UNIVERSITÄT OSNABRÜCK
FACHBEREICH MATHEMATIK / INFORMATIK

COMPUTERGRAFIKPRAKTIKUM

SOMMERSEMESTER 2004

Leitung: Prof. Dr. Oliver Vornberger

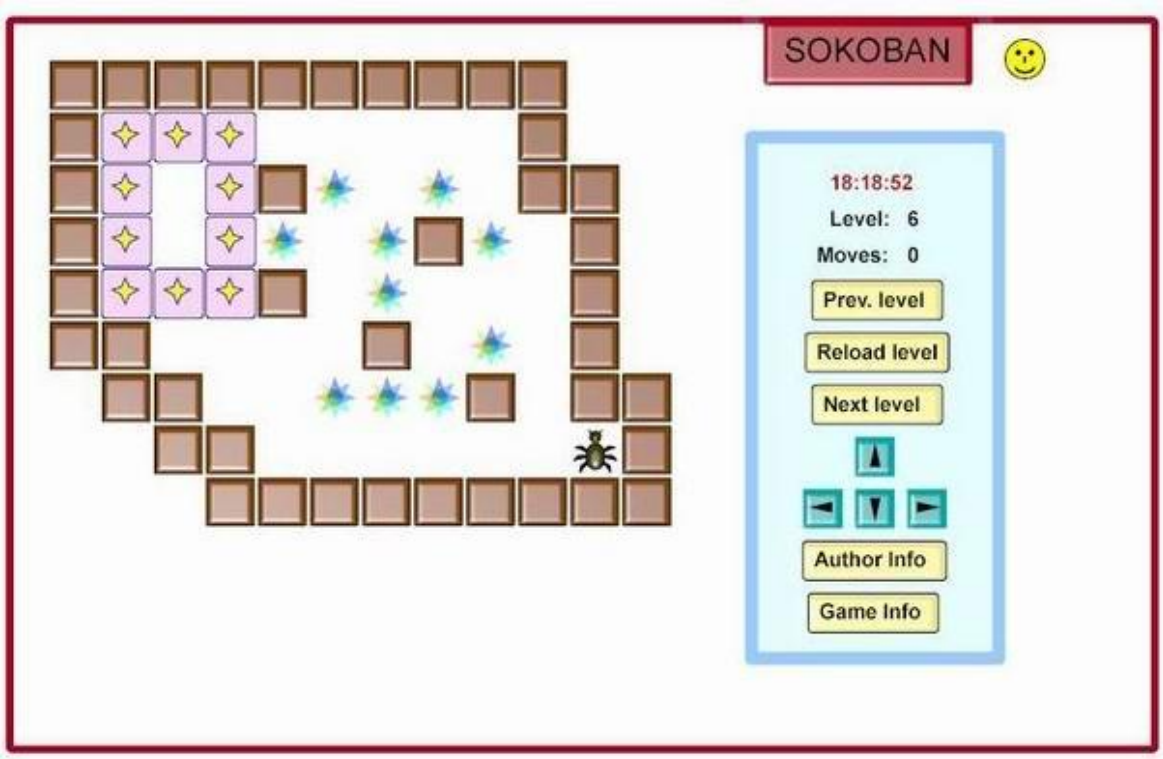
SOKOBAN SPIEL MIT SVG

Erstellt von:
Mustapha Fyjri & Michail Klioutch

Osnabrück, 22 September 2004

1. Aufgabenstellung

Die Aufgabe unseres Praktikums bestand darin, das Spiel Sokoban mit SVG (***Scalable Vector Graphics***) zu implementieren, wobei wir in der Implementation dem Skript (für die dynamische Interaktion wurde JavaScript benutzt) den XML-Code (Extensible Markup Language), in dem reine SVG geschrieben werden können, bevorzugten.



