Praktikum „SEP: Java-Programmierung“ WS 2018/19

Aufgabe 3: Die Ameise

Thomas Lemberger und Martin Spießl

Die Ameise

- Mathematisches Spiel mit einer Ameise, die nach bestimmten Regeln über das Spielfeld läuft und die Farbe von Feldern ändert
- Zelluläre Automaten
- Können alles berechnen, was eine universelle Turing-Maschine berechnen kann (turing-vollständig)

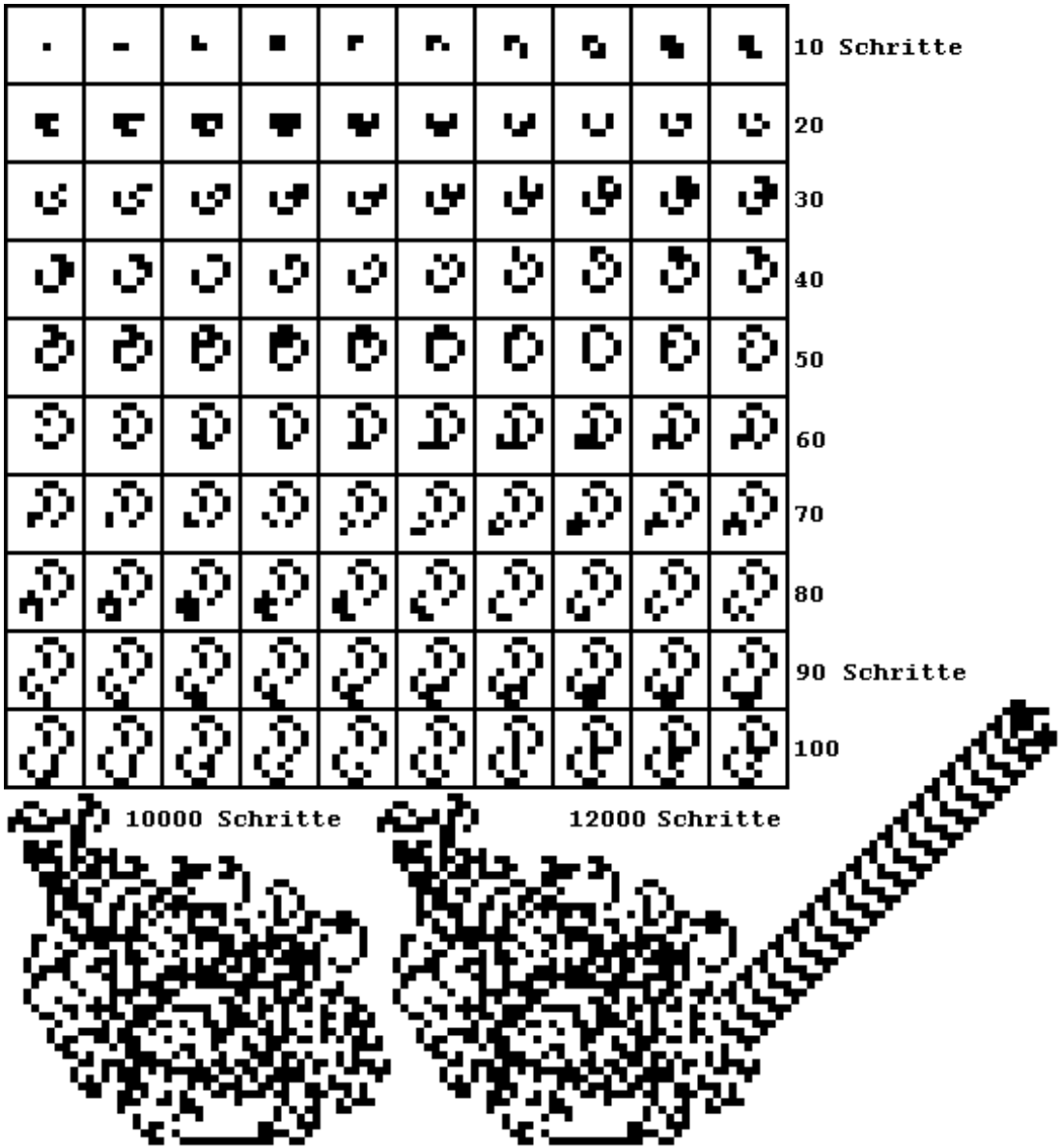
Der Lauf der Ameise (Standard-Edition)

