Übung 08: JUnit

Abgabetermin: 21. 5. 2015, 8:15

Name: _____ **Matrikelnummer**: _____

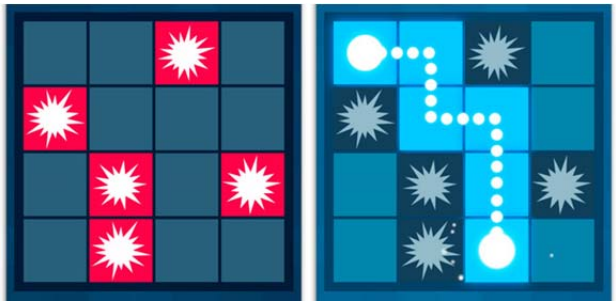
Informatik: ☐ G1 (Prähofer) ☐ G2 (Prähofer) ☐ G3 (Grimmer) ☐ G4 (Grimmer)

WIN: ☐ G1 (Khalil) ☐ G2 (Kusel) ☐ G3 (Kusel)

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Punkte
Übung 8	24	Java-Programm, JUnit-Testfälle, Ausgabe eines Spielablaufs	Java-Programm, JUnit-Testfälle	☐	

RememBrain

In dieser Übung soll das Spiel RememBrain implementiert und mit JUnit ausführlich getestet werden. RememBrain ist ein Gehirntrainingsspiel, welches das Gedächtnis trainiert. Dabei wird in der ersten Spielphase ein Gitter mit Bomben angezeigt, welches sich der Spieler einprägen muss. In der zweiten Spielphase sind diese Bomben nicht mehr sichtbar – der Spieler ist nun gefordert zwei angezeigte Punkte mittels eines Pfades zu verbinden, jedoch ohne ein Feld mit einer Bombe zu überqueren. Wurde durch den Spieler ein entsprechender Pfad zwischen den beiden Punkten erstellt, so wird das Spiel ausgewertet – d.h., es wird überprüft, ob der betretene Pfad tatsächlich keine Bomben enthalten hat. In diesem Fall hat der Spieler gewonnen, ansonsten verloren.



1. Spielphase 2. Spielphase

A) Implementierung des Spiels (14 Punkte)

Spielablauf

In der gesuchten Implementierung werden die Spielfelder für die zwei Spielphasen mit einem Anfangszustand aus einer Datei initialisiert (vgl. "testData1.txt" und "testData2.txt" aus der Vorgabe), welche die Positionen der Bomben sowie die Positionen der zu verbindenden Punkte enthalten. D.h., das Berechnen von gültigen Feldzuständen für die zwei Spielphasen muss nicht selbst erfolgen, sondern man kann davon ausgehen, dass der Anfangszustand aus der Datei gültig ist. Das Feld der *ersten Spielphase*, welches die Bomben enthält, wird in der Folge für eine bestimmte Zeit dem Benutzer angezeigt, in der sich der Benutzer die Positionen der Bomben einprägen soll. In der *zweiten Spielphase* wird nun ein verdecktes Spielfeld mit zwei zu verbindenden Punkten angezeigt (die Punkte stammen ebenfalls aus der Eingabedatei). Der Spieler kann nun durch die Eingabe der Koordinaten im Gitter seinen Pfad eingeben. Sobald ein Pfad zwischen den beiden Punkten erstellt wurde, wird das Spiel ausgewertet – d.h., es wird überprüft, ob der eingegebene Pfad tatsächlich keine Bomben enthalten hat.

Exemplarischer Spielablauf mit Testdaten aus "testData2.txt":

1. Spielphase (Bomben werden dem Benutzer zum Einprägen angezeigt)

```
    0  0  0  0  0
    1  2  3  4  5
01  x  .  .  .  .
02  .  .  .  .  .
03  .  .  .  x  .
04  .  x  .  .  .
05  .  .  .  .  .

================================================================================
================================================================================
```

2. Spielphase (Bomben sind nicht mehr sichtbar; Punkte sollen nun verbunden werden, ohne Bomben zu queren)

```
    0  0  0  0  0
    1  2  3  4  5
01  .  .  .  .  .
02  .  .  .  .  .
03  .  .  p  .  .
04  .  .  .  .  .
05  .  .  .  .  p
Please select a field to build the path in consecutive order
Row: 3
Column: 4

    0  0  0  0  0
    1  2  3  4  5
01  .  .  .  .  .
02  .  .  .  .  .
03  .  .  p  -  .
04  .  .  .  .  .
05  .  .  .  .  p
Please select a field to build the path in consecutive order
Row: 3
Column: 5

    0  0  0  0  0
    1  2  3  4  5
01  .  .  .  .  .
02  .  .  .  .  .
03  .  .  p  -  -
04  .  .  .  .  .
05  .  .  .  .  p
Please select a field to build the path in consecutive order
Row: 4
Column: 5

    0  0  0  0  0
    1  2  3  4  5
01  .  .  .  .  .
02  .  .  .  .  .
03  .  .  p  w  c
04  .  .  .  .  c
05  .  .  .  .  p
Sorry - Try Again!
```

