Hausarbeit im Modul I143 Praxis der Softwareentwicklung

A18a

Falko Partzsch (Matrikelnummer: 8879)
Merlin Ritsch (Matrikelnummer: 8772)
Niklas Witzel (Matrikelnummer: 8712)
Til Zöller (Matrikelnummer: 9366)

4. November 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Installationsanleitung

Folgende Schritte sind zur Ausführung der Experimente im Projekt nötig:

1.1 Schritt 1

Klonen des Git-Repository mithilfe folgender git-Befehle.

```
git clone
https://gitlab2.nordakademie.de/nwitzel/hausarbeit_i143_gamradt_cl.git
```

1.2 Schritt 2

Öffnen des Projekts in einer geeigneten Java IDE. Im Folgenden werden sich die Instruktionen allerdings nur auf IntelliJ beziehen.

1.3 Schritt 3

Das Projekt nutzt Java Version 8. Daher muss sichergestellt werden, dass das Projekt SDK und das Language Level auf Java Version 8 eingestellt ist. Überprüft werden kann dies in den Projekteinstellungen.

1.4 Schritt 4

Nun müssen die Projekt-Dependencies mittels Maven installiert werden. Dazu sollte folgender Maven Befehl ausgeführt werden:

```
mvn clean install
```

2 Nutzungsanleitung

Um ein vordefiniertes Experiment auszuführen, müssen folgende Schritte durchgeführt werden.

2.1 Schritt 1

Öffnen der Main Klasse. Diese beinhaltet eine main-Methode, in der ein Experiment konfiguriert und ausgeführt wird.

2.2 Schritt 2

Konfigurieren und Instantiieren der Abbruchbedingung, die im Experiment genutzt werden soll. Dafür stehen die folgenden Implementationen zur Verfügung.

1. **NoChange** - Das Experiment wird endet, wenn sich der Zustand jeder Zelle in der nächsten Generation des Grids nicht verändert
2. **FixedNumber(n)** - Das Experiment endet nach n Generationen
3. **FixedNumberOrNoChange(n)** - Das Experiment endet, wenn 1. oder 2. zu treffen, je nachdem was als erstes zutrifft

Ein Beispiel für die Instantiierung eines Log-Targets sieht folgendermaßen aus:

```
IAbortCondition abortCondition = new NoChange();
```

Achtung: Bei der Auswahl der Abbruchbedingung sollte beachtet werden, dass ein Experiment unter Umständen nicht beendet wird und unendlich lange ausgeführt wird.

2.3 Schritt 3

Konfigurieren und Instantiieren des Loggers, der im Experiment genutzt werden soll. Dafür stehen die folgenden Implementationen zur Verfügung.

1. **ConsoleLogger** - die Ausgabe des Experiments wird auf die Konsole geschrieben
2. **FileLogger** - die Ausgabe wird in eine Logdatei im Projektroot mit dem Namen des Experiments geschrieben
3. **ConsoleAndFileLogger** - die Ausgabe des Experiments wird auf die Konsole und in eine Logdatei im Projektroot mit dem Namen des Experiments geschrieben
4. **NoneLogger** - die Ausgabe des Experiments wird nicht geschrieben

Ein Beispiel für die Instantiierung eines Log-Targets sieht folgendermaßen aus:

```
IExperimentLogger experimentLogger = new ConsoleLogger();
```

2.4 Schritt 4

Konfigurieren und instantiieren Sie das gewünschte Experiment mit Abbruchbedingung und Log-Target. Dafür stehen die folgenden Experimente zur Auswahl:

1. **GameOfLife-Experiment-1-1** - Grid = Grid2DArray(cols=41, rows=40, neighborhood=Moore, startingPattern=TumblerPattern), Ruleset=GameOfLife
2. **GameOfLife-Experiment-1-2** - Grid = Grid2DList(cols=41, rows=40, neighborhood=Moore, startingPattern=TumblerPattern), Ruleset=GameOfLife
3. **GameOfLife-Experiment-2-1** - Grid = Grid2DArray(cols=100, rows=100, Nachbarschaft=Moore, startingPattern=ChessFieldPattern), Ruleset=GameOfLife
4. **GameOfLife-Experiment-2-2** - Grid = Grid2DList(cols=100, rows=100, neighborhood=Moore, startingPattern=ChessFieldPattern), Ruleset=GameOfLife
5. **GameOfLife-Experiment-3-1** - Grid = Grid2DArray(cols=300, rows=300, neighborhood=Moore, startingPattern=AllDeadPattern), Ruleset=GameOfLife
6. **GameOfLife-Experiment-3-2** - Grid = Grid2DList(cols=300, rows=300, neighborhood=Moore, startingPattern=AllDeadPattern), Ruleset=GameOfLife
7. **Parity-Experiment-1-1** - Grid = Grid2DArray(cols=400, rows=400, neighborhood=VonNeumann, startingPattern=SquarePattern), Ruleset=Parity
8. **Parity-Experiment-1-2** - Grid = Grid2DList(cols=400, rows=400, neighborhood=VonNeumann, startingPattern=SquarePattern), Ruleset=Parity
9. **Parity-Experiment-2-1** - Grid = Grid2DArray(cols=100, rows=100, neighborhood=VonNeumann, startingPattern=ChessFieldPattern), Ruleset=Parity
10. **Parity-Experiment-2-2** - Grid = Grid2DList(cols=100, rows=100, neighborhood=VonNeumann, startingPattern=ChessFieldPattern), Ruleset=Parity

Ein Beispiel für die Instantiierung eines Experiments sieht folgendermaßen aus:

```
IExperiment experiment = new GameOfLife_Experiment_1_1(abortCondition,
experimentLogger);
```

2.5 Schritt 5

Aufrufen der run Methode auf dem zuvor instantiierten Experiment. Ein Beispiel dafür sieht wie folgendermaßen aus:

```
experiment.run();
```

2.6 Kurzfassung der Schritte 2 bis 5

Die obigen Schritte 2 bis 5 lassen sich auch in einer einzigen Zeile zusammenfassen. Das sieht beispielsweise wie folgt aus:

```
new GameOfLife_Experiment_1_1(new NoChange(), new ConsoleLogger()).run();
```

2.7 Schritt 6

Anlegen einer Startkonfiguration für die main-Methode in der Main-Klasse.

2.8 Schritt 7

Startkonfiguration ausführen.

3 Erläuterung der OO-Struktur

In den folgenden Abschnitten wird zunächst die OO-Struktur des Projekts in einem Top-Down Ansatz dargestellt, ehe in einem zweiten Kapitel die Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit der einzelnen Projektmodule erläutert wird. Zusätzlich zu der folgenden schriftlichen Darstellung, kann die Objektorientierte-Struktur mittels des UML-Diagramms im Anhang eingesehen werden.

3.1 Experiment

Das Konstrukt Experiment dient dazu einen Versuchsaufbau zu erstellen. Es setzt die einzelnen Komponenten **Grid**, **Ruleset**, **AbortCondition** und **Logger** zu einer Experiment-Konfiguration zusammen.

Die Schnittstelle aller Experimente wird über das Interface **IExperiment** definiert und die abstrakte Klasse **Experiment** fasst alle gemeinsamen Attribute und Methoden eines Experiments zusammen. Jedes Experiment kann somit maximal flexibel konfiguriert werden. Jedes Experiment besitzt eine run()-Methode, um das jeweilige Experiment auszuführen.

Die von der Aufgabenstellung geforderten unterschiedlichen Experiment-Konfigurationen sind als eigene Klassen realisiert worden, die jeweils von der abstrakten Klasse **Experiment** erben. In diesen Klassen wird innerhalb des Konstruktors die geforderte Konfiguration, bestehend aus einem Grid und einem **Ruleset**, festgelegt. Eine **AbortCondition** und ein **Logger** müssen jeweils direkt beim Instantiieren der vordefinierten Experiment-Klasse vorgegeben werden und sind somit pro Experimentklasse individuell konfigurierbar. Alle Funktionen, die unabhängig von der spezifischen Konfiguration sind, wurden in die abstrakte Oberklasse **Experiment** ausgelagert, um eine Code-Redundanz zu vermeiden.

3.2 Grid

Das Konstrukt **Grid** persistiert den aktuellen Zustand aller Zellen eines 2-dimensionalen zellulären Automaten. Der Zustand, den eine Zelle haben darf, wird mittels des Enums **State** beschrieben.

Das Grid befindet sich eine Ebene unter den Experimenten. Es wird analog zum Experiment über ein Interface **IGrid**, welches die Schnittstelle aller Grid-Implementationen darstellt, und eine abstrakte Klasse **Grid**, welche alle gemeinsamen Attribute und Methoden eine Grids zusammenfasst, abstrahiert.

Die Struktur des dem Experiment übergebenen Grids wird in dem Interface **IGrid** vorgegeben, teilweise durch die abstrakte Klasse Grid implementiert und durch die beiden konkreten Klassen **Grid2DArray** und **Grid2DList** weiter spezifiziert. **Grid2DArray** und **Grid2DList** unterscheiden sich in der zugrunde liegenden Datenstruktur, sind jedoch dank des **IGrid**-Interfaces identisch nutzbar.

3.3 GridConfiguration

Ein Grid braucht immer zwangsläufig auch eine **GridConfiguration**, welche dessen Eigenschaften verwaltet. Zu den Eigenschaften eines **Grids** zählt die Größe (Spalten und Reihen) des Grids, die **Neighborhood** und das initiale **Pattern**. Auf Grund dieses engen Verhältnisses der beiden Konstrukte, ist das Interface **IGridConfiguration** ein inneres Interface des **IGrids**. In der abstrakten Klasse **Grid** findet sich die innere Klasse **GridConfiguration**. Im Konstruktor des Grids wird bei

dessen Initialisierung immer eine **GridConfiguration** mit den übergebenen Parametern erstellt. So ist es nicht notwendig zur Initialisierung eines Grids die GridConfiguration selbstständig erstellen zu müssen.

3.4 Ruleset

In den Rulesets werden die jeweiligen Spielregeln für Game Of Life und Parity definiert.

Auch hier wird das Konstrukt aus einem Interface, einer abstrakten Klasse und mehreren, von der abstrakten Klasse erbenden, Klassen eingesetzt. Die jeweiligen Ruleset-Klassen **GameOfLife** und **Parity** implementieren das Interface **IRuleset**. Das Ruleset-Konstrukt ist eine Ebene unter den Experimenten und wird von diesen genutzt. Mittels dem Ruleset und dem Grid, sowie der Neighborhood kann die nächste Generation des Grids berechnet werden.

3.5 AbortCondition

Das Konstrukt **AbortCondition** legt die Abbruchbedingung eines **Experiments** fest. Eine **AbortCondition** kann beim Instantiieren einer Experiment-Klasse übergeben werden und somit dynamisch je Experiment-Instanz angepasst werden.

Strukturell befinden sich die **AbortConditions** direkt unter den **Experimenten**. Diese erwarten das Interface **IAbortCondition** übergeben zu bekommen, welches von allen **AbortConditions** implementiert wird. Durch diesen Aufbau können einfach weitere Abbruchbedingungen ergänzt werden.

Grundsätzlich gibt es zwei Typen von Abbruchbedingungen. Die simplen **AbortConditions** benötigen keine weiteren Eingaben. Hier existiert auch keine abstrakte Klasse, da es keinen gemeinsam nutzbaren Code gibt, welcher ausgelagert werden könnte. Zu den simplen **AbortConditions** gehört die **NoChange** Klasse / Abbruchbedingung. Im Gegensatz dazu stehen die komplexen **AbortConditions**, die gemeinsam haben, dass sie eine maximale Anzahl von Durchläufen enthalten. Daher erben sie von der abstrakten Klasse **ComplexAbortCondition**. Als **ComplexAbortCondition** stehen **FixedNumber** und **FixedNumberOrNoChange** zur Verfügung.

3.6 Logger

Das Interface **IExperimentLogger** dient als zentrale Schnittstelle für alle Implementierungen von Logging-Algorithmen. Unabhängig vom Ziel, auf welches geloggt werden soll, stehen so dieselben Methoden zur Verfügung.

Das Logging befindet sich eine Ebene unter dem Experiment. Die Struktur des Loggings wird durch das Interface **IExperimentLogger** vorgegeben und dient allen Implementierungen als zentrale Schnittstelle. Zur Implementierung der Logging-Algorithmen wird die Logback Library verwendet. Die abstrakte Klasse **ExperimentLogger** implementiert für alle konkreten Klassen identische Methoden und Attribute, sodass die Kind-Klassen sich ausschließlich auf das Hinzufügen der notwendigen Zielmedien konzentrieren können und nahezu keine eigene Logik aufweisen müssen.

Als konkrete Logger-Implementierungen stehen der **NoneLogger**, der **ConsoleLogger**, der **FileLog-**

ger und der **ConsoleAndFileLogger** zur Verfügung.

3.7 Neighborhood

Das Konstrukt **Neighborhood**, ebenfalls umgesetzt mittels eines Interfaces **INeighborhood**, einer abstrakten Klasse **Neighborhood** und den Implementationen **Moore** und **VonNeumann**, stellt die verschiedenen Nachbarschafts-Relationen dar. Diese werden durch das Grid-Konstrukt genutzt und ist eine eine Ebene unter dem Grid einzuordnen.

Eine Neighborhood-Klasse bietet anschließend die Funktion, mittels des übergebenen Grids des Typen IGrid, die Anzahl der lebendigen Zellen um eine angegebene Zelle herum zu berechnen. Für die Berechnung der Anzahl der lebenden Zellen wird das Enum **Neighbor** genutzt, welches den benötigten Offset zur Berechnung der Koordinaten der nächsten Nachbarzelle beinhaltet. Eine Liste von **Neighbors** wird in den einzelnen **Neighborhood**-Klassen erstellt. Die abstrakte Klasse implementiert anschließend mittels dieser übergebenen List die eigentlichen Funktionen des **INeighborhood**-Interfaces. Diese Struktur ermöglicht eine leichte Erweiterung der möglichen Nachbarn einer Zelle, sowie der möglichen Nachbarschaftsrelationen einer Zelle.

3.8 Pattern

Im Konstrukt **Pattern** werden verschiedene Muster vorab definiert, welche beim initialen Erstellen eines **Grids** der **GridConfiguration** übergeben werden. Sie definieren die erste Generation des Automaten.

Über das Interface **IPattern** implementieren alle **Pattern** Klassen eine Funktion zur Anwendung auf ein **Grid**. Die Pattern sind unterteilt in simple und komplexe Kategorien. Simple Pattern benötigen keine Eingabewerte und können auf jedes **Grid** angewendet werden. Im Sinne der Erweiterbarkeit können simple Pattern ohne Probleme modifiziert oder hinzugefügt werden. Komplexe **Patterns** sind Muster auf der Basis eines vollständig toten **Grids**. Sie haben die Eigenschaften einer Mindestgröße und einer Startposition gemein. Diese werden über Interfaces in der abstrakten Klasse **ComplexPattern** implementiert. Die komplexen Patterns erben von dieser abstrakten Klasse. Der Aufbau sorgt somit für eine gute Erweiterbarkeit.

Als simple Patterns stehen das **AllDeadPattern** und das **ChessFieldPattern** zur Verfügung. Als komplexe Patterns können das **TumblerPattern** und das **SquarePattern** verwendet werden.

4 Darlegung von Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit

4.1 Erweiterbarkeit

In allen Aspekten des Projektes wird bestmögliche Modularität und Erweiterbarkeit gewährleistet. In diesem Sinne spezifiziert stets ein Interface die Schnittstelle von Code-Abschnitten zu anderen Teilen des Programms. Zusätzliche Implementierungen von neuen Datenstrukturen des Grids, Nachbarschafts-

beziehungen, Regelsätzen, Abbruchbedingungen, Mustern, Logging-Zielen oder ganzer Experimente findet dementsprechend stets nach den vordefinierten Methoden der entsprechenden Interfaces statt. Die Funktionsfähigkeit des Programms wird durch Ergänzung neuer Algorithmen nicht eingeschränkt und ist durch die starke Modularität der Experimente weiterhin gewährleistet.

4.2 Wartbarkeit

Konkrete Implementierungen identischer Methoden wurden stets in abstrakte Klassen ausgelagert, damit Änderungen an diesen, während eines Wartungs-Prozesses, nicht in mehreren Klassen getätigt werden müssen. Existieren keine identischen Methoden oder Attribute, wird auf eine abstrakte Klasse verzichtet und das Interface direkt von den konkreten Klassen implementiert. Ein Beispiel hierfür sind die für simple Patterns verantwortlichen Klassen.

Auch die Wahl kurzer und eindeutiger Funktions- und Methodenbezeichner, sowie Klassennamen erleichtern den Einstieg in das Projekt bei Erweiterungen und Wartungen durch Dritte.

Ebenfalls wurde bei der Umsetzung der Aufgabenstellung darauf geachtet, dass jede Klasse jeweils nur eine Aufgabe erledigt, sodass ungewollte Nebeneffekte nicht auftreten können.

4.3 Wiederverwendbarkeit

Auf Grund der Programmierung in der Hochsprache Java ist das Programm auf allen Java-fähigen Geräten ausführbar. Eingeschränkt wird diese Wiederverwendbarkeit dadurch, dass es sich um keine exportierte executable-Jar handelt, sondern um Source Code, dessen Konfiguration die Nutzung einer IDE erfordert. Die Installation und Bedienung ist dementsprechend einer speziellen Zielgruppe mit der entsprechenden Software und Wissensstand vorbehalten und das Programm ist somit nicht für jeden Anwender simpel auf andere Umgebungen übertrag- und ausführbar.

Im Hinblick auf die objektorientierte Struktur können einzelne Komponenten, dank der hohen Modularität, auch in anderen Java-Software-Projekten wiederverwendet werden. Auch die Nutzung von abstrakten Klassen hat die Wiederverwendbarkeit einzelner Funktionen gesteigert und die Implementierung weiterer, von der abstrakten Klasse erbenden Klassen, vereinfacht.

5 Selbstreflexionen

5.1 Falko Partzsch

Dies war für mich die erste umfangreiche Projektarbeit mit einem hohen Programmieraufwand. Dennoch empfand ich die Arbeit in dieser Gruppe als sehr angenehm. Wir hatten keine großen Probleme und konnten uns gegenseitig gut ergänzen und helfen. Nichtsdestotrotz gab es wie zu erwarten Probleme in der Ausführung. Einige Abschnitte des Programms konnten nicht parallel implementiert werden, ohne unangenehme Merge-Konflikte zu erzeugen. Das hat dazu geführt, dass regelmäßig auf die Fortschritte anderer Gruppenmitglieder gewartet werden musste. Da wir rechtzeitig zu Beginn des Semesters mit der Bearbeitung der Aufgaben begonnen haben, stellte dies jedoch kein großartiges Problem dar und hat nicht zu einer Zeitnot geführt. Dieses Vorgehen sollte definitiv für zukünftige

Projektarbeiten beibehalten werden. Während der Bearbeitung haben regelmäßige Meetings stattgefunden, bei denen neue Ziele gesetzt, Aufgaben verteilt und Hilfestellung geleistet wurde.Sowohl die Organisation, als auch die Zusammenarbeit war sehr angenehm und ich werde sie als positives Beispiel für weitere Kooperationen im Gedächtnis behalten.

5.2 Merlin Ritsch

Die gestellten Teilaufgaben wurden meiner Meinung nach effizient gelöst. Mir persönlich hat dabei die Nutzung der zur Verfügung stehenden Zeit besonders gut gefallen. Die Aufgaben wurden in gleichmäßigen Teilen über den Bearbeitungszeitraum bearbeitet, sodass nicht unter Stress gearbeitet werden musste.

Außerdem gefallen hat mir die Zusammenarbeit im Team. Wir haben uns regelmäßig besprochen, um auf dem neuesten Stand zu bleiben. Zudem ist viel Code im Verfahren des Pair-Programmings entstanden, damit die Fehlerquote so gering wie möglich bleibt. Leider haben wir es dabei versäumt, regelmäßig abwechselnd zu committen. Deshalb sind die @author Einträge im Code häufig aussagekräftiger als die Git-Historie. Mir ist außerdem bei drei Commits am 22. Oktober 2020 ein Fauxpas unterlaufen, als ich auf einem anderen PC gearbeitet habe und als “Merlin Ritsch” statt als “16513” committed habe.

Darüber hinaus haben sich gelegentlich Wartezeiten ergeben, als mehrere Gruppenmitglieder gleiche Komponenten anpassen wollten. Da wir im Ergebnis so Merge-Konflikte vermeiden konnten, haben wir die geringen Wartezeiten in Kauf genommen.

In der Planungsphase des Projektes war Niklas Witzel federführend beteiligt. Dies liegt an seinem bemerkenswerten Vorwissen. Dennoch nehme ich für zukünftige Projekte mit, mich in allen Phasen des Projektes gleichermaßen einzubringen. Beibehalten würde ich zukünftig die vielen Pair-Programming Sessions, die besonders bei komplizierten Programmabschnitten zielführend waren und die regelmäßigen Austausche über den Fortschritt.

5.3 Niklas Witzel

Grundsätzlich lief die Erstellung der Hausarbeit erstaunlich gut. Direkt zu Beginn, also nach Veröffentlichung der Aufgabenstellung haben wir uns in der Gruppe getroffen und die Aufgaben im Detail durchgesprochen. Dies war extrem hilfreich, weil wir damit ein einheitliches Verständnis über die Aufgabenstellung entwickelt haben und potenzielle Unklarheiten aus dem Weg schaffen konnten.

Im Anschluss daran haben wir ein erstes UML-Klassendiagramm entwickelt und anhand dieser Überlegung unser Modell in mehrere Module aufgeteilt, die jeweils einen hauptverantwortlichen Bearbeiter bekommen haben. Da ich in unserer Gruppe wohl die meisten Vorkenntnisse hatte, habe ich bei der Modellierung einen Großteil der Ideen eingebracht, die jedoch oftmals ohne Kritik übernommen und angewendet wurden. Dies führte an einer Stelle dazu, dass mitten im Entwicklungsprozess Probleme auftraten, die eine tiefgreifende Umstrukturierung des Codes bedingten. Hier sollte ich mir in Zukunft aktiver Feedback einholen, um solche Problemstellungen zu verhindern.

Die Implementation des Codes lief dann bis auf das oben genannte Problem nahezu reibungslos. Wir trafen uns regelmäßig zu Zoom-Sessions, in denen wir sowohl Features gereviewt, Codeabschnitte re-

factored oder neue Features implementiert haben. Hinderlich war teilweise der Fakt, dass Features von anderen Features vorausgesetzt wurden und somit nicht parallel daran gearbeitet werden konnte. Dies stellte aber aufgrund der guten Zeiteinteilung unsererseits kein nennenswertes Problem dar. Mir fehlt an der Stelle auch eine Idee, wie man dies hätte verhindern können.

5.4 Til Zöller

Insgesamt bin ich mit dem Projektergebnis, der Anwendung, sehr zufrieden. Dadurch, dass wir direkt nach der Veröffentlichung der Aufgabenstellung mit dem Projekt begonnen hatten, konnten wir viele Code-Reviews und Diskussionen durchführen und viele Optimierungen und Umstrukturierungen vornehmen, welche der Qualität der Anwendung zugute kam.

Nicht wirklich gut geklappt hat die Organisation beim Pair-Programming. Große Teile des Codes und der Überlegungen haben wir immer zusammen in regelmäßigen Online-Konferenzen erstellt. Leider haben wir meist nur an einem PC via Bildschirmfreigabe gearbeitet und uns viel zu selten bei den Commits abgewechselt. Dies ist auch in den Git-Graphen zu erkennen, obwohl ich bei einigen Komponenten des Codes zu größeren Teilen beteiligt und verantwortlich war, als die Git-Historie vermuten lässt.

Da dies das erste Programmierprojekt des Studiums ist und somit keinerlei bewährte Vorerfahrung zu solch einem Projekt existierte, möchte ich in zukünftigen Projekten mehr darauf achten, auch bei den Git-Commits, und nicht nur in den Code-Dokumentationen, öfter als Autor aufzutauchen.

Abschließend möchte ich anmerken, dass Niklas Witzel dank seines vielen Vorwissens oft hilfreiche Ideen und Vorschläge einbringen konnte. Dies hatte die Effizienz der Projektplanung und -Umsetzung sehr gesteigert.

A Autoren

Kapitel	**Autor**
Installation	Niklas Witzel
Nutzungsanleitung	Niklas Witzel
Experiment	Til Zöller
Grid	Til Zöller
GridConfiguration	Til Zöller
Ruleset	Merlin Ritsch
AbortCondition	Merlin Ritsch
Logger	Falko Partzsch
Neighborhood	Til Zöller
Pattern	Merlin Ritsch
Darlegung von Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit	Falko Partzsch

B Quellen und Hilfsmittel

- Eric Goebelbecker.(2020, 20.Juli).*A Guide to Logback*.https://www.baeldung.com/logback
- Lennart Gamradt, Felix Plazek, Felix Jan Moritz Welter, Frank Zimmermann.(2020).*Foliensatz Modul I143 - Praxis der Softwareentwicklung*
- Marcus Biel.(2016, 23.Juni).*Shallow vs Deep Copy in Java*[Video]. YouTube.https://www.youtube.com/watch?v=QaCYMgyprtc&
- QOS.ch.(2019).*The logback manual*.http://logback.qos.ch/manual/index.html
- The Coding Train.(2017, 11.Dezember).*Coding Challenge #85: The Game of Life*[Video]. YouTube.https://www.youtube.com/watch?v=FWSR_7kZuYg

C Eidesstattliche Erklärung

Hiermit erklären wir an Eides statt, dass die vorliegende Arbeit ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt haben.