- Jedes Feld hat eine Farbe: weiß oder schwarz
- Ameise führt bei jedem Schritt Aktionen aus:
  1. Wechselt Feldfarbe des aktuellen Feldes
  2. Geht vorwärts
  3. Dreht sich je nach getroffenem Feld:
     - Weißes Feld: Dreht sich 90° rechts
     - Schwarzes Feld: Dreht sich 90° links

Verhalten der Ameise (Standard-Edition)

1. Simplicity (ca. 500 Schritte): Wiederholt kleinere symmetrische Muster
2. Chaos (ca. 10 000 folgende Schritte): Scheinbar zufällige Schritte
3. Emergent Order (alle 104 folgende Schritte): Ameisenstraße

Verhalten der Ameise (Standard-Edition)



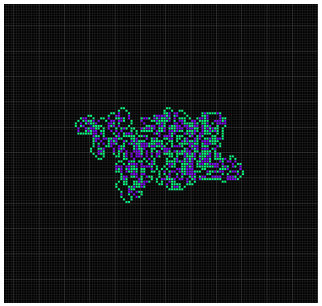
CC BY SA 4.0, Karl Bednarik,
https://de.wikipedia.org/wiki/Ameise_(Turingmaschine)#/media/File:AMEIHUND_Langtons_Ameise.png

Erweiterung: Mehr Konfigurationen

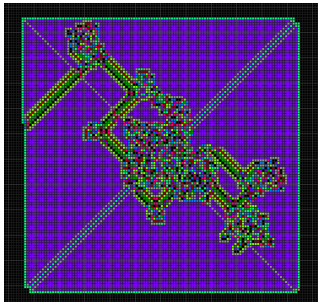
- Felder nicht nur weiß/schwarz, sondern beliebige Sequenz von Farben
- Feld wechselt bei Besuch durch Ameise zur nächsten Farbe
- Jede Farbe hat Rotation zugeordnet (R: 90° rechts, L: 90° links)
- Standard-Edition: `RL`
- Hier: Maximal 12 aufeinanderfolgende Zustände

Bekannte Konfigurationen

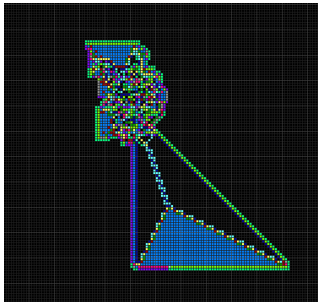
RLR:
Chaotisches
Wachstum



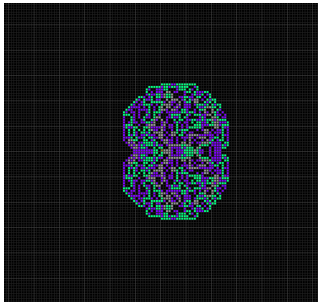
LRRRRRLLR:
Rechtecke füllen



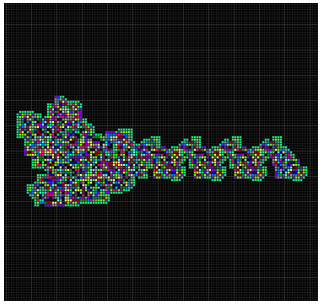
RRLLLRLLLRRR:
Dreieck füllen



LLRR:
Symmetrisches
Wachstum

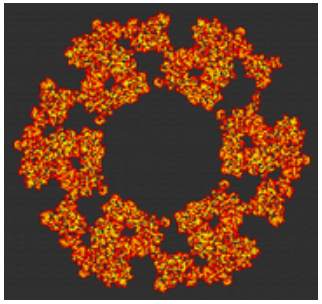


LLRRRLRLRLLR:
Dicker Highway



L2NNL1L2L1:
Hexagon

(nicht Teil der Aufgabe!)



