Einführung in die Computergrafik

Wintersemester 2018/2019

Prof. Dr. Philipp Jenke

HAW HAMBURG | Department Informatik

Aufgabe 7

Hinweis zu den Mathematik-Klassen (`Vector`, `Matrix`, ...): Sie arbeiten hier mit Referenzen. Eine Zuweisung erzeugt also keine Kopie des Vektors. Wenn Sie eine Kopie benötigen, dann verwenden Sie den Kopier-Konstruktor (`new Vector(andererVektor)`) oder die `copy(andererVektor)`-Operation.

Prozedurale Level-Generierung

In diesem Aufgabenblatt generieren Sie prozedural mit einer Grammatik ein Level.

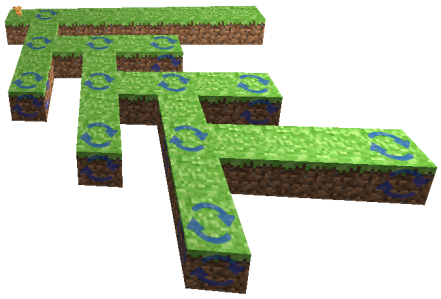

Abbildung 1: Ein prozedural generierter Level.

Als Basis dienen die vorgegenenen Klassen `Grammar` (Informationen zur Grammatik), `Gamefield` (Gitter mit Symbolen = Wort in der Grammatik), `IRule` (Interface für eine Regel in der Grammatik) und `GamefieldImporter` (Generierung eines Levels im Spiel aus einem `Gamefield`).

Grammatik

Die Grammatik arbeitet auf einem 2D-Gitter (umgesetzt in Klasse `Gamefield`). In jeder Gitterzelle kann ein Symbol stehen, das ein Brick repräsentiert. Die Grammatik umfasst die folgenden Symbole:

- B: An diesem Brick startet das Level.
- E: An diesem Brick endet das Level.
- G: Ein regulärer Brick.
- ↓: Hier ist eine Abzweigung in x-Richtung möglich.
- →: Hier ist eine Abzweigung in z-Richtung möglich.
- ↲: Eine Abzweigung von z-Richtung in x-Richtung (und wieder zurück).
- ↳: Eine Abzweigung von x-Richtung in z-Richtung (und wieder zurück).
- ⇅: Eine Kehre in x-Richtung.
- ⇆: Eine Kehre in z-Richtung.

Die Grammatik umfasst einige Regeln, die in der Regeldatei *grammar.grammar* vorgegeben sind. Die Regen haben folgende Syntax:
`pred --> succ : dir`
Eine Regel kann auf ein Symbol `pred` angewendet werden. Das Symbol wird durch eine Liste von neuen Symbolen `succ` ersetzt. Diese neuen Symbole werden in aufeinanderfolgende Zellen entsprechen der Richtung `dir` eingefügt (x oder z).
Der Startzustand (das Axiom) lautet:
B (Zelle 0,0)

→ (Zelle 1,0)
E (Zelle 2,0)

Der Zustand aller Symbole eines Wortes in der Grammatik wird ein einer Datenstruktur `Gamefield` abgelegt. Dieses kapselt ein Gitter. In jeder Zelle steht ein Symbol (`String`) und ein Flag, das angibt, ob das Symbol neu ist (siehe Abschnitt *Ableiten*).

Beispiel

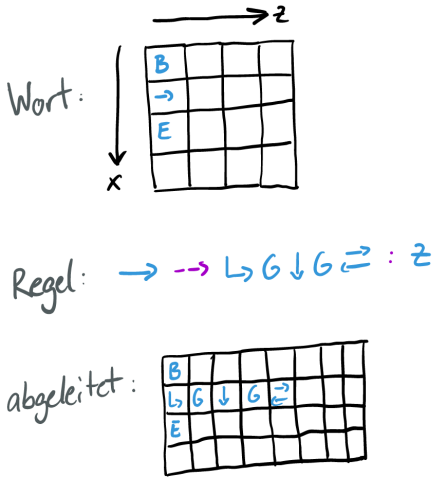

Abbildung 2: Die Ableitung eines Wortes mittels einer Regel.

