Java-basierte Erstellung von Algorithmenanimationen

Guido Rößling, Stephan Mehlhase, Jens Pfau

Rechnerbetriebsgruppe, FB Informatik
TU Darmstadt
Hochschulstr. 10
64289 Darmstadt
{guido, mehlhase, pfau}@rbg.informatik.tu-darmstadt.de

Abstract: In diesem Beitrag wird eine Java-basierte API zur einfachen Erstellung von Algorithmenanimationen vorgestellt. Die Nutzung ist für Lehrkräfte und Studierende sehr einfach erlernbar und bietet sich damit für einen Einsatz im Studium und Selbststudium an.

1 Einleitung

Für dynamische Systeme bietet sich eine ebenfalls dynamische Anzeige an, da diese die zugrundeliegenden Eigenschaften des Systems oft greifbarer präsentieren kann als etwa eine Abfolge von Bildern. Seit etwa 20 Jahren befasst sich das Forschungsgebiet „Algorithmenvisualisierung“ (AV) mit diesem Themengebiet. Mittlerweile gibt es mehrere ausgereifte Systeme – die meisten frei verfügbar und durch die Nutzung von Java praktisch überall lauffähig. Umfragen, Studien und andere Erhebungen zeigen meist, dass ein Großteil der befragten Lehrkräfte davon überzeugt ist, dass der Einsatz von AV-Techniken für die Vermittlung von dynamischen Inhalten hilfreich ist. Gleichzeitig gibt ein Großteil der gleichen Personen an, selbst die Werkzeuge nicht einzusetzen [NR03].

Zu den am meisten genannten Gründen zählt dabei *Zeit*: zum Finden „passender“ Inhalte, zum Erlernen der entsprechenden Werkzeuge, und zum Erstellen oder Anpassen gegebener AV-Inhalte [NR03]. Die erforderlichen Anpassungen decken dabei die gesamte Bandbreite ab: von der Änderung der Farbeinstellungen (etwa für die Anpassung an die „Corporate Identity“) über die Vorgabe von Eingabewerten bis hin zur Anpassung einer gegebenen Visualisierung an eine Variante des gleichen Algorithmus.

In diesem Beitrag stellen wir den aktuellen Implementierungsstand von Erweiterungen des schon länger etablierten AV-Systems ANIMAL [Rö02] vor. Die Erweiterungen haben das Ziel, die Nutzung von Algorithmenvisualisierungen deutlich zu erleichtern und die vom Endnutzer benötigte Zeit möglichst stark zu reduzieren.

2 Kurzvorstellung des ANIMAL-Systems

Das Grundgerüst des ANIMAL-Systems wurde 1998 im Rahmen einer Diplomarbeit an der Universität Siegen entwickelt und im Oktober 1998 erstmals im Rahmen der Grundlehre Informatik eingesetzt. Anschließend wurde es systematisch aufbereitet und umstrukturiert und diente 2002 als Teil einer Dissertation [Rö02]. Seit 2002 wurde das System konstant weiterentwickelt und ist mittlerweile hinreichend ausgereift für einen praktischen Einsatz. Abbildung 1 zeigt ein Beispiel einer Algorithmenanimation in ANIMAL, hier zur wörterbuchbasierten Kompression mittels LZW.

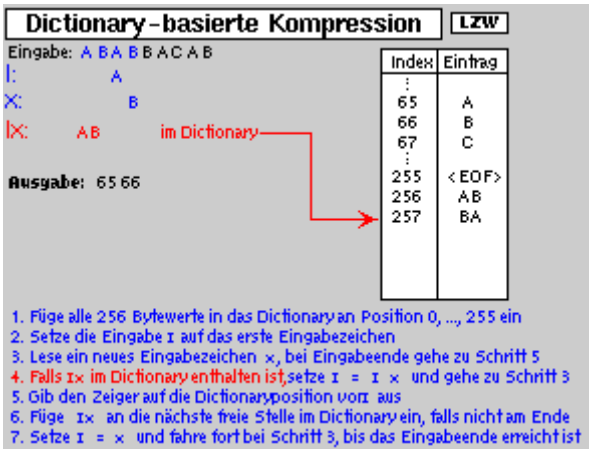

Abbildung 1: Beispiel einer Algorithmenanimation

ANIMAL bietet die folgenden Arten, Inhalte zu erzeugen, wobei nur die erstgenannte Art in der Ursprungarbeit vertreten war:

