Verwendung der Visage Java-API

– Whitepaper –

Andraes Fest

Stand: 27. April 2010
bezieht sich auf Cinderella 2.1 Build 1197

1 Vorbemerkungen

Die interaktive Geometrie-Software Cinderella bietet verschiedene Programmierschnittstellen. Neben der Programmierung in der Software-eigenen Skriptsprache CindyScript und in Python ist es auch möglich, animierte Algorithmen in Java über das in Cinderella eingebaute API zu implementieren. Dieses Interface ist besonders für die Programmierung von Graphenalgorithmen mit der Visage-Erweiterung geeignet.

Allerdings unterscheiden sich die Möglichkeiten, die die Visage Java-API bietet in einigen Details von denen der CindySkript-Schnittstelle. Programme, die in CindyScript geschrieben werden, sind in das Cinderella User-Interface vollständig integriert und ermöglichen wesentlich bessere Interaktion mit der Zeichenoberfläche von Cinderella, z.B. durch Zusammenarbeit mit dem Cinderella-Event-Modell. So können Algorithmen sehr einfach bei jeder Veränderung der Inputinstanz just-in-time ausgeführt und ihr Ergebnis angezeigt werden. Die Visage Java-API bietet dafür geeignete Methoden, um Algorithmen schrittweise ausführen zu können und bei Bedarf einen entsprechenden Pseudocode anzuzeigen.

Die volle Flexibilität der Cinderella Visage-Programmierung entfaltet sich vollends durch das geschickte Zusammenspiel der verschiedenen Programmierschnittstellen. Eine ausführliche Dokumentation beider Schnittstellen sowie eine Reihe von Beispielen werden nach und nach auf der Projektseite

`http://cinderella.de/visage/`

zu finden sein.

2 Vorbereitungen

Zur Verwendung der Cinderella Java-API wird Cinderella 2.1 Build 1197 oder höher benötigt. Eine aktuelle Version des Cinderella-Installationsprogramms kann von `http://beta.cinderella.de/public/` heruntergeladen werden. Dort findet sich jeweils die aktuellste Beta-Version von Cinderella. Diese Cinderella-Version ist zunächst zu installieren.

Für den Einsatz in diesem Kurs benötigen wir die Visage-Schnittstelle von Cinderella. Diese wird zur Zeit mit Cinderella zusammen weiterentwickelt. Für die Programmierung von Visage-Algorithmen empfiehlt es sich deshalb, stets die aktuellste Version von Cinderella zu installieren. Dazu muss lediglich im `lib/`-Verzeichnis innerhalb des Cinderella-Installationspfades die Datei `cindy2.jar` gegen die aktuellste Version ausgetauscht werden, die ebenfalls von `http://beta.cinderella.de/public/` heruntergeladen werden kann. Einen Versions-Log mit den für Visage relevanten Änderungen werden wir auf der Visage-Projekthomepage aufbauen.

Außerdem wird eine Lizenz für Cinderella.2 benötigt. Eine entsprechende Lizenzdatei für den Kurs ADM wird zur Verfügung gestellt. Diese Datei muss in Cinderella geöffnet werden. Anschließend sollte Cinderella neu gestartet werden.

2.1 Visage starten

Visage kann gestartet werden, in dem im „Scripting“-Menü der Punkt „Visage starten“ ausgewählt wird.

Alternativ können Sie die Toolbar von Cinderella ändern und so die Visage-Toolbar auswählen. Dies ist dann zu empfehlen, wenn Sie öfter mit Visage als mit dem Standard-Modus von Cinderella arbeiten wollen. Gehen Sie dazu mit der Maus auf einen freien Bereich der Toolbar, drücken die rechte Maustaste und wählen in dem erscheinenden Popup-Menü den Punkt „Toolbars auswählen“. Sie können dies auch im Datei-Menü finden. Nun erscheint ein kleines Fenster mit einer Auswahl, bei der Sie „Visage“ wählen. Damit ändert sich die Toolbar, die meisten Buttons verschwinden, dafür erscheint ein neuer. Das Auswahlfenster kann nun geschlossen werden.

