PROTOKOLL

"CONWAYS GAME OF LIFE"

Simon Bullik & Nils Kiele | DQI16
Europaschule Schulzentrum SII Utbremen
18. November 2018

Simon Bullik & Nils Kiele

INHALTSVERZEICHNIS

0 ALLGEMEINES **2**

1 AUFGABENSTELLUNG **2**

2 AUSSTATTUNG **2**

3 BENUTZERANLEITUNG **2**

4 FUNKTIONSBESCHREIBUNG **3**

5 ERLÄUTERUNG DER EINZELNEN PROGRAMMTEILE **3**

5.1 IMPORTE 3
5.2 UML-KLASSENDIAGRAMM 4
5.3 KLASSEN 4
5.3.1 IGameOfLife.java *5*
5.3.2 GameOfLife.java *5*
5.3.3 GameOfWildLife.java *5*
5.3.4 Cell.java *5*
5.3.5 VisualGameOfLife.java *5*

6 ERKENNTNISGEWINN **5**

0 Allgemeines

Wir haben uns dazu entschieden, alle Funktionen, Variablennamen und Kommentare unseres Programmcodes auf Englisch zu schreiben. Für die Bearbeitung des Programms haben wir einige Wochen Zeit bekommen und in der Schule wie auch zu Hause gearbeitet.

1 Aufgabenstellung

Aufgabe war es, eine Applikation in Java zu schreiben, die **Conway's Game of Life** simuliert. Hierbei handelt es sich um eine Simulation einer sehr simpel gestrickten *Lebensform*. Auf einem Gitter befinden sich viele kleine Quadrate, die entweder *lebendig* oder *tot* sind. Diese *Zellen* haben immer acht Nachbarn (horizontal, vertikal und diagonal). Außerdem handelt es sich bei dem Gitter um eine *Donutwelt* – der linke Rand grenzt an den rechten, der obere an den unteren und jeweils umgekehrt.

Die nächste Generation entwickelt sich aus der vorherigen Konfiguration, indem folgende Regeln beachtet werden.

- Eine tote Zelle mit genau drei lebenden Nachbarn wird in der Folgegeneration neu geboren.
- Lebende Zellen mit weniger als zwei lebenden Nachbarn sterben in der Folgegeneration an Einsamkeit.
- Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt in der Folgegeneration am Leben.
- Lebende Zellen mit mehr als drei lebenden Nachbarn sterben in der Folgegeneration an Überbevölkerung.

Voraussetzung war, ein vorgegebenes Java-Interface zu implementieren. Darüber hinaus konnte die Darstellung der simulierten Welt in einer GUI dargestellt und zusätzliche Funktionalität programmiert werden.

2 Ausstattung

Als Entwicklungsumgebung haben wir *IntelliJ IDEA* von *JetBrains* verwendet. Für das Verwalten und Einbinden unserer verwendeten *Libraries* haben wir das Build-Management-Tool *Maven* benutzt. Anfangs haben wir die Library *Jansi* darüber eingebunden, um auf einfache Weise eine farbige Ausgabe in der Konsole zu erreichen. Im fertigen Produkt haben wir mithilfe von Maven *JUnit 5* eingebunden, um die gewünschten *Unit-Tests* zu schreiben. Da *Maven* gut mit *IntelliJ* zusammenarbeitet, sollten alle *Dependencies* schnell erfüllt sein und das Programm kann gestartet werden.

3 Benutzeranleitung

Um das Programm auszuführen, sollte es in der *IntelliJ*-IDE geöffnet werden. Von hier aus kann das Programm *GameOfLife.java* gestartet werden. Es sollte sich ein Fenster

öffnen, indem die Simulation angezeigt wird. Gelbe Kästchen sind *lebendig*, während schwarze *tot* sind. Zweimal pro Sekunde erscheint eine neue Generation und der Benutzer kann nicht mit dem Programm interagieren.

