Hausarbeit - Zelluläre Automaten

I143 Praxis der Softwareentwicklung

Gamradt, Plazek, Welter, Prof. Dr. Zimmermann

12.10.2020

1 Allgemeines

Diese Aufgabe ist Prüfungsleistung für das Modul I143 Praxis der Softwareentwicklung. Die Aufgabe wird im Team bearbeitet. Lesen Sie diese Beschreibung sorgfältig durch. Bitte beachten Sie, dass die Umsetzung der Aufgabe im Team auch Teil der Bewertung ist.

Die folgenden Artefakte sind Bestandteil der abzugebenden Dokumentation

- Eine Installations- und Nutzungsanleitung.
- Eine Erläuterung der OO Struktur Ihrer Business-Klassen (UML Klassendiagramm) mit Darlegung der Wart-, Wiederverwend- und Erweiterbarkeit.
- Eine Selbstreflexion (kein Feedback) von jedem Gruppenmitglied: was ist gut gelaufen, was machen Sie im nächsten Projekt anders.
- Eine Liste der genutzten Quellen und Hilfsmittel.
- Eine von allen Gruppenmitgliedern unterschriebene eidesstattliche Erklärung, dass die Aufgabe alleine nur unter Nutzung der genannten Hilfsmittel entwickelt wurde.
- Der Java Source-Code. Der formatierte Source-Code muss gut auf einer DIN A4-Seite hochkant lesbar sein. Dabei ist zu beachten:
  - Der Code ist knapp zu dokumentieren, d. h. jede Klasse hat einen Kommentar, der die Aufgaben und die Kollaborationen der Klasse beschreibt. Methoden- und Parameternamen sollen so sprechend sein, dass ein Kommentar nicht erforderlich ist. Bedenken Sie, dass Methoden einfach und möglichst kurz sein sollen.
  - Die Aufgabe ist eine Gruppenarbeit, in der auch der Teamaspekt eine Rolle spielt. Dennoch müssen gemäß PO §12(1) bei einer Gruppenarbeit die individuellen Leistungen deutlich und verständlich sein. Dazu ist im Source-Code in jeder Klasse mindestens ein (Zahlwort!) Autor anzugeben. Bei

den Git-Commits achten Sie darauf, dass Ihre jeweilige NORDAKADEMIE Nutzerkennungen hinterlegt ist.

- Eine gute Methode um Aktivität nachzuweisen, sind die in gitlab angebotenen Graphen. Wenn diese Angaben fehlen, müssen alle Teilnehmer mit 5,0 bewertet werden!
- Aufgrund der dokumentierten Aktivität können bei wesentlichen Abweichungen der Leistungen die Noten einzelner Teammitglieder unterschiedlich sein. Sorgen Sie deshalb insbesondere beim Pair-Programming dafür, dass beide Paarmitglieder abwechselnd committen.

2 Formales

Der Code muss im gitlab2.nordakademie.de entwickelt werden.

Die Nutzung anderer zentraler git Repositories ist nicht zulässig. Sollte der Server nicht verfügbar sein, informieren Sie umgehend Ihren Dozenten.

Das Projekt muss im gitlab und in der Entwicklungsumgebung den Namen "hausarbeit_i143_<dozent>_<gruppe>" (alles in Kleinbuchstaben) erhalten. Verwenden Sie private Gitlab-Projekte und geben Sie **nur** den Gruppenmitgliedern und dem Prüfer die nötigen Rechte.

Tipp: Wählen Sie eine hohe Commit-Frequenz mit aussagekräftigen Commit-Kommentaren.

Als Teil des Prüfungsprozesses werden ausgewählte Studierende gebeten, ihr Programm im Detail zu erklären, um Zweifel an der Urheberschaft zu klären.

Beginn der Bearbeitung: 12.10.2020, 00:00 Uhr
Spätester Abgabetermin: 02.11.2020, 23:59 Uhr

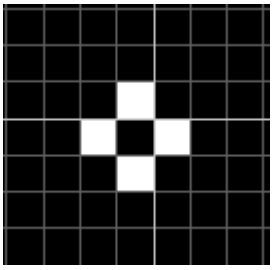
Die Dokumentation soll als druckfertiges DinA4 Dokument im PDF-Format in das root Verzeichnis Ihres gitlab2 Projekts geladen werden.

Ihre Abgabe erfolgt per E-Mail an den Dozenten mit dem Namen des Repositories und der letzten Commit ID.