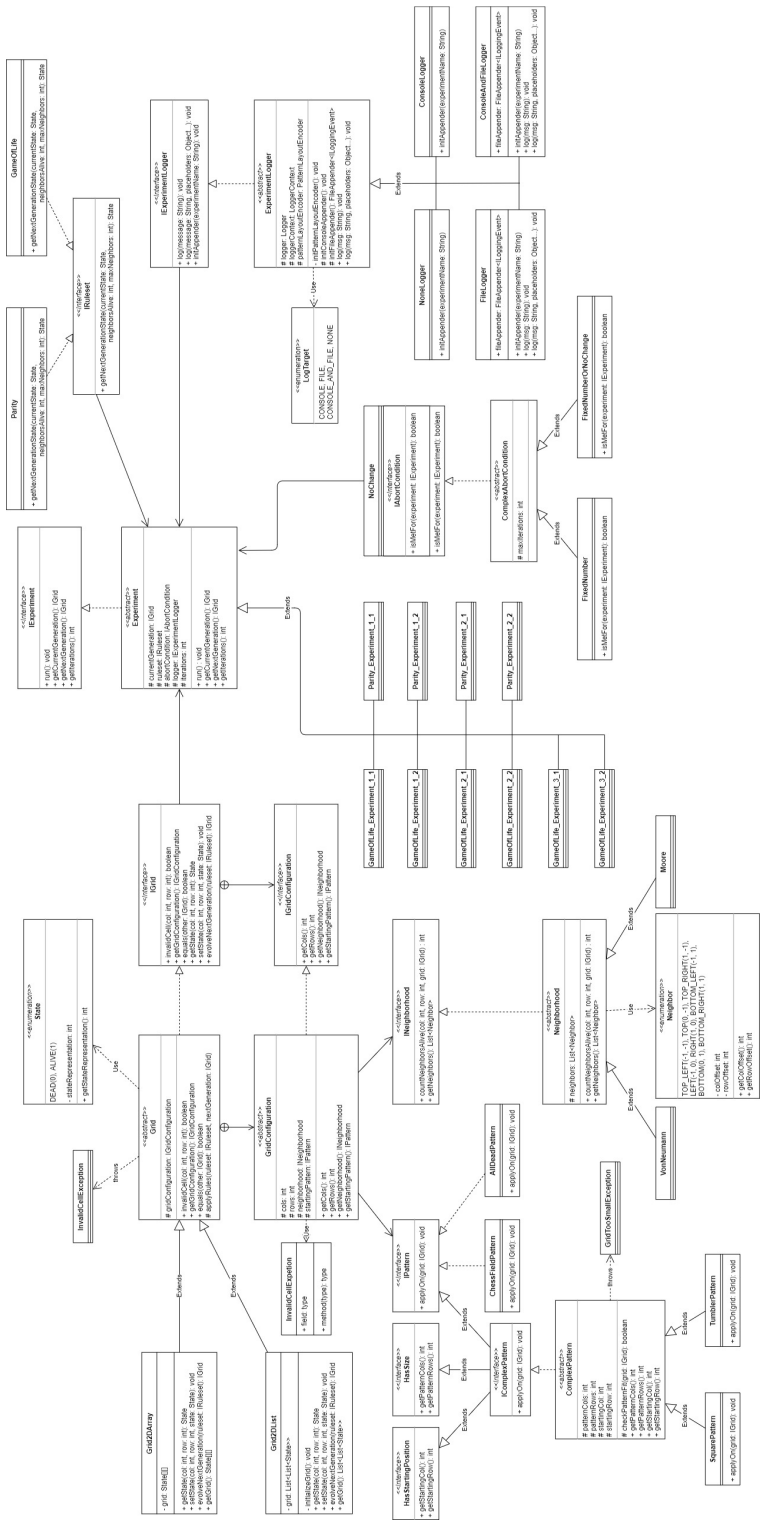
Datum: 03.11.2020 Unterschrift: _____

Datum: 03.11.2020 Unterschrift: _____

Datum: 03.11.2020 Unterschrift: _____

Datum: 03.11.2020 Unterschrift: _____

D UML Klassendiagramm



E Sourcecode

```java
package de.nordakademie.pdse.cellularAutomata;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.abortCondition.complex.FixedNumberOrNoChange;
import de.nordakademie.pdse.cellularAutomata.experiment.GameOfLife_Experiment_1_1;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.log.ConsoleLogger;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;

/**
 * The class <b>Main</b> is the entry point of our application and is used to configure and run experiments
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 * */
public class Main {
    public static void main(String[] args) {
        /*
        Configure abort condition to use in the experiment
        You can use the following options:
            1. NoChange - experiment stops if the state of each cell isn't changing in the next generation of the grid
            2. FixedNumber(n) - experiments stops after n generations
            3. FixedNumberOrNoChange(n) - experiments stops if abort condition 1. or 2. is met
         */
        IAbortCondition abortCondition = new FixedNumberOrNoChange(100);

        /*
        Configure logger to use in the experiment
        You can use the following options:
            1. ConsoleLogger - output of the experiment is printed to console
            2. FileLogger - output is printed to a log file in the project root with the name of the experiment
            3. ConsoleAndFileLogger - output of the experiment is printed to console and toa log file in the project
                root with the name of the experiment
            4. NoneLogger - output of the experiment is not printed anywhere
         */
        IExperimentLogger logger = new ConsoleLogger();

        /*
        Instantiate the desired experiment with abortCondition and logTarget
        You can choose between the following experiments:
            1. GameOfLife_Experiment_1_1 - Grid = Grid2DArray(cols=41, rows=40, neighborhood=Moore, startingPattern=TumblerPattern), Ruleset=GameOfLife
            2. GameOfLife_Experiment_1_2 - Grid = Grid2DList(cols=41, rows=40, neighborhood=Moore, startingPattern=TumblerPattern), Ruleset=GameOfLife
            3. GameOfLife_Experiment_2_1 - Grid = Grid2DArray(cols=100, rows=100, neighborhood=Moore, startingPattern=ChessFieldPattern), Ruleset=GameOfLife
            4. GameOfLife_Experiment_2_2 - Grid = Grid2DList(cols=100, rows=100, neighborhood=Moore, startingPattern=ChessFieldPattern), Ruleset=GameOfLife
            5. GameOfLife_Experiment_3_1 - Grid = Grid2DArray(cols=300, rows=300, neighborhood=Moore, startingPattern=AllDeadPattern), Ruleset=GameOfLife
            6. GameOfLife_Experiment_3_2 - Grid = Grid2DList(cols=300, rows=300, neighborhood=Moore, startingPattern=AllDeadPattern), Ruleset=GameOfLife
            7. Parity_Experiment_1_1 - Grid = Grid2DArray(cols=400, rows=400, neighborhood=VonNeumann, startingPattern=SquarePattern), Ruleset=Parity
            8. Parity_Experiment_1_2 - Grid = Grid2DList(cols=400, rows=400, neighborhood=VonNeumann, startingPattern=SquarePattern), Ruleset=Parity
            9. Parity_Experiment_2_1 - Grid = Grid2DArray(cols=100, rows=100, neighborhood=VonNeumann, startingPattern=ChessFieldPattern), Ruleset=Parity
            10. Parity_Experiment_2_2 - Grid = Grid2DList(cols=100, rows=100, neighborhood=VonNeumann, startingPattern=ChessFieldPattern), Ruleset=Parity
         */
        IExperiment experiment = new GameOfLife_Experiment_1_1(abortCondition, logger);

        // run the experiment
        experiment.run();

        // you can also use this short way to configure and run an experiment
        new GameOfLife_Experiment_1_1(new FixedNumberOrNoChange(100), new ConsoleLogger()).run();
    }
```

```
59 }
60
```

```java
package de.nordakademie.pdse.cellularAutomata.log;

import ch.qos.logback.classic.Logger;
import ch.qos.logback.classic.spi.ILoggingEvent;
import ch.qos.logback.core.FileAppender;

/**
 * The <b>FileLogger</b> represents the implementation of an
 * {@link ExperimentLogger}. The class uses methods from its parent to
 * initialize a {@link FileAppender} and add it to the {@link Logger} which is
 * created by its parent. <br>
 * The {@link Logger} will only print to the file with the delivered name.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 */
public class FileLogger extends ExperimentLogger {

    private FileAppender<ILoggingEvent> fileAppender;

    public FileLogger() {
        super();
    }

    @Override
    public void log(String msg) {
        this.fileAppender.start();
        this.logger.info(msg);
        this.fileAppender.stop();
    }

    @Override
    public void log(String msg, Object... placeholders) {
        this.fileAppender.start();
        this.logger.info(msg, placeholders);
        this.fileAppender.stop();
    }

    @Override
    public void initAppender(String experimentName) {
        this.fileAppender = initFileAppender(experimentName);
    }


}

```

```java
package de.nordakademie.pdse.cellularAutomata.log;

import ch.qos.logback.classic.Logger;

/**
 * The <b>NoneLogger</b> represents the implementation of an
 * {@link ExperimentLogger}. The class will add <b>no</b> appenders to the
 * {@link Logger} which is created by its parent. <br>
 * The {@link Logger} will print <b>nothing</b>.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 */
public class NoneLogger extends ExperimentLogger {

    public NoneLogger() {
        super();
    }

    @Override
    public void initAppender(String experimentName) {
        // no appender is needed
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.log;

import ch.qos.logback.classic.Logger;
import ch.qos.logback.core.ConsoleAppender;


/**
 * The <b>ConsoleLogger</b> represents the implementation of an
 * {@link ExperimentLogger}. The class uses methods from its parent to
 * initialize a {@link ConsoleAppender} and add it to the {@link Logger} which
 * is created by its parent. <br>
 * The {@link Logger} will only print to the console.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 */
public class ConsoleLogger extends ExperimentLogger {

    public ConsoleLogger() {
        super();
    }

    @Override
    public void initAppender(String experimentName) {
        initConsoleAppender();
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.log;

import ch.qos.logback.classic.Logger;
import ch.qos.logback.classic.LoggerContext;
import ch.qos.logback.classic.encoder.PatternLayoutEncoder;
import ch.qos.logback.classic.spi.ILoggingEvent;
import ch.qos.logback.core.ConsoleAppender;
import ch.qos.logback.core.FileAppender;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

/**
 * The abstract class <b>ExperimentLogger</b> serves as abstraction for all
 * {@link IExperimentLogger}. The class generates and holds a {@link Logger}
 * with depending {@link LoggerContext} and {@link PatternLayoutEncoder}. The
 * class also provides methods for initializing different appenders and logging,
 * which are used in several child classes equally.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 */
public abstract class ExperimentLogger implements IExperimentLogger {

    protected final Logger logger;

    protected final LoggerContext loggerContext;
    protected PatternLayoutEncoder patternLayoutEncoder;

    public ExperimentLogger() {
        this.loggerContext = (LoggerContext) LoggerFactory.getILoggerFactory();
        initPatternLayoutEncoder();

        this.logger = (Logger) LoggerFactory.getLogger(IExperiment.class);
        this.logger.setAdditive(false);
        this.logger.detachAndStopAllAppenders();
    }

    private void initPatternLayoutEncoder() {
        this.patternLayoutEncoder = new PatternLayoutEncoder();

        this.patternLayoutEncoder.setPattern("%msg%n");
        this.patternLayoutEncoder.setContext(loggerContext);
        this.patternLayoutEncoder.start();
    }

    protected void initConsoleAppender() {
        ConsoleAppender<ILoggingEvent> consoleAppender = new ConsoleAppender<>();

        consoleAppender.setContext(this.loggerContext);
        consoleAppender.setEncoder(this.patternLayoutEncoder);
        consoleAppender.start();

        this.logger.addAppender(consoleAppender);
    }

    protected FileAppender<ILoggingEvent> initFileAppender(String experimentName) {
        File file = new File(System.getProperty("user.dir") + File.separator + experimentName + ".log"
);

        if (file.exists()) {
            try {
                new PrintWriter(file).close();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
        }

        FileAppender<ILoggingEvent> fileAppender = new FileAppender<>();

```

```
72          fileAppender.setFile(file.getName());
73          fileAppender.setContext(this.loggerContext);
74          fileAppender.setEncoder(this.patternLayoutEncoder);
75
76          this.logger.addAppender(fileAppender);
77
78          return fileAppender;
79      }
80
81      @Override
82      public void log(String msg) {
83          logger.info(msg);
84      }
85
86      @Override
87      public void log(String msg, Object... placeholders) {
88          logger.info(msg, placeholders);
89      }
90
91  }
92
```

```java
package de.nordakademie.pdse.cellularAutomata.log;

/**
 * The <b>IExperimentLogger</b> interface serves as abstraction for all types of
 * loggers for experiments.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 */
public interface IExperimentLogger {
    /**
     * This method is used to log a message.
     *
     * @param msg the message to be logged
     */
    void log(String msg);

    /**
     * This method is used to log a message with any number of placeholders.
     *
     * @param msg  the message to be logged with "{}" as placeholders
     * @param placeholders the placeholders to be inserted in the message
     */
    void log(String msg, Object... placeholders);

    /**
     * This method is used to initialize the desired appender in the logger
     *
     * @param experimentName the name of the experiment the logger is used in
     */
    void initAppender(String experimentName);

}

```

```java
package de.nordakademie.pdse.cellularAutomata.log;

import ch.qos.logback.classic.Logger;
import ch.qos.logback.classic.spi.ILoggingEvent;
import ch.qos.logback.core.ConsoleAppender;
import ch.qos.logback.core.FileAppender;

/**
 * The <b>ConsoleAndFileLogger</b> represents the implementation of an
 * {@link ExperimentLogger}. The class uses methods from its parent to
 * initialize a {@link FileAppender} and a {@link ConsoleAppender} and add them
 * to the {@link Logger} which is created by its parent. <br>
 * The {@link Logger} will print to the file with the delivered name <b>and</b>
 * the console.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 */
public class ConsoleAndFileLogger extends ExperimentLogger {

    private FileAppender<ILoggingEvent> fileAppender;

    public ConsoleAndFileLogger() {
        super();
    }

    @Override
    public void log(String msg) {
        this.fileAppender.start();
        this.logger.info(msg);
        this.fileAppender.stop();
    }

    @Override
    public void log(String msg, Object... placeholders) {
        this.fileAppender.start();
        this.logger.info(msg, placeholders);
        this.fileAppender.stop();
    }

    @Override
    public void initAppender(String experimentName) {
        initConsoleAppender();
        this.fileAppender = initFileAppender(experimentName);
    }

}

```

```java
package de.nordakademie.pdse.cellularAutomata.grid;

import de.nordakademie.pdse.cellularAutomata.neighborhood.INeighborhood;
import de.nordakademie.pdse.cellularAutomata.neighborhood.Neighborhood;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.IRuleset;

/**
 * The abstract class <b>Grid</b> serves as abstraction for all types of grids. It can handle various data structures.
 * The class holds the grid-configuration, which therefore holds the columns, rows, neighborhood relationship and
 * starting pattern.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 */
public abstract class Grid implements IGrid {

    protected final IGridConfiguration gridConfiguration;

    // copy constructor
    public Grid(Grid other) {
        this.gridConfiguration = new GridConfiguration(other.getGridConfiguration());
    }

    public Grid(final int cols, final int rows, final INeighborhood neighborhood, final IPattern startingPattern) {
        this.gridConfiguration = new GridConfiguration(rows, cols, neighborhood, startingPattern);
    }

    @Override
    public boolean invalidCell(int col, int row) {
        return !(col >= 0
                && col < this.gridConfiguration.getCols()
                && row >= 0
                && row < this.gridConfiguration.getRows());
    }

    @Override
    public IGridConfiguration getGridConfiguration() {
        return this.gridConfiguration;
    }

    @Override
    public boolean equals(IGrid other) {
        if(this.gridConfiguration.getCols() != other.getGridConfiguration().getCols())
            return false;

        if(this.gridConfiguration.getRows() != other.getGridConfiguration().getRows())
            return false;

        for(int col = 0; col < this.gridConfiguration.getCols(); col++) {
            for(int row = 0; row < this.gridConfiguration.getRows(); row++) {
                if(this.getState(col, row) != other.getState(col, row)) {
                    return false;
                }
            }
        }

        return true;
    }

    // This method is a helper method for evolveNextGeneration() function. You can look for documentation there
    protected IGrid applyRules(IRuleset ruleset, IGrid nextGeneration) {
        for(int col = 0; col < this.gridConfiguration.getCols(); col++) {
            for(int row = 0; row < this.gridConfiguration.getRows(); row++) {
                nextGeneration.setState(col, row, ruleset.getNextGenerationState(
                        this.getState(col, row),
                        this.getGridConfiguration().getNeighborhood().countNeighborsAlive(col, row, this),
                        this.getGridConfiguration().getNeighborhood().getNeighbors().size()));
```

```
 68             }
 69         }
 70
 71         return nextGeneration;
 72     }
 73
 74     @Override
 75     public String toString() {
 76         StringBuilder builder = new StringBuilder();
 77
 78         for (int row = 0; row < this.gridConfiguration.getRows(); row++) {
 79             for (int col = 0; col < this.gridConfiguration.getCols(); col++) {
 80                 builder.append(getState(col, row).getStateRepresentation());
 81             }
 82             builder.append("\n");
 83         }
 84
 85         return builder.toString();
 86     }
 87
 88     /**
 89      *
 90      * The inner class <b>GridConfiguration</b> serves as encapsulation of the grid's configuration.
 91      * This class holds the number of columns and row of the grid, the desired neighborhood
    relationship
 92      * as well as the starting pattern for the grid.
 93      *
 94      * @author 17015 (til.zoeller@nordakademie.de)
 95      * @author 16513 (merlin.ritsch@nordakademie.de)
 96      */
 97     protected class GridConfiguration implements IGridConfiguration {
 98
 99         protected int cols;
100         protected int rows;
101
102         protected INeighborhood neighborhood;
103         protected IPattern startingPattern;
104
105         // copy constructor
106         protected GridConfiguration(IGridConfiguration other) {
107             this.cols = other.getCols();
108             this.rows = other.getRows();
109
110             this.neighborhood = new Neighborhood(other.getNeighborhood());
111             this.startingPattern = other.getStartingPattern();
112         }
113
114         protected GridConfiguration(int rows, int cols, INeighborhood neighborhood, IPattern
    startingPattern) {
115             if (rows < 1 || cols < 1)
116                 throw new IndexOutOfBoundsException("Rows: " + rows + ", Cols: " + cols);
117
118             this.rows = rows;
119             this.cols = cols;
120
121             this.neighborhood = new Neighborhood(neighborhood.getNeighbors());
122             this.startingPattern = startingPattern;
123         }
124
125         @Override
126         public int getCols() {
127             return this.cols;
128         }
129
130         @Override
131         public int getRows() {
132             return this.rows;
133         }
134
135         @Override
136         public INeighborhood getNeighborhood() {
137             return this.neighborhood;
```

```
138         }
139
140         @Override
141         public IPattern getStartingPattern() {
142             return startingPattern;
143         }
144     }
145 }
146
```

```java
package de.nordakademie.pdse.cellularAutomata.grid;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.neighborhood.INeighborhood;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.IRuleset;

/**
 * The <b>IGrid</b> interface serves as abstraction for all types of grids.
 * It can work with various data structures.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 */
public interface IGrid {
    /**
     * This method is used to check if col and row are invalid identifiers of a cell within the grid
     *
     * @param col the column which should be checked
     * @param row the row which should be checked
     * @return true, if col and rows are invalid identifiers and else false
     */
    boolean invalidCell(int col, int row);

    /**
     * This method is used to access the grid's configuration from outside
     *
     * @return the grid's configuration
     */
    IGridConfiguration getGridConfiguration();

    /**
     * This method is used to compare this grid with another grid. The methods evaluates to true if each cell of this grid
     * has the same state as in the other grid
     *
     * @param other other grid to compare
     * @return true if other grid is equal to this grid, else false
     */
    boolean equals(IGrid other);

    /**
     * This method is used to find out which {@link State} a cell in the {@link Grid} has
     *
     * @param col the column of the cell of which the state should be obtained
     * @param row the row of the cell of which the state should be obtained
     * @return the {@link State} of the cell
     */
    State getState(int col, int row);

    /**
     * This method is used to set the {@link State} of a cell
     * @param col the column of the cell of which the {@link State} should be set
     * @param row the row of the cell of which the {@link State} should be set
     * @param state the new {@link State} to set
     */
    void setState(int col, int row, State state);

    /**
     * This method is used to evolve the next generation of this grid with the given ruleset
     *
     * @param ruleset the ruleset which is used to determine the state of the cells in the next generation
     * @return the next generation grid
     */
    IGrid evolveNextGeneration(IRuleset ruleset);

    /**
     * The <b>IGridConfiguration</b> inner interface serves as abstraction for a configuration of a grid.
     *
     * @author 17015 (til.zoeller@nordakademie.de)
     * @author 16513 (merlin.ritsch@nordakademie.de)
```

```
 70      */
 71     interface IGridConfiguration {
 72         /**
 73          * This method is used to get the defined columns amount.
 74          *
 75          * @return the amount of columns
 76          */
 77         int getCols();
 78 
 79         /**
 80          * This method is used to get the defined rows amount.
 81          *
 82          * @return the amount of rows
 83          */
 84         int getRows();
 85 
 86         /**
 87          * This method is used to get the defined neighborhood type.
 88          *
 89          * @return the neighborhood type
 90          */
 91         INeighborhood getNeighborhood();
 92 
 93         /**
 94          * This method is used to get the grid configurations starting pattern from outside
 95          *
 96          * @return the starting pattern
 97          */
 98         IPattern getStartingPattern();
 99     }
100 }
101 
```

```java
package de.nordakademie.pdse.cellularAutomata.grid;

import java.util.ArrayList;
import java.util.List;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.exceptions.InvalidCellException;
import de.nordakademie.pdse.cellularAutomata.neighborhood.INeighborhood;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.IRuleset;

/**
 * The class <b>Grid2DList</b> represents the implementation of an {@link IGrid} with a two dimensional list as its data structure.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 *
 */
public class Grid2DList extends Grid{

    private List<List<State>> grid;

    // copy constructor
    public Grid2DList(Grid2DList other) {
        super(other);
        initializeGrid();
    }

    public Grid2DList(final int cols, final int rows, final INeighborhood neighborhood, final IPattern startingPattern) {
        super(cols, rows, neighborhood, startingPattern);
        initializeGrid();
        this.gridConfiguration.getStartingPattern().applyOn(this);
    }

    private void initializeGrid() {
        this.grid = new ArrayList<>();

        for(int i = 0; i < this.gridConfiguration.getCols(); i++) {
            List<State> row = new ArrayList<>();

            for(int j = 0; j < this.gridConfiguration.getRows(); j++) {
                row.add(State.DEAD);
            }

            this.grid.add(row);
        }
    }

    @Override
    public State getState(int col, int row) {
        if(invalidCell(col, row)) {
            throw new InvalidCellException(col, row);
        }

        return grid.get(col).get(row);
    }

    @Override
    public void setState(int col, int row, State state) {
        if(invalidCell(col, row)) {
            throw new InvalidCellException(col, row);
        }

        this.grid.get(col).set(row, state);
    }

    @Override
    public IGrid evolveNextGeneration(IRuleset ruleset) {
        IGrid nextGeneration = new Grid2DList(this);

        return applyRules(ruleset, nextGeneration);
```

```
71      }
72
73      public List<List<State>> getGrid() {
74          return this.grid;
75      }
76
77  }
78
```

```java
package de.nordakademie.pdse.cellularAutomata.grid;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.exceptions.InvalidCellException;
import de.nordakademie.pdse.cellularAutomata.neighborhood.INeighborhood;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.IRuleset;

/**
 * The class <b>Grid2DArray</b> represents the implementation of an {@link IGrid} with a two dimensional array as its data structure.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 *
 */
public class Grid2DArray extends Grid {

    private final State[][] grid;

    // copy constructor
    public Grid2DArray(Grid2DArray other) {
        super(other);
        this.grid = new State[other.getGridConfiguration().getCols()][other.getGridConfiguration().getRows()];
    }

    public Grid2DArray(final int cols, final int rows, final INeighborhood neighborhood, final IPattern startingPattern) {
        super(cols, rows, neighborhood, startingPattern);
        this.grid = new State[cols][rows];
        this.gridConfiguration.getStartingPattern().applyOn(this);
    }

    @Override
    public State getState(int col, int row) {
        if(invalidCell(col, row)) {
            throw new InvalidCellException(col, row);
        }

        return grid[col][row];
    }

    @Override
    public void setState(int col, int row, State state) {
        if(invalidCell(col, row)) {
            throw new InvalidCellException(col, row);
        }

        grid[col][row] = state;
    }

    @Override
    public IGrid evolveNextGeneration(IRuleset ruleset) {
        IGrid nextGeneration = new Grid2DArray(this);

        return applyRules(ruleset, nextGeneration);
    }

    public State[][] getGrid() {
        return this.grid;
    }


}

```

```java
package de.nordakademie.pdse.cellularAutomata.enums;

/**
 * The <b>State</b> enum holds all possible states of a grid cell as well as its numeric representation
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 */
public enum State {
    DEAD(0), ALIVE(1);

    private final int stateRepresentation;

    State(int stateRepresentation) {
        this.stateRepresentation = stateRepresentation;
    }

    public int getStateRepresentation() {
        return stateRepresentation;
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.enums;

/**
 * The <b>Neighbor</b> enum holds all possible neighbors of a cell in a two-dimensional grid
 * as well as the correspondent column and row offset
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public enum Neighbor {
    TOP_LEFT(-1,-1),
    TOP(0, -1),
    TOP_RIGHT(1,-1),
    LEFT(-1,0),
    RIGHT(1,0),
    BOTTOM_LEFT(-1,1),
    BOTTOM(0,1),
    BOTTOM_RIGHT(1,1);

    private final int colOffset;
    private final int rowOffset;

    Neighbor(Integer colOffset, Integer rowOffset) {
        this.colOffset = colOffset;
        this.rowOffset = rowOffset;
    }

    public int getColOffset() {
        return colOffset;
    }

    public int getRowOffset() {
        return rowOffset;
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.pattern;

import de.nordakademie.pdse.cellularAutomata.grid.IGrid;

/**
 * The interface <b>IPattern</b> serves as abstraction for all patterns, which could be applied to a grid.
 * The class can work with all different implementations of an {@link IGrid} interface.
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public interface IPattern {
    /**
     * This method is used to apply this pattern on this grid
     *
     * @param grid the grid the pattern should be applied on
     */
    void applyOn(IGrid grid);
}

```

```java
package de.nordakademie.pdse.cellularAutomata.pattern.simple;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;

/**
 * The class <b>AllDeadPattern</b> is used to set all cells of a grid to {@link State}.DEAD
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class AllDeadPattern implements IPattern {
    @Override
    public void applyOn(IGrid grid) {
        for(int col = 0; col < grid.getGridConfiguration().getCols(); col++) {
            for(int row = 0; row < grid.getGridConfiguration().getRows(); row++) {
                grid.setState(col, row, State.DEAD);
            }
        }
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.pattern.simple;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;

/**
 * The class <b>ChessFieldPattern</b> is used to apply an chess field pattern of living and dead cells to the grid.
 * Even rows are alternately initialized with STATE.DEAD and STATE.ALIVE.
 * Odd rows are alternately initialized with STATE.ALIVE and STATE.DEAD.
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class ChessFieldPattern implements IPattern {
    @Override
    public void applyOn(IGrid grid) {
        for(int col = 0; col < grid.getGridConfiguration().getCols(); col++) {
            for(int row = 0; row < grid.getGridConfiguration().getRows(); row++) {
                if (row % 2 == 0) {
                    if (col % 2 == 0)
                        grid.setState(col, row, State.ALIVE);
                    else
                        grid.setState(col, row, State.DEAD);
                }
                else {
                    if (col % 2 == 0)
                        grid.setState(col, row, State.DEAD);
                    else
                        grid.setState(col, row, State.ALIVE);
                }
            }
        }
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.pattern.complex;

import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;

/**
 * The <b>HasSize</b> interface serves as abstraction for all {@link IPattern}s, which have a size
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public interface HasSize {
    /**
     * This method is used to get the pattern's columns
     *
     * @return the pattern's columns
     */
    int getPatternCols();

    /**
     * This method is used to get the pattern's rows
     *
     * @return the pattern's rows
     */
    int getPatternRows();
}