2. Allgemeine Informationen über SVG

Der Begriff SVG bedeutet ***Scalable Vector Graphics***. Diese Sprache erlaubt die Erstellung von 2D Vektorgrafiken basierend auf XML. SVG wurde von mehreren Firmen um 1998 entwickelt, um leichte, dynamische und interaktive Grafiken zu erzeugen. Um SVG-Dokumente zu betrachten braucht man einen SVG-Viewer (Plug In für seinen Browser). SVG ist besser geeignet für Internet-Applikationen (user interface), kann Inhalte (Wirtschaft, Prozesse, Karten ...) in Verbindung mit JavaScript und DOM oder mit XSLT-Transformationen, etc, im XML Format visualisieren.

Die Gründe für eine SVG-Übernahme sind zahlreich: das Anpassen der Visualisierung für verschiedenen Medien, Möglichkeit einer Anwendung der „Styles“, mögliche Indizierung des Texts in Grafiken, Bildergröße nach der Komprimierung, Integrierung in XHTML und in SMIL-Viewer, Benutzung von CSS, scrip-

tabel mit Ecma-/JavaScript via DOM, große und gute Farbenpalette, Benutzung grafischer Filter.

Da SVG ein XML-Format hat, können alle Texteditoren (besser wären XML-Editoren) benutzt werden. Für mehr Informationen siehe W3C Recommendation unter http://www.w3.org/TR/SVG11 für die aktuellste Version.

3. SVG Basis-Strukturen

Eine SVG-Datei fängt mit einer Deklaration von XML-Standardversion an, zum Beispiel:

```
<? xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
```

Danach fügt man den passenden „DocType“ für diese SVG-Version hinzu:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd>
```

Die Wurzel einer SVG-Datei ist ein Element <svg>, also :

```
<svg xmlns=http://www.w3.org/2000/svg xmlns:xlink="http://www.w3.org/1999/xlink" >
        (……...)
</svg>
```

Die Größe des Fensters wird durch die Attribute "width" und "height" definiert, zum Beispiel:

```
 <svg width="400" height="250" xmlns="http://www.w3.org/2000/svg">
```

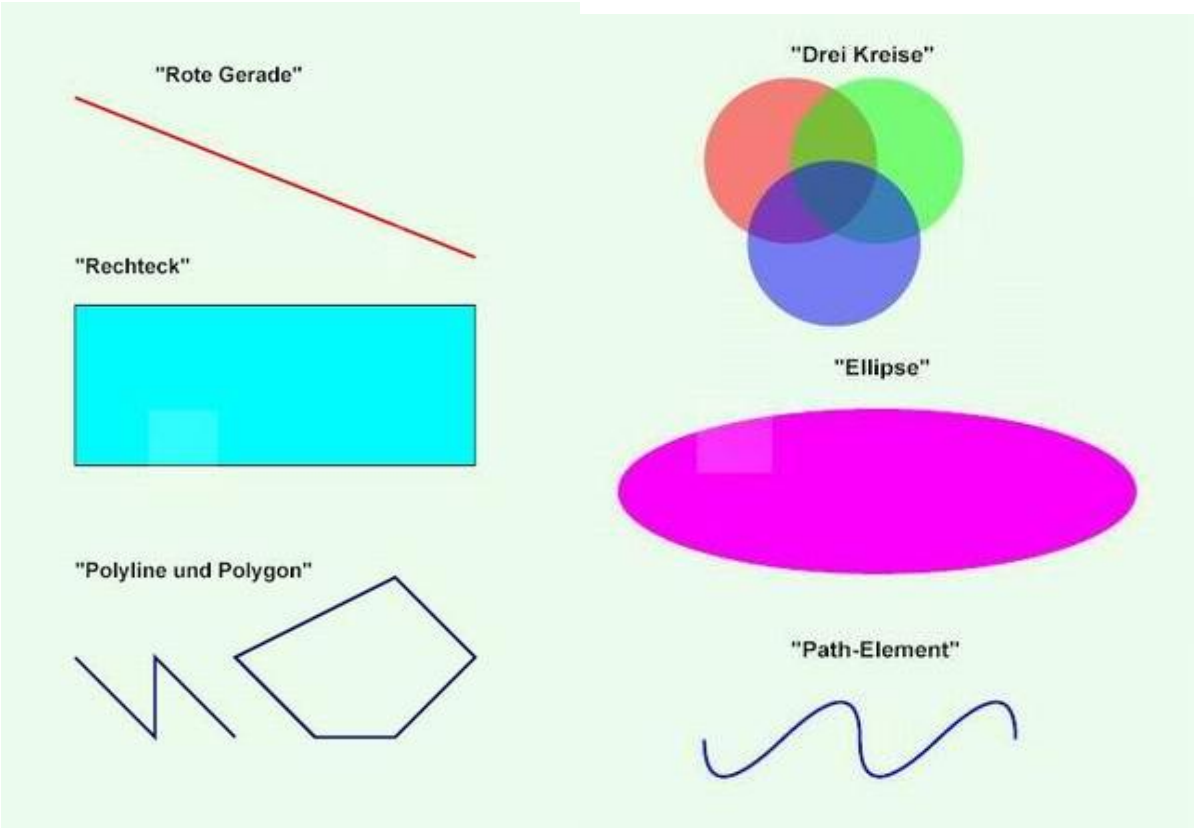
wird unser Programm in einem Fenster 400 * 250 visualisiert.

SVG definiert eine Menge grafischer Elemente; als die wichtigsten sind an dieser Stelle zu nennen: Text mit <text>-Tag, Rechteck mit <rect>-Tag, Kreis mit <circle>- und Ellipse mit <ellipse>-Tag, Gerade mit <line> und "Polyline" mit <polyline>, Polygone mit <polygon>, abstrakte Formen mit <path>.

Jedes grafische Element wird durch ein XML-Element repräsentiert, das durch die geerbten XML-Attribute parametrisierbar ist.
Die Zahl der Attribute ist groß (für alle Details siehe die Spezifikation), daher geben wir an dieser Stelle nur kleine Beispiele, die meisten Elemente haben zahlreiche gemeinsame Attribute wie "id" (identificator) oder "style" oder auch "stroke".
"Stroke" definiert den Rand eines Objektes und besitzt selbst viele Eigenschaften wie die Farbe (fill="red"), die Durchsichtlikeit („opacity“ zwischen 0 und 1), die Breite (width) des Randes und andere. Bei "style" ist dies ähnlich.

Als Beispiel für SVG Elemente sind zu nennen:
Ein Rechteck wird durch <rect> definiert, die Attribute x="?" und y="?" definieren die linke obere Ecke, width="?" und height="?" definieren die Breite und Höhe des

Rechtecks, rx="?" und ry="?" bestimmen, wie stark die Ecken gerundet worden sind.
Kreis und Ellipse werden mit <circle> und <ellipse> eingeführt, haben gemeinsame Attribute cx="?" und cy="?" als Koordinaten des Mittelpunkts, r="?" definiert den Radius des Kreises und rx="?" ry="?" definieren die Halb-Achsen der Ellipse.
Gerade(Line) und "Polyline" werden mit <line> und <polyline> definiert, die Attribute x1="?" und y1="?" definieren die Koordinaten des Startpunktes und x2="?" und y2="?" definieren die Koordinaten des Endpunktes der Geraden, für "Polyline" ist points="?,? ..." das wichtigste Attribut, wobei es sich um eine Liste von Koordinaten handelt.
Ein Polygon wird durch <polygon> definiert und funktioniert genauso wie bei "Polyline". Hier sind die Polygone jedoch geschlossen.
Die Gruppierung von Elementen wird durch <g> (beliebige Elemente) </g> erfolgen und besitzt auch die gleichen Eigenschaften wie alle anderen Elemente.



4. DOM und Scripting

DOM: DOM ist die Abkürzung für ***Document Object Model*** und bezeichnet die Objekthierarchie des Internet Explorers. Zunehmend wird diese Bezeichnung aber auch für die Objektmodelle anderer Browser verwendet. Das DOM des Internet Explorers unterscheidet sich jedoch vom DOM des Netscape Communicators 4.x und vom DOM des Netscape 6.0. Das DOM von Netscape 6.0 hat jedoch sehr viel Ähnlichkeit mit dem XML-DOM und dem SVG-DOM. DOM ist auch ein

plattform- und sprachunabhängiges Interface (API) für den dynamischen Zugriff auf Inhalt, Struktur und Stil von Dokumenten.

Skripte werden ähnlich wie CSS-Deklarationen (Cascading StyleSheets) in ein SVG-Dokument eingefügt. Im <defs>-Bereich bindet man ein Skript über das <script>-Tag ein, wobei man über das Attribut type mit dem Wert "text/ecmascript" oder "text/javascript" die Skriptsprache dem Interpreter bekannt gibt. Der Skript-Code befindet sich im <![CDATA[...]]>-Tag. Das sieht dann folgendermaßen aus:

```
<defs>
  <script type="text/ecmascript"><![CDATA[
    hier kommt der ECMAScript-Code!
  ]]></script>
</defs>
```

Die grundsätzlichen Sprachelemente, die aus Javascript bekannt sind, besitzen natürlich auch ECMAScript. Beispielsweise können Funktionen wie alert() benutzt werden, sowie Objekte wie String, Boolean und Array und deren Funktionen.
Der Unterschied zum Scripting innerhalb einer HTML-Seite ist natürlich das DOM: Der W3C DOM für HTML-Seiten zeigt keine durchgängige Hierarchie auf, man kann auf window, das Anzeigefenster, frames, das Frame-Fenster und document, das eigentliche HTML-Dokument und über document auf die HTML-Elemente zugreifen. Dass SVG ein XML-Format ist, merkt man auch schon am SVG-DOM, denn dieser geht von einer Wurzel aus, dem SVG-Dokument und von dort aus kann man auf alle Elemente, deren Attribute, style-Eigenschaften etc. zugreifen. Dabei wird dynamisch ermittelt, von welcher Art das Element oder Attribut ist.

Die Methoden lassen sich in zwei Arten aufteilen: Zum einen gibt es Methoden, die den Zugriff auf Informationen (Elemente, Attribute, style-Eigenschaften, Text...) ermöglichen, beispielsweise:

getElementById(), getElementsByTagName(), getDocumentElement(), getNodeType(), getNodeName(), getParentNode(), getFirstChild(), getAttributes(), getStyle(), getPropertyValue(), getCSSText() .

Zum anderen gibt es Methoden, die diese Informationen manipulieren können. Zum Beispiel:

createElement(), cloneNode(), appendChild(), removeChild(), setAttribute(), removeAttribute(), setProperty(), removeProperty(), etc.

5. Interaktion

Man verwendet Event-Attribute, um auf Ereignisse zu reagieren. Wenn ein bestimmtes Ereignis (Event) eintritt, wie z.B. der Klick auf ein Element, können Sie als Reaktion auf dieses Ereignis eine EcmaScript-/ JavaScript-Funktion aufrufen

Bsp.: <rect onclick="tuEtwas()"/>

Hier nennen wir ein Paar Event-Attribute, die von SVG unterstützt sind:

* onclick - Klick auf das Element
* onactivate - Aktivierung des Elements durch beliebigen Zeiger (Mausklick, Taste, ..)
* onmousedown - Drücken der linken Maustaste
* onmouseup - Loslassen der linken Maustaste
* onmouseover - Den Bereich des Elements mit dem Mauszeiger betreten
* onmousemove - Bewegen der Maus über dem Bereich des Elements
* onmouseout - Den Bereich des Elements mit dem Mauszeiger verlassen
* onkeypress / onkeydown - Taste gedrückt
* onkeyup - Taste losgelassen
* onfocusin - Das Element bekommt den Fokus
* onfocusout - Das Element verliert den Fokus

Es gibt einen anderen Aspekt von "Event Handling" und dies geschieht durch die Benutzung von SVG DOM. Dort kann das Skript DOM „Eventlistener“ registrieren und wiederum eine Behandlung für ein eingehendes Event beschreiben. Das bedeutet, dass man ein Objekt für das Element hat, welches den Event auslösen soll, und für dieses Objekt die Funktion „addEventListener (Event-Name, Funktionsname, 'false')“ aufruft.

Beispiele aus dem Code:

