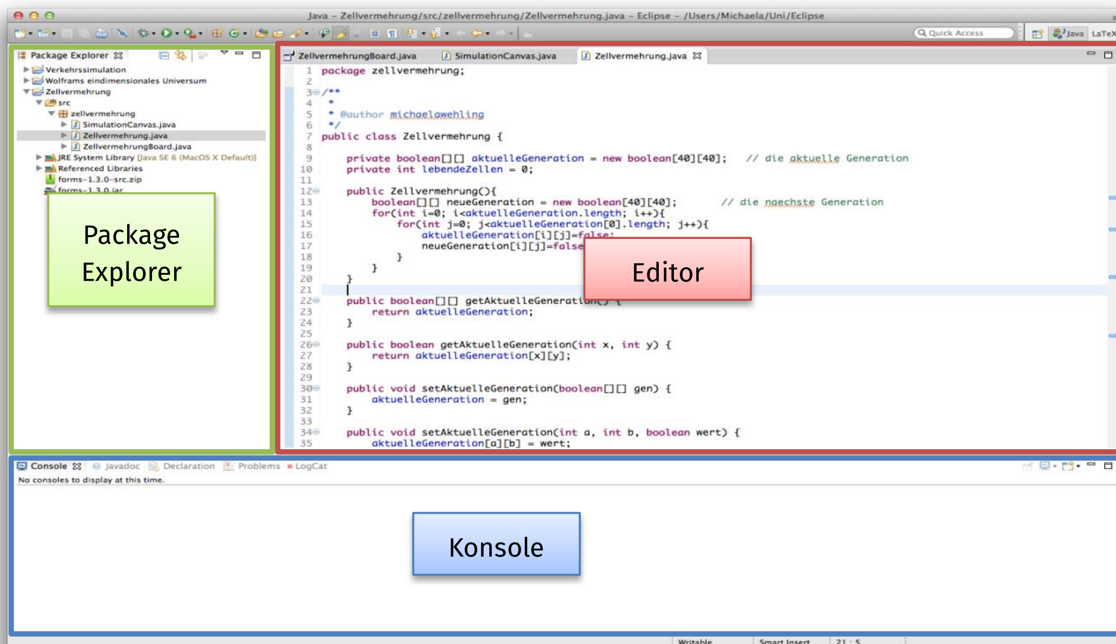
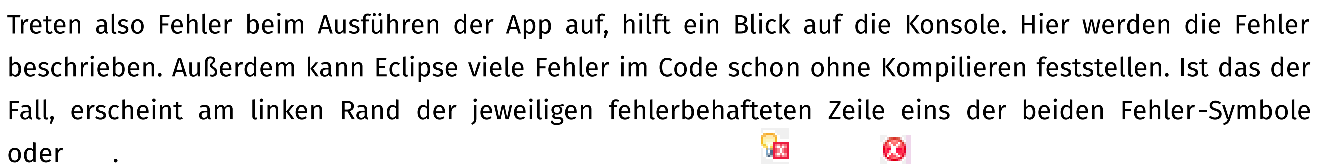
Simulation am Computer – Zellvermehrung

Da die manuelle Ausführung von Regeln auf zellulären Automaten sehr aufwendig ist, ist es hilfreich, den Computer zur Berechnung zur Hilfe zu nehmen. Das Ziel ist es nun, eine Zellvermehrung mit der zuvor gelernten Regel im zweidimensionalen Gitter zu implementieren.

Als Programmierumgebung bietet sich *Eclipse* an. Eclipse stellt euch drei Bereiche zur Verfügung, die für euch interessant sind: Der **Package Explorer** zum Navigieren durch das Projekt, der **Editor** zum Bearbeiten des Quellcodes und die **Konsole** zum Anzeigen von Konsolen-Ausgaben oder Fehlern.



Treten also Fehler beim Ausführen der App auf, hilft ein Blick auf die Konsole. Hier werden die Fehler beschrieben. Außerdem kann Eclipse viele Fehler im Code schon ohne Kompilieren feststellen. Ist das der Fall, erscheint am linken Rand der jeweiligen fehlerbehafteten Zeile eins der beiden Fehler-Symbole oder .



Zum Starten der Simulation genügt ein Klick auf den Run-Button in der oberen Leiste.

Jetzt also ran an die Computer!

1) Öffnet Eclipse.
2) Klickt im **Package Explorer** auf den kleinen Pfeil links neben dem Projekt *Zellvermehrung*, dann auf *src* und schließlich auf *zellvermehrung*.
3) Ein Doppel-Klick auf die **Zellvermehrung.java** öffnet die Datei im Editor.

In dieser Datei wird die Regel für die Simulation zur Zellvermehrung implementiert.

