Online SVG Generator

MM.Prog Studienarbeit

von Simon Heimler (2012)
Interaktive Medien Semester 3
www.svg-generator.de

3. Kurzbeschreibung

Der SVG Generator ist eine in HTML5 / JavaScript umgesetzte Webapplikation die plattformunabhängig in modernen Browsern läuft. Die Anwendung erlaubt es dem User Vektorgrafiken in dem standardisierten Format SVG zu generieren, die im Browser berechnet und sofort dargestellt werden. Es stehen verschiedene Algorithmen zur Auswahl und viele Parameter können frei angepasst werden.

Die Grafik kann anschließend als SVG Datei herunterladen werden. Dieses Format lässt sich in vielen professionellen Grafikprogrammen importieren und weiterverwenden. (Insbesondere Adobe Illustrator)

4. Kurze Installationsanleitung

Es ist keine Installation nötig.
Allerdings sollte ein moderner Browser installiert sein und verwendet werden. Google Chrome liefert aktuell die beste Performance, aufgrund seiner sehr schnellen V8 JavaScript Engine.

5. Kurze Bedienungsanleitung

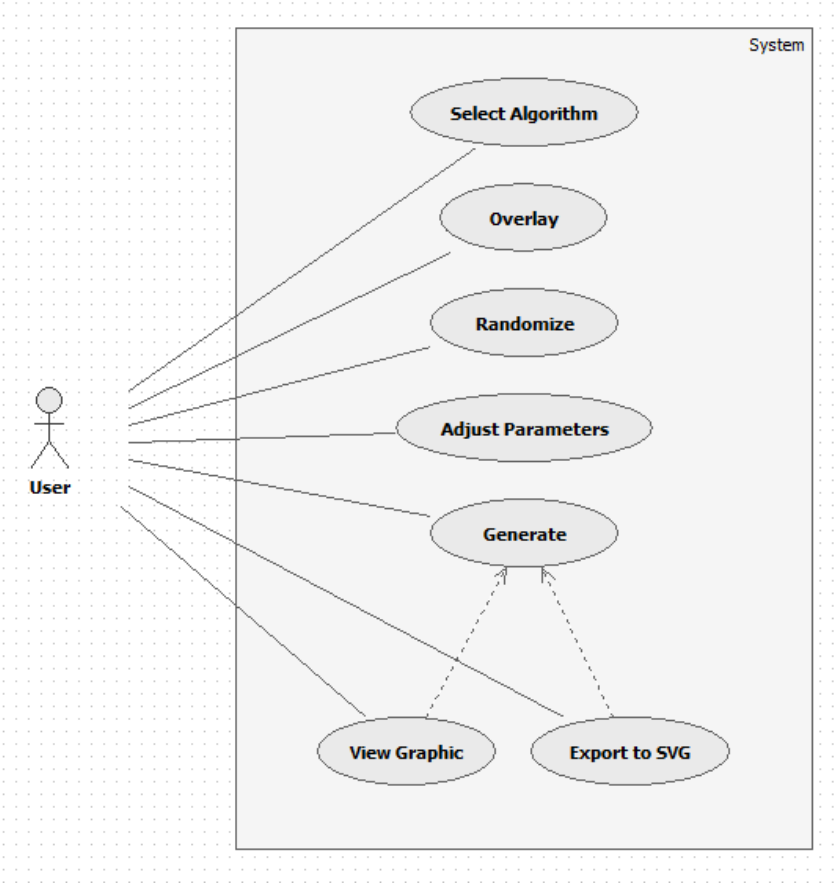
Der SVG Generator wird entweder über online geöffnet (www.svg-generator.de) oder lokal über index.html.

Man kann sich jederzeit über den Button "*Generate*" eine Grafik generieren lassen.

Es gibt viele Anpassungsmöglichkeiten, von denen die generierte Grafik abhängt:

- In der *Drop-Down Auswahlbox* können verschiedene *Algorithmen* gewählt werden, die jeweils grundlegend anders arbeiten, unterschiedliche Standartwerte und Parameter mitbringen.
- Die *Parameter* können in der unteren Seiten-Box angepasst werden. Für die Farbfelder steht ein ColorPicker zu Verfügung. Ein erneuter Klick auf *Generate*! liefert ein neues Ergebnis, das die geänderten Parameter berücksichtigt.
- "*Overlay*!" fügt das neue Bild an das alte Bild an, so lassen sich mehrere verschiedene Parameter und sogar Algorithem miteinander kombinieren.
- Mittels "*Randomize*!" kann man die Parameter mit Zufallswerten füllen und das Ergebnis sofort anzeigen.
- Der "*Export to SVG*" Button speichert die aktuell angezeigte Datei als SVG Grafik ab und bietet sie zum Download an. So kann die Grafik in andere Grafikprogramme importiert werden.