```
function Init(load_evt){  //Initialisierung des Spiels
                    ……….
                    Actual_level();
                    ShowFirstPanel();
                    // initialize event processing
                    svgroot.addEventListener("keydown", keyHandler, false);
                    ani1=svgdoc.getElementById("walking1");
                    ani2=svgdoc.getElementById("walking2");
                    animat3=svgdoc.getElementById("animat3");
                    ..........................
}

<g id="Authorinfo" onclick="hideWindow(authorinfo)" fill-opacity="0.8">
                    (.....................)
</g>
```

```
<a xlink:href="" onclick="getPrevLevel(); Actual_level()">
                    (.....................)
                    <text x="645" y="180" style="font: bold 12pt sans-serif">Prev. level </text>
                    <set attributeName="fill" attributeType="CSS" to="#0F0" begin="mouseover"/>
                    <set attributeName="fill" attributeType="CSS" to="#000" begin="mouseout"/>
  </a>
```

6. Die Level

Die Levels werden bei uns in einem dreidimensionalen Array gespeichert, so dass die erste Dimension für die Nummer des Levels steht, die zweite für die Zeilennummer des Anfangszustandes eines Levels und die dritte für die Spaltennummer. In dem Array werden dann die Zahlen von 0 bis 6 gespeichert, wobei 0 für ein leeres Feld, 1 für die Ameise (im Code Sokob), die 2 für den Diamanten (Objekte, die verschoben werden sollen), 3 für eine Kiste (Feld in das der Diamant verschoben werden soll), 4 für einen Stein (gesperrtes Feld), 5 für einen Diamanten, der bereits in einer Kiste liegt und 6 für die Ameise auf der Kiste (die Ameise darf sich sowohl auf einem leeren Feld, wie auch auf einer Kiste befinden) benutzt wird.
Dieses Array (im Code "Level") wird lediglich für das Laden eines Spiellevels benutzt. Zur Laufzeit wird beim Laden eines Levels ein zweidimensionales Array (act_Level) mit den Verweisen auf die Objekte, die sich auf dem Spielfeld befinden, bis auf die Ameise, die gesondert behandelt wird, erzeugt. Mit Hilfe des DOMs ist es möglich, direkt auf die Elemente des SVG-Dokuments zuzugreifen. Somit mussten wir nur von jedem Objekt ein einzelnes in SVG konstruieren und konnten uns dann dynamisch Kopien (mit cloneNode(false)) von diesen anfertigen und auf dem Feld platzieren. Dabei trafen wir allerdings auf ein Problem, das uns dazu gezwungen hat, nicht die Elemente direkt per Script zu kopieren, sondern erst ihre Kopien, die mit "use" erzeugt wurden.
Ein zusammengesetztes Element (<g>...</g>) hat nämlich von sich aus keine Koordinaten, die man abfragen und leicht manipulieren könnte (man kann die Objekte zwar mit "transform" verschieben und drehen, bekommt aber keine Koordinaten zurück), eine "use"-Kopie dagegen schon. Deswegen wird jedes zusammengesetztes Element bei uns zuerst in dem Definitionsbereich (<defs> ... </defs>) konstruiert und bekommt dann die Identität mit Postfix "proto" (Prototyp), dann mit "use" kopiert und weiterhin wird nur diese Kopie verwendet. Außerdem werden in dieser Funktion auch die Diamanten ("diamcount") gezählt, die Koordinaten der Ameise ("xpos","ypos") gespeichert und die „gameend“ auf „false“ gesetzt.

7. Check_move