Hexagon: CC BY-SA 4.0 Maxter315

https://en.wikipedia.org/wiki/Langton%27s_ant#/media/File:CA3061-81k7.png

Annahmen

- Endliches, fixes Spielfeld
- Spielfeld als Torus
  - Schritt von „rechtestem“ Feld nach rechts: Ameise landet auf „linkstem“ Feld der gleichen Höhe
  - Analog für alle Richtungen
- Koordinate (0, 0): Feld in linker oberer Ecke

Die Ameise: Start

- Start mit leerem Spielfeld mit festgelegter Konfiguration
- Ameise kann beliebig auf Spielfeld positioniert werden
- Ameise ist immer nach links ausgerichtet

Aufgabenstellung zu Aufgabe 3

- Kommandozeilenschnittstelle
  - Shell
  - Prompt `ant>␣`
- Folgende Kommandos sollen unterstützt werden:

`NEW x y c`

Starte neues Spiel mit neuer Spielfeldgröße und gegebener Konfiguration

- x Anzahl der Spalten
- y Anzahl der Zeilen
- c Konfiguration (in RL-Syntax). Mindestens 2, maximal 12 Zustände.
- Leeres Spielfeld

Aufgabenstellung zu Aufgabe 3

`ANT i j`	Setze neue Ameise in i-te Spalte und j-te Zeile. Es darf immer nur eine Ameise existieren.
`UNANT`	Entfernt die Ameise. Felder bleiben bestehen.

Aufgabenstellung zu Aufgabe 3

`STEP [n]`

Berechne den nächsten Schritt oder mache Schritte rückgängig

- Gebe aktuelle (fortlaufende) Schrittzahl nach Berechnung aus
- Optionales Argument `n` $\neq 0$: Anzahl der Schritte, die gegangen werden soll. z. B.:
  - `STEP`: Gehe einen Schritt
  - `STEP 1`: Gehe (auch) einen Schritt
  - `STEP 99999`: Gehe 99999 Schritte
  - `STEP -5`: Reset auf Zustand vor 5 Schritten (inkl. Ameise)

Aufgabenstellung zu Aufgabe 3

`PRINT`

Gib Spielfeld aus

- Als rechteckige Matrix
- Weiße Felder mit ‘0’
- Darauffolgende Farben mit nächster, **einstelliger** Nummer
- Für Farben ‘10’ und ‘11’: ‘A’ und ‘B’.
- Feld mit Ameise:
  - Blick nach oben: ’^’
  - Blick nach rechts: ’>’
  - Blick nach links: ’<’
  - Blick nach unten: ’v’
- Weitere Angaben auf späterer Folie

`CLEAR`

Säubere Spielfeld

- Setzt alle Felder auf Spielfeld weiß entfernt Ameise und resettet Schrittzähler

Aufgabenstellung zu Aufgabe 3

`RESIZE x y`

Ändere die Größe des Spielfeldes **symmetrisch** auf x Spalten und y Zeilen

- Felder, die sich auch auf dem neuen Spielfeld befinden bleiben gleich
- Ameise außerhalb neuen Felds: Gelöscht.
- Schrittzähler wird nicht zurückgesetzt
- Feld soll symmetrisch resized werden.
  - z. B. Vergrößern von Feld der Größe (6,6) auf (10,10): links, rechts, oben und unten kommen jeweils 2 Felder hinzu
  - Bei ungerader Differenz: Zuerst rechts und unten vergrößern/verkleinern.
    z. B. von (6,6) auf (9,9): links und oben +1, rechts und unten +2
    von (6,6) auf (3,3): links und oben -1, rechts und unten -2

Aufgabenstellung zu Aufgabe 3

`HELP`	Gibt einen sinnvollen Hilfetext aus
`QUIT`	Beendet das Programm

- Erster Buchstabe des jeweiligen Kommandos genügt
- Groß- und Kleinschreibung der Befehle soll egal sein
- Fehlermeldungen beginnen mit `Error!␣`
- Hauptdatei heißt `Shell.java`

Textausgabe des Spielfelds (PRINT)