- Grafische Erstellung durch Dialoge und Drag and Drop, basierend auf Primitiven wie *Punkt* oder *Linienzug* sowie Animationseffekten wie *Farbe ändern* oder *Verschieben*;

- Erstellung auf Basis der eingebauten Skriptsprache ANIMALSCRIPT [RF01]. Die Skriptsprache unterstützt alle Möglichkeiten der grafischen Erstellung und erweitert sie um weitere Merkmale. Dabei ist die Erstellung von Skripten erfahrungsgemäß – nach entsprechender Vertrautheit mit der Notation – etwa um den Faktor 5 schneller als die mausklickintensive Erstellung innerhalb einer GUI. Die noch nicht freigegebene Nachfolgefassung – ANIMALSCRIPT2 [Rö04] – wird zusätzlich auch noch Schleifen und Fallunterscheidungen unterstützen;

- Erstellung von ANIMALSCRIPT-Code aus Programmen, etwa indem entsprechende Befehle während eines gegebenen Algorithmus auf die Konsole ausgegeben oder in eine Datei geschrieben werden;

- Erstellung von Animationsinhalten durch Einsatz von *Generatoren* [RA06], mit denen der Benutzer in einem grafischen Wizard in vier Schritten ausgewählte visuelle Eigenschaften wie Farb- oder Tiefenangaben sowie Eingabeparameter für den jeweiligen Algorithmus angeben kann.

Für Informatiker oder Informatik-nahe Nutzer ist die Verwendung von ANIMALSCRIPT sehr einfach zu lernen, da die Notation „sprechende“ Schlüsselwörter verwendet. Dies belegen auch die Erfahrungen der bisherigen Nutzer, die nach kurzer Einarbeitung nur noch selten auf die Dokumentation zurückgreifen mussten. Der Bequemlichkeit vieler Inhaltsersteller wird ebenfalls Rechnung getragen, da die meisten Parameter der vorhandenen Befehle optional sind.

Gegenüber der manuellen Erzeugung von Animationen bietet die Annotation von Quellcode mit entsprechenden Befehlen den Vorteil, für den gleichen Algorithmus bei Varianz der Eingabeparameter (fast) beliebig viele Animationen erstellen zu können. Sowohl aus ästhetischer Sicht als auch aus Gründen der Softwarewartung ist die Durchsetzung des Algorithmuscodes mit wiederkehrenden Ausgabebefehlen allerdings wenig ratsam. Aus diesem Grund wurde beschlossen, eine Java-API für die Erzeugung von ANIMALSCRIPT-Code zu entwickeln.

Die Hauptanforderungen an die API können dabei wie folgt umrissen werden:

- Saubere Kapselung des visualisierungsbezogenen Codes vom Algorithmus durch die Ersetzung *System.out.println*-artiger Ausgaben in Methodenaufrufe;
- Berücksichtigung der Einstellungen im *Generator-Framework* [RA06];
- Möglichst einfache Nutzung für Programmierer wie Endnutzer;
- Wiederverwendbarkeit von grafischen Eigenschaften, um etwa alle Texte auf die gleiche Art zeichnen zu können;
- Erweiterbarkeit der API um andere Ausgabeformate, wie die Notationen von GAIGS [NG02], JAWAA2 [Rod02]oder *Scalable Vector Graphics (SVG).*

Als Ergebnis entstand die in diesem Beitrag vorgestellte ANIMALSCRIPT-API. Im Folgenden wird zunächst auf die grundsätzliche Nutzung der API eingegangen, und anschließend eine Evaluation der API in Bezug auf die genannten Kriterien gegeben.