Wenn Sie auf den neuen Visage-Button drücken, erscheint – ebenso wie bei Auswahl von ein „Visage starten“ im „Scripting“-Menü – ein neues Auswahlfenster. Hier kann man verschiedene vorprogrammierte Graphen-Algorithmen wählen, z.B. Breitensuche, Tiefensuche oder Dijkstra, oder später auch selbst

implementierte Algorithmen. Je nach Algorithmus erscheinen oder verschwinden wieder einige Buttons zum Graphen konstruieren. Spielen Sie ruhig ein wenig damit rum! (Aber erschrecken Sie nicht, wenn ab und an Cinderella dabei abstürzt oder nicht mehr reagiert. Wie bereits erwähnt, Visage ist noch in der Entwicklung es ist noch nicht alles fehlerfrei. Starten Sie dann Cinderella neu und probieren etwas anderes. Wenn Ihnen solche Fehler auffallen, wäre es schön, wenn Sie uns benachrichtigen, was Sie vorher genau gemacht haben und was für eine Fehlermeldung dabei erschien. Schicken Sie Ihren Fehlerbericht bitte an visage@cermat.org) !!!!! Wenn Sie selbst entwickelte Algorithmen verwenden wollen, wählen Sie „use external Algorithmus“ und geben den vollständigen Klassennamen ein. Es steht auch ein graphisches Auswahlfenster für eigene Algorithmen zur Verfügung.

2.2 Ausgaben auf die Konsole

Cinderella schaltet Ausgaben auf die Konsole standardmäßig aus. Beim eigenen Entwickeln von Algorithmen möchte man jedoch gerne oftmals eigene Ausgaben mittels System.out.println(...) erzeugen oder die Fehlermeldungen der Exceptions auf System.err einsehen. Um also die Konsolen-Ausgaben zu sehen gehe man wie folgt vor:

Mac OS X

Die Kosolenausgaben erfolgen in der Konsole des gleichnamigen Programs ,,Konsole” im Verzeichnis Dienstprogramme. Vor dem Start von cinderella also einfach das Konsolen-Programm starten!

Linux

Im Installationspfad von Cinderella existiert eine Datei mit dem Namen `Cinderella.lax`. Diese Datei mit einem Texteditor öffnen! dort sucht man die Abschnitte ,,LAX.STDERR.REDIRECT” und ,,LAX.STDOUT.REDIRECT” und ändert die jeweiligen Einträge in
`lax.stderr.redirect=console`
bzw. `lax.stdout.redirect=console`
Bitte speichern.

Wird nun Cinderella aus einem xterm heraus gestarten, so erscheinen alle Ausgaben in dem xterm.

Windows

Wie unter Linux ist die Datei `Cinderella.lax` zu ändern. Während des Cinderella-Starts bitte auf der Tastatur die CTRL/STRG-Taste drücken, damit das Konsolenfenster erscheint. (Das habe ich noch nicht getestet, wurde mir aber so von den Cinderella-Entwicklern so gesagt!)

2.3 Verwendung der API

Bei Programmierung mit der Visage API muss die Library `cindy2.jar` aus `lib`-Verzeichnis des Cinderella-Installationspfades eingebunden werden. Diese Library kann auch direkt von `http://beta.cinderella.de/public/cindy2.jar` heruntergeladen werden.

Wir empfehlen die Verwendung einer Java IDE, z.B. Eclipse. Dort kann die Library direkt in ein eigenes Projekt gelinkt werden. Zu beachten ist, dass die meisten Namen der Klassen und Methoden von Cinderella verschlüsselt sind. Diese sind jedoch für die Verwendung der API irrelevant. Von Interesse sind nur die Klassen (und dort nur die Methoden), deren Namen in Klartext vorliegen. Man orientiere sich dazu an der Javadoc Dokumentation der API, die auf der Projekt-Homepage eingesehen werden kann.

Ein eigener Graphen-Algorithmus muss von der abstrakten Klasse `de.cinderella.api.visage.GraphAlgorithm` abgeleitet werden. Zu implementieren sind folgende Mathoden:

- `public void init()`
  In dieser Methode erfolgt die Initialisierung des Algorithmus. Sie wird aufgerufen, sobald der Algorithmus in Cinderella geladen, der Rewind-Button gedrückt oder ein neues Element gezeichnet wird.

- `public void runAlgorithm()`
  Hier wird der eigentliche Algorithmus implementiert. Der Ablauf des Algorithmus kann durch Einfügen des `stepDone()`-Befehls an beliebiger Stelle angehalten werden.