Neben dieser simpel gehaltenen Version von *Conway's Game of Life* haben wir ein weiteres Programm entwickelt, das etwas aufregender ist. Um dieses auszuführen, muss *GameOfWildLife.java* gestartet werden. In diesem Programm folgen Zellen den gleichen Regeln wie zuvor, aber anstelle von gelber Farbe werden lebendige Zellen bunt dargestellt. Im Laufe der Zeit passen sich die Farben der Zellen innerhalb einer Gruppe an während *neugeborene* Zellen mit zufälligen Farben frischen Wind in die Simulation bringen. Außerdem können durch Klicken der Maus auf der Oberfläche tote Zellen zum Leben erweckt werden. Wird gleichzeitig die *Shift*-Taste gedrückt, können lebendige Zellen getötet werden. Viel Spaß beim Ausprobieren. ☺

4 Funktionsbeschreibung

Unser Programm erfüllt die Anforderungen der Aufgabe. Wir haben uns dazu entschieden, *GameOfLife* und *GameOfWildLife* klar voneinander zu trennen. Grund dafür ist, dass *GameOfLife* das Interface implementiert, da wir in diesem Programm mit einem zweidimensionalen Array von *ints* arbeiten. Auch mit *GameOfWildlife* haben wir das Interface *IGameOfLife* implementiert, aber viele Methoden sind hier nicht sinnvoll benutzbar, da wir statt *ints* eigens geschriebene *Cell*-Objekte verwendet haben, was oft nicht mit dem Interface vereinbar ist.
Um die korrekte Funktionsweise des Programms zu überprüfen, haben wir Unit-Test geschrieben, die sich im *tests*-Ordner befinden. Diese überprüfen verschiedene Spielfeldkonstellationen und testen auf korrekte Ergebnisse nach einer bestimmten Anzahl von Generationen.

5 Erläuterung der einzelnen Programmteile

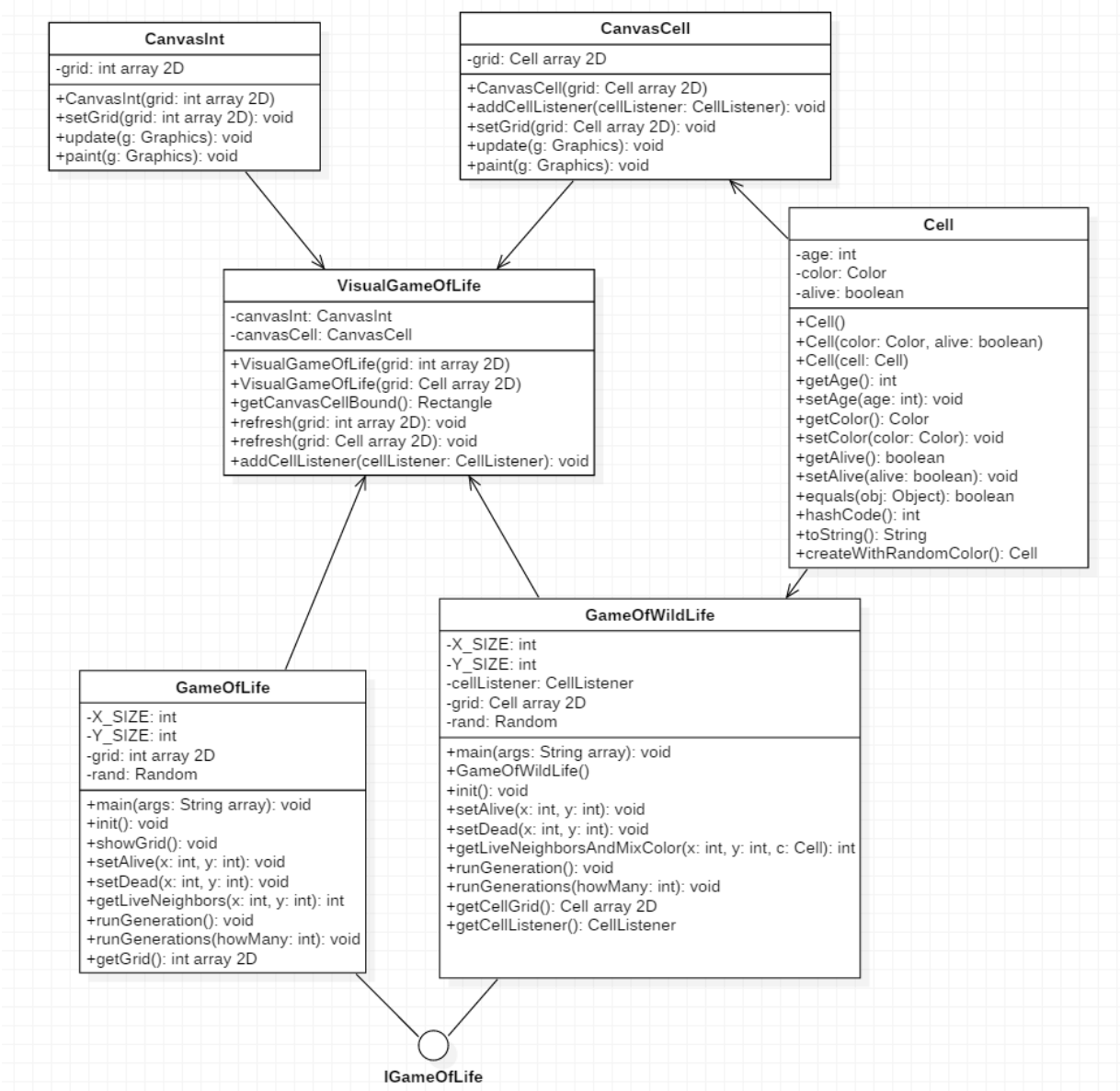
Hier werden alle wesentlichen Elemente der Software erklärt. Detaillierte Angaben zur Umsetzung sind im Programmcode als Kommentare zu finden.