3 Grundstruktur der AnimalScript-API

Die AnimalScript-API wurde auf Basis von Schnittstellen und abstrakten Klassen entworfen. Dabei wurde das *Factory* Design-Pattern intensiv genutzt, um die API-Methoden auf eine konkrete Ausgabesprache – hier AnimalScript – abzubilden. Dazu ist zunächst ein konkretes Objekt des gewünschten Ausgabesprachentyps zu erzeugen, wie in Abbildung 2 gezeigt. Dabei ist *Language* die abstrakte – und sprachunabhängige – Oberklasse der konkreten Sprache *AnimalScript*. Die Konstruktorparameter sind der Reihe nach Animationstitel, Autor sowie Breite und Höhe der Animation. Die Schrittkontrolle erlaubt es dem Autor, mehrere Befehle in einem Schritt zusammenzufassen; andernfalls entspricht jeder Befehl einem neuen Schritt.

```
// Anlegen einer neuen Sprachinstanz zur Inhaltserstellung
// Parameter: Titel, Autor, Breite, Hoehe
Language lang = new AnimalScript("Quicksort Animation",
                      "Dr. Guido Rößling", 640, 480);
// Schrittkontrolle aktivieren
lang.setStepMode(true);
```

Abbildung 2: Erzeugung einer konkreten Ausgabespracheinstanz

3.1 Erzeugung und von grafischen Objekten

Ausgehend von dem erzeugten Objekt können nun über eine geerbte *Factory*-Methode neue Objekte erzeugt und anschließend animiert werden, wie in Abbildung 3 gezeigt.

```
// Erzeuge ein neues Textobjekt
// Parameter: Platzierung, Text, Name, Sichtbarkeit, Format
Text aText = lang.newText(new Coordinates(10, 30),
              "Quicksort", "title", null, textProps);
```

Abbildung 3: Erzeugung eines neuen Objekts über die entsprechende *Factory*-Methode

Die Parameter der Methode beschreiben die Platzierung des Objekts, seinen Inhalt – hier den Text – und seinen Namen. Anstelle von *null* kann bei der Sichtbarkeit auch eine Instanz des Typs *Hidden* übergeben werden, damit das Objekt nicht gezeichnet wird.

Die API unterstützt derzeit die folgenden grafischen Primitive: Punkte, Ellipsen und Ellipsensegmente, Kreise und Kreissegmente, Quadrate, Dreiecke, Rechtecke, Polygone, Linien und Linienzüge sowie Texte.

Durch eine entsprechende Kombination dieser Primitiven lassen sich zudem komplexere Objekte einfach erstellen. In Abschnitt 3.6 werden einige weitere grafische Objekte beschrieben, die intern auf diesen Primitiven beruhen.

3.2 Definition der visuellen Eigenschaften

Das in Abbildung 2 referenzierte *textProps*-Objekt enthält die visuellen Eigenschaften des zu erzeugenden Objekts. Es kann wie in Abbildung 4 gezeigt erstellt werden.

```
// Texteigenschaften: rot, nicht zentriert, SansSerif 16
TextProperties textProps = new TextProperties();
textProps.set(AnimationPropertiesKeys.COLOR_PROPERTY,
                  Color.RED);
textProps.set(AnimationPropertiesKeys.CENTERED_PROPERTY,
                  false);
textProps.set(AnimationPropertiesKeys.FONT_PROPERTY,
                  new Font("SansSerif", Font.PLAIN, 16));
```

Abbildung 4: Definition der visuellen Eigenschaften für ein Textobjekt

In vielen Systemen werden anstelle der hier genutzten Properties direkte API-Aufrufe eingesetzt, die Farbe, Ausrichtung und Zeichensatz des gegebenen Textobjekts direkt nutzen. Der Nutzen des hier verfolgten Ansatzes wird dagegen klar, wenn ein zweites Objekt gleicher Art angelegt werden soll: die bereits definierten Properties können nun ohne weitere Modifikation direkt übergeben werden (siehe Abbildung 5).

```
// Erzeuge ein weiteres Textobjekt
// Parameter: Platzierung, Text, Name, Sichtbarkeits- und
// visuelle Eigenschaften
Text anotherText = lang.newText(new Coordinates(10, 55),
               "Beschreibung", "descr", null, textProps);
```

