutzen die *testField*-Methode um unterschiedliche Karten als Inputs zu testen. Dabei werden Edge-Cases wie *map=null*, *map* enthält Elemente die *null* sind, *map* enthält keine Elemente und normale Kartenlayouts getestet.

Aufgabe 3 In geregelten Bahnen

Durch die *hasNeighbor*-Methode, lässt sich die Bewegung des Spielers auf die Wege beschränken:

```
68              if(!field.hasNeighbor(player.getX(), player.getY(), player.getRotation()) ||
                  (x==player.getX() && y==player.getY())) continue;
69
70              player.setLocation(x, y);
```

Bevor der Spieler in dem Main-Gameloop nach dem drücken einer der Pfeiltasten in die zugehörige Richtung bewegt wird, wird gecheckt, ob in der gefragten Richtung, überhaupt ein angeschlossenes Weg-Feld existiert. Sollte dies nicht der Fall sein, wird der nachfolge Code durch *continue* übersprungen und der nächste Durchgang der *while*-Schleife direkt ausgeführt. Somit schreitet das Spiel erst voran, wenn der Spieler über einen validen Weg läuft.

Auch Bewegung der *NPC*s lässt sich durch *hasNeighbor* vereinfachen:

```
24              if(!RPGGame.getField().hasNeighbor(avatar.getX(), avatar.getY(), avatar.
                  getRotation())){
25                  avatar.setRotation((avatar.getRotation()+2)%4);
26              }
27              int _direction = avatar.getRotation();
28              Move(_direction);
```

Der *NPC* läuft solange in seine Anfangsrichtung, bis der Weg in diese Richtung endet (*hasNeighbor* in die Laufrichtung *false* zurückgibt). Wenn dies passiert, dreht sich der *NPC* um 180° und läuft in die entgegengesetzte Richtung, bis er erneut auf das Ende des Weges trifft. Diese Schleife führt dazu, dass der *NPC* auf einer gerade Strecken hin und her patrouilliert.