**Reflexion Prüfungsarbeit Programmieren „Game of Life“ Mat.Nr. 4434822**
Weitere Projektmitglieder: 2883670, 8254649, 8388391

Aufgabenstellung
Die Aufagenstellung eine eigene Version des bekannten „Game of Life“ zu implementieren stellte sich auf Grund der guten Zusammenarbeit in unserem Team als recht einfach dar. Da die Teammitglieder auf Grund von Wohnort/Arbeitsstelle recht weit verstreut waren, entschieden wir uns unser Projekt mittels Git als Versioningsystem zu koordinieren. Als Hoster für ein online Repository fiel die Wahl auf den bekannten Anbieter GitHub.com, da mehrere Teammitglieder hier bereits Zugänge hatten.

Aus der anfänglichen Aufgabenverteilung habe ich die Implementierung der Regeln welche von der Simulation-Engine verwendet werden und das implementieren der „Hauptklasse“ (GameOfLife.java) übernommen.

Entscheidungen
Bei der Implementierung der Regeln verwendete ich eine abstrakte Klasse „Rules“ welche ein Vorbild für Regeln darstellt. Um die „Standardregeln“ des Game of Life abzubilden habe ich die Klasse RPopulation erstellt. Die Abfrage der Regel-Bedingungen erfolgt dabei mit einfachen if-else-Blöcken. Für das zählen der lebenden Zellen habe ich das neighbours-Array in einen Stream umgewandelt und gefiltert, da dies eine sehr übersichtliche und elegante Lösung darstellt.
Bei der Hauptklasse fielen die Schwerpunkte auf das Parsen der Übergabeparameter, den Start der Engine und das Fehlermanagement.
Da die GameOfLife Klasse die Schnittstelle zwischen den restlichen Komponenten darstellt, fange ich hier alle Exceptions ab, welche zu einem Abbruch der Ausführung des Programms führen. Dabei wird eine Fehlermeldung auf den Fehlerstream geschrieben und das Programm mit dem Exitcode von „-1“ beendet. Hierfür habe ich eine eigene Methode verwendet, da das Programm so an nur einer einzigen Stelle im Code beendet wird, wodurch sich etwaiges Troubleshooting vereinfacht.
Um die Zeit, welche zwischen zwei Generationen gewartet wird exakt zu halten, wird vor und nach der Ausführung der Engine eine Zeitmessung vorgenommen und das entsprechende Delta von der Wartezeit abgezogen. So ist die gewartete Zeit auch auf langsameren Systemen korrekt (mit Ausnahme, wenn die Wartezeit kleiner als die Zeit der Ausführung ist, dann ist das System selbst der begrenzende Faktor, es wird nicht mehr gewartet.)

Erweiterbarkeit
Auf Grund der geschaffenen Rule-Klasse, kann das Programm recht einfach um weitere Regeln erweitert werden. Falls jedoch andere Kriterien als die Bevölkerungsdichte der Zellen als Grundlage für die Regelbearbeitung verwendet werden sollen, so muss mehr Code verändert werden. Auf Grund des Designs sind die notwendigen Anpassungen jedoch einfach zu identifizieren.

Zusammenarbeit
Dadurch, dass ich mit der Hauptklasse die den gemeinsamen Punkt für die Klassen der anderen Teammitglieder implementiert habe, musste meine Abstimmung natürlich besonders eng gehalten werden. Da die Kommunikation über entsprechende Kanäle im Internet jedoch problemlos war und alle Mitglieder des Teams hoch motiviert gearbeitet haben, lief auch hierbei alles ideal.