Abbildung 5: Wiederverwendung der visuellen Eigenschaften für ein weiteres Textobjekt

3.3 Platzierung von visuellen Objekten

Die explizite Platzierung des zweiten Textes an festen Koordinaten 25 Pixel unterhalb des ersten Textes erscheint eher willkürlich. Die Bestimmung der Position eines Textobjektes basiert auf dessen unteren linken Ecke der Basislinie, also ohne Berücksichtigung der Tiefenlinien von Buchstaben wie *p*, *q* oder *y*. Es wäre daher sinnvoller, statt einer manuellen Bestimmung der (fixen) Koordinaten mit relativer Positionierung arbeiten zu können, wie in Abbildung 6 gezeigt.

```
// Erzeuge ein drittes – relativ platziertes! - Textobjekt
// Parameter: Platzierung, Text, Name, Sichtbarkeits- und
// visuelle Eigenschaften
Text anotherText = lang.newText(
   new Offset(0, 25, aText, AnimalScript.DIRECTION_SW),
   "Beschreibung", "descr", null, textProps);
```

Abbildung 6: Relative Platzierung anstelle von festgesetzten Koordinaten

Abbildung 6 nutzt ein *Offset*-Objekt, das anhand des Offset (Δx, Δy), gefolgt von der Angaben des Basisobjekts – hier unseres ersten Textes – und der Ausrichtung definiert wird. Die Ausrichtung erfolgt auf Basis eines Rechtecks, das das gesamte Referenzobjekt umschließt (die sogenannte *bounding box*). Von diesem Rechteck sind alle Eck- und Seitenmittelpunkte sowie die Mitte als Bezugspunkt wählbar.

3.4 Erstellung von Animationseffekten und Zeitdefinition

Neben der Erstellung von Objekten können diese natürlich auch animiert werden, etwa indem sie verschoben werden. Dazu stehen einige Grundeffekte zur Verfügung, wie Farbänderung, Rotation, Änderung der Sichtbarkeit und das Verschieben von Objekten. Die meisten Effekte können durch eine konkrete Methodenangabe genauer spezifiziert werden, um Subeffekte zu erzeugen. So sorgt etwa das Verschieben eines Objektes dafür, dass das gesamte Objekt verschoben wird; die Angabe eines konkreten Verschiebetyps wie „**`translateNodes 1 2 3`**“ hingegen führt zur Verschiebung nur der Knoten 1, 2 und 3 des ausgewählten Objekts.

Analog kann, wie in Abbildung 6 gezeigt, die Füllfarbe eines Quadrat-Objekts durch die Angabe des entsprechenden Namens – hier *fillColor* für die Füllfarbe – geändert werden. Die Nutzung von Methodennamen zur Spezifikation der konkreten Aktion erspart die Definition zahlreicher Methoden und erlaubt eine flexible Parametrisierung. Die Zeitdauer des Animationseffekts wird entweder auf Basis von Millisekunden mit einem Objekt des Typs *MsTiming* oder auf Basis von Animationsframes mit einem Objekt des Typs *TicksTiming* spezifiziert. In Abbildung 7 wird also dafür gesorgt, dass sich die Füllfarbe des Quadrats über 20 Frames hinweg auf Orange ändert.

```
// Passe die Fuellfarbe eines Quadrats an; wechselt in
// 20 Frames auf Orange, ohne zeitliche Verzögerung.
// Parameter: Methodenname, Zielfarbe, Startzeitpunkt (null
// wenn direkt), Dauer
square.changeColor("fillColor", Color.ORANGE,
                   null, new TicksTiming(20));
```

Abbildung 7: Spezifikation eines Animationsschrittes mit Angabe der Zeitdauer

3.5 Animationsschritte

Animationseffekte können zusätzlich in Schritte zusammengefasst werden, um die Animation in sinnvolle Einheiten zu zerlegen. Dabei kann jeder Schritt einen oder mehrere Animationseffekte umfassen. Wird anfangs – wie in Abbildung 1 gezeigt – der Schrittmodus aktiviert, so muss jeder Animationsschritt mit einem Aufruf der Methode *nextStep()* der Sprachinstanz gekapselt werden. Wurde der Modus nicht aktiviert, so findet jede Objekterstellung und –animation in einem eigenen Animationsschritt statt.