6. Use-Cases-Diagramm



7. Use-Cases-Beschreibung

Der Benutzer öffnet den SVG Generator. Folgende Möglichkeiten bieten sich:

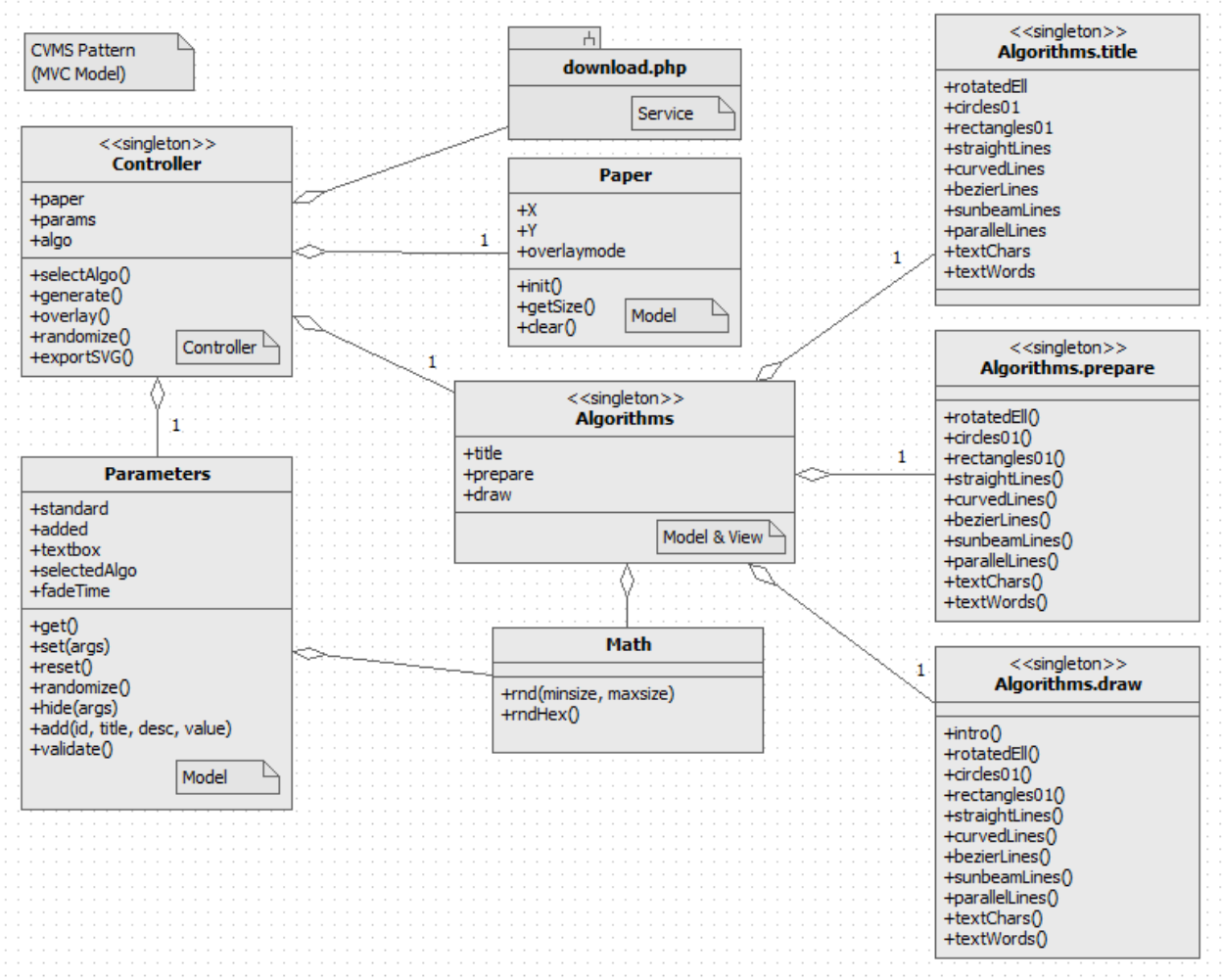
Schritt 1

- **Algorithmus auswählen** (optional, Standard ist definiert)
- **Parameter anpassen** (optional, vernünftige Standards, von Algorithmus abhängig)
- **Start**: Startet den Generator, Ausgabe erfolgt im selben Fenster direkt neben der Steuerung

Schritt 2

- Bild wurde generiert, Benutzer kann Ergebnis ansehen.
- Er kann auch die Parameter jetzt anpassen (optional)
- Oder einen neuen Algorithus auswählen (optional)
- **Export to SVG**: SVG-Grafik als Datei exportieren (optional, für andere Programme)
- **Generate**: Generiert neue Grafik entsprechend den aktuellen Einstellungen

8. UML-Klassen-Diagramm



(View ist hier nicht abgebildet, da sie in HTML / CSS umgesetzt ist)

9. UML-Klassen-Beschreibung

Als Design-Pattern wurde CVMS verwendet.