```

```java
package de.nordakademie.pdse.cellularAutomata.pattern.complex;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;

/**
 * The class <b>SquarePattern</b> is used to apply a 2x2 square of living cells to the grid at given starting position
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class SquarePattern extends ComplexPattern {
    public SquarePattern(int startingCol, int startingRow) {
        super(startingCol, startingRow, 2, 2);
    }

    @Override
    public void applyOn(IGrid grid) {
        checkPatternFit(grid);

        for(int col = 0; col < grid.getGridConfiguration().getCols(); col++) {
            for(int row = 0; row < grid.getGridConfiguration().getRows(); row++) {
                grid.setState(col, row, State.DEAD);
            }
        }

        grid.setState(startingCol, startingRow, State.ALIVE);
        grid.setState(startingCol + 1, startingRow, State.ALIVE);
        grid.setState(startingCol, startingRow + 1, State.ALIVE);
        grid.setState(startingCol + 1, startingRow + 1, State.ALIVE);
    }
}
```

```java
package de.nordakademie.pdse.cellularAutomata.pattern.complex;

import de.nordakademie.pdse.cellularAutomata.exceptions.GridTooSmallException;
import de.nordakademie.pdse.cellularAutomata.exceptions.InvalidCellException;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;

/**
 * The abstract class <b>IComplexPattern</b> serves as abstraction for all complex patterns, which could be applied to a grid.
 *  A complex pattern has a defined size and starting position.
 *  This class can work with all different implementations of an {@link IGrid} interface.
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public abstract class ComplexPattern implements IComplexPattern {
    protected final int startingCol;
    protected final int startingRow;

    protected final int patternCols;
    protected final int patternRows;

    public ComplexPattern(int startingCol, int startingRow, int patternCols, int patternRows) {
        this.startingCol = startingCol;
        this.startingRow = startingRow;

        this.patternCols = patternCols;
        this.patternRows = patternRows;
    }

    protected void checkPatternFit(IGrid grid) {
        if (grid.invalidCell(startingCol, startingRow))
            throw new InvalidCellException(startingCol, startingRow);

        if (grid.getGridConfiguration().getCols() < (startingCol + patternCols)
                || grid.getGridConfiguration().getRows() < (startingRow + patternRows))
            throw new GridTooSmallException(this);
    }

    @Override
    public int getPatternCols() {
        return this.patternCols;
    }

    @Override
    public int getPatternRows() {
        return this.patternRows;
    }

    @Override
    public int getStartingCol() {
        return this.startingCol;
    }

    @Override
    public int getStartingRow() {
        return this.startingRow;
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.pattern.complex;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;

/**
 * The class <b>TumblerPattern</b> is used to apply the tumbler pattern out of living cells to a grid at given starting position
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class TumblerPattern extends ComplexPattern {
    public TumblerPattern(int startingCol, int startingRow) {
        super(startingCol, startingRow, 7, 6);
    }

    @Override
    public void applyOn(IGrid grid) {
        checkPatternFit(grid);

        for(int col = 0; col < grid.getGridConfiguration().getCols(); col++) {
            for(int row = 0; row < grid.getGridConfiguration().getRows(); row++) {
                grid.setState(col, row, State.DEAD);
            }
        }

        grid.setState(getStartingCol(), getStartingRow() + 3, State.ALIVE);
        grid.setState(getStartingCol(), getStartingRow() + 4, State.ALIVE);
        grid.setState(getStartingCol(), getStartingRow() + 5, State.ALIVE);
        grid.setState(getStartingCol() + 1, getStartingRow(), State.ALIVE);
        grid.setState(getStartingCol() + 1, getStartingRow() + 1, State.ALIVE);
        grid.setState(getStartingCol() + 1, getStartingRow() + 5, State.ALIVE);
        grid.setState(getStartingCol() + 2, getStartingRow(), State.ALIVE);
        grid.setState(getStartingCol() + 2, getStartingRow() + 1, State.ALIVE);
        grid.setState(getStartingCol() + 2, getStartingRow() + 2, State.ALIVE);
        grid.setState(getStartingCol() + 2, getStartingRow() + 3, State.ALIVE);
        grid.setState(getStartingCol() + 2, getStartingRow() + 4, State.ALIVE);
        grid.setState(getStartingCol() + 4, getStartingRow(), State.ALIVE);
        grid.setState(getStartingCol() + 4, getStartingRow() + 1, State.ALIVE);
        grid.setState(getStartingCol() + 4, getStartingRow() + 2, State.ALIVE);
        grid.setState(getStartingCol() + 4, getStartingRow() + 3, State.ALIVE);
        grid.setState(getStartingCol() + 4, getStartingRow() + 4, State.ALIVE);
        grid.setState(getStartingCol() + 5, getStartingRow(), State.ALIVE);
        grid.setState(getStartingCol() + 5, getStartingRow() + 1, State.ALIVE);
        grid.setState(getStartingCol() + 5, getStartingRow() + 5, State.ALIVE);
        grid.setState(getStartingCol() + 6, getStartingRow() + 3, State.ALIVE);
        grid.setState(getStartingCol() + 6, getStartingRow() + 4, State.ALIVE);
        grid.setState(getStartingCol() + 6, getStartingRow() + 5, State.ALIVE);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.pattern.complex;

import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;

/**
 * The interface <b>IComplexPattern</b> serves as abstraction for all complex patterns, which could be applied to a grid.
 * This class can work with all different implementations of an {@link IGrid} interface.
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public interface IComplexPattern extends IPattern, HasSize, HasStartingPosition {}

```

```java
package de.nordakademie.pdse.cellularAutomata.pattern.complex;

import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;

/**
 * The <b>HasStartingPosition</b> interface serves as abstraction for all {@link IPattern}s,
 * which have a starting position
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public interface HasStartingPosition {
    /**
     * This method is used to get the pattern's starting column
     *
     * @return the pattern's starting column
     */
    int getStartingCol();

    /**
     * This method is used to get the pattern's starting row
     *
     * @return the pattern's starting row
     */
    int getStartingRow();
}

```

```java
package de.nordakademie.pdse.cellularAutomata.ruleset;

import de.nordakademie.pdse.cellularAutomata.enums.State;

/**
 * The class <b>Parity</b> represents the implementation of a {@link IRuleset} with rules of Parity Model.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class Parity extends Ruleset {

    @Override
    public State getNextGenerationState(State currentState, int neighborsAlive, int maxNeighbors) {

        checkForIllegalInput(neighborsAlive, maxNeighbors);

        if (neighborsAlive % 2 == 0) {
            return State.DEAD;
        }

        return State.ALIVE;
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.ruleset;

/**
 * The abstract class <b>Ruleset</b> serves as abstraction for all types of Rulesets. It holds redundant methods.
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */

public abstract class Ruleset implements IRuleset {
    protected void checkForIllegalInput(int neighborsAlive, int maxNeighbors) {
        if(maxNeighbors < 0) {
            throw new IllegalArgumentException("MaxNeighbors should be greater than or equal to 0 but is " + maxNeighbors + ".");
        }

        if(neighborsAlive < 0 || neighborsAlive > maxNeighbors) {
            throw new IllegalArgumentException("NeighborsAlive should be between 0 and " + maxNeighbors + " but was " + neighborsAlive + ".");
        }
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.ruleset;

import de.nordakademie.pdse.cellularAutomata.enums.State;

/**
 * The <b>IRuleset</b> interface serves as abstraction for all rulesets.
 * It can work with various grids.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public interface IRuleset {
    /**
     * This method is used to get the {@link State} of a cell for the next generation.
     *
     * @param currentState State of a cell from this generation
     * @param neighborsAlive amount of living neighbors of a cell from this generation
     * @param maxNeighbors maximal allowed neighbors of a cell due to neighborhood relationship
     * @return State of the cell for next generation
     */
    State getNextGenerationState(State currentState, int neighborsAlive, int maxNeighbors);
}

```

```java
package de.nordakademie.pdse.cellularAutomata.ruleset;

import de.nordakademie.pdse.cellularAutomata.enums.State;

/**
 * The class <b>GameOfLife</b> represents the implementation of a {@link IRuleset} with rules of GameOfLife
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class GameOfLife extends Ruleset {

    @Override
    public State getNextGenerationState(State currentState, int neighborsAlive, int maxNeighbors) {

        checkForIllegalInput(neighborsAlive, maxNeighbors);

        if(currentState == State.ALIVE) {
            if(neighborsAlive < 2 || neighborsAlive > 3) {
                return State.DEAD;
            }

            return State.ALIVE;
        }
        else {
            if(neighborsAlive == 3) {
                return State.ALIVE;
            }

            return State.DEAD;
        }
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.exceptions;

/**
 * The exception <b>InvalidCellException</b> is thrown, if a cell is invalid.
 * For context information we also store the column and row of the invalid cell.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 *
 */
public class InvalidCellException extends RuntimeException {
    public InvalidCellException(int col, int row) {
        super("Invalid cell with following coordinates: " + col + ", " + row);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.exceptions;

import de.nordakademie.pdse.cellularAutomata.pattern.complex.IComplexPattern;

/**
 * The exception <b>GridTooSmallException</b> is thrown, if a grid is too small for applying a {@link IComplexPattern}
 * to it at a desired starting column and row
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class GridTooSmallException extends RuntimeException{
    public GridTooSmallException(IComplexPattern pattern) {
        super("Grid is too small. " +
                "Tried to insert pattern " + pattern + " into grid at " +
                "col " + pattern.getStartingCol() + " and " +
                "row " + pattern.getStartingRow() + ". " +
                "Grid must be at least " +
                (pattern.getStartingCol() + pattern.getPatternCols()) + " cols wide and " +
                (pattern.getStartingRow() + pattern.getPatternRows()) + " rows high.");
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.ruleset.IRuleset;

/**
 * The abstract class <b>Experiment</b> serves as abstraction for all Experiments. It can handle various Inputs.
 * The class holds the currentGeneration and the generation after currentGeneration, the used ruleset and the
 * maximal number of iterations.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public abstract class Experiment implements IExperiment {

    protected IGrid currentGeneration;
    protected final IRuleset ruleset;
    protected final IAbortCondition abortCondition;
    protected final IExperimentLogger logger;

    protected int iterations = 0;

    public Experiment(IGrid currentGeneration, IRuleset ruleset, IAbortCondition abortCondition, IExperimentLogger logger) {
        this.currentGeneration = currentGeneration;
        this.ruleset = ruleset;
        this.abortCondition = abortCondition;
        this.logger = logger;

        this.logger.initAppender(this.getClass().getSimpleName());
    }

    @Override
    public void run() {
        boolean firstIteration = true;

        do {
            if(!firstIteration)
                this.currentGeneration = getNextGeneration();
            else
                firstIteration = false;

            logger.log("### ({})", iterations);
            logger.log(this.currentGeneration.toString());

            iterations++;
        } while (!abortCondition.isMetFor(this));
    }

    @Override
    public IGrid getCurrentGeneration() {return currentGeneration;}

    @Override
    public IGrid getNextGeneration() {return currentGeneration.evolveNextGeneration(ruleset);}

    @Override
    public int getIterations() { return iterations; }

    @Override
    public IExperimentLogger getExperimentLogger() { return logger; }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;

/**
 * The <b>IExperiment</b> interface serves as abstraction for all Experiments.
 * It can run Experiments with various configurations.
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public interface IExperiment {

    /**
     * This method is used to run an Experiment.
     */
    void run();

    /**
     * This method is used to get the current generation of <b>IGrid</b>
     *
     * @return the current generation of <b>IGrid</b>
     */
    IGrid getCurrentGeneration();

    /**
     * This method is used to get the next generation of <b>IGrid</b> based on the current generation of Grid
     *
     * @return the current generation of <b>IGrid</b>
     */
    IGrid getNextGeneration();

    /**
     * This method is used to get the logger of the <b>Experiment</b>
     *
     * @return logger of <b>Experiment</b>
     */
    IExperimentLogger getExperimentLogger();

    /**
     * This method is used to get the current Iteration of an Experiment run
     *
     * @return current Iteration as integer number
     */
    int getIterations();

}
```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DArray;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.neighborhood.VonNeumann;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.SquarePattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.Parity;

/**
 * The class <b>Parity_Experiment_1_1</b> represents the implementation of an {@link Experiment} with ruleset
 * Parity, pattern MIDDLE_SQUARE, data structure 2DArray and Neighborhood VON_NEUMANN.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class Parity_Experiment_1_1 extends Experiment {

    public Parity_Experiment_1_1(IAbortCondition abortCondition, IExperimentLogger logger) {
        super(
                new Grid2DArray(400, 400, new VonNeumann(), new SquarePattern(199, 199)),
                new Parity(),
                abortCondition,
                logger);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DList;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.neighborhood.VonNeumann;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.SquarePattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.Parity;

/**
 * The class <b>Parity_Experiment_1_2</b> represents the implementation of an {@link Experiment} with ruleset
 * Parity, pattern MIDDLE_SQUARE, data structure 2DArray and Neighborhood VON_NEUMANN.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class Parity_Experiment_1_2 extends Experiment {

    public Parity_Experiment_1_2(IAbortCondition abortCondition, IExperimentLogger logger) {
        super(
                new Grid2DList(400, 400, new VonNeumann(), new SquarePattern(199, 199)),
                new Parity(),
                abortCondition,
                logger);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DArray;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.neighborhood.VonNeumann;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.ChessFieldPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.Parity;

/**
 * The class <b>Parity_Experiment_2_1</b> represents the implementation of an {@link Experiment} with ruleset
 * Parity, pattern CHESS_FIELD, data structure 2DArray and Neighborhood VON_NEUMANN.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class Parity_Experiment_2_1 extends Experiment {

    public Parity_Experiment_2_1(IAbortCondition abortCondition, IExperimentLogger logger) {
        super(
                new Grid2DArray(100, 100, new VonNeumann(), new ChessFieldPattern()),
                new Parity(),
                abortCondition,
                logger);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DList;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.neighborhood.VonNeumann;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.ChessFieldPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.Parity;

/**
 * The class <b>Parity_Experiment_2_2</b> represents the implementation of an {@link Experiment} with ruleset
 * Parity, pattern CHESS_FIELD, datas tructure 2DList and Neighborhood VON_NEUMANN.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class Parity_Experiment_2_2 extends Experiment {

    public Parity_Experiment_2_2(IAbortCondition abortCondition, IExperimentLogger logger) {
        super(
                new Grid2DList(100, 100, new VonNeumann(), new ChessFieldPattern()),
                new Parity(),
                abortCondition,
                logger);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DArray;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.neighborhood.Moore;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.TumblerPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.GameOfLife;

/**
 * The class <b>GameOfLife_Experiment_1_1</b> represents the implementation of an {@link Experiment} with ruleset
 * GameOfLife, pattern Tumbler, data structure 2DArray and Neighborhood Moore.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class GameOfLife_Experiment_1_1 extends Experiment {

    public GameOfLife_Experiment_1_1(IAbortCondition abortCondition, IExperimentLogger logger) {
        super(
                new Grid2DArray(41, 40, new Moore(), new TumblerPattern(17, 17)),
                new GameOfLife(),
                abortCondition,
                logger);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DList;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.neighborhood.Moore;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.TumblerPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.GameOfLife;

/**
 * The class <b>GameOfLife_Experiment_1_2</b> represents the implementation of an {@link Experiment} with ruleset
 * GameOfLife, pattern Tumbler, data structure 2DList and Neighborhood Moore.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class GameOfLife_Experiment_1_2 extends Experiment {

    public GameOfLife_Experiment_1_2(IAbortCondition abortCondition, IExperimentLogger logger) {
        super(
                new Grid2DList(41, 40, new Moore(), new TumblerPattern(17, 17)),
                new GameOfLife(),
                abortCondition,
                logger);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DArray;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.neighborhood.Moore;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.ChessFieldPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.GameOfLife;

/**
 * The class <b>GameOfLife_Experiment_2_1</b> represents the implementation of an {@link Experiment} with ruleset
 * GameOfLife, pattern CHESS_FIELD, data structure 2DArray and Neighborhood Moore.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class GameOfLife_Experiment_2_1 extends Experiment {

    public GameOfLife_Experiment_2_1(IAbortCondition abortCondition, IExperimentLogger logger) {
        super(
                new Grid2DArray(100, 100, new Moore(), new ChessFieldPattern()),
                new GameOfLife(),
                abortCondition,
                logger);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DList;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.neighborhood.Moore;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.ChessFieldPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.GameOfLife;

/**
 * The class <b>GameOfLife_Experiment_2_2</b> represents the implementation of an {@link Experiment} with ruleset
 * GameOfLife, pattern CHESS_FIELD, data structure 2DList and Neighborhood Moore.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class GameOfLife_Experiment_2_2 extends Experiment {

    public GameOfLife_Experiment_2_2(IAbortCondition abortCondition, IExperimentLogger logger) {
        super(
                new Grid2DList(100, 100, new Moore(), new ChessFieldPattern()),
                new GameOfLife(),
                abortCondition,
                logger);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DArray;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.neighborhood.Moore;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.AllDeadPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.GameOfLife;

/**
 * The class <b>GameOfLife_Experiment_3_1</b> represents the implementation of an {@link Experiment} with ruleset
 * GameOfLife, pattern ALL_DEAD, data structure 2DArray and Neighborhood Moore.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class GameOfLife_Experiment_3_1 extends Experiment {

    public GameOfLife_Experiment_3_1(IAbortCondition abortCondition, IExperimentLogger logger) {
        super(
                new Grid2DArray(300, 300, new Moore(), new AllDeadPattern()),
                new GameOfLife(),
                abortCondition,
                logger);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DList;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import de.nordakademie.pdse.cellularAutomata.neighborhood.Moore;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.AllDeadPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.GameOfLife;

/**
 * The class <b>GameOfLife_Experiment_3_2</b> represents the implementation of an {@link Experiment} with ruleset
 * GameOfLife, pattern ALL_DEAD, data structure 2DList and Neighborhood Moore.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */
public class GameOfLife_Experiment_3_2 extends Experiment {

    public GameOfLife_Experiment_3_2(IAbortCondition abortCondition, IExperimentLogger logger) {
        super(
                new Grid2DList(300, 300, new Moore(), new AllDeadPattern()),
                new GameOfLife(),
                abortCondition,
                logger);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.neighborhood;

import de.nordakademie.pdse.cellularAutomata.enums.Neighbor;

import java.util.Arrays;

/**
 * The class <b>Moore</b> is the implementation of the <i>Moore</i> neighborhood relationship
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class Moore extends Neighborhood {
    public Moore() {
        super(Arrays.asList(Neighbor.TOP_LEFT, Neighbor.TOP, Neighbor.TOP_RIGHT, Neighbor.LEFT, Neighbor.RIGHT,
                Neighbor.BOTTOM_LEFT, Neighbor.BOTTOM, Neighbor.BOTTOM_RIGHT));
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.neighborhood;

import de.nordakademie.pdse.cellularAutomata.enums.Neighbor;

import java.util.Arrays;

/**
 * The class <b>VonNeumann</b> is the implementation of the <i>Von Neumann</i> neighborhood
relationship
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class VonNeumann extends Neighborhood {
    public VonNeumann() {
        super(Arrays.asList(Neighbor.TOP, Neighbor.LEFT, Neighbor.RIGHT, Neighbor.BOTTOM));
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.neighborhood;

import de.nordakademie.pdse.cellularAutomata.enums.Neighbor;
import de.nordakademie.pdse.cellularAutomata.exceptions.InvalidCellException;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;

import java.util.List;

/**
 * The class <b>Neighborhood</b> serves as abstraction for all types of neighborhood relationships.
 * It can handle all different implementations of an {@link IGrid} interface as well as various data structures.
 * The class holds a list of {@link Neighbor}s, which are the way to define different neighborhood relationships.
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class Neighborhood implements INeighborhood {

    protected final List<Neighbor> neighbors;

    // copy constructor
    public Neighborhood(INeighborhood other) {
        this.neighbors = other.getNeighbors();
    }

    public Neighborhood(List<Neighbor> neighbors) {
        this.neighbors = neighbors;
    }

    @Override
    public int countNeighborsAlive(int col, int row, IGrid grid) {
        if (grid.invalidCell(col, row))
            throw new InvalidCellException(col, row);

        int sum = 0;

        for(Neighbor neighbor : neighbors) {
            int neighborCol = col + neighbor.getColOffset();
            int neighborRow = row + neighbor.getRowOffset();

            if(grid.invalidCell(neighborCol, neighborRow))
                continue;

            sum += grid.getState(neighborCol, neighborRow).getStateRepresentation();
        }

        return sum;
    }

    @Override
    public List<Neighbor> getNeighbors() {
        return this.neighbors;
    }
}
```

```java
package de.nordakademie.pdse.cellularAutomata.neighborhood;

import de.nordakademie.pdse.cellularAutomata.enums.Neighbor;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;

import java.util.List;

/**
 * The <b>INeighborhood</b> interface serves as abstraction for all types of neighborhood relationships.
 * It can work with all different implementations of an {@link IGrid} interface.
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public interface INeighborhood {
    /**
     * This method is used to count the live neighbors of a cell
     *
     * @param col the column of the cell of which the live neighbors should be counted
     * @param row the row of the cell of which the live neighbors should be counted
     * @return the count of live neighbors
     */
    int countNeighborsAlive(int col, int row, IGrid grid);

    /**
     * This method is used to get the list of neighbors from outside
     *
     * @return the list of neighbors
     */
    List<Neighbor> getNeighbors();
}

```

```java
package de.nordakademie.pdse.cellularAutomata.abortCondition;

import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;

/**
 * The <b>IAbortCondition</b> interface serves as abstraction for all abort conditions.
 * It can check different abort conditions for Experiments.
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */

public interface IAbortCondition {
    /**
     * This method is used to determine whether an Experiment fulfills its abort condition or not
     *
     * @param experiment the Experiment that should be checked on completion
     * @return boolean that determines if an experiment is done or not.
     */
     boolean isMetFor(IExperiment experiment);
}

```

```java
package de.nordakademie.pdse.cellularAutomata.abortCondition.Simple;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;

/**
 * The class <b>NoChange</b> represents the implementation of an
 * {@link de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition} with the abort condition of no change
 * between the current generation of a grid and its next generation.
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */

public class NoChange implements IAbortCondition {
    @Override
    public boolean isMetFor(IExperiment experiment) {
        int iterations;
        IExperimentLogger logger;
        if(experiment.getCurrentGeneration().equals(experiment.getNextGeneration())) {
            experiment.getExperimentLogger().log("### ({})", experiment.getIterations());
            experiment.getExperimentLogger().log(experiment.getCurrentGeneration().toString());
            return true;
        }
        return false;
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.abortCondition.complex;

import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;

/**
 * The class <b>FixedNumber</b> represents the implementation of an
 * {@link de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition} with the abort condition of reaching
 * the maximal number of allowed iterations.
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */

public class FixedNumber extends ComplexAbortCondition{

    public FixedNumber(int maxIterations) {
        super(maxIterations);
    }

    @Override
    public boolean isMetFor(IExperiment experiment) {
        if(experiment.getIterations() < 0) {
            throw new IllegalArgumentException("Current Iteration in Experiment cannot be negative but is: " + experiment.getIterations());
        }

        return (experiment.getIterations() >= this.maxIterations);
    }
}

```

```java
package de.nordakademie.pdse.cellularAutomata.abortCondition.complex;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;

/**
 * The abstract class <b>ComplexAbortCondition</b> serves as abstraction for all abort conditions with a number of
 * maximal iterations to walk through.
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */

public abstract class ComplexAbortCondition implements IAbortCondition {

    protected final int maxIterations;

    protected ComplexAbortCondition(int maxIterations) {
        this.maxIterations = maxIterations;
    }
}
```

```java
package de.nordakademie.pdse.cellularAutomata.abortCondition.complex;

import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;

/**
 * The class <b>FixedNumberOrNoChange</b> represents the implementation of an
 * {@link de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition} with the abort condition of reaching
 * the maximal number of allowed iterations or no change between the current generation of a grid and its next
 * generation, whatever happens first.
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */

public class FixedNumberOrNoChange extends ComplexAbortCondition {

    public FixedNumberOrNoChange(int maxIterations) {
        super(maxIterations);
    }

    @Override
    public boolean isMetFor(IExperiment experiment) {
        if(experiment.getIterations() >= this.maxIterations) return true;
        if(experiment.getCurrentGeneration().equals(experiment.getNextGeneration())) {
            experiment.getExperimentLogger().log("### ({})", experiment.getIterations());
            experiment.getExperimentLogger().log(experiment.getCurrentGeneration().toString());
            return true;
        }
        return false;
    }
}

```

```
1 <configuration>
2   <root level="info" />
3 </configuration>
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.log;

import de.nordakademie.pdse.cellularAutomata.log.FileLogger;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;

/**
 * Test class for {@link FileLogger}.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 *
 */
public class FileLoggerTest {
    private FileLogger fileLogger;
    private File logFile;

    @Before
    public void beforeEach() {
        this.fileLogger = new FileLogger();

        this.fileLogger.initAppender("experiment");

        this.logFile = new File(System.getProperty("user.dir") + File.separator + "experiment.log");
    }

    @After
    public void afterEach() {
        this.logFile.delete();
    }

    @Test
    public void testLogMessageInFile() throws FileNotFoundException {
        String testMessage = "test log message";
        this.fileLogger.log(testMessage);

        Scanner fileScanner = new Scanner(this.logFile);
        StringBuilder logContent = new StringBuilder();

        while (fileScanner.hasNextLine())
            logContent.append(fileScanner.nextLine());

        fileScanner.close();

        assertThat(logContent.toString(), is(testMessage));
    }

    @Test
    public void testLogMessageWithPlaceholderInFile() throws FileNotFoundException {
        String testMessage = "test {} message";
        this.fileLogger.log(testMessage, "log");

        Scanner fileScanner = new Scanner(this.logFile);
        StringBuilder logContent = new StringBuilder();
        while (fileScanner.hasNextLine())
            logContent.append(fileScanner.nextLine());
        fileScanner.close();

        assertThat(logContent.toString(), is("test log message"));
    }

    @Test
    public void testLogMessageWithMultiplePlaceholdersInFile()
            throws FileNotFoundException {
        String testMessage = "test {} {} {}";
```

```
73          this.fileLogger.log(testMessage, 1, "log", "message");
74
75          Scanner fileScanner = new Scanner(this.logFile);
76          StringBuilder logContent = new StringBuilder();
77          while (fileScanner.hasNextLine())
78              logContent.append(fileScanner.nextLine());
79          fileScanner.close();
80
81          assertThat(logContent.toString(), is("test 1 log message"));
82      }
83  }
84
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.log;

import de.nordakademie.pdse.cellularAutomata.log.NoneLogger;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.util.Scanner;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;

/**
 * Test class for {@link NoneLogger}.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 *
 */
public class NoneLoggerTest {

    private ByteArrayOutputStream outContent = new ByteArrayOutputStream();
    private PrintStream originalOut = System.out;

    private NoneLogger noneLogger;
    private File logFile;

    @Before
    public void beforeEach() throws FileNotFoundException {
        this.noneLogger = new NoneLogger();
        noneLogger.initAppender("experiment");
        this.logFile = new File(System.getProperty("user.dir") + File.separator  + "experiment.log");