- Nicht nur Zahlen für Farben, sondern auch echte Farbe
- ANSI Escape Sequences

```
public static final String ANSI_RESET = "\u001B[0m";
public static final String COLOR_0 = "\u001B[47m";
public static final String COLOR_1 = "\u001B[37;40m";
public static final String COLOR_2 = "\u001B[42m";
public static final String COLOR_3 = "\u001B[41m";
public static final String COLOR_4 = "\u001B[37;44m";
public static final String COLOR_5 = "\u001B[43m";
public static final String COLOR_6 = "\u001B[46m";
public static final String COLOR_7 = "\u001B[45m";
public static final String COLOR_8 = "\u001B[36;41m";
public static final String COLOR_9 = "\u001B[31;44m";
public static final String COLOR_10 = "\u001B[34;43m";
public static final String COLOR_11 = "\u001B[32;45m";
```

  - Benutzung:
    System.out. print (COLOR_0 + "0" + ANSI_RESET)
  - Funktioniert nicht auf Windows Commandline
  - Eclipse: ANSI-Escape-Console

    https://marketplace.eclipse.org/content/ansi-escape-console

Implementierung

- Alle Konventionen aus vorherigen Aufgabenstellungen gelten weiter. z. B.:
  - Sinnvolle Paketstruktur
  - Separation of Concerns, Geheimnisprinzip
  - DRY, KISS, etc.

Implementierung: Performance

- Einen Schritt (vorwärts) ausführen muss in $O(1)$ möglich sein
- Speicheraufwand für Resize muss in $O(n)$ liegen, für n neue/gelöschte Felder. D.h.: Gesamtes Spielfeld kopieren nicht erlaubt.
- Speicheraufwand für Funktion zum „Zurückspulen“ sollte unabhängig von Spielfeldgröße sein

Implementierung

- Schnittstelle des *Models* zur `Shell`

```
public interface Grid {
    void setAnt(Ant object, int col, int row); // set ant into cell
    Map<Coordinate, Ant> getAnts(); // get all ants on Grid
    void clearAnts(); // delete all ants on Grid

    void performStep(); // compute next step
    void performStep(int number); // compute next n steps
    void reset(int number); // reset to state n steps ago

    int getWidth(); // x-dimension
    int getHeight(); // y-dimension


    List<Cell> getColumn(int i); // get column i (starting at 0)
    List<Cell> getRow(int j); // get row j (starting at 0)

    void resize(int cols, int rows); // resize grid
    void clear(); // clear grid

    int getStepCount(); // get number of steps
}

public interface Cell {
    // get representation of state
    // choose return type yourself!
    ? getState();
}

public class Ant {
    // get direction of ant
    public Direction getOrientation() { ... }
}
```

Implementierung

- Strikte Trennung der `Shell` von der Spiellogik
  - Zugriff nur über public-Methoden der vorherigen Folie
  - `Grid ant = new SingleAntGrid(...);`
  - Insbesondere `ant.toString()` für `PRINT`-Befehl **nicht** gültig!

  $\Rightarrow$ Komplette Präsentation muss im User-Interface (Shell.java) umgesetzt werden
- Blickrichtung `Direction` der Ameise beliebiger Typ
- Koordinate `Coordinate` auf Feld beliebiger Typ
- `null` als Rückgabeparameter von public-Methoden streng verboten
- Eigene Exceptions nicht notwendig, aber checked erlaubt
- Unchecked exceptions:
  - **Nur** NumberFormatException darf gefangen werden
  - Anderen unchecked Exceptions vorbeugen.

Hinweise (nicht verpflichtend)

- Mögliches Speichern der Zellen:
  - `Cell[][]`
  - `List<List<Cell>>`
  - `LinkedHashMap<Coordinate, Cell>`
- Arbeiten mit Arrays und Collections: Klassen `Arrays` und `Collections`.
  z. B.: `Arrays.asList`, `Collections.singletonMap`

Zusätzliche Abgaben

- Datei Tests.txt wie bei letzten Abgaben
- Dieses Mal neu: Datei Git.txt
  Inhalt:
  - 1. Zeile: Remote-URL (`git remote -v` oder auf Webseite)

  - Danach: Letzter Commit des Repo (`git show`)
    - Hashcode
    - Commit-Message
    - Author-Zeile bitte entfernen
    - Andere Infos optional
- Beide Dateien zum Bestehen notwendig!

Beispiel Git.txt:

```
git@gitlab.lrz.de:masp/sep-ws-18.git
commit 74e1c8da09c2edaf000d4e35718c0f508f6be363
Date:   Sat Dec 1 11:30:32 2018 +0100

    Add example solution
```