Bevor ein Spielzug vollzogen wird, muss geprüft werden, ob dieser überhaupt erlaubt ist. Dies geschieht in der Check_move Funktion, die als Parameter die Richtung bekommt, wobei mit 1 oben, mit 2 rechts, mit 3 unten und mit 4 links gemeint sind. Dann werden in einem „Switch“ die Koordinaten des nächsten Feldes in die Bewegungsrichtung neben der Ameise berechnet. Mit der "getAttribute("id")" wird dann die Art des Objekts in dem nächsten Feld herausgefunden. Falls ein Stein oder der Rand im Weg steht, wird "false" zurückgegeben, genauso, wie wenn "gamepause" oder "gameende" (globale Booleanvariable) "true" sind. Weiter wird nachgeprüft, ob ein Diamant ("diamant" oder "diabox") im nächsten Feld liegt, denn dann muss auch das übernächste Feld angeschaut werden (es darf zum Beispiel keine Bewegung stattfinden, falls zwei Diamanten hintereinander stehen). In einem „Switch“ werden dazu die Koordinaten des übernächsten Feldes berechnet und mit "if-else" dann entschieden, ob die Bewegung erlaubt ist oder nicht (in einem Fall wird "true" in anderem "false" zurückgegeben).

8. Move

Zuerst wird mit "Check_move" geprüft, ob die Bewegung erlaubt ist und falls es nicht der Fall ist, geschieht nichts. Sonst werden sofort die Koordinaten der Ameise verändert und diese im Dokument mit "setAttribute()" verschoben. Danach wird noch geprüft, ob ein Diamant verschoben werden muss, also ob in dem Feld, in dem sich die Ameise jetzt befindet, noch ein "diamant" oder "diamantaus" (diabox) zu finden ist. Dieser wird dann gelöscht und an seiner Stelle ein leeres Feld (das "null"-Objekt) bzw. Kasten eingefügt. Dann wird das nächste Feld angeschaut und darin ein "diamant" oder ein "diamantaus" eingefügt (falls dort ein Kasten war "diamantaus", sonst einfach "diamant"). Dabei wird der Zähler der Diamanten ("diamcount") hoch- bzw. runtergesetzt (abhängig davon, ob einer geschaffen oder gelöscht wird).
Am Ende wird noch geprüft, ob "diamcount" die "0" erreicht hat, denn dann ist der Level beendet und die "gameend" wird auf "true" gesetzt, sowie ein "GameEndInfo"fenster ("endgameinfo") geöffnet.

9. Init, ShowFirstPanel, Unshow

Die "Init" Funktion wird beim Laden der SVG mit "onload="Init(evt)"" gestartet und speichert Verweise auf alle Objekte der SVG, die später benutzt werden auf globale Variable (inklusive des Dokumentelements und der Wurzel der Graphik). Außerdem bindet sie den „EvenListener“ für die Pfeiltasten der Tastatur an, damit das Spiel auch mit der Tastatur gespielt werden kann, und startet das erste Fenster, das in "ShowFirstPanel"noch ein paar Komponenten (Diamanten

und die vergrößerte Ameise) bekommt. Beim ersten Klick auf dieses Fenster wird mit "Unshow" das Fenster gelöscht, sowie der erste Level geladen.

10. Restliche Funktionen

"Uhrzeit" lädt die aktuelle Systemzeit und gibt sie dann in einem Textelement aus, wonach sie nach einer Sekunde Pause sich selber aufruft, so dass im Textelement immer die aktuelle Systemzeit zu finden ist. Die "getWindow" bzw. "hideWindow" zeigen bzw. entfernen Objekte, die sie als Parameter bekommt, wobei "getWindow" auch eine Animation an das Objekt anbindet, die das langsame Auftauchen des Objekts sorgt.
Diese Funktionen werden zur Fenster-Erzeugung bzw. -Entfernung benutzt.
"getNextLevel" und "getPrevLevel" schalten mit "loadLevel" den Level hoch bzw. runter, wobei auch die Anzahl der Level beachtet wird (sie werden im Kreis geladen, so dass nach dem letzten das erste Level auftaucht und umgekehrt).

11. Die SVG Elemente

Die SVG-Elemente (sämtliche Objekte auf dem Feld, sowie Fenster, Buttons und Infofelder (zum Beispiel die Zeit)) werden mit dem XML-Code aus den Basiselementen (Rechtecke, Kreise, Polylines usw.) gebaut mit "<g id="..." >...</g>" gruppiert (außer der einfachen Elemente, wie der Stein). Dies geschieht in dem Definitionsbereich (außer der Elemente, die immer da bleiben, wie die Rechte Tabelle und die Ameise), damit sie nur dynamisch gezeichnet werden.

Literatur

http://www.w3.org/TR/SVG11/
http://svglbc.datenverdrahten.de/
http://www.helma-spona.de/PDF/glossESSVG.pdf