        System.setOut(new PrintStream(outContent));

        if (this.logFile.exists())
            new PrintWriter(this.logFile.getAbsoluteFile()).close();
    }

    @After
    public void afterEach() {
        System.setOut(originalOut);
    }

    @Test
    public void testLogMessageOnNone() throws FileNotFoundException {
        NoneLogger noneLogger = new NoneLogger();
        String testMessage = "test no message";
        noneLogger.log(testMessage);

        if (this.logFile.exists()) {
            Scanner fileScanner = new Scanner(this.logFile);
            StringBuilder logContent = new StringBuilder();
            while (fileScanner.hasNextLine())
                logContent.append(fileScanner.nextLine());
            fileScanner.close();

            assertThat(logContent.toString(), is(""));
        }

        assertThat(outContent.toString(), is(""));
    }

}

```

```java
package unit.de.nordakademie.pdse.cellularAutomata.log;

import de.nordakademie.pdse.cellularAutomata.log.ConsoleLogger;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;

/**
 * Test class for {@link ConsoleLogger}.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 *
 */
public class ConsoleLoggerTest {

    private ByteArrayOutputStream outContent;
    private ConsoleLogger consoleLogger;

    @Before
    public void beforeEach() {
        this.outContent = new ByteArrayOutputStream();
        this.consoleLogger = new ConsoleLogger();
        consoleLogger.initAppender("experiment");

        System.setOut(new PrintStream(outContent));
    }

    @After
    public void afterEach() {
        System.setOut(System.out);
    }

    @Test
    public void testLogMessageOnConsole() {
        String testMessage = "test log message";
        this.consoleLogger.log(testMessage);
        assertThat(outContent.toString(), is(testMessage + "\r\n"));
    }

    @Test
    public void testLogMessageWithPlaceholderOnConsole() {
        String testMessage = "test {} message";
        this.consoleLogger.log(testMessage, "log");
        assertThat(outContent.toString(), is("test log message" + "\r\n"));
    }

    @Test
    public void testLogMessageWithMultiplePlaceholdersOnConsole() {
        String testMessage = "test {} {} {}";
        this.consoleLogger.log(testMessage, 1, "log", "message");
        assertThat(outContent.toString(), is("test 1 log message" + "\r\n"));
    }
}

```

```java
package unit.de.nordakademie.pdse.cellularAutomata.log;

import de.nordakademie.pdse.cellularAutomata.log.ConsoleAndFileLogger;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.util.Scanner;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;

/**
 * Test class for {@link ConsoleAndFileLogger}.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 *
 */
public class ConsoleAndFileLoggerTest {

    private ByteArrayOutputStream outContent;

    private File logFile;
    private ConsoleAndFileLogger consoleAndFileLogger;

    @Before
    public void beforeEach() {
        this.outContent = new ByteArrayOutputStream();
        this.logFile = new File(System.getProperty("user.dir") + File.separator + "experiment.log");
        this.consoleAndFileLogger = new ConsoleAndFileLogger();

        this.consoleAndFileLogger.initAppender("experiment");

        System.setOut(new PrintStream(outContent));
    }

    @After
    public void afterEach() {
        System.setOut(System.out);
        this.logFile.delete();
    }

    @Test
    public void testLogMessageOnConsoleAndFile() throws FileNotFoundException {
        String testMessage = "test log message";
        this.consoleAndFileLogger.log(testMessage);

        Scanner fileScanner = new Scanner(this.logFile);
        StringBuilder logContent = new StringBuilder();
        while (fileScanner.hasNextLine())
            logContent.append(fileScanner.nextLine());
        fileScanner.close();

        assertThat(outContent.toString(), is(testMessage + "\r\n"));
        assertThat(logContent.toString(), is(testMessage));
    }

    @Test
    public void testLogMessageWithPlaceholderOnConsoleAndFile() throws FileNotFoundException {
        String testMessage = "test {} message";
        this.consoleAndFileLogger.log(testMessage, "log");

        Scanner fileScanner = new Scanner(this.logFile);
        StringBuilder logContent = new StringBuilder();
        while (fileScanner.hasNextLine())
            logContent.append(fileScanner.nextLine());
        fileScanner.close();

```

```
73          assertThat(outContent.toString(), is("test log message" + "\r\n"));
74          assertThat(logContent.toString(), is("test log message"));
75      }
76
77      @Test
78      public void testLogMessageWithMultiplePlaceholdersOnConsoleAndFile() throws FileNotFoundException
    {
79          String testMessage = "test {} {} message";
80          this.consoleAndFileLogger.log(testMessage, 1, "log");
81
82          Scanner fileScanner = new Scanner(this.logFile);
83          StringBuilder logContent = new StringBuilder();
84          while (fileScanner.hasNextLine())
85              logContent.append(fileScanner.nextLine());
86          fileScanner.close();
87
88          assertThat(outContent.toString(), is("test 1 log message" + "\r\n"));
89          assertThat(logContent.toString(), is("test 1 log message"));
90      }
91  }
92
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.grid;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.exceptions.InvalidCellException;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DList;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.neighborhood.INeighborhood;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.IRuleset;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Matchers.anyInt;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

/**
 * Test class for {@link Grid2DList}.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class Grid2DListTest {

    private Grid2DList grid;

    private IRuleset gameOfLifeMock;
    private IRuleset parityMock;

    private INeighborhood neighborhoodMock;

    private IPattern startingPatternMock;

    @Before
    public void beforeEach() {
        initRulesetMocks();

        this.neighborhoodMock = mock(INeighborhood.class);
        this.startingPatternMock = mock(IPattern.class);
    }

    private void initRulesetMocks() {
        gameOfLifeMock = mock(IRuleset.class);
        when(gameOfLifeMock.getNextGenerationState(any(State.class), anyInt(), anyInt())).thenAnswer(
invocationOnMock -> {
            State currentState = (State) invocationOnMock.getArguments()[0];
            int neighborsAlive = (int) invocationOnMock.getArguments()[1];
            int maxNeighbors = (int) invocationOnMock.getArguments()[2];

            if(maxNeighbors < 0) {
                throw new IllegalArgumentException("MaxNeighbors should be greater than or equal to 0
but is " + maxNeighbors + ".");
            }

            if(neighborsAlive < 0 || neighborsAlive > maxNeighbors) {
                throw new IllegalArgumentException("NeighborsAlive should be between 0 and " +
maxNeighbors + " but was " + neighborsAlive + ".");
            }

            if(currentState == State.ALIVE) {
                if(neighborsAlive < 2 || neighborsAlive > 3) {
                    return State.DEAD;
                }

                return State.ALIVE;
            }
            else {
                if(neighborsAlive == 3) {
```

```
70                      return State.ALIVE;
71                  }
72
73                  return State.DEAD;
74              }
75          });
76
77          parityMock = mock(IRuleset.class);
78          when(parityMock.getNextGenerationState(any(State.class), anyInt(), anyInt())).thenAnswer(
    invocationOnMock -> {
79              int neighborsAlive = (int) invocationOnMock.getArguments()[1];
80              int maxNeighbors = (int) invocationOnMock.getArguments()[2];
81
82              if(maxNeighbors < 0) {
83                  throw new IllegalArgumentException("MaxNeighbors should be greater than or equal to 0
    but is " + maxNeighbors + ".");
84              }
85
86              if(neighborsAlive < 0 || neighborsAlive > maxNeighbors) {
87                  throw new IllegalArgumentException("NeighborsAlive should be between 0 and " +
    maxNeighbors + " but was " + neighborsAlive + " .");
88              }
89
90              if (neighborsAlive % 2 == 0) {
91                  return State.DEAD;
92              }
93
94              return State.ALIVE;
95          });
96      }
97
98      // region test constructor
99      @Test
100     public void testCreate1by1Grid() {
101         this.grid = new Grid2DList(1, 1, neighborhoodMock, startingPatternMock);
102
103         assertThat(grid.getGrid().size(), is(1));
104         assertThat(grid.getGrid().get(0).size(), is(1));
105     }
106
107     @Test
108     public void testCreate10by100Grid() {
109         grid = new Grid2DList(10, 100, neighborhoodMock, startingPatternMock);
110
111         assertThat(grid.getGrid().size(), is(10));
112         assertThat(grid.getGrid().get(0).size(), is(100));
113     }
114
115     @Test
116     public void testCreate9999by9999Grid() {
117         grid = new Grid2DList(9999, 9999, neighborhoodMock, startingPatternMock);
118
119         assertThat(grid.getGrid().size(), is(9999));
120         assertThat(grid.getGrid().get(0).size(), is(9999));
121     }
122     // endregion
123
124     // region test setState
125     @Test
126     public void testSetState() {
127         grid = new Grid2DList(10, 10, neighborhoodMock, startingPatternMock);
128         grid.setState(4, 5, State.ALIVE);
129
130         assertThat(grid.getState(4, 5), is(State.ALIVE));
131     }
132
133     @Test(expected = InvalidCellException.class)
134     public void testSetStateAtNegativeCol() {
135         Grid2DList grid = new Grid2DList(10, 10, neighborhoodMock, startingPatternMock);
136         grid.setState(-1, 0, State.ALIVE);
137     }
138
```

```
139     @Test(expected = InvalidCellException.class)
140     public void testSetStateAtNegativeRow() {
141         Grid2DList grid = new Grid2DList(10, 10, neighborhoodMock, startingPatternMock);
142         grid.setState(0, -1, State.ALIVE);
143     }
144 
145     @Test(expected = InvalidCellException.class)
146     public void testSetStateAtNegativeColAndRow() {
147         Grid2DList grid = new Grid2DList(10, 10, neighborhoodMock, startingPatternMock);
148         grid.setState(-1, -1, State.ALIVE);
149     }
150     // endregion
151 
152     // region test getState
153     @Test
154     public void testGetStateAt0And0() {
155         grid = new Grid2DList(10, 10, neighborhoodMock, startingPatternMock);
156         grid.setState(0, 0, State.ALIVE);
157 
158         assertThat(grid.getState(0, 0), is(State.ALIVE));
159     }
160 
161     @Test(expected = InvalidCellException.class)
162     public void testGetStateAtNegativeCol() {
163         grid = new Grid2DList(10, 10, neighborhoodMock, startingPatternMock);
164         grid.getState(-1, 0);
165     }
166 
167     @Test(expected = InvalidCellException.class)
168     public void testGetStateAtNegativeRow() {
169         grid = new Grid2DList(10, 10, neighborhoodMock, startingPatternMock);
170         grid.getState(0, -1);
171     }
172 
173     @Test(expected = InvalidCellException.class)
174     public void testGetStateAtNegativeColAndRow() {
175         grid = new Grid2DList(10, 10, neighborhoodMock, startingPatternMock);
176         grid.getState(-1, -1);
177     }
178     // endregion
179 
180     // region test equals
181     @Test
182     public void testEqualsForEqualGrids() {
183         grid = new Grid2DList(1, 1, neighborhoodMock, startingPatternMock);
184         IGrid other = new Grid2DList(1, 1, neighborhoodMock, startingPatternMock);
185 
186         assertThat(grid.equals(other), is(true));
187     }
188 
189     @Test
190     public void testEqualsForUnequalGrids() {
191         grid = new Grid2DList(1, 1, neighborhoodMock, startingPatternMock);
192         IGrid other = new Grid2DList(1, 1, neighborhoodMock, startingPatternMock);
193         other.setState(0, 0, State.ALIVE);
194 
195         assertThat(grid.equals(other), is(false));
196     }
197 
198     @Test
199     public void testEqualsForUnequalGridCols() {
200         grid = new Grid2DList(1, 1, neighborhoodMock, startingPatternMock);
201         IGrid other = new Grid2DList(2, 1, neighborhoodMock, startingPatternMock);
202 
203         assertThat(grid.equals(other), is(false));
204     }
205 
206     @Test
207     public void testEqualsForUnequalGridRows() {
208         grid = new Grid2DList(1, 1, neighborhoodMock, startingPatternMock);
209         IGrid other = new Grid2DList(1, 2, neighborhoodMock, startingPatternMock);
210 
```

```
211         assertThat(grid.equals(other), is(false));
212     }
213     // endregion
214 
215     // region test evolveNextGeneration
216     @Test
217     public void testEvolveNextGenerationForPatternChessFieldRulesetGameOfLife() {
218         grid = new Grid2DList(2, 2, neighborhoodMock, startingPatternMock);
219 
220         grid.setState(0,0, State.DEAD);
221         grid.setState(0,1, State.ALIVE);
222         grid.setState(1,0, State.ALIVE);
223         grid.setState(1,1, State.DEAD);
224 
225         IGrid other = new Grid2DList(2, 2, neighborhoodMock, startingPatternMock);
226 
227         assertThat(grid.evolveNextGeneration(gameOfLifeMock).equals(other), is(true));
228     }
229 
230     @Test
231     public void testEvolveNextGenerationForPatternChessFieldRulesetParity() {
232         grid = new Grid2DList(2, 2, neighborhoodMock, startingPatternMock);
233 
234         grid.setState(0,0, State.DEAD);
235         grid.setState(0,1, State.ALIVE);
236         grid.setState(1,0, State.ALIVE);
237         grid.setState(1,1, State.DEAD);
238 
239         IGrid other = new Grid2DList(2, 2, neighborhoodMock, startingPatternMock);
240 
241         assertThat(grid.evolveNextGeneration(parityMock).equals(other), is(true));
242     }
243     // endregion
244 
245     // region test grid configuration
246     @Test
247     public void testGetGridConfiguration() {
248         grid = new Grid2DList(3,3, neighborhoodMock, startingPatternMock);
249         assertThat(grid.getGridConfiguration(), isA(IGrid.IGridConfiguration.class));
250     }
251 
252     @Test
253     public void testGridConfigurationGetCols() {
254         grid = new Grid2DList(3,3, neighborhoodMock, startingPatternMock);
255         assertThat(grid.getGridConfiguration().getCols(), is(3));
256     }
257 
258     @Test
259     public void testGridConfigurationGetRows() {
260         grid = new Grid2DList(3,3, neighborhoodMock, startingPatternMock);
261         assertThat(grid.getGridConfiguration().getRows(), is(3));
262     }
263 
264     @Test(expected = IndexOutOfBoundsException.class)
265     public void testGridConfigurationIndexOutOfBoundsExceptionCols() {
266         grid = new Grid2DList(0,3, neighborhoodMock, startingPatternMock);
267     }
268 
269     @Test(expected = IndexOutOfBoundsException.class)
270     public void testGridConfigurationIndexOutOfBoundsExceptionRows() {
271         grid = new Grid2DList(3,0, neighborhoodMock, startingPatternMock);
272     }
273 
274     @Test(expected = IndexOutOfBoundsException.class)
275     public void testGridConfigurationIndexOutOfBoundsExceptionRowsAndCols() {
276         grid = new Grid2DList(0,0, neighborhoodMock, startingPatternMock);
277     }
278 
279     @Test
280     public void testGridConfigurationGetNeighborhood() {
281         grid = new Grid2DList(2,2, neighborhoodMock, startingPatternMock);
282         assertThat(grid.getGridConfiguration().getNeighborhood(), isA(INeighborhood.class));
```

```
283     }
284
285     @Test
286     public  void testGridConfigurationGetStartingPattern() {
287         grid = new Grid2DList(2,2, neighborhoodMock, startingPatternMock);
288         assertThat(grid.getGridConfiguration().getStartingPattern(), isA(IPattern.class));
289     }
290     // endregion
291 }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.grid;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.exceptions.InvalidCellException;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DArray;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.neighborhood.INeighborhood;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.IRuleset;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Matchers.anyInt;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

/**
 * Test class for {@link Grid2DArray}.
 *
 * @author 16593 (falko_vincent.partzsch@nordakademie.de)
 * @author 16455 (niklas.witzel@nordakademie.de)
 *
 */
public class Grid2DArrayTest {

    private Grid2DArray grid;

    private IRuleset gameOfLifeMock;
    private IRuleset parityMock;

    private INeighborhood neighborhoodMock;

    private IPattern startingPatternMock;

    @Before
    public void beforeEach() {
        initRulesetMocks();

        this.neighborhoodMock = mock(INeighborhood.class);
        this.startingPatternMock = mock(IPattern.class);
    }

    private void initRulesetMocks() {
        gameOfLifeMock = mock(IRuleset.class);
        when(gameOfLifeMock.getNextGenerationState(any(State.class), anyInt(), anyInt())).thenAnswer(
invocationOnMock -> {
            State currentState = (State) invocationOnMock.getArguments()[0];
            int neighborsAlive = (int) invocationOnMock.getArguments()[1];
            int maxNeighbors = (int) invocationOnMock.getArguments()[2];

            if(maxNeighbors < 0) {
                throw new IllegalArgumentException("MaxNeighbors should be greater than or equal to 0
but is " + maxNeighbors + ".");
            }

            if(neighborsAlive < 0 || neighborsAlive > maxNeighbors) {
                throw new IllegalArgumentException("NeighborsAlive should be between 0 and " +
maxNeighbors + " but was " + neighborsAlive + ".");
            }

            if(currentState == State.ALIVE) {
                if(neighborsAlive < 2 || neighborsAlive > 3) {
                    return State.DEAD;
                }

                return State.ALIVE;
            }
            else {
```

```
 70                 if(neighborsAlive == 3) {
 71                     return State.ALIVE;
 72                 }
 73
 74                 return State.DEAD;
 75             }
 76         });
 77
 78         parityMock = mock(IRuleset.class);
 79         when(parityMock.getNextGenerationState(any(State.class), anyInt(), anyInt())).thenAnswer(
    invocationOnMock -> {
 80             int neighborsAlive = (int) invocationOnMock.getArguments()[1];
 81             int maxNeighbors = (int) invocationOnMock.getArguments()[2];
 82
 83             if(maxNeighbors < 0) {
 84                 throw new IllegalArgumentException("MaxNeighbors should be greater than or equal to 0
    but is " + maxNeighbors + ".");
 85             }
 86
 87             if(neighborsAlive < 0 || neighborsAlive > maxNeighbors) {
 88                 throw new IllegalArgumentException("NeighborsAlive should be between 0 and " +
    maxNeighbors + " but was " + neighborsAlive + " .");
 89             }
 90
 91             if (neighborsAlive % 2 == 0) {
 92                 return State.DEAD;
 93             }
 94
 95             return State.ALIVE;
 96         });
 97     }
 98
 99     // region test constructor
100     @Test
101     public void testCreate1by1Grid() {
102         this.grid = new Grid2DArray(1, 1, neighborhoodMock, startingPatternMock);
103
104         assertThat(grid.getGrid().length, is(1));
105         assertThat(grid.getGrid()[0].length, is(1));
106     }
107
108     @Test
109     public void testCreate10by100Grid() {
110         grid = new Grid2DArray(10, 100, neighborhoodMock, startingPatternMock);
111
112         assertThat(grid.getGrid().length, is(10));
113         assertThat(grid.getGrid()[0].length, is(100));
114     }
115
116     @Test
117     public void testCreate9999by9999Grid() {
118         grid = new Grid2DArray(9999, 9999, neighborhoodMock, startingPatternMock);
119
120         assertThat(grid.getGrid().length, is(9999));
121         assertThat(grid.getGrid()[0].length, is(9999));
122     }
123     // endregion
124
125     // region test setState
126     @Test
127     public void testSetState() {
128         grid = new Grid2DArray(10, 10, neighborhoodMock, startingPatternMock);
129         grid.setState(4, 5, State.ALIVE);
130
131         assertThat(grid.getState(4, 5), is(State.ALIVE));
132     }
133
134     @Test(expected = InvalidCellException.class)
135     public void testSetStateAtNegativeCol() {
136         Grid2DArray grid = new Grid2DArray(10, 10, neighborhoodMock, startingPatternMock);
137         grid.setState(-1, 0, State.ALIVE);
138     }
```

```
139
140     @Test(expected = InvalidCellException.class)
141     public void testSetStateAtNegativeRow() {
142         Grid2DArray grid = new Grid2DArray(10, 10, neighborhoodMock, startingPatternMock);
143         grid.setState(0, -1, State.ALIVE);
144     }
145
146     @Test(expected = InvalidCellException.class)
147     public void testSetStateAtNegativeColAndRow() {
148         Grid2DArray grid = new Grid2DArray(10, 10, neighborhoodMock, startingPatternMock);
149         grid.setState(-1, -1, State.ALIVE);
150     }
151     // endregion
152
153     // region test getState
154     @Test
155     public void testGetStateAt0And0() {
156         grid = new Grid2DArray(10, 10, neighborhoodMock, startingPatternMock);
157         grid.setState(0, 0, State.ALIVE);
158
159         assertThat(grid.getState(0, 0), is(State.ALIVE));
160     }
161
162     @Test(expected = InvalidCellException.class)
163     public void testGetStateAtNegativeCol() {
164         grid = new Grid2DArray(10, 10, neighborhoodMock, startingPatternMock);
165         grid.getState(-1, 0);
166     }
167
168     @Test(expected = InvalidCellException.class)
169     public void testGetStateAtNegativeRow() {
170         grid = new Grid2DArray(10, 10, neighborhoodMock, startingPatternMock);
171         grid.getState(0, -1);
172     }
173
174     @Test(expected = InvalidCellException.class)
175     public void testGetStateAtNegativeColAndRow() {
176         grid = new Grid2DArray(10, 10, neighborhoodMock, startingPatternMock);
177         grid.getState(-1, -1);
178     }
179     // endregion
180
181     // region test equals
182     @Test
183     public void testEqualsForEqualGrids() {
184         grid = new Grid2DArray(1, 1, neighborhoodMock, startingPatternMock);
185         IGrid other = new Grid2DArray(1, 1, neighborhoodMock, startingPatternMock);
186
187         assertThat(grid.equals(other), is(true));
188     }
189
190     @Test
191     public void testEqualsForUnequalGrids() {
192         grid = new Grid2DArray(1, 1, neighborhoodMock, startingPatternMock);
193         IGrid other = new Grid2DArray(1, 1, neighborhoodMock, startingPatternMock);
194         other.setState(0, 0, State.ALIVE);
195
196         assertThat(grid.equals(other), is(false));
197     }
198
199     @Test
200     public void testEqualsForUnequalGridCols() {
201         grid = new Grid2DArray(1, 1, neighborhoodMock, startingPatternMock);
202         IGrid other = new Grid2DArray(2, 1, neighborhoodMock, startingPatternMock);
203
204         assertThat(grid.equals(other), is(false));
205     }
206
207     @Test
208     public void testEqualsForUnequalGridRows() {
209         grid = new Grid2DArray(1, 1, neighborhoodMock, startingPatternMock);
210         IGrid other = new Grid2DArray(1, 2, neighborhoodMock, startingPatternMock);
```

```
211
212         assertThat(grid.equals(other), is(false));
213     }
214     // endregion
215
216     // region test evolveNextGeneration
217     @Test
218     public void testEvolveNextGenerationForPatternChessFieldRulesetGameOfLife() {
219         grid = new Grid2DArray(2, 2, neighborhoodMock, startingPatternMock);
220         grid.setState(0,0, State.DEAD);
221         grid.setState(0,1, State.ALIVE);
222         grid.setState(1,0, State.ALIVE);
223         grid.setState(1,1, State.DEAD);
224
225         IGrid other = new Grid2DArray(2, 2, neighborhoodMock, startingPatternMock);
226         other.setState(0,0, State.DEAD);
227         other.setState(0,1, State.DEAD);
228         other.setState(1,0, State.DEAD);
229         other.setState(1,1, State.DEAD);
230
231         assertThat(grid.evolveNextGeneration(gameOfLifeMock).equals(other), is(true));
232     }
233
234     @Test
235     public void testEvolveNextGenerationForPatternChessFieldRulesetParity() {
236         grid = new Grid2DArray(2, 2, neighborhoodMock, startingPatternMock);
237         grid.setState(0,0, State.DEAD);
238         grid.setState(0,1, State.ALIVE);
239         grid.setState(1,0, State.ALIVE);
240         grid.setState(1,1, State.DEAD);
241
242         IGrid other = new Grid2DArray(2, 2, neighborhoodMock, startingPatternMock);
243         other.setState(0,0,State.DEAD);
244         other.setState(0,1,State.DEAD);
245         other.setState(1,0,State.DEAD);
246         other.setState(1,1,State.DEAD);
247
248         assertThat(grid.evolveNextGeneration(parityMock).equals(other), is(true));
249     }
250     // endregion
251
252     // region test grid configuration
253     @Test
254     public void testGetGridConfiguration() {
255         grid = new Grid2DArray(3,3, neighborhoodMock, startingPatternMock);
256         assertThat(grid.getGridConfiguration(), isA(IGrid.IGridConfiguration.class));
257     }
258
259     @Test
260     public void testGridConfigurationGetCols() {
261         grid = new Grid2DArray(3,3, neighborhoodMock, startingPatternMock);
262         assertThat(grid.getGridConfiguration().getCols(), is(3));
263     }
264
265     @Test
266     public void testGridConfigurationGetRows() {
267         grid = new Grid2DArray(3,3, neighborhoodMock, startingPatternMock);
268         assertThat(grid.getGridConfiguration().getRows(), is(3));
269     }
270
271     @Test(expected = IndexOutOfBoundsException.class)
272     public void testGridConfigurationIndexOutOfBoundsExceptionCols() {
273         grid = new Grid2DArray(0,3, neighborhoodMock, startingPatternMock);
274     }
275
276     @Test(expected = IndexOutOfBoundsException.class)
277     public void testGridConfigurationIndexOutOfBoundsExceptionRows() {
278         grid = new Grid2DArray(3,0, neighborhoodMock, startingPatternMock);
279     }
280
281     @Test(expected = IndexOutOfBoundsException.class)
282     public void testGridConfigurationIndexOutOfBoundsExceptionRowsAndCols() {
```

```
283         grid = new Grid2DArray(0,0, neighborhoodMock, startingPatternMock);
284     }
285 
286     @Test
287     public void testGridConfigurationGetNeighborhood() {
288         grid = new Grid2DArray(2,2, neighborhoodMock, startingPatternMock);
289         assertThat(grid.getGridConfiguration().getNeighborhood(), isA(INeighborhood.class));
290     }
291 
292     @Test
293     public  void testGridConfigurationGetStartingPattern() {
294         grid = new Grid2DArray(2,2, neighborhoodMock, startingPatternMock);
295         assertThat(grid.getGridConfiguration().getStartingPattern(), isA(IPattern.class));
296     }
297     // endregion
298 }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.pattern.simple;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.AllDeadPattern;
import org.junit.Before;
import org.junit.Test;

import static org.mockito.Matchers.anyInt;
import static org.mockito.Matchers.eq;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

/**
 * Test class for {@link AllDeadPattern}
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class AllDeadPatternTest {

    private IPattern allDeadPattern;

    private IGrid gridMock;
    private IGrid.IGridConfiguration gridConfigurationMock;