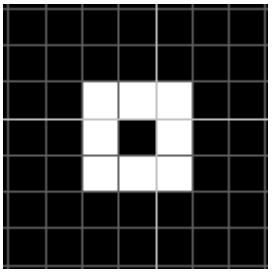
3 Aufgabenstellung

Berechnung von 2-dimensionalen zellulären Automaten

Zelluläre Automaten sind eine Möglichkeit Systeme zu modellieren. Die entstehenden Modelle bestehen aus n*m Zellen (das "Grid"), die jeweils den gleichen Regelsatz und Zustandsraum aufweisen. Die Regeln basieren auf dem eigenen Zustand und den Zustand der benachbarten Zellen. Die verwendete Nachbarschaft soll zwischen "von-Neumann" (links, rechts, oben, unten; also vier benachbarte Zellen)



und "Moore" (links, rechts, oben, unten, linksoben, rechtsoben, linksunten, rechtsunten; also acht benachbarte Zellen)



umgestellt werden können. An den Rändern wird die Nachbarschaft "beschnitten". Der Zustandsraum einer Zelle beinhaltet die Zustände "lebendig = 1" und "tot = 0".

Pro Schritt wird der Zustand einer jeden Zelle neu berechnet. Dabei kann der Zustand zu t-1 von allen berücksichtigten Zellen (eigener und der der Nachbarn) verwendet.

```
MIT JEDER ZELLE DES GRIDS TUE
        ZELLE.ZUSTAND (T) =
                REGELSATZ (ZELLE.ZUSTAND (T-1),
                        ZUSTAND DER NACHBARZELLEN (T-1))
```

Ihre Aufgaben

1. Erstellen Sie Schnittstellen für Ihre zentralen Klassen und dokumentieren Sie den Klassenentwurf in Ihrer Dokumentation.
2. Entwickeln Sie zwei alternative Datenstrukturen mit Java-Klassen zur Verwaltung des Grids im Hauptspeicher. Entwickeln Sie diese Test First.
3. Implementieren Sie einen Algorithmus zur Berechnung von zellulären Modellen. Entwickeln Sie diesen Test First.
4. Setzen Sie das Game-Of-Life Modell und das Parity Modell (siehe Anhang) um.
5. Berechnen Sie die ersten 100 Schritte basierend auf den im Anhang befindlichen Experimentdefinitionen und beiden Datenstrukturen (siehe 2.) und geben Sie das Ergebnis in einer Datei aus (siehe Punkt 7).

Legen Sie die Experimente in getrennten Klassen an (Namensschema: ("GameOfLife" | "Parity") "_Experiment_" Nummer "_" ("1"|"2")). (1 oder 2 für die Datenstrukturen aus 2.) Die von den Experimenten geschriebene Log-Datei soll die Endung .log haben und so heißen, wie das Experiment. Sie soll im Wurzelverzeichnis Ihres Projekts liegen.

6. Es soll mehrere Varianten für die Abbruchbedingung ihrer Berechnung geben:
   a) Abbruch nach einer festen Anzahl an Schritten.
   b) Abbruch, wenn sich der Zustand keiner Zelle mehr ändert.
   c) Abbruch, wenn a oder b eintritt, was auch immer zuerst passiert.
7. Die Ausgabe der Simulation soll in ein Log erfolgen. Die vorgeschriebene Formatierung des Logs ist im Anhang definiert. Auch hier soll es mehrere Varianten geben:
   a) Ausgaben werden auf die Konsole geschrieben.
   b) Ausgaben werden in eine Datei geschrieben.
   c) Ausgaben werden auf die Konsole und in eine Datei geschrieben.
   d) Ausgaben werden gar nicht geschrieben.

Weitere Informationen zu zellulären Automaten können Sie z.B. auf der Seite von Wolfram finden. Ein eindrucksvolles Beispiel für die Mächtigkeit stellt der Wireworld Primzahlcomputer dar.

Weitere Anforderungen

1. Gehen Sie bei der Entwicklung der Geschäftslogik (Business Logik) Test-First vor.
2. Die Programmiersprache ist Java. Die zu verwendende Java-Version ist 8. Als Referenzinstallation zählt die Installation der in der Vorlesung genutzten Entwicklungsumgebung der VDI der NORDAKADEMIE.
3. Der Einsatz des Mocking Frameworks Mockito ist verpflichtend.
4. Nutzen Sie die in der Vorlesung erklärten objektorientierten Techniken, wenn immer es Ihnen möglich ist. Wiederverwendbarkeit, Erweiterbarkeit und Wartbarkeit sind wesentliche Bewertungskriterien.
5. Eine Benutzeroberfläche sowie eine parallele Verarbeitung sind **nicht** gewünscht.