5.1 Importe

Es folgt eine Übersicht der von uns verwendeten Importe in unserem Projekt.

1. **java.awt.*** benutzen wir für die graphische Darstellung.
2. **java.util.Objects** verwenden wir für die *hashCode*-Methode in *Cell.java*.
3. **java.util.Random** wird für zufällige Farben und Zustände von Zellen benutzt.

5.2 UML-Klassendiagramm



Dieses Diagramm bietet einen Überblick über die Struktur unseres Programmes. Wir haben uns am UML-Standard orientiert und auf diese Weise die Implementation des Interfaces sowie Assoziationen dargestellt. Die Klasse *GameOfWildLife* besitzt zusätzlich eine untergeordnete Klasse *CellListener*, die aus Gründen der Übersichtlichkeit nicht in diesem Diagramm dargestellt ist.

5.3 Klassen

Es folgen die Klassen unseres Programms. Die meisten Klassen haben ihre entsprechend benannten Java-Datei; in *VisualGameOfLife.java* befinden sich allerdings mehrere relativ kleine Klassen, da sie eng miteinander zu tun haben.

5.3.1 IGameOfLife.java

Diese Datei stellt das Interface dar, das wir im Unterricht zur Verfügung gestellt bekommen haben. Diese Datei ist von uns unverändert.

5.3.2 GameOfLife.java

Das klassische *Game of Life* ist in dieser Datei implementiert. Mithilfe der Methode *showGrid* kann das Spielfeld in der Konsole ausgegeben werden. Da die Darstellung mit der GUI aber deutlich übersichtlicher ist, wird diese Methode aktuell nicht verwendet.

5.3.3 GameOfWildLife.java

Die deutlich „wildere" Variante der Simulation ist in dieser Datei zu finden. Bunte Zellen und interaktives Generieren und Auslöschen von Zellen (näher beschrieben in der 3 Benutzeranleitung) sind in dieser Klasse zur Verfügung gestellt.

5.3.4 Cell.java

Objekte dieser Klasse werden in *GameOfWildlife* benötigt. So können Farbwerte und Alter verschiedener Zellen einfach gespeichert werden. Zudem überschreibt diese Klasse etliche Standardmethoden wie *equals* und *toString* und besitzt verschiedene Arten von Konstruktoren (allgemein, mit Parametern und Copy-Konstruktor).

5.3.5 VisualGameOfLife.java

Diese Datei sorgt für die grafische Darstellung unseres Spielfeldes. Sowohl *int*- als auch *Cell*-Arrays können mit der GUI ausgegeben werden.

6 Erkenntnisgewinn

Wir haben mit diesem Projekt unsere Java-Kenntnisse ausgebaut. Auch wenn die Logik von *Conway's Game of Life* sehr simpel ist, entwickeln sich spannende Gebilde und Situationen in der Simulation. Abgesehen davon, dass wir Java nun flüssiger schreiben können, haben wir uns zudem mit der Verwendung einer grafischen Oberfläche (GUI) beschäftigt. Auch das Implementieren eines Interfaces sowie das Ausprägen von Unit-Tests waren spannende und lehrreiche Herausforderungen.

Die Teamarbeit ist uns gut gelungen; wir haben Aufgaben gleichmäßig verteilt und uns gegenseitig geholfen. Mit unserem Produkt sind wir zufrieden.

- codingWhale & SimonIT