Regeln

Implementieren Sie die Klasse `Rule`, die jeweils eine Regel in der Grammatik repräsentieren soll. Sie implementiert das Interface `IRule`. Die Klasse soll außerdem eine Methode

```
public static Rule fromString(
  String grammarLine)
```

haben, die dazu dient, eine Regel (also pred, succ und dir) aus einer Textzeile zu generieren. Diese Methode wird dann in `Grammar.fromGrammarFile()` aufgerufen (in den Vorgaben auskommentiert). Hilfreich beim Parsen der Textzeile sind die `String`-Methoden `split()` und `trim()`.

Ableiten

In jedem Ableitungsdurchgang werden nur die Symbole abgeleitet, die bereits da sind, nicht die, die durch eine Ableitung neu entstanden sind. Daher wird in jeder Zelle des `Gamefield` ein Flag verwaltet. Vor einem Ableitungsdurchgang setzt man das Flag für alle Zellen zurück (`resetIsNew()`). Es werden nur die Symbole abgleitet für die das Flag `false` ist. Werden bei der Umsetzung einer Ableitung neue Symbole mit `set(int, int, String)` eingetragen, wird das Flag automatisch auf `true` gesetzt.
Schreiben Sie für die Ableitung mit der Grammatik eine Klasse `Evaluator` mit einer Methode

```
Gamefield derive(Gamefield gamefield,
  int numberOfIterations).
```

Darin würd für alle (alten) Symbole geprüft, ob eine der Regeln passt. Wenn ja, dann wird die Regel angewendet. Es wird also das Symbol durch seine Nachfolge-Symbole (`succ`) ersetzt. Zuvor muss aber geprüft werden, ob alle benötigten Felder frei und innerhalb der Gittergrenzen sind. Die Regel gibt an (`dir`), ob die Nachfolge-Symbole entlang der x- oder z-Achse ausgebreitet werden. Beim Eintragen der Nachfolge-Symbole wird in den entsprechenden Zellen das Flag automatisch auf `true` gesetzt. Diese Symbole werden in der aktuellen Ableitung nicht mehr verändert. Die Ableitung des aktuellen Wortes wird `numberOfIterations` durchgeführt.
Beispiel einer Ableitung:

$$\begin{matrix} B \\ \rightarrow \\ E \end{matrix} \quad \rightarrow \quad \begin{matrix} B \\ \hookrightarrow G \downarrow G \leftrightarrows \\ G \\ G \\ E \end{matrix}$$

Der `Evaluator` kommt dann in `PlatformerScene.onSetup()` zum Einsatz. Dort ist ein passenden Code-Fragment bereits vorgesehen und in der Vorgabe auskommentiert. Verwenden Sie dann diesen Code anstelle von

```
game.fromJson(
  "misc/platformer/level01.json");
```

(Einlesen eines Levels aus JSON).

Freiwillig können Sie die Funktionalität noch so erweitern, dass auch Gegner im Level vorkommen und dann zufällig Münzen im Level verteilt sind.

Im Rahmen des Praktikums wird über die Aufgaben hinweg ein Jump'n'Run Computerspiel (Platformer) entwickelt. In jedem Aufgabenblatt entwickelt sich das Spiel ein wenig weiter. Das Spiel ist in das Framework für die Lehrveranstaltung integriert. Eine Anleitung zum Einrichten des Frameworks finden Sie in der Dokumentation zum Framework auf der EMIL-Seite zur Veranstaltung. Innerhalb des Frameworks ist bereits Funktionalität zum Spiel vorgegeben. Eine Dokumentation dieser Funktionalität findet sich im doc-Ordner des Packages für den Platformer.
Das Spiel kann sowohl auf dem Smartphone unter Android verwendet werden als auch in einer Desktop-Anwendung. Ich empfehle, im Praktikum auf die Desktop-Entwicklung zu setzen, weil sich diese viel einfacher und schneller debuggen und kompilieren lässt. Informationen zur Einrichtung finden Sie im doc*-Ordner des Frameworks.*