3.6 Zusätzliche Eigenschaften für Nutzung in der Informatik

Für den produktiven Einsatz in der Informatik bietet die API neben den bereits erwähnten Primitiven einige angepasste Datenstrukturen: Int- und String-Arrays, Int- und String-Matrizen, Graphen (mit den üblichen Eigenschaften, wie gerichtet / ungerichtet, gewichtet / ungewichtet), Arrayzeiger, um die Position von Laufvariablen auf Arrays transparenter zu gestalten, Listenelemente mit einer vom Nutzer vorgegebenen Anzahl Zeigern, sowie Codeblöcke mit Einrückung und farblicher Hervorhebung.

Alle Elemente bieten die üblichen Operationen. So können bei Listenelementen die Zeiger leicht auf andere Elemente sowie auf Bildschirmkoordinaten gesetzt werden. Bei Feldern und Matrizen können Elemente oder Zellen farblich hervorgehoben werden; zudem gibt es Methoden zum Ersetzen oder Vertauschen von Elementen. In Abbildung 8 wird ein Auszug aus einer Beispielanimation gezeigt, die die folgenden Informatik-spezifischen Elemente nutzt:

- Ein Int-Array mit der Hervorhebung der beiden bereits sortierten Elemente,
- Zwei Arrayzeiger, um die aktuellen Positionen von *i* und *j* hervorzuheben,
- Quellcodeanzeige mit Einrückung und Hervorhebung der aktuellen Codezeile.

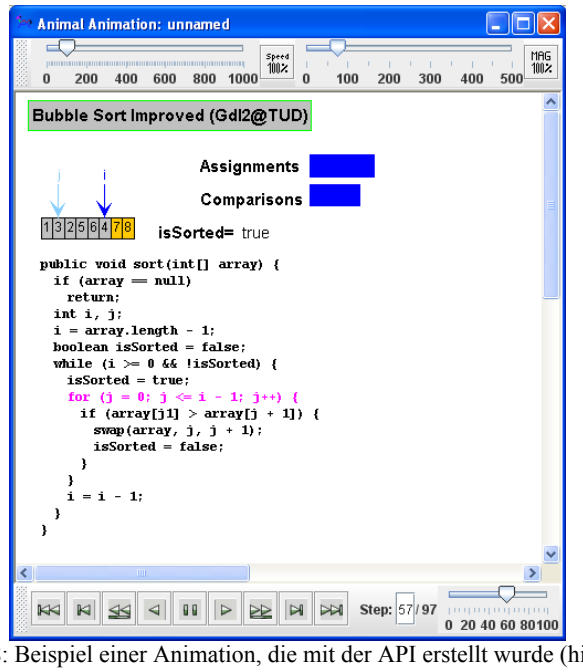

Abbildung 8: Beispiel einer Animation, die mit der API erstellt wurde (hier: Bubble Sort)

4 Einsatz für das Selbststudium

Die Struktur der API macht die Nutzung durch Studierende einfach, sofern diese über grundlegende Java-Kenntnisse verfügen. Gleichzeitig kann die API beim Einsatz im Rahmen der Grundlehre – vor allem in den Vorlesungen zu Algorithmen und Datenstrukturen – wertvolle Hilfe leisten. Mit den Datenstrukturen können die Algorithmen „ganz normal" programmiert werden, da nur geringe Anpassungen wie die Erstellung der Sprachinstanz und der jeweiligen „Properties" erforderlich sind. Diese können von der Lehrkraft gegebenenfalls auch bereitgestellt oder durch entsprechende Hilfsmethoden oder –klassen gekapselt werden.