- `public boolean modeUndirectedEdges()`
  `public boolean modeDirectedEdges()`
  `public int modeSpecialVertices()`

Diese Methoden legen fest, ob der Algorithmus gerichtete oder ungerichtete Kanten verarbeiten kann und ob spezielle Knoten wie Start- und Zielknoten verwendet werden.

- `public Color getDefaultVertexColor()`
  `public Color getDefaultEdgeColor()`
  Diese Methoden legen Standardfarben für Knoten und Kanten fest.

- `public String intlKey()`
  Dies ist ein Eindeutig zu vergebener Name, der als Schlüssel dient, um dem Algorithmus eine sprachenspeziefische Beschreibung zuzuordnen.

Wir empfehlen (z.B. bei Verwendung von Eclipse), denn Quellcode in einem src-Verzeichnis und die compilierten class-Dateien in einem bin-Verzeichnis getrennt voneinander zu verwalten. Bei der Auswahl des eigenen Algorithmus in Cinderella kann dann das bin-Verzeichniss als Basis-Pfad für die eignen Algorithmen ausgewählt werden. Es ist natürlich sehr empfehlenswert, alle eigenen Dateien einen eigenen Package zuzuordnen. In der ADM verlangen wir ein Package mit dem Namen der Benutzer-Gruppe im Unix-Pool also in etwa gunaXXX, wobei XXX die Gruppennummer ist! Falls das Eclipse-Projekt unter `~/eclipse-workspace/guna` gespeichert ist, und dort die `class`-Dateien in dem Verzeichnis `bin` im Package `gunaXXX` stehen, so sollte als Basis-Pfad für die Algorithmen `~/eclipse-workspace/guna/bin` und als Klassenname `gunaXXX.MyAlgo` angegeben werden.

Zeichnen Sie nun einen Graphen in Cinderella. Um Ihren Algorithmus schrittweise oder vollständig ausführen zu lassen, wählen Sie Ihren Algorithmus über den Visage-Button aus. In der Toolbar erscheint ein Play-Button, um den Algorithmus zu starten, sowie ein Rewind-Button, um ihn neu zu initialisieren. Es gibt zusätzlich einen Informations-Button, über den eingestellt werden kann, ob der Algorithmus schrittweise oder in einem Zug ausgeführt werden soll. Wird der Algorithmus schrittweise ausgeführt, so wird jeder Schritt über den Play-Button gestartet.

3 Ein Beispiel-Algorithmus

Einen ersten Eindruck von der Implementation soll das folgende Beispiel liefern. Hierbei sollen nach einander zunächst alle Knoten eines Graphen und anschließend alle Kanten des Graphen einmal aufblinken. Nach dem

Blinken aller Knoten sowie nach dem Blinken jeder einzelnen Kante wird der Algorithmus unterbrochen, bis der Benutzer den nächsten Schritt ausführen läßt.

```
package myvisage;

import de.cinderella.api.visage.Edge;
import de.cinderella.api.visage.Graph;
import de.cinderella.api.visage.GraphAlgorithm;
import de.cinderella.api.visage.Vertex;

import java.awt.*;
import java.util.Iterator;



public class MyTestAlgorithm extends GraphAlgorithm {


    Graph G;
    private Color DEFAULT = Color.blue;

    public void init() {
        G = getGraph();

        for (Edge e : G.edges()) {
            e.setSize(1);
            e.setColor(DEFAULT);
        }
        for (Vertex v : G.vertices()) {
            v.setSize(3);
            v.setColor(Color.green);
        }

    }

    public void runAlgorithm() {
        Iterator<Vertex> V = G.vertices().iterator();
```

```
    Vertex v;

    while (V.hasNext()) {
        v = V.next();
        flash(v, Color.blue, 200, 1);
        v.setAttribute("Mark","unvisited");
    }
    stepDone();
    for (Edge e: G.edges()) {
     e.flash(Color.red, 200);
     e.setSize(1);
     e.setSize(5);
        stepDone();
    }
}


public boolean modeUndirectedEdges() {
    return true;
}

public boolean modeDirectedEdges() {
    return false;
}

public int modeSpecialVertices() {
    return 0;
}

public Color getDefaultVertexColor() {
    return Color.green;
}

public Color getDefaultEdgeColor() {
    return Color.blue;
}

public String intlKey() {
```

```
        return "mytestalgorithm";
    }

}
```