    @Before
    public void beforeEach() {
        this.allDeadPattern = new AllDeadPattern();

        gridConfigurationMock = mock(IGrid.IGridConfiguration.class);
        gridMock = mock(IGrid.class);

        when(gridMock.getGridConfiguration()).thenReturn(gridConfigurationMock);
    }

    @Test
    public void testApplyOnFor3x3Grid() {
        when(gridConfigurationMock.getCols()).thenReturn(3);
        when(gridConfigurationMock.getRows()).thenReturn(3);

        allDeadPattern.applyOn(gridMock);

        verify(gridMock, times(3 * 3)).setState(anyInt(), anyInt(), eq(State.DEAD));
    }

    @Test
    public void testApplyOnFor6x6Grid() {
        when(gridConfigurationMock.getCols()).thenReturn(6);
        when(gridConfigurationMock.getRows()).thenReturn(6);

        allDeadPattern.applyOn(gridMock);

        verify(gridMock, times(6 * 6)).setState(anyInt(), anyInt(), eq(State.DEAD));
    }
}

```

```java
package unit.de.nordakademie.pdse.cellularAutomata.pattern.simple;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.ChessFieldPattern;
import org.junit.Before;
import org.junit.Test;

import static org.mockito.Matchers.anyInt;
import static org.mockito.Matchers.eq;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

/**
 * Test class for {@link ChessFieldPattern}
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class ChessFieldPatternTest {

    private IPattern chessFieldPattern;

    private IGrid gridMock;
    private IGrid.IGridConfiguration gridConfigurationMock;

    @Before
    public void beforeEach() {
        this.chessFieldPattern = new ChessFieldPattern();

        gridConfigurationMock = mock(IGrid.IGridConfiguration.class);
        gridMock = mock(IGrid.class);

        when(gridMock.getGridConfiguration()).thenReturn(gridConfigurationMock);
    }

    @Test
    public void testApplyOnFor3x3Grid() {
        when(gridConfigurationMock.getCols()).thenReturn(3);
        when(gridConfigurationMock.getRows()).thenReturn(3);

        chessFieldPattern.applyOn(gridMock);

        verify(gridMock, times(3 * 3 / 2)).setState(anyInt(), anyInt(), eq(State.DEAD));
        verify(gridMock, times(3 * 3 / 2 + 1)).setState(anyInt(), anyInt(), eq(State.ALIVE));
    }

    @Test
    public void testApplyOnFor6x6Grid() {
        when(gridConfigurationMock.getCols()).thenReturn(6);
        when(gridConfigurationMock.getRows()).thenReturn(6);

        chessFieldPattern.applyOn(gridMock);

        verify(gridMock, times(6 * 6 / 2)).setState(anyInt(), anyInt(), eq(State.DEAD));
        verify(gridMock, times(6 * 6 / 2)).setState(anyInt(), anyInt(), eq(State.ALIVE));
    }
}

```

```java
package unit.de.nordakademie.pdse.cellularAutomata.pattern.complex;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.exceptions.GridTooSmallException;
import de.nordakademie.pdse.cellularAutomata.exceptions.InvalidCellException;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.IComplexPattern;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.SquarePattern;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.TumblerPattern;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.anyInt;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

/**
 * Test class for {@link SquarePattern}
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class SquarePatternTest {

    private IComplexPattern squarePattern;

    private IGrid gridMock;
    private IGrid.IGridConfiguration gridConfigurationMock;

    @Before
    public void beforeEach() {
        this.gridMock = mock(IGrid.class);
        this.gridConfigurationMock = mock(IGrid.IGridConfiguration.class);

        when(gridMock.getGridConfiguration()).thenReturn(gridConfigurationMock);
    }

    @Test(expected = InvalidCellException.class)
    public void testApplyOnWithInvalidStartingPosition() {
        this.squarePattern = new SquarePattern(-1,-1);

        when(gridMock.invalidCell(-1,-1)).thenReturn(true);

        squarePattern.applyOn(gridMock);
    }

    @Test(expected = GridTooSmallException.class)
    public void testApplyOnWithNotEnoughColumns() {
        this.squarePattern = new SquarePattern(0,0);

        when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
        when(gridConfigurationMock.getCols()).thenReturn(1);
        when(gridConfigurationMock.getRows()).thenReturn(10);

        squarePattern.applyOn(gridMock);
    }

    @Test(expected = GridTooSmallException.class)
    public void testApplyOnWithNotEnoughRows() {
        this.squarePattern = new TumblerPattern(0,0);

        when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
        when(gridConfigurationMock.getCols()).thenReturn(10);
        when(gridConfigurationMock.getRows()).thenReturn(1);

        squarePattern.applyOn(gridMock);
    }

    @Test
    public void testApplyOnFor2x2Grid() {
```

```
 73         this.squarePattern = new SquarePattern(0,0);
 74
 75         when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
 76         when(gridConfigurationMock.getCols()).thenReturn(2);
 77         when(gridConfigurationMock.getRows()).thenReturn(2);
 78
 79         squarePattern.applyOn(gridMock);
 80
 81         verify(gridMock).setState(0, 0, State.ALIVE);
 82         verify(gridMock).setState(0, 1, State.ALIVE);
 83         verify(gridMock).setState(1, 0, State.ALIVE);
 84         verify(gridMock).setState(1, 1, State.ALIVE);
 85     }
 86
 87     @Test
 88     public void testApplyOnFor10x10Grid() {
 89         this.squarePattern = new SquarePattern(4,4);
 90
 91         when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
 92         when(gridConfigurationMock.getCols()).thenReturn(10);
 93         when(gridConfigurationMock.getRows()).thenReturn(10);
 94
 95         squarePattern.applyOn(gridMock);
 96
 97         verify(gridMock).setState(4, 4, State.ALIVE);
 98         verify(gridMock).setState(4, 5, State.ALIVE);
 99         verify(gridMock).setState(5, 4, State.ALIVE);
100         verify(gridMock).setState(5, 5, State.ALIVE);
101     }
102
103     @Test
104     public void testGetPatternCols() {
105         this.squarePattern = new SquarePattern(0,0);
106         assertThat(squarePattern.getPatternCols(), is(2));
107     }
108
109     @Test
110     public void testGetPatternRows() {
111         this.squarePattern = new SquarePattern(0,0);
112         assertThat(squarePattern.getPatternRows(), is(2));
113     }
114
115     @Test
116     public void testGetStartingCol() {
117         this.squarePattern = new SquarePattern(5,7);
118         assertThat(squarePattern.getStartingCol(), is(5));
119     }
120
121     @Test
122     public void testGetStartingRow() {
123         this.squarePattern = new SquarePattern(5,7);
124         assertThat(squarePattern.getStartingRow(), is(7));
125     }
126 }
127
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.pattern.complex;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.exceptions.GridTooSmallException;
import de.nordakademie.pdse.cellularAutomata.exceptions.InvalidCellException;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.IComplexPattern;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.TumblerPattern;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.anyInt;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

/**
 * Test class for {@link TumblerPattern}
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class TumblerPatternTest {

    private IComplexPattern tumblerPattern;

    private IGrid gridMock;
    private IGrid.IGridConfiguration gridConfigurationMock;

    @Before
    public void beforeEach() {
        this.gridMock = mock(IGrid.class);
        this.gridConfigurationMock = mock(IGrid.IGridConfiguration.class);

        when(gridMock.getGridConfiguration()).thenReturn(gridConfigurationMock);
    }

    @Test(expected = InvalidCellException.class)
    public void testApplyOnWithInvalidStartingPosition() {
        this.tumblerPattern = new TumblerPattern(-1,-1);

        when(gridMock.invalidCell(-1,-1)).thenReturn(true);

        tumblerPattern.applyOn(gridMock);
    }

    @Test(expected = GridTooSmallException.class)
    public void testApplyOnWithNotEnoughColumns() {
        this.tumblerPattern = new TumblerPattern(0,0);

        when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
        when(gridConfigurationMock.getCols()).thenReturn(3);
        when(gridConfigurationMock.getRows()).thenReturn(10);

        tumblerPattern.applyOn(gridMock);
    }

    @Test(expected = GridTooSmallException.class)
    public void testApplyOnWithNotEnoughRows() {
        this.tumblerPattern = new TumblerPattern(0,0);

        when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
        when(gridConfigurationMock.getCols()).thenReturn(10);
        when(gridConfigurationMock.getRows()).thenReturn(3);

        tumblerPattern.applyOn(gridMock);
    }

    @Test
    public void testApplyOnFor7x6Grid() {
        this.tumblerPattern = new TumblerPattern(0,0);
```

```
 73
 74          when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
 75          when(gridConfigurationMock.getCols()).thenReturn(7);
 76          when(gridConfigurationMock.getRows()).thenReturn(6);
 77
 78          tumblerPattern.applyOn(gridMock);
 79
 80          verify(gridMock).setState(0, 3, State.ALIVE);
 81          verify(gridMock).setState(0, 4, State.ALIVE);
 82          verify(gridMock).setState(0, 5, State.ALIVE);
 83          verify(gridMock).setState(1, 0, State.ALIVE);
 84          verify(gridMock).setState(1, 1, State.ALIVE);
 85          verify(gridMock).setState(1, 5, State.ALIVE);
 86          verify(gridMock).setState(2, 0, State.ALIVE);
 87          verify(gridMock).setState(2, 1, State.ALIVE);
 88          verify(gridMock).setState(2, 2, State.ALIVE);
 89          verify(gridMock).setState(2, 3, State.ALIVE);
 90          verify(gridMock).setState(2, 4, State.ALIVE);
 91          verify(gridMock).setState(4, 0, State.ALIVE);
 92          verify(gridMock).setState(4, 1, State.ALIVE);
 93          verify(gridMock).setState(4, 2, State.ALIVE);
 94          verify(gridMock).setState(4, 3, State.ALIVE);
 95          verify(gridMock).setState(4, 4, State.ALIVE);
 96          verify(gridMock).setState(5, 0, State.ALIVE);
 97          verify(gridMock).setState(5, 1, State.ALIVE);
 98          verify(gridMock).setState(5, 5, State.ALIVE);
 99          verify(gridMock).setState(6, 3, State.ALIVE);
100          verify(gridMock).setState(6, 4, State.ALIVE);
101          verify(gridMock).setState(6, 5, State.ALIVE);
102      }
103
104      @Test
105      public void testApplyOnFor10x10Grid() {
106          this.tumblerPattern = new TumblerPattern(2,2);
107
108          when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
109          when(gridConfigurationMock.getCols()).thenReturn(10);
110          when(gridConfigurationMock.getRows()).thenReturn(10);
111
112          tumblerPattern.applyOn(gridMock);
113
114          verify(gridMock).setState(2, 5, State.ALIVE);
115          verify(gridMock).setState(2, 6, State.ALIVE);
116          verify(gridMock).setState(2, 7, State.ALIVE);
117          verify(gridMock).setState(3, 2, State.ALIVE);
118          verify(gridMock).setState(3, 3, State.ALIVE);
119          verify(gridMock).setState(3, 7, State.ALIVE);
120          verify(gridMock).setState(4, 2, State.ALIVE);
121          verify(gridMock).setState(4, 3, State.ALIVE);
122          verify(gridMock).setState(4, 4, State.ALIVE);
123          verify(gridMock).setState(4, 5, State.ALIVE);
124          verify(gridMock).setState(4, 6, State.ALIVE);
125          verify(gridMock).setState(6, 2, State.ALIVE);
126          verify(gridMock).setState(6, 3, State.ALIVE);
127          verify(gridMock).setState(6, 4, State.ALIVE);
128          verify(gridMock).setState(6, 5, State.ALIVE);
129          verify(gridMock).setState(6, 6, State.ALIVE);
130          verify(gridMock).setState(7, 2, State.ALIVE);
131          verify(gridMock).setState(7, 3, State.ALIVE);
132          verify(gridMock).setState(7, 7, State.ALIVE);
133          verify(gridMock).setState(8, 5, State.ALIVE);
134          verify(gridMock).setState(8, 6, State.ALIVE);
135          verify(gridMock).setState(8, 7, State.ALIVE);
136      }
137
138      @Test
139      public void testGetPatternCols() {
140          this.tumblerPattern = new TumblerPattern(0,0);
141          assertThat(tumblerPattern.getPatternCols(), is(7));
142      }
143
144      @Test
```

```
145     public void testGetPatternRows() {
146         this.tumblerPattern = new TumblerPattern(0,0);
147         assertThat(tumblerPattern.getPatternRows(), is(6));
148     }
149 
150     @Test
151     public void testGetStartingCol() {
152         this.tumblerPattern = new TumblerPattern(5,7);
153         assertThat(tumblerPattern.getStartingCol(), is(5));
154     }
155 
156     @Test
157     public void testGetStartingRow() {
158         this.tumblerPattern = new TumblerPattern(5,7);
159         assertThat(tumblerPattern.getStartingRow(), is(7));
160     }
161 }
162 
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.ruleset;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.ruleset.Parity;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;

/**
 * Test class for the {@link Parity}.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */

public class ParityTest {

    private Parity parity;

    @Before
    public void beforeEach() {
        this.parity = new Parity();
    }

    @Test
    public void getStateFromAliveCellWithoutAliveNeighbors() {
        assertThat(parity.getNextGenerationState(State.ALIVE, 0, 4), is(State.DEAD));
    }

    @Test
    public void getStateFromAliveCellWithOneAliveNeighbor() {
        assertThat(parity.getNextGenerationState(State.ALIVE, 1, 4), is(State.ALIVE));
    }

    @Test
    public void getStateFromAliveCellWithTwoAliveNeighbors() {
        assertThat(parity.getNextGenerationState(State.ALIVE, 2, 4), is(State.DEAD));
    }

    @Test
    public void getStateFromAliveCellWithThreeAliveNeighbors() {
        assertThat(parity.getNextGenerationState(State.ALIVE, 3, 4), is(State.ALIVE));
    }

    @Test
    public void getStateFromAliveCellWithFourAliveNeighbors() {
        assertThat(parity.getNextGenerationState(State.ALIVE, 4, 4), is(State.DEAD));
    }

    @Test(expected = IllegalArgumentException.class)
    public void getStateFromAliveCellWithToManyAliveNeighbors() {
        parity.getNextGenerationState(State.ALIVE, 5, 4);
    }

    @Test(expected = IllegalArgumentException.class)
    public void getStateFromAliveCellWithNegativeAliveNeighbors() {
        parity.getNextGenerationState(State.ALIVE, -1, 4);
    }

    @Test(expected = IllegalArgumentException.class)
    public void getStateFromAliveCellWithNegativeMaxNeighbors() {
        parity.getNextGenerationState(State.ALIVE, 0, -1);
    }

    @Test
    public void getStateFromDeadCellWithoutAliveNeighbors() {
        assertThat(parity.getNextGenerationState(State.DEAD, 0, 4), is(State.DEAD));
    }

    @Test
```

```
 73     public void getStateFromDeadCellWithOneAliveNeighbor() {
 74         assertThat(parity.getNextGenerationState(State.DEAD, 1, 4), is(State.ALIVE));
 75     }
 76
 77     @Test
 78     public void getStateFromDeadCellWithTwoAliveNeighbors() {
 79         assertThat(parity.getNextGenerationState(State.DEAD, 2, 4), is(State.DEAD));
 80     }
 81
 82     @Test
 83     public void getStateFromDeadCellWithThreeAliveNeighbors() {
 84         assertThat(parity.getNextGenerationState(State.DEAD, 3, 4), is(State.ALIVE));
 85     }
 86
 87     @Test
 88     public void getStateFromDeadCellWithFourAliveNeighbors() {
 89         assertThat(parity.getNextGenerationState(State.DEAD, 4, 4), is(State.DEAD));
 90     }
 91
 92     @Test(expected = IllegalArgumentException.class)
 93     public void getStateFromDeadCellWithToManyAliveNeighbors() {
 94         parity.getNextGenerationState(State.DEAD, 5, 4);
 95     }
 96
 97     @Test(expected = IllegalArgumentException.class)
 98     public void getStateFromDeadCellWithNegativeAliveNeighbors() {
 99         parity.getNextGenerationState(State.DEAD, -1, 4);
100     }
101
102     @Test(expected = IllegalArgumentException.class)
103     public void getStateFromDeadCellWithNegativeMaxNeighbors() {
104         parity.getNextGenerationState(State.DEAD, 0, -1);
105     }
106
107 }
108
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.ruleset;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.ruleset.GameOfLife;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;

/**
 * Test class for the {@link GameOfLife}.
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 * @author 16513 (merlin.ritsch@nordakademie.de)
 */

public class GameOfLifeTest {

    private GameOfLife gameOfLife;

    @Before
    public void beforeEach() {
        this.gameOfLife = new GameOfLife();
    }

    @Test
    public void getStateFromAliveCellWithoutAliveNeighbors() {
        assertThat(gameOfLife.getNextGenerationState(State.ALIVE, 0, 4), is(State.DEAD));
    }

    @Test
    public void getStateFromAliveCellWithOneAliveNeighbor() {
        assertThat(gameOfLife.getNextGenerationState(State.ALIVE, 1, 4), is(State.DEAD));
    }

    @Test
    public void getStateFromAliveCellWithTwoAliveNeighbors() {
        assertThat(gameOfLife.getNextGenerationState(State.ALIVE, 2, 4), is(State.ALIVE));
    }

    @Test
    public void getStateFromAliveCellWithThreeAliveNeighbors() {
        assertThat(gameOfLife.getNextGenerationState(State.ALIVE, 3, 4), is(State.ALIVE));
    }

    @Test
    public void getStateFromAliveCellWithFourAliveNeighbors() {
        assertThat(gameOfLife.getNextGenerationState(State.ALIVE, 4, 4), is(State.DEAD));
    }

    @Test(expected = IllegalArgumentException.class)
    public void getStateFromAliveCellWithToManyAliveNeighbors() {
        gameOfLife.getNextGenerationState(State.ALIVE, 5, 4);
    }

    @Test(expected = IllegalArgumentException.class)
    public void getStateFromAliveCellWithNegativeAliveNeighbors() {
        gameOfLife.getNextGenerationState(State.ALIVE, -1, 4);
    }

    @Test(expected = IllegalArgumentException.class)
    public void getStateFromAliveCellWithNegativeMaxNeighbors() {
        gameOfLife.getNextGenerationState(State.ALIVE, 0, -1);
    }

    @Test
    public void getStateFromDeadCellWithoutAliveNeighbors() {
        assertThat(gameOfLife.getNextGenerationState(State.DEAD, 0, 4), is(State.DEAD));
    }

    @Test
```

```
 73     public void getStateFromDeadCellWithOneAliveNeighbor() {
 74         assertThat(gameOfLife.getNextGenerationState(State.DEAD, 1, 4), is(State.DEAD));
 75     }
 76 
 77     @Test
 78     public void getStateFromDeadCellWithTwoAliveNeighbors() {
 79         assertThat(gameOfLife.getNextGenerationState(State.DEAD, 2, 4), is(State.DEAD));
 80     }
 81 
 82     @Test
 83     public void getStateFromDeadCellWithThreeAliveNeighbors() {
 84         assertThat(gameOfLife.getNextGenerationState(State.DEAD, 3, 4), is(State.ALIVE));
 85     }
 86 
 87     @Test
 88     public void getStateFromDeadCellWithFourAliveNeighbors() {
 89         assertThat(gameOfLife.getNextGenerationState(State.DEAD, 4, 4), is(State.DEAD));
 90     }
 91 
 92     @Test(expected = IllegalArgumentException.class)
 93     public void getStateFromDeadCellWithToManyAliveNeighbors() {
 94         gameOfLife.getNextGenerationState(State.DEAD, 5, 4);
 95     }
 96 
 97     @Test(expected = IllegalArgumentException.class)
 98     public void getStateFromDeadCellWithNegativeAliveNeighbors() {
 99         gameOfLife.getNextGenerationState(State.DEAD, -1, 4);
100     }
101 
102     @Test(expected = IllegalArgumentException.class)
103     public void getStateFromDeadCellWithNegativeMaxNeighbors() {
104         gameOfLife.getNextGenerationState(State.DEAD, 0, -1);
105     }
106 
107 
108 
109 }
110 
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.experiment.Parity_Experiment_1_1;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;

/**
 * Test class for {@link Parity_Experiment_1_1}
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class Parity_Experiment_1_1Test {

    private IExperiment experiment;
    private IAbortCondition abortConditionMock;

    @Before
    public void beforeEach() {
        this.abortConditionMock = mock(IAbortCondition.class);
        IExperimentLogger experimentLoggerMock = mock(IExperimentLogger.class);
        this.experiment = new Parity_Experiment_1_1(abortConditionMock,
                experimentLoggerMock);
    }

    @Test
    public void testRunWithoutNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(true);

        experiment.run();

        verify(abortConditionMock, times(1)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(1));
    }

    @Test
    public void testRunWithOneNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, true);

        experiment.run();

        verify(abortConditionMock, times(2)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(2));
    }

    @Test
    public void testRunWithFiveNewGenerations() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, false, false, false
, false, true);

        experiment.run();

        verify(abortConditionMock, times(6)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(6));
    }

    @Test
    public void testGetCurrentGeneration() {
        assertThat(experiment.getCurrentGeneration(), isA(IGrid.class));
    }

    @Test
```

```
72      public void testGetIteration() {
73          assertThat(experiment.getIterations(), isA(Integer.class));
74      }
75
76      @Test
77      public void testGetExperimentLogger() {
78          assertThat(experiment.getExperimentLogger(), isA(IExperimentLogger.class));
79      }
80  }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.experiment.Parity_Experiment_1_2;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;

/**
 * Test class for {@link Parity_Experiment_1_2}
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class Parity_Experiment_1_2Test {

    private IExperiment experiment;
    private IAbortCondition abortConditionMock;

    @Before
    public void beforeEach() {
        this.abortConditionMock = mock(IAbortCondition.class);
        IExperimentLogger experimentLoggerMock = mock(IExperimentLogger.class);
        this.experiment = new Parity_Experiment_1_2(abortConditionMock,
                experimentLoggerMock);
    }

    @Test
    public void testRunWithoutNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(true);

        experiment.run();

        verify(abortConditionMock, times(1)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(1));
    }

    @Test
    public void testRunWithOneNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, true);

        experiment.run();

        verify(abortConditionMock, times(2)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(2));
    }

    @Test
    public void testRunWithFiveNewGenerations() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, false, false, false
, false, true);

        experiment.run();

        verify(abortConditionMock, times(6)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(6));
    }

    @Test
    public void testGetCurrentGeneration() {
        assertThat(experiment.getCurrentGeneration(), isA(IGrid.class));
    }

    @Test
```

```
72      public void testGetIteration() {
73          assertThat(experiment.getIterations(), isA(Integer.class));
74      }
75
76      @Test
77      public void testGetExperimentLogger() {
78          assertThat(experiment.getExperimentLogger(), isA(IExperimentLogger.class));
79      }
80  }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.experiment.Parity_Experiment_2_1;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;

/**
 * Test class for {@link Parity_Experiment_2_1}
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class Parity_Experiment_2_1Test {

    private IExperiment experiment;
    private IAbortCondition abortConditionMock;

    @Before
    public void beforeEach() {
        this.abortConditionMock = mock(IAbortCondition.class);
        IExperimentLogger experimentLoggerMock = mock(IExperimentLogger.class);
        this.experiment = new Parity_Experiment_2_1(abortConditionMock,
                experimentLoggerMock);
    }

    @Test
    public void testRunWithoutNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(true);

        experiment.run();

        verify(abortConditionMock, times(1)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(1));
    }

    @Test
    public void testRunWithOneNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, true);

        experiment.run();

        verify(abortConditionMock, times(2)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(2));
    }

    @Test
    public void testRunWithFiveNewGenerations() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, false, false, false
, false, true);

        experiment.run();

        verify(abortConditionMock, times(6)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(6));
    }

    @Test
    public void testGetCurrentGeneration() {
        assertThat(experiment.getCurrentGeneration(), isA(IGrid.class));
    }

    @Test
```

```
72      public void testGetIteration() {
73          assertThat(experiment.getIterations(), isA(Integer.class));
74      }
75
76      @Test
77      public void testGetExperimentLogger() {
78          assertThat(experiment.getExperimentLogger(), isA(IExperimentLogger.class));
79      }
80  }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.experiment.Parity_Experiment_2_2;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;

/**
 * Test class for {@link Parity_Experiment_2_2}
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class Parity_Experiment_2_2Test {

    private IExperiment experiment;
    private IAbortCondition abortConditionMock;

    @Before
    public void beforeEach() {
        this.abortConditionMock = mock(IAbortCondition.class);
        IExperimentLogger experimentLoggerMock = mock(IExperimentLogger.class);
        this.experiment = new Parity_Experiment_2_2(abortConditionMock,
                experimentLoggerMock);
    }

    @Test
    public void testRunWithoutNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(true);

        experiment.run();

        verify(abortConditionMock, times(1)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(1));
    }

    @Test
    public void testRunWithOneNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, true);

        experiment.run();

        verify(abortConditionMock, times(2)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(2));
    }

    @Test
    public void testRunWithFiveNewGenerations() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, false, false, false
, false, true);

        experiment.run();

        verify(abortConditionMock, times(6)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(6));
    }

    @Test
    public void testGetCurrentGeneration() {
        assertThat(experiment.getCurrentGeneration(), isA(IGrid.class));
    }

    @Test
```

```
72      public void testGetIteration() {
73          assertThat(experiment.getIterations(), isA(Integer.class));
74      }
75
76      @Test
77      public void testGetExperimentLogger() {
78          assertThat(experiment.getExperimentLogger(), isA(IExperimentLogger.class));
79      }
80  }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.GameOfLife_Experiment_1_1;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;

/**
 * Test class for {@link GameOfLife_Experiment_1_1}
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class GameOfLife_Experiment_1_1Test {

    private IExperiment experiment;
    private IAbortCondition abortConditionMock;

    @Before
    public void beforeEach() {
        this.abortConditionMock = mock(IAbortCondition.class);
        IExperimentLogger experimentLoggerMock = mock(IExperimentLogger.class);
        this.experiment = new GameOfLife_Experiment_1_1(abortConditionMock,
                experimentLoggerMock);
    }

    @Test
    public void testRunWithoutNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(true);

        experiment.run();

        verify(abortConditionMock, times(1)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(1));
    }

    @Test
    public void testRunWithOneNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, true);

        experiment.run();

        verify(abortConditionMock, times(2)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(2));
    }

    @Test
    public void testRunWithFiveNewGenerations() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, false, false, false
, false, true);

        experiment.run();

        verify(abortConditionMock, times(6)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(6));
    }

    @Test
    public void testGetCurrentGeneration() {
        assertThat(experiment.getCurrentGeneration(), isA(IGrid.class));
    }

    @Test
```

```
72      public void testGetIteration() {
73          assertThat(experiment.getIterations(), isA(Integer.class));
74      }
75
76      @Test
77      public void testGetExperimentLogger() {
78          assertThat(experiment.getExperimentLogger(), isA(IExperimentLogger.class));
79      }
80  }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.GameOfLife_Experiment_1_2;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;

/**
 * Test class for {@link GameOfLife_Experiment_1_2}
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class GameOfLife_Experiment_1_2Test {

    private IExperiment experiment;
    private IAbortCondition abortConditionMock;

