Analyse «Game of Life»

Verteilte Systeme und Komponenten

Gruppe 5 (Bucher, Kiser, Meyer, Sägesser)

04.03.2018

Inhaltsverzeichnis

Allgemein

Bei der vorliegenden Applikation handelt es sich um eine Implementierung von John Horton Comways Simulation *Game of Life* als Java Applet.

Bei der Simulation gibt es ein Raster mit Zellen, die entweder den Satus aktiv oder inaktiv bzw. lebendig oder tot haben. Zu Beginn der Simulation werden einige lebendige Zellen vordefiniert. Mit jedem Zyklus entstehen, bleiben oder verschwinden die Zellen gemäss folgenden Regeln:

1. Eine lebendige Zelle mit weniger als zwei lebendigen Nachbarzellen stirbt (Unterbevölkerung).
2. Eine lebendige Zelle mit zwei oder drei lebendigen Nachbarzellen überlebt.
3. Eine lebendige Zelle mit mehr als drei lebendigen Nachbarzellen stirbt (Überbevölkerung).
4. Eine tote Zelle mit genau drei lebendigen Nachbarzellen wird lebendig (Fortpflanzung).

Dieses Regelwerk ist in der Methode `org.bitstorm.gameoflife.GameOfLifeGrid.next()` implementiert. Als Nachbarn gelten nicht nur die Zellen oberhalb, unterhalb, rechts, links von einer Zelle,

sondern auch die diagonal angrenzenden (8er-Nachbarschaft). Es gibt auch abweichende Regelwerke (sogenannte *alternative Welten*), die hier jedoch nicht von Interesse sind.

Aufbau

Packages

- `org.bitstorm`: Dies ist das Oberpackage, das die ganze Applikation beinhaltet.
- `org.bitstorm.gameoflife`: In diesem Package liegt der Code zur Erzeugung der grafischen Benutzeroberfläche und die eigentliche Spiellogik.
- `org.bitstorm.util`: Dieses Package beinhaltet Hilfsklassen wie Dialoge und Utility-Klassen für allgemeine Aufgaben.

Klassen

- `org.bitstorm.gameoflife`
  - `Cell` repräsentiert eine einzelne Zelle. Diese kennt ihre Position und die Anzahl aktiver Nachbarzellen.
  - `CellGrid` definiert ein Interface, das die Funktionalität eines Rasters bestimmt. Es kann der Status einer Zelle an einer bestimmten Position (mittels x/y-Koordinaten) und die Dimension des Grids abgefragt werden. Das Raster kann auch in der Grösse angepasst werden.
  - `CellGridCanvas` erweitert ein AWT-Canvas und behandelt die Interaktion des Benutzers mit dem Raster (Event-Handling). Der Benutzer kann per Mausklick den Status von Zellen ändern. Die Benutzeroberfläche ist skalierbar, die einzelnen Zellen haben jedoch eine feste Grösse: Beim Skalieren wird also nicht die Grösse der Zellen, sondern deren Anzahl angepasst. (So kann es passieren, dass Statusinformationen über Zellen am Rand nach Verkleinerung und anschliessender Vergrösserung des Fensters verlorengehen.) Die Interaktion mit dem Raster erfolgt über das `CellGrid`-Interface.
  - `GameOfLife` erweitert ein AWT-Applet und baut die ganze Benutzeroberfläche auf. Neben dem eigentlichen Spielfeld (Zellenraster) gibt es oben noch eine Menüleiste. (Die Steuerelemente am untere Bildschirmrand werden in einer anderen Klasse umgesetzt.) Die Berechnung der nächsten Generation wird in einen eigenen Thread ausgelagert, wozu das `Runnable`-Interface implementiert wird. Das Applet reagiert auf verschiedene Ereignisse, etwa auf die Veränderung von Zoom-Stufe und Geschwindigkeit.
  - `GameOfLifeControls` erweitert ein AWT-Panel und beinhaltet die Steuerelemente im unteren Bereich des Applets. Dies sind: Das Dropdown mit vordefinierten Figuren, Schaltflächen zur Steuerung des Ablaufs (*Next*, *Start*, Auswahl der Geschwindigkeit), eine Auswahl für die Zellengrösse und eine Statusanzeige für die Generation (Nummer des aktuellen Zyklus). Die Steuerelemente werden aufgebaut und mit (konstanten) Werten befüllt. Das Event-Handling für die Steuerelemente im unteren Panel ist hier umgesetzt.