Vorgabe für Implementierung

Um Ihnen diese Aufgabe zu erleichtern, ist ein Klassengerüst vorgegeben (siehe `UE08_Vorgabe.zip`), welches folgende Pakete sowie Klassen enthält:

Paket `remembrain.model` (Umsetzung des Spiels):

- *Interface* `Field.java`: Gibt die Methoden vor, die eine Implementierung von RememBrain anbieten muss.
- *Klasse* `FieldImpl.java`: In dieser Klasse soll das Spiel implementiert werden. In der Vorgabe sind hier die leeren Methodenrümpfe des Interfaces `Field.java` enthalten.
- *Enumeration* `CellState.java`: Gibt mögliche Zustände vor, die eine Zelle im Feld annehmen kann. Eine Zelle kann entweder den Zustand `BOMB` (= Zelle enthält eine Bombe), den Zustand `EMPTY` (= Zelle ist leer), den Zustand `POINT` (= Zelle enthält einen zu verbindenden Punkt), den Zustand `PATH` (= Zelle enthält ein Teilstück des vom Benutzer eingegebenen Pfades), den Zustand `CORRECT` (= Zelle enthält ein korrektes Teilstück des Pfades, d.h., einen Teil, der keine Bombe enthalten hat) oder schließlich den Zustand `WRONG` (= Zelle enthält ein Teilstück des Pfades, welches eine Bombe enthalten hat) annehmen.

- *Enumeration* `GameState.java`: Gibt mögliche Zustände vor, die das Spiel annehmen kann. Das Spiel kann entweder den Zustand `INITIAL` (= Spiel befindet sich im Initialzustand, d.h., in der Phase in der der Benutzer sich das Feld merken kann), den Zustand `SELECTION` (= Spiel befindet sich in der Lösungsphase, d.h., der Benutzer kann die beiden angezeigten Punkte mit einem Pfad verbinden) oder den Zustand `END` (= Spiel wurde aufgelöst, d.h., es wurde ein Pfad zwischen den beiden Punkten hergestellt und darauf aufbauend ausgewertet) annehmen.

Paket `remembrain.textui` (Konsolen-basierte Schnittstelle):

- *Klasse* `TextUI.java`: Implementiert eine Konsolen-basierte Anwendung, um mit dem Spiel zu interagieren. Hierfür liest die Anwendung den Initialzustands der Felder aus einer Vorgabedatei ein (z.B. aus "testData1.txt" oder "testData2.txt" aus der Vorgabe), erzeugt eine neue Instanz des Feldes und fordert des Weiteren zur schrittweisen Eingabe von Werten für die Erstellung eines Pfades auf. Bitte beachten Sie, dass die Größe des Feldes (`WIDTH` und `HEIGHT`) sowie die einzulesende Datei (`FILE_NAME`) über Konstante konfiguriert werden können.

Hinweise zur Implementierung:

Um das Spiel zu implementieren, muss die Klasse `FieldImpl.java` implementiert werden. Sehen Sie für die Initialisierung einer Instanz der Klasse eine Konstruktor mit zwei Parametern vor, welcher die zwei initialen Feldzustände für die zwei Spielphasen übernehmen kann (d.h., ein `initialFieldToRemember`, welches die Bomben enthält, sowie ein `intitialFieldRemembered`, welches die beiden Punkte, welche verbunden werden sollen, enthält).

Darüber hinaus muss eine gültige Implementierung des Interfaces `Field.java` erstellt werden, welche folgende Methoden vorgibt:

- `public int getWidth();`
  Diese Methode soll dazu verwendet werden, die *Breite* des Spielfeldes ermitteln zu können.
- `public int getHeight();`
  Diese Methode soll dazu verwendet werden, die *Höhe* des Spielfeldes ermitteln zu können.
- `public GameState getGameState();`
  Diese Methode soll dazu verwendet werden, den aktuellen *Zustand* des Spiels ermitteln zu können.
- `public boolean getGameResult();`
  Diese Methode soll dazu verwendet werden, den *Endzustand* des Spiels ermitteln zu können – d.h., `true` im Falle, dass der Spieler gewonnen hat bzw. `false` andernfalls. Die Methode soll nur in der Phase `GameState.END` aufgerufen werden können, da sie nur dann ein sinnvolles Ergebnis zurückliefern kann. D.h., falls diese Methode in einer anderen Spielphase aufgerufen wird, soll eine entsprechende Exception (`java.lang.IllegalStateException`) geworfen werden.
- `CellState getCellStateToRemember(int row, int col);`
  Diese Methode soll dazu verwendet werden, den Zustand einer Zelle zu ermitteln, welche in der initialen Phase eingeprägt werden soll. Daher soll diese Methode auch nur in der Phase `GameState.INITIAL` aufgerufen werden können, andernfalls soll eine Exception (`java.lang.IllegalStateException`) geworfen werden.
- `CellState getRememberedCellState(int row, int col);`
  Diese Methode soll dazu verwendet werden, den Zustand einer Zelle zu ermitteln, welche in der Erinnerungsphase rekonstruiert wurde. Daher soll diese Methode auch nur in den Phasen `GameState.SELECTION` und `GameState.END` aufgerufen werden können, andernfalls soll eine Exception (`java.lang.IllegalStateException`) geworfen werden.
- `public void select(int row, int col);`
  Diese Methode soll dazu verwendet werden, ein Pfadteilstück auf eine bestimmte Zelle zu setzen. Daher soll diese Methode nur in der Phase `GameState.SELECTION` aufgerufen werden können, andernfalls soll eine Exception (`java.lang.IllegalStateException`) geworfen werden.
- `public void timeToRemember(int millis);`
  Diese Methode soll dazu verwendet werden, die Zeitspanne für die Einprägungsphase zu bestimmen bzw. einzuhalten. Am Ende der Zeitspanne wechselt der Spielzustand von `GameState.INITIAL` auf `GameState.SELECTION`. Daher darf diese Methode auch nur in der Phase `GameState.INITIAL` aufgerufen werden, andernfalls soll eine Exception (`java.lang.IllegalStateException`) geworfen werden.

- `public boolean isValidMove(int row, int col);`
  Diese Methode soll dazu verwendet werden, zu bestimmen, ob ein Zug gültig ist. Ein Zug ist nur dann gültig, wenn er entweder den Pfad von einem Punkt beginnend startet oder den bereits begonnen Pfad weiter fortsetzt. Diese Methode darf nur in der Phase `GameState.SELECTION` aufgerufen werden, andernfalls soll eine Exception (`java.lang.IllegalStateException`) geworfen werden.

Beachten Sie auch, dass bei Übergabe von ungültigen Indizes für Spalten oder Zeilen wie auch bei Übergabe von ungültigen Werten jeweils eine entsprechende Exception (`java.lang.IllegalArgumentException`) geworfen werden soll. Beachten Sie schließlich, dass Sie – wo sinnvoll – entsprechende Assertions einfügen.

Hinweis für die Implementierung:

- Die größte Herausforderung in der Implementierung liegt darin, zu bestimmen, ob ein gültiger Pfad zwischen den beiden Punkten erstellt wurde. Hierfür ist es hilfreich sich einen Hilfsalgorithmus zu schreiben, welcher sich einerseits „merkt" welche Koordinaten er bereits besucht hat bei der Pfadsuche (um nicht im Kreis nach einem Pfad zu suchen) bzw. welche Koordinaten er noch besuchen muss, wobei diese Menge mit den Koordinaten von einem der beiden zu verbindenden Punkte initialisiert wird. Folglich wird bei einem der beiden Punkte begonnen und es wird überprüft ob ausgehend von diesem Punkt ein oder mehrere angrenzende Pfadteilstücke betreten werden können. Dies wird solange fortgesetzt, solange weitere Pfadteilstücke betreten werden können oder schlussendlich der andere Punkt erreicht worden ist.

B) Testen des Spiels mit JUnit (10 Punkte)

Schreiben sie anschließend einen umfassenden JUnit-Test für Ihre RememBrain-Implementierung. Der JUnit-Test soll im Package `remembrain.model.test` implementiert werden.

Hinweise für den Test:

- Beginnen Sie jeden Test mit einem wohldefinierten Anfangszustand.
- Testen Sie alle Methoden der öffentlichen Schnittstelle möglichst umfassend.
- Beachten Sie, nicht nur gültige Verwendungen der Methoden zu testen, sondern bewusst auch ungültige Verwendungen zu testen und zu überprüfen, ob das Verhalten im Fehlerfall korrekt ist laut obiger Spezifikation.
- Um den aktuellen Zustand des RememBrain-Feldes effizient mit einem erwarteten Zustand vergleichen zu können, kann es sinnvoll sein, sich entsprechende Hilfsmethoden zu schreiben.

Beachten Sie schließlich, dass die kommende Übung 9 auf dieser Übung aufbauen wird – d.h., mit einer sauberen Ausarbeitung dieser Übung schaffen Sie sich eine gute Ausgangsbasis für die folgende Übung!