    @Before
    public void beforeEach() {
        this.abortConditionMock = mock(IAbortCondition.class);
        IExperimentLogger experimentLoggerMock = mock(IExperimentLogger.class);
        this.experiment = new GameOfLife_Experiment_1_2(abortConditionMock,
                experimentLoggerMock);
    }

    @Test
    public void testRunWithoutNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(true);

        experiment.run();

        verify(abortConditionMock, times(1)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(1));
    }

    @Test
    public void testRunWithOneNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, true);

        experiment.run();

        verify(abortConditionMock, times(2)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(2));
    }

    @Test
    public void testRunWithFiveNewGenerations() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, false, false, false
, false, true);

        experiment.run();

        verify(abortConditionMock, times(6)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(6));
    }

    @Test
    public void testGetCurrentGeneration() {
        assertThat(experiment.getCurrentGeneration(), isA(IGrid.class));
    }

    @Test
```

```
72      public void testGetIteration() {
73          assertThat(experiment.getIterations(), isA(Integer.class));
74      }
75
76      @Test
77      public void testGetExperimentLogger() {
78          assertThat(experiment.getExperimentLogger(), isA(IExperimentLogger.class));
79      }
80  }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.GameOfLife_Experiment_2_1;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;

/**
 * Test class for {@link GameOfLife_Experiment_2_1}
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class GameOfLife_Experiment_2_1Test {

    private IExperiment experiment;
    private IAbortCondition abortConditionMock;

    @Before
    public void beforeEach() {
        this.abortConditionMock = mock(IAbortCondition.class);
        IExperimentLogger experimentLoggerMock = mock(IExperimentLogger.class);
        this.experiment = new GameOfLife_Experiment_2_1(abortConditionMock,
                experimentLoggerMock);
    }

    @Test
    public void testRunWithoutNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(true);

        experiment.run();

        verify(abortConditionMock, times(1)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(1));
    }

    @Test
    public void testRunWithOneNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, true);

        experiment.run();

        verify(abortConditionMock, times(2)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(2));
    }

    @Test
    public void testRunWithFiveNewGenerations() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, false, false, false
, false, true);

        experiment.run();

        verify(abortConditionMock, times(6)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(6));
    }

    @Test
    public void testGetCurrentGeneration() {
        assertThat(experiment.getCurrentGeneration(), isA(IGrid.class));
    }

    @Test
```

```
72      public void testGetIteration() {
73          assertThat(experiment.getIterations(), isA(Integer.class));
74      }
75
76      @Test
77      public void testGetExperimentLogger() {
78          assertThat(experiment.getExperimentLogger(), isA(IExperimentLogger.class));
79      }
80  }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.experiment;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

import org.junit.Before;
import org.junit.Test;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.GameOfLife_Experiment_2_2;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;

/**
 * Test class for {@link GameOfLife_Experiment_2_2}
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class GameOfLife_Experiment_2_2Test {

    private IExperiment experiment;
    private IAbortCondition abortConditionMock;

    @Before
    public void beforeEach() {
        this.abortConditionMock = mock(IAbortCondition.class);
        IExperimentLogger experimentLoggerMock = mock(IExperimentLogger.class);
        this.experiment = new GameOfLife_Experiment_2_2(abortConditionMock,
                experimentLoggerMock);
    }

    @Test
    public void testRunWithoutNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(true);

        experiment.run();

        verify(abortConditionMock, times(1)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(1));
    }

    @Test
    public void testRunWithOneNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, true);

        experiment.run();

        verify(abortConditionMock, times(2)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(2));
    }

    @Test
    public void testRunWithFiveNewGenerations() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, false, false, false
, false, true);

        experiment.run();

        verify(abortConditionMock, times(6)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(6));
    }

    @Test
    public void testGetCurrentGeneration() {
```

```
72          assertThat(experiment.getCurrentGeneration(), isA(IGrid.class));
73      }
74
75      @Test
76      public void testGetIteration() {
77          assertThat(experiment.getIterations(), isA(Integer.class));
78      }
79
80      @Test
81      public void testGetExperimentLogger() {
82          assertThat(experiment.getExperimentLogger(), isA(IExperimentLogger.class));
83      }
84  }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.GameOfLife_Experiment_3_1;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;

/**
 * Test class for {@link GameOfLife_Experiment_3_1}
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class GameOfLife_Experiment_3_1Test {

    private IExperiment experiment;
    private IAbortCondition abortConditionMock;

    @Before
    public void beforeEach() {
        this.abortConditionMock = mock(IAbortCondition.class);
        IExperimentLogger experimentLoggerMock = mock(IExperimentLogger.class);
        this.experiment = new GameOfLife_Experiment_3_1(abortConditionMock,
                experimentLoggerMock);
    }

    @Test
    public void testRunWithoutNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(true);

        experiment.run();

        verify(abortConditionMock, times(1)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(1));
    }

    @Test
    public void testRunWithOneNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, true);

        experiment.run();

        verify(abortConditionMock, times(2)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(2));
    }

    @Test
    public void testRunWithFiveNewGenerations() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, false, false, false
, false, true);

        experiment.run();

        verify(abortConditionMock, times(6)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(6));
    }

    @Test
    public void testGetCurrentGeneration() {
        assertThat(experiment.getCurrentGeneration(), isA(IGrid.class));
    }

    @Test
```

```
72      public void testGetIteration() {
73          assertThat(experiment.getIterations(), isA(Integer.class));
74      }
75
76      @Test
77      public void testGetExperimentLogger() {
78          assertThat(experiment.getExperimentLogger(), isA(IExperimentLogger.class));
79      }
80  }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.experiment.GameOfLife_Experiment_3_2;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.isA;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;

/**
 * Test class for {@link GameOfLife_Experiment_3_2}
 *
 * @author 16513 (merlin.ritsch@nordakademie.de)
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class GameOfLife_Experiment_3_2Test {

    private IExperiment experiment;
    private IAbortCondition abortConditionMock;
    private IExperimentLogger experimentLoggerMock;

    @Before
    public void beforeEach() {
        this.abortConditionMock = mock(IAbortCondition.class);
        this.experimentLoggerMock = mock(IExperimentLogger.class);
        this.experiment = new GameOfLife_Experiment_3_2(abortConditionMock,
                experimentLoggerMock);
    }

    @Test
    public void testRunWithoutNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(true);

        experiment.run();

        verify(abortConditionMock, times(1)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(1));
    }

    @Test
    public void testRunWithOneNewGeneration() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, true);

        experiment.run();

        verify(abortConditionMock, times(2)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(2));
    }

    @Test
    public void testRunWithFiveNewGenerations() {
        when(abortConditionMock.isMetFor(any(IExperiment.class))).thenReturn(false, false, false, false
, false, true);

        experiment.run();

        verify(abortConditionMock, times(6)).isMetFor(any(IExperiment.class));
        assertThat(experiment.getIterations(), is(6));
    }

    @Test
    public void testGetCurrentGeneration() {
        assertThat(experiment.getCurrentGeneration(), isA(IGrid.class));
    }

```

```
72      @Test
73      public void testGetIteration() {
74          assertThat(experiment.getIterations(), isA(Integer.class));
75      }
76
77      @Test
78      public void testGetExperimentLogger() {
79          assertThat(experiment.getExperimentLogger(), isA(IExperimentLogger.class));
80      }
81  }
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.neighborhood;

import de.nordakademie.pdse.cellularAutomata.enums.Neighbor;
import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.exceptions.InvalidCellException;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.neighborhood.INeighborhood;
import de.nordakademie.pdse.cellularAutomata.neighborhood.Moore;
import org.junit.Before;
import org.junit.Test;

import java.util.Arrays;
import java.util.List;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.anyInt;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

/**
 * Test class for {@link Moore}
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class MooreTest {

    private INeighborhood mooreNeighborhood;
    private IGrid gridMock;

    @Before
    public void beforeEach() {
        this.mooreNeighborhood = new Moore();
        this.gridMock = mock(IGrid.class);
    }

    @Test(expected = InvalidCellException.class)
    public void testCountNeighborsAliveWithInvalidCell() {
        when(gridMock.invalidCell(-1, -1)).thenReturn(true);

        mooreNeighborhood.countNeighborsAlive(-1,-1, gridMock);
    }

    @Test
    public void testCountNeighborsAliveWithZeroNeighborsAlive() {
        when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
        when(gridMock.getState(anyInt(), anyInt())).thenReturn(State.DEAD);

        assertThat(mooreNeighborhood.countNeighborsAlive(1,1, gridMock), is(0));
    }

    @Test
    public void testCountNeighborsAliveWithFiveNeighborsAlive() {
        when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);

        when(gridMock.getState(anyInt(), anyInt())).thenReturn(State.DEAD);
        when(gridMock.getState(0,0)).thenReturn(State.ALIVE);
        when(gridMock.getState(0,1)).thenReturn(State.ALIVE);
        when(gridMock.getState(1,1)).thenReturn(State.ALIVE);
        when(gridMock.getState(1,2)).thenReturn(State.ALIVE);
        when(gridMock.getState(2,0)).thenReturn(State.ALIVE);
        when(gridMock.getState(2,1)).thenReturn(State.ALIVE);

        assertThat(mooreNeighborhood.countNeighborsAlive(1,1, gridMock), is(5));
    }

    @Test
    public void testCountNeighborsAliveWithSideCell() {
        when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
        when(gridMock.invalidCell(1,-1)).thenReturn(true);

        when(gridMock.getState(anyInt(), anyInt())).thenReturn(State.DEAD);
```

```
 73          when(gridMock.getState(0,0)).thenReturn(State.ALIVE);
 74          when(gridMock.getState(0,1)).thenReturn(State.ALIVE);
 75          when(gridMock.getState(0,2)).thenReturn(State.ALIVE);
 76          when(gridMock.getState(1,0)).thenReturn(State.ALIVE);
 77          when(gridMock.getState(1,1)).thenReturn(State.ALIVE);
 78          when(gridMock.getState(1,2)).thenReturn(State.ALIVE);
 79          when(gridMock.getState(2,0)).thenReturn(State.ALIVE);
 80          when(gridMock.getState(2,1)).thenReturn(State.ALIVE);
 81          when(gridMock.getState(2,2)).thenReturn(State.ALIVE);
 82
 83          assertThat(mooreNeighborhood.countNeighborsAlive(1,0, gridMock), is(5));
 84      }
 85
 86      @Test
 87      public void testCountNeighborsAliveWithCornerCell() {
 88          when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
 89          when(gridMock.invalidCell(0,-1)).thenReturn(true);
 90          when(gridMock.invalidCell(-1,0)).thenReturn(true);
 91
 92          when(gridMock.getState(anyInt(), anyInt())).thenReturn(State.DEAD);
 93          when(gridMock.getState(0,0)).thenReturn(State.ALIVE);
 94          when(gridMock.getState(0,1)).thenReturn(State.ALIVE);
 95          when(gridMock.getState(0,2)).thenReturn(State.ALIVE);
 96          when(gridMock.getState(1,0)).thenReturn(State.ALIVE);
 97          when(gridMock.getState(1,1)).thenReturn(State.ALIVE);
 98          when(gridMock.getState(1,2)).thenReturn(State.ALIVE);
 99          when(gridMock.getState(2,0)).thenReturn(State.ALIVE);
100          when(gridMock.getState(2,1)).thenReturn(State.ALIVE);
101          when(gridMock.getState(2,2)).thenReturn(State.ALIVE);
102
103          assertThat(mooreNeighborhood.countNeighborsAlive(0,0, gridMock), is(3));
104      }
105
106      @Test
107      public void testGetNeighbors() {
108          List<Neighbor> neighbors = Arrays.asList(Neighbor.TOP_LEFT, Neighbor.TOP, Neighbor.TOP_RIGHT,
    Neighbor.LEFT,
109                  Neighbor.RIGHT, Neighbor.BOTTOM_LEFT, Neighbor.BOTTOM, Neighbor.BOTTOM_RIGHT);
110
111          assertThat(mooreNeighborhood.getNeighbors().size(), is(8));
112          assertThat(mooreNeighborhood.getNeighbors(), is(neighbors));
113      }
114 }
115
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.neighborhood;

import de.nordakademie.pdse.cellularAutomata.enums.Neighbor;
import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.exceptions.InvalidCellException;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.neighborhood.INeighborhood;
import de.nordakademie.pdse.cellularAutomata.neighborhood.VonNeumann;
import org.junit.Before;
import org.junit.Test;

import java.util.Arrays;
import java.util.List;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.anyInt;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

/**
 * Test class for {@link VonNeumann}
 *
 * @author 16455 (niklas.witzel@nordakademie.de)
 */
public class VonNeumannTest {

    private INeighborhood vonNeumannNeighborhood;
    private IGrid gridMock;

    @Before
    public void beforeEach() {
        this.vonNeumannNeighborhood = new VonNeumann();
        this.gridMock = mock(IGrid.class);
    }

    @Test(expected = InvalidCellException.class)
    public void testCountNeighborsAliveWithInvalidCell() {
        when(gridMock.invalidCell(-1, -1)).thenReturn(true);

        vonNeumannNeighborhood.countNeighborsAlive(-1,-1, gridMock);
    }

    @Test
    public void testCountNeighborsAliveWithZeroNeighborsAlive() {
        when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
        when(gridMock.getState(anyInt(), anyInt())).thenReturn(State.DEAD);

        assertThat(vonNeumannNeighborhood.countNeighborsAlive(1,1, gridMock), is(0));
    }

    @Test
    public void testCountNeighborsAliveWithFiveNeighborsAlive() {
        when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);

        when(gridMock.getState(anyInt(), anyInt())).thenReturn(State.DEAD);
        when(gridMock.getState(0,0)).thenReturn(State.ALIVE);
        when(gridMock.getState(0,1)).thenReturn(State.ALIVE);
        when(gridMock.getState(1,1)).thenReturn(State.ALIVE);
        when(gridMock.getState(1,2)).thenReturn(State.ALIVE);
        when(gridMock.getState(2,0)).thenReturn(State.ALIVE);
        when(gridMock.getState(2,1)).thenReturn(State.ALIVE);

        assertThat(vonNeumannNeighborhood.countNeighborsAlive(1,1, gridMock), is(3));
    }

    @Test
    public void testCountNeighborsAliveWithSideCell() {
        when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
        when(gridMock.invalidCell(1,-1)).thenReturn(true);

        when(gridMock.getState(anyInt(), anyInt())).thenReturn(State.DEAD);
```

```
 73          when(gridMock.getState(0,0)).thenReturn(State.ALIVE);
 74          when(gridMock.getState(0,1)).thenReturn(State.ALIVE);
 75          when(gridMock.getState(0,2)).thenReturn(State.ALIVE);
 76          when(gridMock.getState(1,0)).thenReturn(State.ALIVE);
 77          when(gridMock.getState(1,1)).thenReturn(State.ALIVE);
 78          when(gridMock.getState(1,2)).thenReturn(State.ALIVE);
 79          when(gridMock.getState(2,0)).thenReturn(State.ALIVE);
 80          when(gridMock.getState(2,1)).thenReturn(State.ALIVE);
 81          when(gridMock.getState(2,2)).thenReturn(State.ALIVE);
 82
 83          assertThat(vonNeumannNeighborhood.countNeighborsAlive(1,0, gridMock), is(3));
 84      }
 85
 86      @Test
 87      public void testCountNeighborsAliveWithCornerCell() {
 88          when(gridMock.invalidCell(anyInt(), anyInt())).thenReturn(false);
 89          when(gridMock.invalidCell(0,-1)).thenReturn(true);
 90          when(gridMock.invalidCell(-1,0)).thenReturn(true);
 91
 92          when(gridMock.getState(anyInt(), anyInt())).thenReturn(State.DEAD);
 93          when(gridMock.getState(0,0)).thenReturn(State.ALIVE);
 94          when(gridMock.getState(0,1)).thenReturn(State.ALIVE);
 95          when(gridMock.getState(0,2)).thenReturn(State.ALIVE);
 96          when(gridMock.getState(1,0)).thenReturn(State.ALIVE);
 97          when(gridMock.getState(1,1)).thenReturn(State.ALIVE);
 98          when(gridMock.getState(1,2)).thenReturn(State.ALIVE);
 99          when(gridMock.getState(2,0)).thenReturn(State.ALIVE);
100          when(gridMock.getState(2,1)).thenReturn(State.ALIVE);
101          when(gridMock.getState(2,2)).thenReturn(State.ALIVE);
102
103          assertThat(vonNeumannNeighborhood.countNeighborsAlive(0,0, gridMock), is(2));
104      }
105
106      @Test
107      public void testGetNeighbors() {
108          List<Neighbor> neighbors = Arrays.asList(Neighbor.TOP, Neighbor.LEFT, Neighbor.RIGHT, Neighbor
    .BOTTOM);
109
110          assertThat(vonNeumannNeighborhood.getNeighbors().size(), is(4));
111          assertThat(vonNeumannNeighborhood.getNeighbors(), is(neighbors));
112      }
113  }
114
```

```java
package unit.de.nordakademie.pdse.cellularAutomata.abortCondition;

import de.nordakademie.pdse.cellularAutomata.abortCondition.Simple.NoChange;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Matchers.any;
import static org.mockito.Matchers.anyInt;
import static org.mockito.Mockito.doNothing;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

/**
 * Test class for {@link NoChange}
 *
 *  @author 16513 (merlin.ritsch@nordakademie.de)
 *  @author 17015 (til.zoeller@nordakademie.de)
 */
public class NoChangeTest {

    private NoChange noChange;

    private IExperiment experimentMock;
    private IGrid currentGridMock;
    private IExperimentLogger experimentLoggerMock;

    @Before
    public void beforeEach() {
        noChange = new NoChange();

        experimentMock = mock(IExperiment.class);
        currentGridMock = mock(IGrid.class);
        experimentLoggerMock = mock(IExperimentLogger.class);

        when(experimentMock.getCurrentGeneration()).thenReturn(currentGridMock);
        when(experimentMock.getExperimentLogger()).thenReturn(experimentLoggerMock);
    }

    @Test
    public void testIsMetForEqualGrids() {
        when(currentGridMock.equals(any(IGrid.class))).thenReturn(true);
        assertThat(noChange.isMetFor(experimentMock), is(true));
    }

    @Test
    public void testIsDoneWithUnequal() {
        when(currentGridMock.equals(any (IGrid.class))).thenReturn(false);
        assertThat(noChange.isMetFor(experimentMock), is(false));
    }
}

```

```java
package unit.de.nordakademie.pdse.cellularAutomata.abortCondition;

import de.nordakademie.pdse.cellularAutomata.abortCondition.complex.FixedNumber;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import org.junit.Before;
import org.junit.Test;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;

import static org.mockito.Mockito.*;

/**
 * Test class for {@link FixedNumber}
 *
 *  @author 16513 (merlin.ritsch@nordakademie.de)
 *  @author 17015 (til.zoeller@nordakademie.de)
 */
public class FixedNumberTest {

    private FixedNumber fixedNumber;
    private IExperiment experimentMock;

    @Before
    public void beforeEach() {
        fixedNumber = new FixedNumber(10);
        experimentMock = mock(IExperiment.class);
    }

    @Test
    public void testStopAfterMaxIterations() {
        when(experimentMock.getIterations()).thenReturn(10);
        assertThat(fixedNumber.isMetFor(experimentMock), is(true));
    }

    @Test
    public void testContinueWhenIterationGreaterThanMaxIteration() {
        when(experimentMock.getIterations()).thenReturn(11);
        assertThat(fixedNumber.isMetFor(experimentMock), is(true));
    }

    @Test
    public void testContinueWhenIterationSmallerThanMaxIteration() {
        when(experimentMock.getIterations()).thenReturn(9);
        assertThat(fixedNumber.isMetFor(experimentMock), is(false));
    }

    @Test (expected = IllegalArgumentException.class)
    public void testIsDoneWithIllegalMaxIterations() {
        when(experimentMock.getIterations()).thenReturn(-5);
        fixedNumber.isMetFor(experimentMock);
    }



}

```

```java
package unit.de.nordakademie.pdse.cellularAutomata.abortCondition;

import de.nordakademie.pdse.cellularAutomata.abortCondition.complex.FixedNumberOrNoChange;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.log.IExperimentLogger;
import org.junit.Before;
import org.junit.Test;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;

import static org.mockito.Mockito.*;

/**
 * Test class for {@link FixedNumberOrNoChange}
 *
 *  @author 16513 (merlin.ritsch@nordakademie.de)
 *  @author 17015 (til.zoeller@nordakademie.de)
 */
public class FixedNumberOrNoChangeTest {

    private FixedNumberOrNoChange fixedNumberOrNoChange;

    private IExperiment experimentMock;
    private IGrid currentGridMock;
    private IExperimentLogger experimentLoggerMock;

    @Before
    public void beforeEach() {
        fixedNumberOrNoChange = new FixedNumberOrNoChange(10);

        experimentMock = mock(IExperiment.class);
        currentGridMock = mock(IGrid.class);
        experimentLoggerMock = mock(IExperimentLogger.class);

        when(experimentMock.getCurrentGeneration()).thenReturn(currentGridMock);
        when(experimentMock.getExperimentLogger()).thenReturn(experimentLoggerMock);
    }

    @Test
    public void testStopEqualsMaxIterationsAndChange() {
        when(currentGridMock.equals(any (IGrid.class))).thenReturn(false);
        when(experimentMock.getIterations()).thenReturn(10);

        assertThat(fixedNumberOrNoChange.isMetFor(experimentMock), is(true));
    }

    @Test
    public void testContinueBeforeMaxIterationAndChange() {
        when(currentGridMock.equals(any (IGrid.class))).thenReturn(false);
        when(experimentMock.getIterations()).thenReturn(9);

        assertThat(fixedNumberOrNoChange.isMetFor(experimentMock), is(false));
    }

    @Test
    public void testIsDoneBeforeMaxIterationAndNoChange() {
        when(currentGridMock.equals(any (IGrid.class))).thenReturn(true);
        when(experimentMock.getIterations()).thenReturn(9);

        assertThat(fixedNumberOrNoChange.isMetFor(experimentMock), is(true));
    }

    @Test
    public void testIsDoneEqualsMaxIterationAndNoChange() {
        when(currentGridMock.equals(any (IGrid.class))).thenReturn(true);
        when(experimentMock.getIterations()).thenReturn(10);

        assertThat(fixedNumberOrNoChange.isMetFor(experimentMock), is(true));
    }

    @Test
```

```
73      public void testIsDoneAfterMaxIterationAndChange() {
74          when(currentGridMock.equals(any (IGrid.class))).thenReturn(false);
75          when(experimentMock.getIterations()).thenReturn(11);
76
77          assertThat(fixedNumberOrNoChange.isMetFor(experimentMock), is(true));
78      }
79
80
81
82
83
84  }
85
```

```java
package integration.de.nordakademie.pdse.cellularAutomata.grid;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DList;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.neighborhood.INeighborhood;
import de.nordakademie.pdse.cellularAutomata.neighborhood.Moore;
import de.nordakademie.pdse.cellularAutomata.neighborhood.VonNeumann;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.SquarePattern;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.TumblerPattern;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.AllDeadPattern;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.ChessFieldPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.GameOfLife;
import de.nordakademie.pdse.cellularAutomata.ruleset.IRuleset;
import de.nordakademie.pdse.cellularAutomata.ruleset.Parity;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;

/**
 * Integration-test class for {@link Grid2DList}
 * to test the interaction with the classes of {@link IRuleset}, {@link INeighborhood} and {@link IPattern}
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class Grid2DListIntegrationTest {

    private IRuleset gameOfLife;
    private IRuleset parity;

    private INeighborhood moore;
    private INeighborhood vonNeumann;

    private IPattern chessField;
    private IPattern allDead;
    private IPattern tumblerPattern;
    private IPattern squarePattern;

    @Before
    public void beforeEach() {
        this.gameOfLife = new GameOfLife();
        this.parity = new Parity();

        this.moore = new Moore();
        this.vonNeumann = new VonNeumann();

        this.chessField = new ChessFieldPattern();
        this.allDead = new AllDeadPattern();
        this.tumblerPattern = new TumblerPattern(0, 0);
        this.squarePattern = new SquarePattern(0, 0);
    }

    @Test
    public void testEvolveNextGenerationWithMooreNeighborhoodAndChessFieldPatternAndGameOfLifeRuleset() {
        IGrid grid = new Grid2DList(3, 3, this.moore, this.chessField);

        IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);

        assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
        assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
        assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
        assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
        assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
        assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
        assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
        assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
        assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
```

```
 71     }
 72
 73     @Test
 74     public void testEvolveNextGenerationWithMooreNeighborhoodAndChessFieldPatternAndParityRuleset() {
 75         IGrid grid = new Grid2DList(3, 3, this.moore, this.chessField);
 76
 77         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
 78
 79         assertThat(nextGenerationGrid.getState(0, 0), is(State.ALIVE));
 80         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
 81         assertThat(nextGenerationGrid.getState(0, 2), is(State.ALIVE));
 82         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
 83         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
 84         assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
 85         assertThat(nextGenerationGrid.getState(2, 0), is(State.ALIVE));
 86         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
 87         assertThat(nextGenerationGrid.getState(2, 2), is(State.ALIVE));
 88     }
 89
 90     @Test
 91     public void testEvolveNextGenerationWithMooreNeighborhoodAndAllDeadPatternAndGameOfLifeRuleset() {
 92         IGrid grid = new Grid2DList(3, 3, this.moore, this.allDead);
 93
 94         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
 95
 96         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
 97         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
 98         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
 99         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
100         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
101         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
102         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
103         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
104         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
105     }
106
107     @Test
108     public void testEvolveNextGenerationWithMooreNeighborhoodAndAllDeadPatternAndParityRuleset() {
109         IGrid grid = new Grid2DList(3, 3, this.moore, this.allDead);
110
111         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
112
113         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
114         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
115         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
116         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
117         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
118         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
119         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
120         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
121         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
122     }
123
124     @Test
125     public void testEvolveNextGenerationWithMooreNeighborhoodAndTumblerPatternAndGameOfLifeRuleset() {
126         IGrid grid = new Grid2DList(7, 6, this.moore, this.tumblerPattern);
127
128         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
129
130         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
131         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
132         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
133         assertThat(nextGenerationGrid.getState(0, 3), is(State.DEAD));
134         assertThat(nextGenerationGrid.getState(0, 4), is(State.ALIVE));
135         assertThat(nextGenerationGrid.getState(0, 5), is(State.ALIVE));
136
137         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
138         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
139         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
140         assertThat(nextGenerationGrid.getState(1, 3), is(State.DEAD));
141         assertThat(nextGenerationGrid.getState(1, 4), is(State.DEAD));
142         assertThat(nextGenerationGrid.getState(1, 5), is(State.ALIVE));
```