CC BY SA

Implementierung des Zellulären Automaten

Die Simulation soll auf einem 40×40 großem Spielfeld laufen, d.h. das Spielfeld wird in einem 40×40 großem zweidimensionalen Array abgespeichert. Da in einem einzelnen Feld entweder eine lebendige oder eine tote Zelle liegen kann, ist das Array vom Typ Boolean. Damit ihr die Klasse Zellvermehrung nicht komplett selbst schreiben müsst, ist hier schon einiges vorgegeben. Die Klasse enthält zwei Attribute:

- **boolean**[][] aktuelleGeneration = **new boolean**[40][40]: Repräsentiert das Spielfeld und speichert die lebendigen oder toten Zellen ab.
- **int** lebendeZellen = 0: Speichert die Anzahl der lebenden Zellen in der aktuellen Generation.

Wie jede Klasse hat auch unsere Klasse get- und set-Methoden für ihre Attribute, um von anderen Klassen aus zugreifen zu können.

Darüber hinaus werden natürlich noch andere Methoden zur Berechnung der nächsten Generation etc. benötigt. Die Rümpfe der benötigten vier Methoden sind schon vorgegeben. In den Rümpfen steht jeweils als Kommentar // TODO, d.h. hier seid ihr gefragt und dürft die Funktionalität implementieren. Schreibt also euren Code für jeden Schritt in die entsprechende Methode. Was genau die einzelnen Methoden machen sollen, wird im Folgenden erklärt.

Schritt 1 – Array für eine neue Simulation initialisieren

Die Methode **public void** initalisiereArray() soll das Array aktuelleGeneration leeren, d.h. es darf keine lebendige Zelle mehr im Array gespeichert sein. Welcher Wert muss also in jedem Feld stehen?

Wie realisiere ich *for-Schleifen & Arrays* in Java?

Eine for-Schleife, die 10-mal etwas machen soll, wird folgendermaßen deklariert:

```
for(int i=0; i<10; i++){
        // mache etwas
}
```

Ein zweidimensionales Array vom Typ Boolean wird durch folgenden Befehl deklariert:

```
boolean[][] aktuelleGeneration = new boolean[40][40];
```

Die Länge eines Arrays enthält man durch die Befehle

aktuelleGeneration.length (Anzahl der Spalten)

bzw.

aktuelleGeneration[0].length (Anzahl der Zeilen).

Wie formuliere ich *UND-/ ODER-Verknüpfungen* in Java?

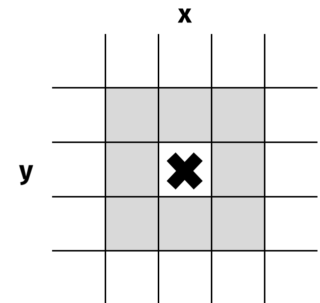
UND-Verknüfung:	if(Bedingung1 && Bedingung2){...}
ODER-Vernüfpung:	if(Bedingung1 || Bedingung2){...}

CC BY SA

Schritt 2 – Lebende Zellen zählen

Damit nachher im Applet zur Simulation auch die Anzahl der lebenden Zellen angezeigt werden, wird die Methode **public void** zaehleLebendeZellen() benötigt. Bei einem Durchlauf durch das Array aktuelleGeneration sollen also die lebenden Zellen gezählt werden.

Schritt 3 – Lebendige Nachbarn zählen



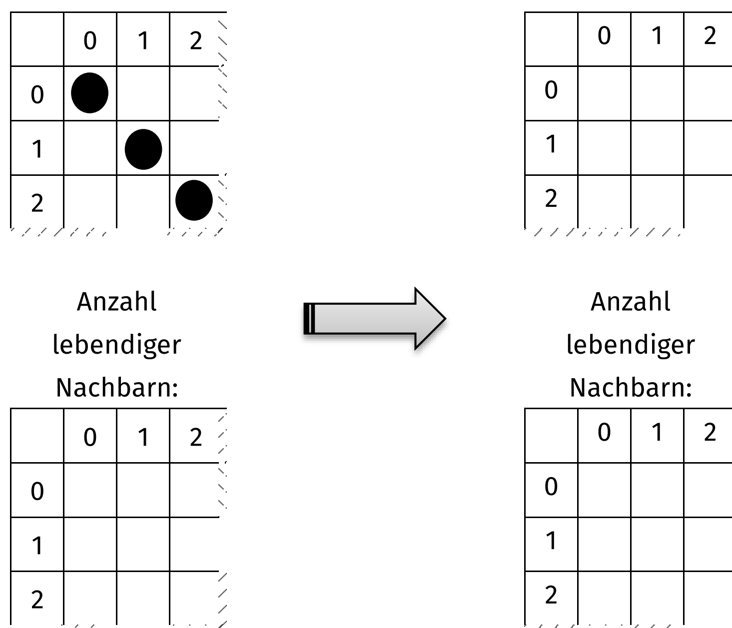
Bevor die Methode zur Berechnung der nächsten Generation geschrieben werden kann, soll uns die Methode **public int** zaehleNachbarn(**int** x, **int** y) ein wenig Arbeit abnehmen. Diese zählt nämlich für eine Zelle an der Position (x, y) des Arrays aktuelleGeneration die lebendigen Nachbarn.