Fragen müssen im Moodle Diskussionsforum gestellt werden, damit diese von allen eingesehen werden können. Bitte nicht per E-Mail direkt an den Dozenten. Bitte beachten Sie, dass Sie in der Rolle des Auftragnehmers Unklarheiten aktiv beseitigen müssen. Ohne Kommunikation ist das in der Regel nicht möglich.

4 Wie kann ich bei dieser Prüfungsleistung durchfallen?

Dazu gibt es viele verschiedene Möglichkeiten, einige sind:

- Abgabetermin versäumen.
- Mangelhafte oder unglaubwürdige git-Historie.
- Keine Erklärung oder keine Zuordnung der individuellen Leistung möglich.
- Dokumentation fehlt, ist unlesbar oder unangemessen. Eine gute Orientierung für Angemessenheit ihrer Dokumentation bieten die Transferleistungen.
- Unleserlicher, nicht formatierter Quellcode: strategie X= new strategie(Q,W,t17),
- Programm zeigt Compilefehler, die nicht auf unterschiedliche Plattformen zurückzuführen sind. Testen Sie Ihr Projekt auf der Referenzplattform.
- Ein Programm, das nicht oder offensichtlich fehlerhaft läuft.
- Ein Programm, das OO-Regeln ignoriert (z. B. Klassen und/oder Methoden erfüllen mehrere Aufgaben).
- Fehlende oder überflüssige Tests.

Eine Übersicht der Bewertungskriterien ist in den Folien der letzten Veranstaltung aufgeführt.

5 Anhang - Game of Life

Regelsatz

1. Eine lebendige Zelle bleibt lebendig, wenn Sie genau 2 oder 3 lebendige Nachbarn besitzt. Andernfalls stirbt sie.
2. Eine tote Zelle bleibt tot bis sie exakt 3 lebendige Nachbarn hat.

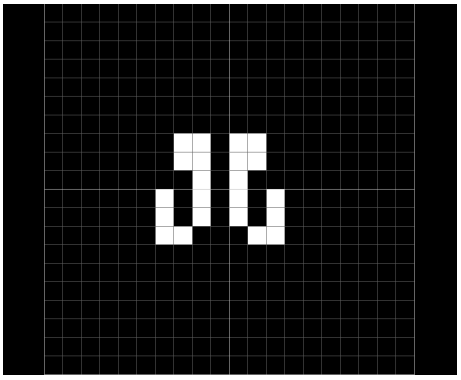
5.1 Startkonfiguration

Konfiguration 1

Grid-Size: 40x41

Nachbarschaft: Moore

Mittig platziert werden soll folgendes Muster:



Konfiguration 2

Grid-Size: 100x100

Gerade Zeilen werden abwechselnd mit "tot" "lebendig" initialisiert.
Ungerade Zeilen werden abwechselnd mit "lebendig" "tot" initialisiert.

Nachbarschaft: Moore

Konfiguration 3

Grid-Size: 300x300

Alle Zellen sind "tot"

Nachbarschaft: Moore

6 Anhang - Parity Modell

Regelsatz

- Eine Zelle ist genau dann lebendig, wenn in ihrer von-Neumann Nachbarschaft eine ungerade Anzahl lebendiger Zellen existiert, andernfalls ist sie tot.

6.1 Startkonfiguration

Konfiguration 1

Grid-Size: 400x400

Alle Zellen werden mit "tot" initialisiert. Lebendig sind lediglich 4 Zellen in der Mitte.

Konfiguration 2

Grid-Size: 100x100

Gerade Zeilen werden abwechselnd mit "tot" "lebendig" initialisiert.
Ungerade Zeilen werden abwechselnd mit "lebendig" "tot" initialisiert.

7 Anhang - Vorlage für die Formatierung der Log-Ausgabe (Konsole / Datei)

Die folgende Ausgabe ist für einen 10x7 zellulären Automaten und zwei aufeinanderfolgende Zustände (n und n+1). Tot soll mit 0, lebendig mit 1 codiert werden. Zwischen den Ausgaben des Zustands soll "###" gefolgt von der Schrittnummer ausgegeben werden. Die erste Zeile / erste Spalte beginnt links oben.

```
...

### (n)
0000000000
0000000000
0000000000
0000010000
0000000000
0000000000
0000000000

### (n+1)
0000000000
0000000000
0000010000
0000111000
0000010000
0000000000
0000000000

...
```