Für die Studierenden ändert sich dann nicht viel gegenüber der „klassischen" direkten Programmierung der Aufgaben. Sie erhalten aber direkt eine grafische Anzeige des Ablaufs ihres Algorithmus, die bei der Fehlersuche sehr helfen kann – und gleichzeitig die Motivation gegenüber den üblichen *System.err.println-Debugging* deutlich erhöhen kann. Diese Nutzung soll im kommenden Wintersemester in der Lehre erprobt werden, da die API leider nicht rechtzeitig für das Sommersemester fertiggestellt werden konnte.

5 Zusammenfassung

In diesem Beitrag wurde eine Java-basierte API zur einfachen und flexiblen Erstellung von Algorithmenanimationen vorgestellt. Kombiniert mit dem Animationssystem ANIMAL kann diese API sowohl von Lehrkräften als auch von Studierenden sehr gut genutzt werden, um einfache oder komplexe Algorithmen und Datenstrukturen zu animieren und so deren dynamisches Verhalten besser zu verstehen. Zusätzlich hilft die Animation sehr bei der Fehlersuche, da sie Fehler im Wortsinne sichtbar macht.

Die ersten informellen Befragungen der API-Nutzer haben neben einer Anzahl möglicher Verbesserungen auch gezeigt, dass die Nutzung der API die Erstellung von Animationsinhalten subjektiv deutlich erleichtert hat. Gleichzeitig wird, verglíchen mit der bisherigen Praxis der Auszeichnung des Quelltextes mit ANIMALSCRIPT-Befehlen, der Quelltext deutlich besser lesbar und damit auch einfacher zu warten. Die API lässt sich leicht durch darauf aufbauende APIs erweitern, die wiederkehrende Funktionalitäten wie die in Abbildung 8 gezeigten Schrittzähler bei Sortierungsverfahren automatisieren.

Verglichen mit vielen anderen Animationssystemen kombiniert ANIMAL auf Basis der neuen API nun eine komfortablere Erstellung der Inhalte bei gleichzeitig großen Freiheiten des Benutzers. Die Art der Darstellung der Inhalt bleibt weiterhin fest in Hand des Programmierers, wird aber durch die API deutlich einfacher festlegbar. Wir erhoffen uns daher eine weitere Steigerung der Nutzung von ANIMAL nach entsprechender Bewerbung der neuen Möglichkeiten.

Literaturverzeichnis

[NG02] Naps, T.; Grissom, S.: The effective use of quicksort visualizations in the classroom. Journal of Computing in Small Colleges 18, 1 (Oktober 2002); S. 88-96.

[NR03] Naps, T.; Rößling, G.: Evaluating the Educational Impact of Visualization. inroads - Paving the Way Towards Excellence in Computing Education, volume 35, Number 4. ACM Press, New York, NY, 2003; S. 124-136.

[Rod02] Rodger, S. H.: Introducing Computer Science Through Animation and Virtual Worlds. Proceedings of the thirty-third SIGCSE Technical Symposium on Computer Science Education, 2002: S. 186-190.

[Rö02] Rößling, G.: ANIMAL-FARM: An Extensible Framework for Algorithm Visualization. Dissertation, Universität Siegen, Fachbereich Elektrotechnik und Technische Informatik, 2002.

[RA06] Rößling, G.; Ackermann, T.: A Framework for Generating AV Content on-the-fly. Proceedings of the Fourth Program Visualization Workshop, Florence, Italy. 2006; S. 112-117.

[RF01] Rößling, G.; Freisleben, B.: Software Visualization Generation Using ANIMALSCRIPT. Informatik / Informatique, Special Issue on Visualization of Software, Band 8, Nummer 2. ICTswitzerland, Zürich, Switzerland, 2001; S. 35-40.

[Rö04] Rößling, G.; Gliesche, F.; Jajeh, T.; Widjaja, T.: Enhanced Expressiveness in Scripting Using ANIMALSCRIPT V2. In: Proceedings of the Third International Program Visualization Workshop, Warwick, England. Ari Korhonen (Hrsg.). University of Warwick, Coventry, UK, 2004; S. 10-17.