```
143
144         assertThat(nextGenerationGrid.getState(2, 0), is(State.ALIVE));
145         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
146         assertThat(nextGenerationGrid.getState(2, 2), is(State.ALIVE));
147         assertThat(nextGenerationGrid.getState(2, 3), is(State.ALIVE));
148         assertThat(nextGenerationGrid.getState(2, 4), is(State.ALIVE));
149         assertThat(nextGenerationGrid.getState(2, 5), is(State.DEAD));
150
151         assertThat(nextGenerationGrid.getState(3, 0), is(State.DEAD));
152         assertThat(nextGenerationGrid.getState(3, 1), is(State.DEAD));
153         assertThat(nextGenerationGrid.getState(3, 2), is(State.DEAD));
154         assertThat(nextGenerationGrid.getState(3, 3), is(State.DEAD));
155         assertThat(nextGenerationGrid.getState(3, 4), is(State.DEAD));
156         assertThat(nextGenerationGrid.getState(3, 5), is(State.DEAD));
157
158         assertThat(nextGenerationGrid.getState(4, 0), is(State.ALIVE));
159         assertThat(nextGenerationGrid.getState(4, 1), is(State.DEAD));
160         assertThat(nextGenerationGrid.getState(4, 2), is(State.ALIVE));
161         assertThat(nextGenerationGrid.getState(4, 3), is(State.ALIVE));
162         assertThat(nextGenerationGrid.getState(4, 4), is(State.ALIVE));
163         assertThat(nextGenerationGrid.getState(4, 5), is(State.DEAD));
164
165         assertThat(nextGenerationGrid.getState(5, 0), is(State.ALIVE));
166         assertThat(nextGenerationGrid.getState(5, 1), is(State.DEAD));
167         assertThat(nextGenerationGrid.getState(5, 2), is(State.DEAD));
168         assertThat(nextGenerationGrid.getState(5, 3), is(State.DEAD));
169         assertThat(nextGenerationGrid.getState(5, 4), is(State.DEAD));
170         assertThat(nextGenerationGrid.getState(5, 5), is(State.ALIVE));
171
172         assertThat(nextGenerationGrid.getState(6, 0), is(State.DEAD));
173         assertThat(nextGenerationGrid.getState(6, 1), is(State.DEAD));
174         assertThat(nextGenerationGrid.getState(6, 2), is(State.DEAD));
175         assertThat(nextGenerationGrid.getState(6, 3), is(State.DEAD));
176         assertThat(nextGenerationGrid.getState(6, 4), is(State.ALIVE));
177         assertThat(nextGenerationGrid.getState(6, 5), is(State.ALIVE));
178     }
179
180     @Test
181     public void testEvolveNextGenerationWithMooreNeighborhoodAndTumblerPatternAndParityRuleset() {
182         IGrid grid = new Grid2DList(7, 6, this.moore, this.tumblerPattern);
183
184         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
185
186         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
187         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
188         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
189         assertThat(nextGenerationGrid.getState(0, 3), is(State.ALIVE));
190         assertThat(nextGenerationGrid.getState(0, 4), is(State.ALIVE));
191         assertThat(nextGenerationGrid.getState(0, 5), is(State.DEAD));
192
193         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
194         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
195         assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
196         assertThat(nextGenerationGrid.getState(1, 3), is(State.ALIVE));
197         assertThat(nextGenerationGrid.getState(1, 4), is(State.DEAD));
198         assertThat(nextGenerationGrid.getState(1, 5), is(State.ALIVE));
199
200         assertThat(nextGenerationGrid.getState(2, 0), is(State.ALIVE));
201         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
202         assertThat(nextGenerationGrid.getState(2, 2), is(State.ALIVE));
203         assertThat(nextGenerationGrid.getState(2, 3), is(State.DEAD));
204         assertThat(nextGenerationGrid.getState(2, 4), is(State.DEAD));
205         assertThat(nextGenerationGrid.getState(2, 5), is(State.DEAD));
206
207         assertThat(nextGenerationGrid.getState(3, 0), is(State.DEAD));
208         assertThat(nextGenerationGrid.getState(3, 1), is(State.DEAD));
209         assertThat(nextGenerationGrid.getState(3, 2), is(State.DEAD));
210         assertThat(nextGenerationGrid.getState(3, 3), is(State.DEAD));
211         assertThat(nextGenerationGrid.getState(3, 4), is(State.DEAD));
212         assertThat(nextGenerationGrid.getState(3, 5), is(State.DEAD));
213
214         assertThat(nextGenerationGrid.getState(4, 0), is(State.ALIVE));
```

```
215         assertThat(nextGenerationGrid.getState(4, 1), is(State.DEAD));
216         assertThat(nextGenerationGrid.getState(4, 2), is(State.ALIVE));
217         assertThat(nextGenerationGrid.getState(4, 3), is(State.DEAD));
218         assertThat(nextGenerationGrid.getState(4, 4), is(State.DEAD));
219         assertThat(nextGenerationGrid.getState(4, 5), is(State.DEAD));
220 
221         assertThat(nextGenerationGrid.getState(5, 0), is(State.ALIVE));
222         assertThat(nextGenerationGrid.getState(5, 1), is(State.DEAD));
223         assertThat(nextGenerationGrid.getState(5, 2), is(State.ALIVE));
224         assertThat(nextGenerationGrid.getState(5, 3), is(State.ALIVE));
225         assertThat(nextGenerationGrid.getState(5, 4), is(State.DEAD));
226         assertThat(nextGenerationGrid.getState(5, 5), is(State.ALIVE));
227 
228         assertThat(nextGenerationGrid.getState(6, 0), is(State.DEAD));
229         assertThat(nextGenerationGrid.getState(6, 1), is(State.DEAD));
230         assertThat(nextGenerationGrid.getState(6, 2), is(State.DEAD));
231         assertThat(nextGenerationGrid.getState(6, 3), is(State.ALIVE));
232         assertThat(nextGenerationGrid.getState(6, 4), is(State.ALIVE));
233         assertThat(nextGenerationGrid.getState(6, 5), is(State.DEAD));
234     }
235 
236     @Test
237     public void testEvolveNextGenerationWithMooreNeighborhoodAnSquarePatternAndGameOfLifeRuleset() {
238         IGrid grid = new Grid2DList(3, 3, this.moore, this.squarePattern);
239 
240         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
241 
242         assertThat(nextGenerationGrid.getState(0, 0), is(State.ALIVE));
243         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
244         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
245         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
246         assertThat(nextGenerationGrid.getState(1, 1), is(State.ALIVE));
247         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
248         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
249         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
250         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
251     }
252 
253     @Test
254     public void testEvolveNextGenerationWithMooreNeighborhoodAnSquarePatternAndParityRuleset() {
255         IGrid grid = new Grid2DList(3, 3, this.moore, this.squarePattern);
256 
257         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
258 
259         assertThat(nextGenerationGrid.getState(0, 0), is(State.ALIVE));
260         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
261         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
262         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
263         assertThat(nextGenerationGrid.getState(1, 1), is(State.ALIVE));
264         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
265         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
266         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
267         assertThat(nextGenerationGrid.getState(2, 2), is(State.ALIVE));
268     }
269 
270     @Test
271     public void
    testEvolveNextGenerationWithVonNeumannNeighborhoodAndChessFieldPatternAndGameOfLifeRuleset() {
272         IGrid grid = new Grid2DList(3, 3, this.vonNeumann, this.chessField);
273 
274         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
275 
276         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
277         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
278         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
279         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
280         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
281         assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
282         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
283         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
284         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
285     }
```

```
286
287     @Test
288     public void testEvolveNextGenerationWithVonNeumannNeighborhoodAndChessFieldPatternAndParityRuleset
    () {
289         IGrid grid = new Grid2DList(3, 3, this.vonNeumann, this.chessField);
290
291         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
292
293         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
294         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
295         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
296         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
297         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
298         assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
299         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
300         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
301         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
302     }
303
304     @Test
305     public void
    testEvolveNextGenerationWithVonNeumannNeighborhoodAndAllDeadPatternAndGameOfLifeRuleset() {
306         IGrid grid = new Grid2DList(3, 3, this.vonNeumann, this.allDead);
307
308         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
309
310         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
311         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
312         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
313         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
314         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
315         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
316         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
317         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
318         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
319     }
320
321     @Test
322     public void testEvolveNextGenerationWithVonNeumannNeighborhoodAndAllDeadPatternAndParityRuleset
    () {
323         IGrid grid = new Grid2DList(3, 3, this.vonNeumann, this.allDead);
324
325         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
326
327         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
328         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
329         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
330         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
331         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
332         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
333         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
334         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
335         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
336     }
337
338     @Test
339     public void
    testEvolveNextGenerationWithVonNeumannNeighborhoodAndTumblerPatternAndGameOfLifeRuleset() {
340         IGrid grid = new Grid2DList(7, 6, this.vonNeumann, this.tumblerPattern);
341
342         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
343
344         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
345         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
346         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
347         assertThat(nextGenerationGrid.getState(0, 3), is(State.DEAD));
348         assertThat(nextGenerationGrid.getState(0, 4), is(State.ALIVE));
349         assertThat(nextGenerationGrid.getState(0, 5), is(State.ALIVE));
350
351         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
352         assertThat(nextGenerationGrid.getState(1, 1), is(State.ALIVE));
353         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
```

```
354         assertThat(nextGenerationGrid.getState(1, 3), is(State.DEAD));
355         assertThat(nextGenerationGrid.getState(1, 4), is(State.ALIVE));
356         assertThat(nextGenerationGrid.getState(1, 5), is(State.DEAD));
357 
358         assertThat(nextGenerationGrid.getState(2, 0), is(State.ALIVE));
359         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
360         assertThat(nextGenerationGrid.getState(2, 2), is(State.ALIVE));
361         assertThat(nextGenerationGrid.getState(2, 3), is(State.ALIVE));
362         assertThat(nextGenerationGrid.getState(2, 4), is(State.DEAD));
363         assertThat(nextGenerationGrid.getState(2, 5), is(State.DEAD));
364 
365         assertThat(nextGenerationGrid.getState(3, 0), is(State.DEAD));
366         assertThat(nextGenerationGrid.getState(3, 1), is(State.DEAD));
367         assertThat(nextGenerationGrid.getState(3, 2), is(State.DEAD));
368         assertThat(nextGenerationGrid.getState(3, 3), is(State.DEAD));
369         assertThat(nextGenerationGrid.getState(3, 4), is(State.DEAD));
370         assertThat(nextGenerationGrid.getState(3, 5), is(State.DEAD));
371 
372         assertThat(nextGenerationGrid.getState(4, 0), is(State.ALIVE));
373         assertThat(nextGenerationGrid.getState(4, 1), is(State.ALIVE));
374         assertThat(nextGenerationGrid.getState(4, 2), is(State.ALIVE));
375         assertThat(nextGenerationGrid.getState(4, 3), is(State.ALIVE));
376         assertThat(nextGenerationGrid.getState(4, 4), is(State.DEAD));
377         assertThat(nextGenerationGrid.getState(4, 5), is(State.DEAD));
378 
379         assertThat(nextGenerationGrid.getState(5, 0), is(State.ALIVE));
380         assertThat(nextGenerationGrid.getState(5, 1), is(State.ALIVE));
381         assertThat(nextGenerationGrid.getState(5, 2), is(State.DEAD));
382         assertThat(nextGenerationGrid.getState(5, 3), is(State.DEAD));
383         assertThat(nextGenerationGrid.getState(5, 4), is(State.ALIVE));
384         assertThat(nextGenerationGrid.getState(5, 5), is(State.DEAD));
385 
386         assertThat(nextGenerationGrid.getState(6, 0), is(State.DEAD));
387         assertThat(nextGenerationGrid.getState(6, 1), is(State.DEAD));
388         assertThat(nextGenerationGrid.getState(6, 2), is(State.DEAD));
389         assertThat(nextGenerationGrid.getState(6, 3), is(State.DEAD));
390         assertThat(nextGenerationGrid.getState(6, 4), is(State.ALIVE));
391         assertThat(nextGenerationGrid.getState(6, 5), is(State.ALIVE));
392     }
393 
394     @Test
395     public void testEvolveNextGenerationWithVonNeumannNeighborhoodAndTumblerPatternAndParityRuleset
    () {
396         IGrid grid = new Grid2DList(7, 6, this.vonNeumann, this.tumblerPattern);
397 
398         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
399 
400         assertThat(nextGenerationGrid.getState(0, 0), is(State.ALIVE));
401         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
402         assertThat(nextGenerationGrid.getState(0, 2), is(State.ALIVE));
403         assertThat(nextGenerationGrid.getState(0, 3), is(State.ALIVE));
404         assertThat(nextGenerationGrid.getState(0, 4), is(State.DEAD));
405         assertThat(nextGenerationGrid.getState(0, 5), is(State.DEAD));
406 
407         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
408         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
409         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
410         assertThat(nextGenerationGrid.getState(1, 3), is(State.DEAD));
411         assertThat(nextGenerationGrid.getState(1, 4), is(State.ALIVE));
412         assertThat(nextGenerationGrid.getState(1, 5), is(State.ALIVE));
413 
414         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
415         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
416         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
417         assertThat(nextGenerationGrid.getState(2, 3), is(State.DEAD));
418         assertThat(nextGenerationGrid.getState(2, 4), is(State.ALIVE));
419         assertThat(nextGenerationGrid.getState(2, 5), is(State.DEAD));
420 
421         assertThat(nextGenerationGrid.getState(3, 0), is(State.DEAD));
422         assertThat(nextGenerationGrid.getState(3, 1), is(State.DEAD));
423         assertThat(nextGenerationGrid.getState(3, 2), is(State.DEAD));
424         assertThat(nextGenerationGrid.getState(3, 3), is(State.DEAD));
```

```
425         assertThat(nextGenerationGrid.getState(3, 4), is(State.DEAD));
426         assertThat(nextGenerationGrid.getState(3, 5), is(State.DEAD));
427
428         assertThat(nextGenerationGrid.getState(4, 0), is(State.DEAD));
429         assertThat(nextGenerationGrid.getState(4, 1), is(State.ALIVE));
430         assertThat(nextGenerationGrid.getState(4, 2), is(State.DEAD));
431         assertThat(nextGenerationGrid.getState(4, 3), is(State.DEAD));
432         assertThat(nextGenerationGrid.getState(4, 4), is(State.ALIVE));
433         assertThat(nextGenerationGrid.getState(4, 5), is(State.DEAD));
434
435         assertThat(nextGenerationGrid.getState(5, 0), is(State.DEAD));
436         assertThat(nextGenerationGrid.getState(5, 1), is(State.DEAD));
437         assertThat(nextGenerationGrid.getState(5, 2), is(State.DEAD));
438         assertThat(nextGenerationGrid.getState(5, 3), is(State.DEAD));
439         assertThat(nextGenerationGrid.getState(5, 4), is(State.ALIVE));
440         assertThat(nextGenerationGrid.getState(5, 5), is(State.ALIVE));
441
442         assertThat(nextGenerationGrid.getState(6, 0), is(State.ALIVE));
443         assertThat(nextGenerationGrid.getState(6, 1), is(State.ALIVE));
444         assertThat(nextGenerationGrid.getState(6, 2), is(State.ALIVE));
445         assertThat(nextGenerationGrid.getState(6, 3), is(State.ALIVE));
446         assertThat(nextGenerationGrid.getState(6, 4), is(State.DEAD));
447         assertThat(nextGenerationGrid.getState(6, 5), is(State.DEAD));
448     }
449
450     @Test
451     public void testEvolveNextGenerationWithVonNeumannNeighborhoodAnSquarePatternAndGameOfLifeRuleset
    () {
452         IGrid grid = new Grid2DList(3, 3, this.vonNeumann, this.squarePattern);
453
454         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
455
456         assertThat(nextGenerationGrid.getState(0, 0), is(State.ALIVE));
457         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
458         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
459         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
460         assertThat(nextGenerationGrid.getState(1, 1), is(State.ALIVE));
461         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
462         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
463         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
464         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
465     }
466
467     @Test
468     public void testEvolveNextGenerationWithVonNeumannNeighborhoodAnSquarePatternAndParityRuleset() {
469         IGrid grid = new Grid2DList(3, 3, this.vonNeumann, this.squarePattern);
470
471         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
472
473         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
474         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
475         assertThat(nextGenerationGrid.getState(0, 2), is(State.ALIVE));
476         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
477         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
478         assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
479         assertThat(nextGenerationGrid.getState(2, 0), is(State.ALIVE));
480         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
481         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
482     }
483 }
484
```

```java
package integration.de.nordakademie.pdse.cellularAutomata.grid;

import de.nordakademie.pdse.cellularAutomata.enums.State;
import de.nordakademie.pdse.cellularAutomata.grid.Grid2DArray;
import de.nordakademie.pdse.cellularAutomata.grid.IGrid;
import de.nordakademie.pdse.cellularAutomata.neighborhood.INeighborhood;
import de.nordakademie.pdse.cellularAutomata.neighborhood.Moore;
import de.nordakademie.pdse.cellularAutomata.neighborhood.VonNeumann;
import de.nordakademie.pdse.cellularAutomata.pattern.IPattern;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.SquarePattern;
import de.nordakademie.pdse.cellularAutomata.pattern.complex.TumblerPattern;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.AllDeadPattern;
import de.nordakademie.pdse.cellularAutomata.pattern.simple.ChessFieldPattern;
import de.nordakademie.pdse.cellularAutomata.ruleset.GameOfLife;
import de.nordakademie.pdse.cellularAutomata.ruleset.IRuleset;
import de.nordakademie.pdse.cellularAutomata.ruleset.Parity;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;

/**
 * Integration-test class for {@link Grid2DArray}
 * to test the interaction with the classes of {@link IRuleset}, {@link INeighborhood} and {@link IPattern}
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class Grid2DArrayIntegrationTest {

    private IRuleset gameOfLife;
    private IRuleset parity;

    private INeighborhood moore;
    private INeighborhood vonNeumann;

    private IPattern chessField;
    private IPattern allDead;
    private IPattern tumblerPattern;
    private IPattern squarePattern;

    @Before
    public void beforeEach() {
        this.gameOfLife = new GameOfLife();
        this.parity = new Parity();

        this.moore = new Moore();
        this.vonNeumann = new VonNeumann();

        this.chessField = new ChessFieldPattern();
        this.allDead = new AllDeadPattern();
        this.tumblerPattern = new TumblerPattern(0, 0);
        this.squarePattern = new SquarePattern(0, 0);
    }

    @Test
    public void testEvolveNextGenerationWithMooreNeighborhoodAndChessFieldPatternAndGameOfLifeRuleset() {
        IGrid grid = new Grid2DArray(3, 3, this.moore, this.chessField);

        IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);

        assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
        assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
        assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
        assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
        assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
        assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
        assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
        assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
        assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
```

```
 71     }
 72
 73     @Test
 74     public void testEvolveNextGenerationWithMooreNeighborhoodAndChessFieldPatternAndParityRuleset() {
 75         IGrid grid = new Grid2DArray(3, 3, this.moore, this.chessField);
 76
 77         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
 78
 79         assertThat(nextGenerationGrid.getState(0, 0), is(State.ALIVE));
 80         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
 81         assertThat(nextGenerationGrid.getState(0, 2), is(State.ALIVE));
 82         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
 83         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
 84         assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
 85         assertThat(nextGenerationGrid.getState(2, 0), is(State.ALIVE));
 86         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
 87         assertThat(nextGenerationGrid.getState(2, 2), is(State.ALIVE));
 88     }
 89
 90     @Test
 91     public void testEvolveNextGenerationWithMooreNeighborhoodAndAllDeadPatternAndGameOfLifeRuleset() {
 92         IGrid grid = new Grid2DArray(3, 3, this.moore, this.allDead);
 93
 94         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
 95
 96         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
 97         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
 98         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
 99         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
100         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
101         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
102         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
103         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
104         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
105     }
106
107     @Test
108     public void testEvolveNextGenerationWithMooreNeighborhoodAndAllDeadPatternAndParityRuleset() {
109         IGrid grid = new Grid2DArray(3, 3, this.moore, this.allDead);
110
111         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
112
113         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
114         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
115         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
116         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
117         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
118         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
119         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
120         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
121         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
122     }
123
124     @Test
125     public void testEvolveNextGenerationWithMooreNeighborhoodAndTumblerPatternAndGameOfLifeRuleset() {
126         IGrid grid = new Grid2DArray(7, 6, this.moore, this.tumblerPattern);
127
128         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
129
130         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
131         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
132         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
133         assertThat(nextGenerationGrid.getState(0, 3), is(State.DEAD));
134         assertThat(nextGenerationGrid.getState(0, 4), is(State.ALIVE));
135         assertThat(nextGenerationGrid.getState(0, 5), is(State.ALIVE));
136
137         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
138         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
139         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
140         assertThat(nextGenerationGrid.getState(1, 3), is(State.DEAD));
141         assertThat(nextGenerationGrid.getState(1, 4), is(State.DEAD));
142         assertThat(nextGenerationGrid.getState(1, 5), is(State.ALIVE));
```

```
143
144         assertThat(nextGenerationGrid.getState(2, 0), is(State.ALIVE));
145         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
146         assertThat(nextGenerationGrid.getState(2, 2), is(State.ALIVE));
147         assertThat(nextGenerationGrid.getState(2, 3), is(State.ALIVE));
148         assertThat(nextGenerationGrid.getState(2, 4), is(State.ALIVE));
149         assertThat(nextGenerationGrid.getState(2, 5), is(State.DEAD));
150
151         assertThat(nextGenerationGrid.getState(3, 0), is(State.DEAD));
152         assertThat(nextGenerationGrid.getState(3, 1), is(State.DEAD));
153         assertThat(nextGenerationGrid.getState(3, 2), is(State.DEAD));
154         assertThat(nextGenerationGrid.getState(3, 3), is(State.DEAD));
155         assertThat(nextGenerationGrid.getState(3, 4), is(State.DEAD));
156         assertThat(nextGenerationGrid.getState(3, 5), is(State.DEAD));
157
158         assertThat(nextGenerationGrid.getState(4, 0), is(State.ALIVE));
159         assertThat(nextGenerationGrid.getState(4, 1), is(State.DEAD));
160         assertThat(nextGenerationGrid.getState(4, 2), is(State.ALIVE));
161         assertThat(nextGenerationGrid.getState(4, 3), is(State.ALIVE));
162         assertThat(nextGenerationGrid.getState(4, 4), is(State.ALIVE));
163         assertThat(nextGenerationGrid.getState(4, 5), is(State.DEAD));
164
165         assertThat(nextGenerationGrid.getState(5, 0), is(State.ALIVE));
166         assertThat(nextGenerationGrid.getState(5, 1), is(State.DEAD));
167         assertThat(nextGenerationGrid.getState(5, 2), is(State.DEAD));
168         assertThat(nextGenerationGrid.getState(5, 3), is(State.DEAD));
169         assertThat(nextGenerationGrid.getState(5, 4), is(State.DEAD));
170         assertThat(nextGenerationGrid.getState(5, 5), is(State.ALIVE));
171
172         assertThat(nextGenerationGrid.getState(6, 0), is(State.DEAD));
173         assertThat(nextGenerationGrid.getState(6, 1), is(State.DEAD));
174         assertThat(nextGenerationGrid.getState(6, 2), is(State.DEAD));
175         assertThat(nextGenerationGrid.getState(6, 3), is(State.DEAD));
176         assertThat(nextGenerationGrid.getState(6, 4), is(State.ALIVE));
177         assertThat(nextGenerationGrid.getState(6, 5), is(State.ALIVE));
178     }
179
180     @Test
181     public void testEvolveNextGenerationWithMooreNeighborhoodAndTumblerPatternAndParityRuleset() {
182         IGrid grid = new Grid2DArray(7, 6, this.moore, this.tumblerPattern);
183
184         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
185
186         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
187         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
188         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
189         assertThat(nextGenerationGrid.getState(0, 3), is(State.ALIVE));
190         assertThat(nextGenerationGrid.getState(0, 4), is(State.ALIVE));
191         assertThat(nextGenerationGrid.getState(0, 5), is(State.DEAD));
192
193         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
194         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
195         assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
196         assertThat(nextGenerationGrid.getState(1, 3), is(State.ALIVE));
197         assertThat(nextGenerationGrid.getState(1, 4), is(State.DEAD));
198         assertThat(nextGenerationGrid.getState(1, 5), is(State.ALIVE));
199
200         assertThat(nextGenerationGrid.getState(2, 0), is(State.ALIVE));
201         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
202         assertThat(nextGenerationGrid.getState(2, 2), is(State.ALIVE));
203         assertThat(nextGenerationGrid.getState(2, 3), is(State.DEAD));
204         assertThat(nextGenerationGrid.getState(2, 4), is(State.DEAD));
205         assertThat(nextGenerationGrid.getState(2, 5), is(State.DEAD));
206
207         assertThat(nextGenerationGrid.getState(3, 0), is(State.DEAD));
208         assertThat(nextGenerationGrid.getState(3, 1), is(State.DEAD));
209         assertThat(nextGenerationGrid.getState(3, 2), is(State.DEAD));
210         assertThat(nextGenerationGrid.getState(3, 3), is(State.DEAD));
211         assertThat(nextGenerationGrid.getState(3, 4), is(State.DEAD));
212         assertThat(nextGenerationGrid.getState(3, 5), is(State.DEAD));
213
214         assertThat(nextGenerationGrid.getState(4, 0), is(State.ALIVE));
```

```
215         assertThat(nextGenerationGrid.getState(4, 1), is(State.DEAD));
216         assertThat(nextGenerationGrid.getState(4, 2), is(State.ALIVE));
217         assertThat(nextGenerationGrid.getState(4, 3), is(State.DEAD));
218         assertThat(nextGenerationGrid.getState(4, 4), is(State.DEAD));
219         assertThat(nextGenerationGrid.getState(4, 5), is(State.DEAD));
220
221         assertThat(nextGenerationGrid.getState(5, 0), is(State.ALIVE));
222         assertThat(nextGenerationGrid.getState(5, 1), is(State.DEAD));
223         assertThat(nextGenerationGrid.getState(5, 2), is(State.ALIVE));
224         assertThat(nextGenerationGrid.getState(5, 3), is(State.ALIVE));
225         assertThat(nextGenerationGrid.getState(5, 4), is(State.DEAD));
226         assertThat(nextGenerationGrid.getState(5, 5), is(State.ALIVE));
227
228         assertThat(nextGenerationGrid.getState(6, 0), is(State.DEAD));
229         assertThat(nextGenerationGrid.getState(6, 1), is(State.DEAD));
230         assertThat(nextGenerationGrid.getState(6, 2), is(State.DEAD));
231         assertThat(nextGenerationGrid.getState(6, 3), is(State.ALIVE));
232         assertThat(nextGenerationGrid.getState(6, 4), is(State.ALIVE));
233         assertThat(nextGenerationGrid.getState(6, 5), is(State.DEAD));
234     }
235
236     @Test
237     public void testEvolveNextGenerationWithMooreNeighborhoodAnSquarePatternAndGameOfLifeRuleset() {
238         IGrid grid = new Grid2DArray(3, 3, this.moore, this.squarePattern);
239
240         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
241
242         assertThat(nextGenerationGrid.getState(0, 0), is(State.ALIVE));
243         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
244         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
245         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
246         assertThat(nextGenerationGrid.getState(1, 1), is(State.ALIVE));
247         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
248         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
249         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
250         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
251     }
252
253     @Test
254     public void testEvolveNextGenerationWithMooreNeighborhoodAnSquarePatternAndParityRuleset() {
255         IGrid grid = new Grid2DArray(3, 3, this.moore, this.squarePattern);
256
257         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
258
259         assertThat(nextGenerationGrid.getState(0, 0), is(State.ALIVE));
260         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
261         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
262         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
263         assertThat(nextGenerationGrid.getState(1, 1), is(State.ALIVE));
264         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
265         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
266         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
267         assertThat(nextGenerationGrid.getState(2, 2), is(State.ALIVE));
268     }
269
270     @Test
271     public void
    testEvolveNextGenerationWithVonNeumannNeighborhoodAndChessFieldPatternAndGameOfLifeRuleset() {
272         IGrid grid = new Grid2DArray(3, 3, this.vonNeumann, this.chessField);
273
274         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
275
276         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
277         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
278         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
279         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
280         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
281         assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
282         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
283         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
284         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
285     }
```