Hinweis: Bei den Randzellen werden nur die vorhandenen fünf Nachbarn und bei Eckzellen die vorhandenen drei Nachbarn berücksichtigt. Der Zustand der betrachteten Zelle darf natürlich nicht mitgezählt werden!

Schritt 4 – Die nächste Generation berechnen

In der Methode **public void** berechneNaechsteGeneration() sollen nun die Regeln der Simulation implementiert werden. Sie ist also das Herzstück und berechnet für jede Zelle des Spielfeldes, ob diese in der nächsten Generation tot oder lebendig ist. Für jede Zelle sollen also die Nachbarn gezählt und anhand derer entschieden werden, was mit dieser Zelle in der nächsten Generation passiert.

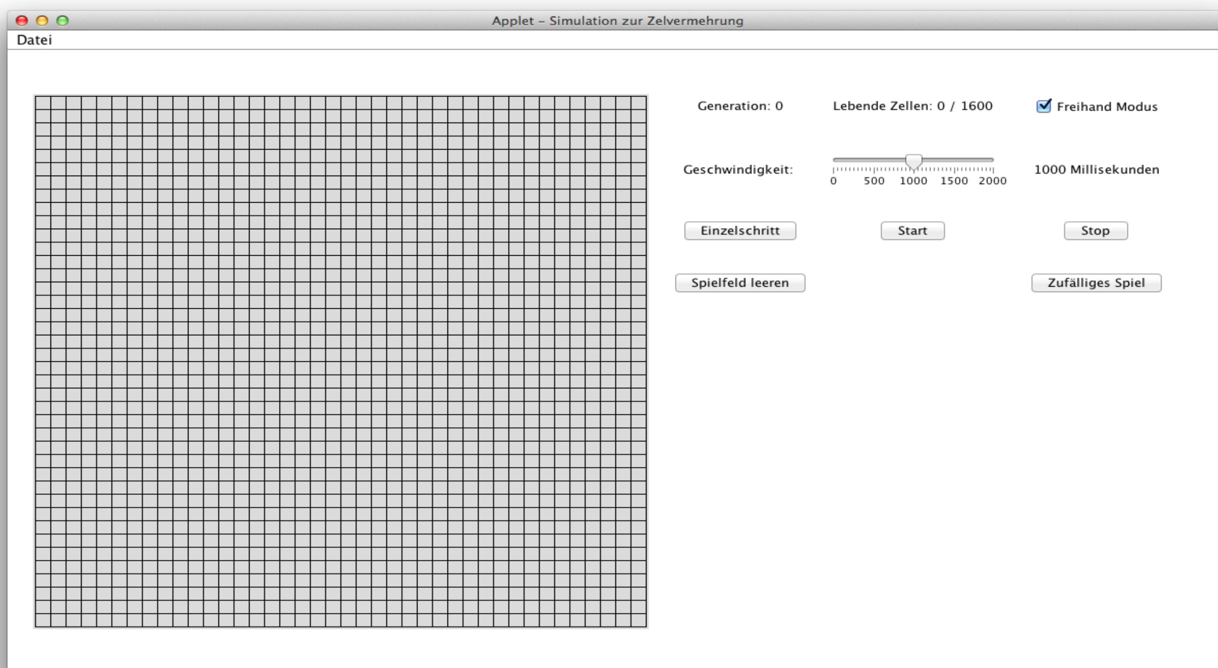
Aber aufgepasst! Was passiert, wenn ihr den neuen Zustand einer Zelle sofort im Array aktuelleGeneration ändert? Welche Auswirkung hat die Änderung für die Berechnung des Folgezustandes der nächsten Zelle? Überlegt euch, wie ihr das Problem beheben könnt und eure Daten nicht sofort überschreibt, obwohl ihr sie noch braucht. Dazu hilft euch das Minimalbeispiel:



cc BY SA

Die Simulation testen

Habt ihr alle Methoden implementiert und Eclipse zeigt euch keine Fehler an, könnt ihr auf den Run-Button klicken und die Simulation testen! Es öffnet sich automatisch ein neues Fenster mit der Simulation:



Aufgabe 1:

Durch Klicken auf das Spielfeld könnt ihr selbst verschiedene Startmuster erzeugen und die nächsten Generationen berechnen lassen. Testet die Muster vom Spielbrett und vergleicht sie mit eurer selbst gelegten Lösung. Stimmen beide Lösungen überein?

Aufgabe 2:

Nach wie vielen Generationen haben sich die beiden Startmuster wieder vermehrt und wie viele Wiederholungen gibt es jetzt? Notiert diese Informationen für die nächsten 6 Vermehrungen.

____

____

Aufgabe 3:

Denkt euch nun eigene Muster aus, der Größe 4×4, 5×5 und 6×6 aus. Verändern sich die Anzahl an Wiederholungen und die dazwischen liegenden Generationen? Notiert eure Beobachtungen.

____

____

CC BY SA