  - `GameOfLifeControlsEvent` ist eine Erweiterung der `Event`-Klasse, die zusätzlich über Angaben zur Geschwindigkeit, Zoom-Stufe und zum Namen der Figur verfügt. So gelangen die Informationen immer direkt zum Event-Handler und müssen nicht noch manuell abgefragt werden. Dies erlaubt eine bessere Kapselung der Steuerelemente.
  - `GameOfLifeControlsListener` ist ein Interface, welches das `EventListener`-Interface erweitert, und vorgibt, welche Ereignisse vom Applet gehandhabt werden müssen. Das Interface wird von der Klasse `GameOfLife` implementiert. `GameOfLifeControls` kommuniziert über dieses Interface mit der Klasse `GameOfLife`.
  - `GameOfLifeGrid` implementiert das Interface `CellGrid`. Die Methode `next()` setzt das eingangs erläuterte Regelwerk um.
  - `Shape` repräsentiert eine vorgegebene Form (z.B. einen *Glider*), wie sie im unteren Panel ausgewählt werden kann. Es sind Informationen zum Namen, zu den aktivierten Zellen und zur Dimension vorhanden und abfragbar.
  - `ShapeCollection` beinhaltet mehrere statisch definierte Instanzen von `Shape`. Das Feld `COLLECTION` kann um weitere Formen erweitert werden, welche dann sogleich auf der Benutzeroberfläche zur Auswahl bereit stehen.
  - `ShapeException` erweitert `Exception` zur Behandlung von Fehlern, die mit einer `Shape` zusammenhängen. Sie wird geworfen, wenn ein nicht definiertes Shape angefordert wird oder ein vordefiniertes Shape nicht auf das Raster passt.
  - `StandaloneGameOfLife` ist ein Wrapper für das Applet, das in `GameOfLife` definiert ist. Es dient dazu, das Applet in einem eigenen Fenster anzuzeigen. (Der eigentliche Zweck von Applets ist es, diese im Browserfenster auszuführen, was auf den meisten Umgebungen aus Sicherheitsgründen deaktiviert ist.)
    * `AppletFrame` ist in der gleichen Datei definiert wie `StandaloneGameOfLife` und implementiert einen Wrapper, womit das Applet in einem eigenständigen Fenster angezeigt wird.
    * `GameOfLifeGridIO` ist als innere Klasse von `StandaloneGameOfLife` definiert und bietet zusätzliche Möglichkeiten zum Öffnen und Speichern von Spielständen. Diese Möglichkeiten bestehen in einem Browser-Applet aus Sicherheitsgründen nicht.
- `org.bitstorm.util`
  - `AboutDialog` implementiert einen AWT-Dialog und setzt das Fenster um, das bei einem Klick auf den Menüeintrag *Help*, *About* erscheint.
  - `AlertBox` implementiert einen AWT-Dialog für Warnungen, der etwa beim gescheiterten Öffnen von Spielständen erscheint.
  - `EasyFile` kapselt das Lesen und Schreiben von Textdateien, wie es für das Öffnen und Speichern von Spielständen benötigt wird.
  - `ImageComponent` kümmert sich um die Anzeige von Bildern, wie sie im `AboutDialog` und in der `AlertBox` verwendet werden.
  - `LineEnumerator` implementiert eine `Enumeration` über einen Text, der mittles `CR` (frühe Macs), `LF` (Unix) oder `CRLF` (Windows) in Zeilen umgebrochen wurde.
  - `TextFileDialog` implementiert einen AWT-Dialog zur Anzeige eines längere Textes – Lizenzangaben (Menü: *Help – License*), Handbuch (Menü: *Help – Manual*) –, der zu diesem Zweck in einem vertikal scrollbaren Textfeld angezeigt wird.

Log-Ereignisse

Im Folgenden sollen Ereignisse im Ablauf des Spiels und der Applikation ermittelt werden, die für das Logging von Interesse sein könnten. Weil das zu verwendende Interface mit den dazugehörigen Log-Levels noch nicht definiert ist, wird hier der Vorschlag derjenigen Interface-Gruppe verwendet, in der ein Gruppenmitglied der Gruppe 5 involviert ist. Es werden folgende Levels definiert: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`.

`TRACE`

- Allgemein: Meldungen zum *exakten* Nachvollziehen des Ablaufs der Applikation
- Statusänderungen einzelner Zellen
- Statusmeldungen bei der Initialisierung der Applikation (Aufbau der GUI-Elemente)
- Meldungen über Benutzerinteraktionen, die *nicht* das Spiel betreffen
  - Öffnen/Schliessen eines Dialogs
  - Mausbewegungen

`DEBUG`

- Allgemein: Meldungen, die bei der Fehlersuche hilfreich sein könnten
- Meldungen über Benutzerinteraktionen, die das Spiel betreffen
  - Hinzufügen/Entfernen einzelner Zellen
  - Auswahl von Parametern wie Geschwindigkeit und Form

`INFO`

- Allgemein: Meldungen zum *groben* Nachvollziehen des Ablaufs der Applikation
  - Meldungen über das Starten oder Beenden einer Simulation
  - Meldungen über das Entstehen einer Generation und die Anzahl darin lebendiger Zellen
  - Meldungen über das Starten/Beenden der Applikation

`WARN`

- Allgemein: Meldungen über mögliche Probleme, die das Weiterlaufen der Applikation nicht gefährden.
  - Skalierung der Benutzeroberfläche auf ein kritisches Level
    - zu klein: es können keine sinnvollen Simulationen mehr durchgeführt werden
    - zu gross: die Simulation (oder das feingranulare Loggen) könnte zu aufwändig werden
  - Auftreten einer `ShapeException`

  - Meldung, dass sich die neue Generation nicht mehr von der letzten (bzw. vorletzten) Generation unterscheidet und ein Weiterlaufen der Simulation sinnlos ist.

ERROR

- Allgemein: Unerwartete Probleme, in der Regel abgefangene Exceptions, von denen sich eine Anwendung «erholen» kann.
  - `IOException` beim Öffnen/Speichern von Spielständen
  - `InterruptedException` bei der Behandlung von Threads

FATAL

- Allgemein: Unerwartete Probleme, die ein Weiterlaufen einer Applikation unmöglich oder gar gefährlich (Datenverlust, Hardwarebeschädigung) machen. (Manche Logger-Frameworks behandeln Nachrichten auf dieser Stufe mit einem sofortigen Stopp des Prozesses – nachdem die Nachricht geloggt wurde!)
  - Fehler, die im obersten `catch`-Block abgefangen werden (bei GUI-Applikationen oberhalb des Event-Loops)
  - Speziell vordefinierte Exceptions, die bei Auftreten ein sofortiges aber geordnetes Beenden der Applikation zur Folge haben sollen.