```
286
287     @Test
288     public void testEvolveNextGenerationWithVonNeumannNeighborhoodAndChessFieldPatternAndParityRuleset
    () {
289         IGrid grid = new Grid2DArray(3, 3, this.vonNeumann, this.chessField);
290
291         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
292
293         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
294         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
295         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
296         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
297         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
298         assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
299         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
300         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
301         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
302     }
303
304     @Test
305     public void
    testEvolveNextGenerationWithVonNeumannNeighborhoodAndAllDeadPatternAndGameOfLifeRuleset() {
306         IGrid grid = new Grid2DArray(3, 3, this.vonNeumann, this.allDead);
307
308         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
309
310         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
311         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
312         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
313         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
314         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
315         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
316         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
317         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
318         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
319     }
320
321     @Test
322     public void testEvolveNextGenerationWithVonNeumannNeighborhoodAndAllDeadPatternAndParityRuleset
    () {
323         IGrid grid = new Grid2DArray(3, 3, this.vonNeumann, this.allDead);
324
325         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
326
327         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
328         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
329         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
330         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
331         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
332         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
333         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
334         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
335         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
336     }
337
338     @Test
339     public void
    testEvolveNextGenerationWithVonNeumannNeighborhoodAndTumblerPatternAndGameOfLifeRuleset() {
340         IGrid grid = new Grid2DArray(7, 6, this.vonNeumann, this.tumblerPattern);
341
342         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
343
344         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
345         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
346         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
347         assertThat(nextGenerationGrid.getState(0, 3), is(State.DEAD));
348         assertThat(nextGenerationGrid.getState(0, 4), is(State.ALIVE));
349         assertThat(nextGenerationGrid.getState(0, 5), is(State.ALIVE));
350
351         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
352         assertThat(nextGenerationGrid.getState(1, 1), is(State.ALIVE));
353         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
```

```
354         assertThat(nextGenerationGrid.getState(1, 3), is(State.DEAD));
355         assertThat(nextGenerationGrid.getState(1, 4), is(State.ALIVE));
356         assertThat(nextGenerationGrid.getState(1, 5), is(State.DEAD));
357 
358         assertThat(nextGenerationGrid.getState(2, 0), is(State.ALIVE));
359         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
360         assertThat(nextGenerationGrid.getState(2, 2), is(State.ALIVE));
361         assertThat(nextGenerationGrid.getState(2, 3), is(State.ALIVE));
362         assertThat(nextGenerationGrid.getState(2, 4), is(State.DEAD));
363         assertThat(nextGenerationGrid.getState(2, 5), is(State.DEAD));
364 
365         assertThat(nextGenerationGrid.getState(3, 0), is(State.DEAD));
366         assertThat(nextGenerationGrid.getState(3, 1), is(State.DEAD));
367         assertThat(nextGenerationGrid.getState(3, 2), is(State.DEAD));
368         assertThat(nextGenerationGrid.getState(3, 3), is(State.DEAD));
369         assertThat(nextGenerationGrid.getState(3, 4), is(State.DEAD));
370         assertThat(nextGenerationGrid.getState(3, 5), is(State.DEAD));
371 
372         assertThat(nextGenerationGrid.getState(4, 0), is(State.ALIVE));
373         assertThat(nextGenerationGrid.getState(4, 1), is(State.ALIVE));
374         assertThat(nextGenerationGrid.getState(4, 2), is(State.ALIVE));
375         assertThat(nextGenerationGrid.getState(4, 3), is(State.ALIVE));
376         assertThat(nextGenerationGrid.getState(4, 4), is(State.DEAD));
377         assertThat(nextGenerationGrid.getState(4, 5), is(State.DEAD));
378 
379         assertThat(nextGenerationGrid.getState(5, 0), is(State.ALIVE));
380         assertThat(nextGenerationGrid.getState(5, 1), is(State.ALIVE));
381         assertThat(nextGenerationGrid.getState(5, 2), is(State.DEAD));
382         assertThat(nextGenerationGrid.getState(5, 3), is(State.DEAD));
383         assertThat(nextGenerationGrid.getState(5, 4), is(State.ALIVE));
384         assertThat(nextGenerationGrid.getState(5, 5), is(State.DEAD));
385 
386         assertThat(nextGenerationGrid.getState(6, 0), is(State.DEAD));
387         assertThat(nextGenerationGrid.getState(6, 1), is(State.DEAD));
388         assertThat(nextGenerationGrid.getState(6, 2), is(State.DEAD));
389         assertThat(nextGenerationGrid.getState(6, 3), is(State.DEAD));
390         assertThat(nextGenerationGrid.getState(6, 4), is(State.ALIVE));
391         assertThat(nextGenerationGrid.getState(6, 5), is(State.ALIVE));
392     }
393 
394     @Test
395     public void testEvolveNextGenerationWithVonNeumannNeighborhoodAndTumblerPatternAndParityRuleset
    () {
396         IGrid grid = new Grid2DArray(7, 6, this.vonNeumann, this.tumblerPattern);
397 
398         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
399 
400         assertThat(nextGenerationGrid.getState(0, 0), is(State.ALIVE));
401         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
402         assertThat(nextGenerationGrid.getState(0, 2), is(State.ALIVE));
403         assertThat(nextGenerationGrid.getState(0, 3), is(State.ALIVE));
404         assertThat(nextGenerationGrid.getState(0, 4), is(State.DEAD));
405         assertThat(nextGenerationGrid.getState(0, 5), is(State.DEAD));
406 
407         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
408         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
409         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
410         assertThat(nextGenerationGrid.getState(1, 3), is(State.DEAD));
411         assertThat(nextGenerationGrid.getState(1, 4), is(State.ALIVE));
412         assertThat(nextGenerationGrid.getState(1, 5), is(State.ALIVE));
413 
414         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
415         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
416         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
417         assertThat(nextGenerationGrid.getState(2, 3), is(State.DEAD));
418         assertThat(nextGenerationGrid.getState(2, 4), is(State.ALIVE));
419         assertThat(nextGenerationGrid.getState(2, 5), is(State.DEAD));
420 
421         assertThat(nextGenerationGrid.getState(3, 0), is(State.DEAD));
422         assertThat(nextGenerationGrid.getState(3, 1), is(State.DEAD));
423         assertThat(nextGenerationGrid.getState(3, 2), is(State.DEAD));
424         assertThat(nextGenerationGrid.getState(3, 3), is(State.DEAD));
```

```
425         assertThat(nextGenerationGrid.getState(3, 4), is(State.DEAD));
426         assertThat(nextGenerationGrid.getState(3, 5), is(State.DEAD));
427
428         assertThat(nextGenerationGrid.getState(4, 0), is(State.DEAD));
429         assertThat(nextGenerationGrid.getState(4, 1), is(State.ALIVE));
430         assertThat(nextGenerationGrid.getState(4, 2), is(State.DEAD));
431         assertThat(nextGenerationGrid.getState(4, 3), is(State.DEAD));
432         assertThat(nextGenerationGrid.getState(4, 4), is(State.ALIVE));
433         assertThat(nextGenerationGrid.getState(4, 5), is(State.DEAD));
434
435         assertThat(nextGenerationGrid.getState(5, 0), is(State.DEAD));
436         assertThat(nextGenerationGrid.getState(5, 1), is(State.DEAD));
437         assertThat(nextGenerationGrid.getState(5, 2), is(State.DEAD));
438         assertThat(nextGenerationGrid.getState(5, 3), is(State.DEAD));
439         assertThat(nextGenerationGrid.getState(5, 4), is(State.ALIVE));
440         assertThat(nextGenerationGrid.getState(5, 5), is(State.ALIVE));
441
442         assertThat(nextGenerationGrid.getState(6, 0), is(State.ALIVE));
443         assertThat(nextGenerationGrid.getState(6, 1), is(State.ALIVE));
444         assertThat(nextGenerationGrid.getState(6, 2), is(State.ALIVE));
445         assertThat(nextGenerationGrid.getState(6, 3), is(State.ALIVE));
446         assertThat(nextGenerationGrid.getState(6, 4), is(State.DEAD));
447         assertThat(nextGenerationGrid.getState(6, 5), is(State.DEAD));
448     }
449
450     @Test
451     public void testEvolveNextGenerationWithVonNeumannNeighborhoodAnSquarePatternAndGameOfLifeRuleset
    () {
452         IGrid grid = new Grid2DArray(3, 3, this.vonNeumann, this.squarePattern);
453
454         IGrid nextGenerationGrid = grid.evolveNextGeneration(gameOfLife);
455
456         assertThat(nextGenerationGrid.getState(0, 0), is(State.ALIVE));
457         assertThat(nextGenerationGrid.getState(0, 1), is(State.ALIVE));
458         assertThat(nextGenerationGrid.getState(0, 2), is(State.DEAD));
459         assertThat(nextGenerationGrid.getState(1, 0), is(State.ALIVE));
460         assertThat(nextGenerationGrid.getState(1, 1), is(State.ALIVE));
461         assertThat(nextGenerationGrid.getState(1, 2), is(State.DEAD));
462         assertThat(nextGenerationGrid.getState(2, 0), is(State.DEAD));
463         assertThat(nextGenerationGrid.getState(2, 1), is(State.DEAD));
464         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
465     }
466
467     @Test
468     public void testEvolveNextGenerationWithVonNeumannNeighborhoodAnSquarePatternAndParityRuleset() {
469         IGrid grid = new Grid2DArray(3, 3, this.vonNeumann, this.squarePattern);
470
471         IGrid nextGenerationGrid = grid.evolveNextGeneration(parity);
472
473         assertThat(nextGenerationGrid.getState(0, 0), is(State.DEAD));
474         assertThat(nextGenerationGrid.getState(0, 1), is(State.DEAD));
475         assertThat(nextGenerationGrid.getState(0, 2), is(State.ALIVE));
476         assertThat(nextGenerationGrid.getState(1, 0), is(State.DEAD));
477         assertThat(nextGenerationGrid.getState(1, 1), is(State.DEAD));
478         assertThat(nextGenerationGrid.getState(1, 2), is(State.ALIVE));
479         assertThat(nextGenerationGrid.getState(2, 0), is(State.ALIVE));
480         assertThat(nextGenerationGrid.getState(2, 1), is(State.ALIVE));
481         assertThat(nextGenerationGrid.getState(2, 2), is(State.DEAD));
482     }
483 }
484
```

```java
package integration.de.nordakademie.pdse.cellularAutomata.experiment;

import de.nordakademie.pdse.cellularAutomata.abortCondition.IAbortCondition;
import de.nordakademie.pdse.cellularAutomata.abortCondition.Simple.NoChange;
import de.nordakademie.pdse.cellularAutomata.abortCondition.complex.FixedNumber;
import de.nordakademie.pdse.cellularAutomata.abortCondition.complex.FixedNumberOrNoChange;
import de.nordakademie.pdse.cellularAutomata.experiment.GameOfLife_Experiment_1_1;
import de.nordakademie.pdse.cellularAutomata.experiment.GameOfLife_Experiment_3_1;
import de.nordakademie.pdse.cellularAutomata.experiment.IExperiment;
import de.nordakademie.pdse.cellularAutomata.log.ConsoleAndFileLogger;
import de.nordakademie.pdse.cellularAutomata.log.ConsoleLogger;
import de.nordakademie.pdse.cellularAutomata.log.FileLogger;
import de.nordakademie.pdse.cellularAutomata.log.NoneLogger;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.util.Scanner;

import static org.hamcrest.CoreMatchers.any;
import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.not;
import static org.junit.Assert.assertThat;

/**
 * Integration-test class for {@link IExperiment}
 * to test the interaction with the classes of {@link IExperiment} and {@link IAbortCondition}
 *
 * @author 17015 (til.zoeller@nordakademie.de)
 */
public class ExperimentIntegrationTest {

    private ByteArrayOutputStream outContent;
    private PrintStream originalOut;

    private IExperiment experiment;

    private IAbortCondition noChange;
    private IAbortCondition fixedNumber;
    private IAbortCondition fixedNumberOrNoChange;

    @Before
    public void beforeEach() {
        this.noChange = new NoChange();
        this.fixedNumber = new FixedNumber(2);
        this.fixedNumberOrNoChange = new FixedNumberOrNoChange(2);

        this.originalOut = System.out;
        this.outContent = new ByteArrayOutputStream();
        System.setOut(new PrintStream(outContent));
    }

    @After
    public void restore() {
        System.setOut(originalOut);
    }

    @Test
    public void testExperimentWithNoChangeAbortConditionAndConsoleLogger() {
        this.experiment = new GameOfLife_Experiment_3_1(this.noChange, new ConsoleLogger());
        this.experiment.run();

        assertThat(outContent.toString(), is(any(String.class)));
        assertThat(outContent.toString(), containsString("0"));
        // Replacing iteration number "(1)" with an empty string, to check only the "1"s from the grid
        assertThat(outContent.toString().replace("(1)", ""), not(containsString("1")));

```

```
 73         assertThat(this.experiment.getIterations(), is(1));
 74     }
 75 
 76     @Test
 77     public void testExperimentWithNoChangeAbortConditionAndFileLogger() throws FileNotFoundException {
 78         this.experiment = new GameOfLife_Experiment_3_1(this.noChange, new FileLogger());
 79         this.experiment.run();
 80 
 81         File logFile = new File(System.getProperty("user.dir")
 82                 + File.separator + "GameOfLife_Experiment_3_1.log");
 83         Scanner fileScanner = new Scanner(logFile);
 84         StringBuilder logContent = new StringBuilder();
 85         while (fileScanner.hasNextLine())
 86             logContent.append(fileScanner.nextLine());
 87         fileScanner.close();
 88 
 89         assertThat(logFile.delete(), is(true));
 90 
 91         assertThat(logContent.toString(), is(any(String.class)));
 92         assertThat(logContent.toString(), containsString("0"));
 93         // Replacing iteration number "(1)" with an empty string, to check only the "1"s from the grid
 94         assertThat(logContent.toString().replace("(1)", ""), not(containsString("1")));
 95 
 96         assertThat(this.experiment.getIterations(), is(1));
 97     }
 98 
 99     @Test
100     public void testExperimentWithNoChangeAbortConditionAndConsoleAndFileLogger() throws
    FileNotFoundException {
101         this.experiment = new GameOfLife_Experiment_3_1(this.noChange, new ConsoleAndFileLogger());
102         this.experiment.run();
103 
104         File logFile = new File(System.getProperty("user.dir")
105                 + File.separator + "GameOfLife_Experiment_3_1.log");
106         Scanner fileScanner = new Scanner(logFile);
107         StringBuilder logContent = new StringBuilder();
108         while (fileScanner.hasNextLine())
109             logContent.append(fileScanner.nextLine());
110         fileScanner.close();
111 
112         assertThat(logFile.delete(), is(true));
113 
114         assertThat(logContent.toString(), is(any(String.class)));
115         assertThat(logContent.toString(), containsString("0"));
116         // Replacing iteration number "(1)" with an empty string, to check only the "1"s from the grid
117         assertThat(logContent.toString().replace("(1)", ""), not(containsString("1")));
118 
119         assertThat(outContent.toString(), is(any(String.class)));
120         assertThat(outContent.toString(), containsString("0"));
121         // Replacing iteration number "(1)" with an empty string, to check only the "1"s from the grid
122         assertThat(outContent.toString().replace("(1)", ""), not(containsString("1")));
123 
124         assertThat(this.experiment.getIterations(), is(1));
125     }
126 
127     @Test
128     public void testExperimentWithNoChangeAbortConditionAndNoLogger() {
129         this.experiment = new GameOfLife_Experiment_3_1(this.noChange, new NoneLogger());
130         this.experiment.run();
131 
132         File logFile = new File(System.getProperty("user.dir")
133                 + File.separator + "GameOfLife_Experiment_3_1.log");
134 
135         assertThat(logFile.exists(), is(false));
136 
137         assertThat(outContent.toString().isEmpty(), is(true));
138 
139         assertThat(this.experiment.getIterations(), is(1));
140     }
141 
142     @Test
143     public void testExperimentWithFixedNumberAbortConditionAndConsoleLogger() {
```

```
144         this.experiment = new GameOfLife_Experiment_1_1(this.fixedNumber, new ConsoleLogger());
145         this.experiment.run();
146 
147         assertThat(outContent.toString(), is(any(String.class)));
148         assertThat(outContent.toString(), containsString("0"));
149         assertThat(outContent.toString(), containsString("1"));
150 
151         assertThat(this.experiment.getIterations(), is(2));
152     }
153 
154     @Test
155     public void testExperimentWithFixedNumberAbortConditionAndFileLogger() throws
    FileNotFoundException {
156         this.experiment = new GameOfLife_Experiment_1_1(this.fixedNumber, new FileLogger());
157         this.experiment.run();
158 
159         File logFile = new File(System.getProperty("user.dir")
160                 + File.separator + "GameOfLife_Experiment_1_1.log");
161         Scanner fileScanner = new Scanner(logFile);
162         StringBuilder logContent = new StringBuilder();
163         while (fileScanner.hasNextLine())
164             logContent.append(fileScanner.nextLine());
165         fileScanner.close();
166 
167         assertThat(logFile.delete(), is(true));
168 
169         assertThat(logContent.toString(), is(any(String.class)));
170         assertThat(logContent.toString(), containsString("0"));
171         assertThat(logContent.toString(), containsString("1"));
172 
173         assertThat(this.experiment.getIterations(), is(2));
174     }
175 
176     @Test
177     public void testExperimentWithFixedNumberAbortConditionAndConsoleAndFileLogger() throws
    FileNotFoundException {
178         this.experiment = new GameOfLife_Experiment_1_1(this.fixedNumber, new ConsoleAndFileLogger());
179         this.experiment.run();
180 
181         File logFile = new File(System.getProperty("user.dir")
182                 + File.separator + "GameOfLife_Experiment_1_1.log");
183         Scanner fileScanner = new Scanner(logFile);
184         StringBuilder logContent = new StringBuilder();
185         while (fileScanner.hasNextLine())
186             logContent.append(fileScanner.nextLine());
187         fileScanner.close();
188 
189         assertThat(logFile.delete(), is(true));
190 
191         assertThat(logContent.toString(), is(any(String.class)));
192         assertThat(logContent.toString(), containsString("0"));
193         assertThat(logContent.toString(), containsString("1"));
194 
195         assertThat(outContent.toString(), is(any(String.class)));
196         assertThat(outContent.toString(), containsString("0"));
197         assertThat(outContent.toString(), containsString("1"));
198 
199         assertThat(this.experiment.getIterations(), is(2));
200     }
201 
202     @Test
203     public void testExperimentWithFixedNumberAbortConditionAndNoLogger() {
204         this.experiment = new GameOfLife_Experiment_1_1(this.fixedNumber, new NoneLogger());
205         this.experiment.run();
206 
207         File logFile = new File(System.getProperty("user.dir")
208                 + File.separator + "GameOfLife_Experiment_1_1.log");
209 
210         assertThat(logFile.exists(), is(false));
211 
212         assertThat(outContent.toString().isEmpty(), is(true));
213 
```

```
214         assertThat(this.experiment.getIterations(), is(2));
215     }
216 
217     @Test
218     public void
    testExperimentWithNoChangingGridAndFixedNumberAndNoChangeAbortConditionAndConsoleLogger() {
219         this.experiment = new GameOfLife_Experiment_3_1(this.fixedNumberOrNoChange, new ConsoleLogger
    ());
220         this.experiment.run();
221 
222         assertThat(outContent.toString(), is(any(String.class)));
223         assertThat(outContent.toString(), containsString("0"));
224         // Replacing iteration number "(1)" with an empty string, to check only the "1"s from the grid
225         assertThat(outContent.toString().replace("(1)", ""), not(containsString("1")));
226 
227         assertThat(this.experiment.getIterations(), is(1));
228     }
229 
230     @Test
231     public void testExperimentWithNoChangingGridAndFixedNumberAndNoChangeAbortConditionAndFileLogger()
232             throws FileNotFoundException {
233         this.experiment = new GameOfLife_Experiment_3_1(this.fixedNumberOrNoChange, new FileLogger());
234         this.experiment.run();
235 
236         File logFile = new File(System.getProperty("user.dir")
237                 + File.separator + "GameOfLife_Experiment_3_1.log");
238         Scanner fileScanner = new Scanner(logFile);
239         StringBuilder logContent = new StringBuilder();
240         while (fileScanner.hasNextLine())
241             logContent.append(fileScanner.nextLine());
242         fileScanner.close();
243 
244         assertThat(logFile.delete(), is(true));
245 
246         assertThat(logContent.toString(), is(any(String.class)));
247         assertThat(logContent.toString(), containsString("0"));
248         // Replacing iteration number "(1)" with an empty string, to check only the "1"s from the grid
249         assertThat(logContent.toString().replace("(1)", ""), not(containsString("1")));
250 
251         assertThat(this.experiment.getIterations(), is(1));
252     }
253 
254     @Test
255     public void
    testExperimentWithNoChangingGridAndFixedNumberAndNoChangeAbortConditionAndConsoleAndFileLogger()
256             throws FileNotFoundException {
257         this.experiment = new GameOfLife_Experiment_3_1(this.fixedNumberOrNoChange, new
    ConsoleAndFileLogger());
258         this.experiment.run();
259 
260         File logFile = new File(System.getProperty("user.dir")
261                 + File.separator + "GameOfLife_Experiment_3_1.log");
262         Scanner fileScanner = new Scanner(logFile);
263         StringBuilder logContent = new StringBuilder();
264         while (fileScanner.hasNextLine())
265             logContent.append(fileScanner.nextLine());
266         fileScanner.close();
267 
268         assertThat(logFile.delete(), is(true));
269 
270         assertThat(logContent.toString(), is(any(String.class)));
271         assertThat(logContent.toString(), containsString("0"));
272         // Replacing iteration number "(1)" with an empty string, to check only the "1"s from the grid
273         assertThat(logContent.toString().replace("(1)", ""), not(containsString("1")));
274 
275         assertThat(outContent.toString(), is(any(String.class)));
276         assertThat(outContent.toString(), containsString("0"));
277         // Replacing iteration number "(1)" with an empty string, to check only the "1"s from the grid
278         assertThat(outContent.toString().replace("(1)", ""), not(containsString("1")));
279 
280         assertThat(this.experiment.getIterations(), is(1));
281     }
```

```
282
283     @Test
284     public void testExperimentWithNoChangingGridAndFixedNumberAndNoChangeAbortConditionAndNoLogger() {
285         this.experiment = new GameOfLife_Experiment_3_1(this.fixedNumberOrNoChange, new NoneLogger());
286         this.experiment.run();
287
288         File logFile = new File(System.getProperty("user.dir")
289                 + File.separator + "GameOfLife_Experiment_3_1.log");
290
291         assertThat(logFile.exists(), is(false));
292
293         assertThat(outContent.toString().isEmpty(), is(true));
294
295         assertThat(this.experiment.getIterations(), is(1));
296     }
297
298     @Test
299     public void testExperimentWithChangingGridAndFixedNumberAndNoChangeAbortConditionAndConsoleLogger
    () {
300         this.experiment = new GameOfLife_Experiment_1_1(this.fixedNumberOrNoChange, new ConsoleLogger
    ());
301         this.experiment.run();
302
303         assertThat(outContent.toString(), is(any(String.class)));
304         assertThat(outContent.toString(), containsString("0"));
305         assertThat(outContent.toString(), containsString("1"));
306
307         assertThat(this.experiment.getIterations(), is(2));
308     }
309
310     @Test
311     public void testExperimentWithChangingGridAndFixedNumberAndNoChangeAbortConditionAndFileLogger()
312             throws FileNotFoundException {
313         this.experiment = new GameOfLife_Experiment_1_1(this.fixedNumberOrNoChange, new FileLogger());
314         this.experiment.run();
315
316         File logFile = new File(System.getProperty("user.dir")
317                 + File.separator + "GameOfLife_Experiment_1_1.log");
318         Scanner fileScanner = new Scanner(logFile);
319         StringBuilder logContent = new StringBuilder();
320         while (fileScanner.hasNextLine())
321             logContent.append(fileScanner.nextLine());
322         fileScanner.close();
323
324         assertThat(logFile.delete(), is(true));
325
326         assertThat(logContent.toString(), is(any(String.class)));
327         assertThat(logContent.toString(), containsString("0"));
328         assertThat(logContent.toString(), containsString("1"));
329
330         assertThat(this.experiment.getIterations(), is(2));
331     }
332
333     @Test
334     public void
    testExperimentWithChangingGridAndFixedNumberAndNoChangeAbortConditionAndConsoleAndFileLogger()
335             throws FileNotFoundException {
336         this.experiment = new GameOfLife_Experiment_1_1(this.fixedNumberOrNoChange, new
    ConsoleAndFileLogger());
337         this.experiment.run();
338
339         File logFile = new File(System.getProperty("user.dir")
340                 + File.separator + "GameOfLife_Experiment_1_1.log");
341         Scanner fileScanner = new Scanner(logFile);
342         StringBuilder logContent = new StringBuilder();
343         while (fileScanner.hasNextLine())
344             logContent.append(fileScanner.nextLine());
345         fileScanner.close();
346
347         assertThat(logFile.delete(), is(true));
348
349         assertThat(logContent.toString(), is(any(String.class)));
```

```
350         assertThat(logContent.toString(), containsString("0"));
351         assertThat(logContent.toString(), containsString("1"));
352 
353         assertThat(outContent.toString(), is(any(String.class)));
354         assertThat(outContent.toString(), containsString("0"));
355         assertThat(outContent.toString(), containsString("1"));
356 
357         assertThat(this.experiment.getIterations(), is(2));
358     }
359 
360     @Test
361     public void testExperimentWithChangingGridAndFixedNumberAndNoChangeAbortConditionAndNoLogger() {
362         this.experiment = new GameOfLife_Experiment_1_1(this.fixedNumberOrNoChange, new NoneLogger());
363         this.experiment.run();
364 
365         File logFile = new File(System.getProperty("user.dir")
366                 + File.separator + "GameOfLife_Experiment_1_1.log");
367 
368         assertThat(logFile.exists(), is(false));
369 
370         assertThat(outContent.toString().isEmpty(), is(true));
371 
372         assertThat(this.experiment.getIterations(), is(2));
373     }
374 
375 
376 }
377 
```