Das **Model** verteilt sich auf *Paper*, *Parameter* und *Algorithms*.

Paper enthält den Zustand und die Funktionen der Fläche auf die die Grafik generiert wird.
Parameter enthält den Zustand der Parameter und die zugehörigen Funktionen um mit ihnen zu arbeiten.

Eine Besonderheit ist der *Algorithms* Singleton. Er besteht aus drei Objekten, Algorithms.title, Algorithms.prepare, Algorithms.draw. Diese enthalten jeweils wieder die Funktionen/Variablen der verschiedenen Algorithmen. So hat z.B. der Algorithmus "textChars" drei Einträge: Algorithms.title.textChars, Algorithms.prepare.textChars und Algorithms.draw.textChars. In Algorithms.draw wird auch auf das SVG Paper gezeichnet. Hier spielt also eine View Komponente mit ein. Dennoch halte ich es für sinnvoll hier nicht künstlich zu trennen: Wenn man dem Generator einen neuen Algorithmus hinzufügen will, muss man nur in der Algorithms.js die drei zusammengehörigen Funktionen anhängen.

Diese will ich hier kurz exemplarisch erklären: Nehmen wir einen neuen Algorithmus namens „Tree Graphic“ und der ID „tree“. Ich füge erst *Algorithsms.title.tree* = „Tree Graphic“; an.
Jetzt kommt *Algorithms.prepare.tree* = function() {…} dran. Hier werden Parameter Defaults gesetzt, neue hinzugefügt, unnötige entfernt.
Das letzendliche Berechnen und Zeichnen der Grafik übernimmt *Algorithms.draw.tree* = function() {…}. Hier werden die Berechnungen durchgeführt (Logik) und mittels RaphaelJS auf das Paper gezeichnet.

Das Vorgehen hilft auch dabei die Algorithms Singleton Objekte mittels for-Schleifen durchzulaufen. (Konkret wird hier Algorithms.title dazu verwendet) Die Optionen der Select Box können daher automatisch generiert werden, einfach nur durch einen weiteren Eintrag in der Algorithms.js. Es sind keine weiteren Änderungen an der View oder sonst irgendwo notwendig.

Die **View** wurde in HTML / CSS umgesetzt. Der JavaScript Code enthält keine Stilinformationen, es werden nach Möglichkeit nur CSS Klassen und ID’s zugewiesen.
Um Interaktivität zu gewährleisten generiert der JavaScript Code die HTML Struktur je nach Situation um. So muss die index.html nie neugeladen werden.

Der **Controller** wurde als Singleton angelegt, ebenfalls *Algorithms*, da beide keinesfalls mehrfach vorkommen sollten. Der Controller erstellt jeweils eine neue Instanz von Paper und Parameters. (Hier wären mehrere Instanzen denkbar, aber nicht implementiert) Diese initialisieren über den Konstruktor die Parameter. „*Parameters*“ generiert dabei auch automatisch die Optionen in der Select-Box auf Basis der in Algorithms.js eingetragenen Algorithmen.

Bsp: *Algorithms* greift also auf die *Parameters* und das *Paper* immer über den Controller zu. (Bsp: Controller.paper.getSize();)

Als **Service** benötigt es *download.php*, die vom jQuery Plugin *jquery.generateFile.js* eingebunden wird. Dieses serverseitige Skript ändert die Header des SVG Exports, so dass die SVG Datei vom Browser heruntergeladen werden kann. Rein clientseitiges JavaScript unterstützt dies aus Sicherheitsgründen nicht.

Die Standard-JavaScript Klasse "*Math*" habe ich um Math.rnd(minsize, maxsize) und Math.rndHex erweitert, um die Berechnungen in Algorithms.js und Parameters.js zu erleichtern.

Für Produktionseinsatz habe ich unter */js/* ein Batch Skript *_MinifyJS.bat* erstellt, dass mit Hilfe des Google Closure Compilers eine komprimiertie all.min.js erstellt. Im HTML ist dies aktuell auskommentiert.

10. Quellenangaben

Folgende Libraries und Snippets wurden verwendet und eingearbeitet:

- CSS: html5doctor.com Reset Stylesheet (http://richclarkdesign.com)
- JS: jQuery: (http://www.jquery.com)
- JS: Raphaeljs: (http://www.raphaeljs.com/)
- JS: jQuery Color Picker: (http://www.eyecon.ro/colorpicker/)
- JS: jQuery Colorbox: Nur für den info.html Bereich. (http://jacklmoore.com/colorbox/)
- PHP/JS: jQuery File Export: (http://tutorialzine.com/2011/05/generating-files-javascript-php)

Wo möglich bzw. sinnvoll habe ich sie in dem */lib/* Ordner eingebunden.