HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Bearbeitungsbeginn: 01.03.2021

Vorgelegt am 31.08.2021

Thesis

zur Erlangung des Grades

Bachelor of Science

im Studiengang Medieninformatik

an der Fakultät Digitale Medien

Marius König

Matrikelnummer: 254573

Aktualisierung des im Game-Editor FUDGE verwendeten Layout-Managers

Erstbetreuer: Prof. Jirka Dell'Oro-Friedl

Zweitbetreuer: Prof. Dr. Norbert Schnell

Abstract

Der Game-Editor FUDGE (Furtwangen Univerity Didactic Game Editor) wird an der Hochschule Furtwangen entwickelt. FUDGE ist für die Lehre optimiert und setzt auf Webtechnologien. Mit FUDGE ist es möglich Spiele mit Typescript, HTML und CSS zu entwickeln. Der grafische Editor von FUDGE setzt ebenfalls auf Webtechnologien. Verwendet werden neben Typescript, HTML und CSS auch Electron und GoldenLayout. Electron ermöglicht es Desktopanwendungen mit Webtechnologien zu entwickeln. GoldenLayout ist ein inzwischen mit Typescript entwickelter LayoutManager. Damit können Bestandteile einer Website platziert und angeordnet werden. Eine Änderung des Layouts, wie bspw. das Verschieben von Fenstern oder Tabs ist zur Laufzeit der Anwendung möglich.

Die in FUDGE verwendete GoldenLayout Version ist veraltet. GoldenLayout ist inzwischen umfangreich überarbeitet worden. Veröffentlichungen bis einschließlich der Version 1.59 wurden mit Javascript entwickelt. Dabei wurde auch jQuery eingesetzt. Die neue GoldenLayout-Verson wird mit Typescript entwickelt und verwendet jQuery nicht mehr.

Ziel dieser Arbeit ist die Aktualisierung des im GoldenLayout verwendeten Layout-Managers. Dabei soll die neue GoldenLayout-Version eingesetzt und der Editor entsprechend angepasst werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

HTML	Hypertext Markup Language
FUDGE	Furwangen Univerty Didactic Game Editor
CSS	Cascading Style Sheet
SDK	Software Development Kit
GUI	Graphical User Interface
UI	User Interface
ES	ECMAScript
NPM	Node Package Manager
UMD	Universal Module Definiton
AMD	Asynchronous Module Definition
SVG	Scalable Vectors Grapics
DOM	Document Object Model

1 Einleitung

Es gibt viele Game-Engines bzw. Game-Editoren. FUDGE (Furtwangen University Didactic Game Editior) ist ein Game-Editor, der für die Lehre optimiert ist[1]. Game-Engines wie Unity3D oder die Unreal-Engine setzten ihre Installation voraus. Unterschiedliche Versionen werden gleichzeitig gepflegt. Bei diesen Game-Engines muss die zum Projekt passende Version der Engine installiert werden, damit das Projekt im Editor geöffnet werden kann. Jede Version dieser Engines belegt mehrere Gigabyte an Speicher.

Mit FUDGE ist es einfacher Projekte zu erstellen und zu verteilen. FUDGE setzt auf Web-Technologien. FUDGE Projekte können in Web-Browsern ausgeführt werden, die WebGL unterstützen. Für die Darstellung der 3D-Grafik wird die JavaScript-Programmierschnittstelle WebGL verwendet.

FUDGE wird an der Hochschule Furtwangen entwickelt. Der Projektleiter ist Professor Dell'Oro-Friedl. Viele Bachelor- und Masterarbeiten haben das Ziel FUDGE weiterzuentwickeln.

Der FUDGE-Editor ist eine mit Electron entwickelte Desktop-Anwendung. Electron ist ein Framework, welches die Entwicklung von Desktop-Anwendungen mithilfe HTML, CSS und Javascript ermöglicht. Darüber hinaus ermöglicht Electron den Zugriff auf das Dateisystem des Betriebssystems. Es ist somit möglich Dateien wie bspw. Projektdateien zu speichern und zu lesen. FUDGE nutzt den Layout-Manager GoldenLayout. Mit einem Layout-Manager wie GoldenLayout ist das kontrollierte Platzieren von User-Interface-Bestandteilen auf dem Anwendungsfenster (Website) möglich.

Das ursprüngliche Ziel dieser Arbeit war es, einen auf GoldenLayout basierenden Layout-Manager zu entwickeln. Dieser sollte mit Typescript entwickelt werden. Anlass dafür ist, dass die im FUDGE-Editor verwendete Version von GoldenLayout die Javascript-Library jQuery verwendet. Die Verwendung von jQuery kann zu einer schlechteren Leistung und Ladezeit einer Web-Anwendung führen[2].

---

[1] Jirka Dell'Oro-Friedl, „GitHub Wiki - JirkaDellOro/FUDGE: Furtwangen University Didactic Game Editor." Zuletzt geprüft am 30.08.2021, https://github.com/JirkaDellOro/FUDGE/wiki.

[2] Marco Trombino, „You might not need jQuery: A 2018 Performance case study." Zuletzt geprüft am 13.08.2021, https://medium.com/@trombino.marco/you-might-not-need-jquery-a-2018-performance-case-study-aa6531d0b0c3.

Zu Beginn der Bearbeitungszeit ist eine neue GoldenLayout-Version veröffentlicht worden. GoldenLayout wird seit der Version 2.0 mit Typescript entwickelt. Die Javascript-Library jQuery wird nicht mehr verwendet[3]. Das neue Ziel dieser Arbeit ist die Aktualisierung des im Game-Editor FUDGE verwendeten Layout-Managers. GoldenLayout 2 soll verwendet und der Editor um weitere Ansichten bzw. Panels erweitert werden. Dazu wird der Quellcode des FUDGE-Editors angepasst bzw. verändert.

[3] GoldenLayout, „GoldenLayout-Readme." GoldenLayout, zuletzt geprüft am 30.08.2021, https://github.com/golden-layout/golden-layout/blob/master/README.md.

2 Verwendete Technologien des FUDGE-Editors

2.1 FUDGE

FUDGE (Furtwangen Didactic Game Editor) ist ein für die Lehre optimierter Game-Editor[4]. Durch die Verwendung von Web-Technologien ist eine plattformunabhängige Verwendung möglich. (Windows, MacOS, GNU/Linux). FUDGE-Projekte können entweder mit dem auf Electron und GoldenLayout basierenden Editor oder ausschließlich mit Programmcode entwickelt werden.

2.2 Electron

Electron ist ein Framework, welches die Entwicklung von Desktop-Anwendungen mithilfe von Web-Technologien ermöglicht. Das sind HTML, CSS und JavaScript. Electron ist ein CommonJS-Modul**.** Siehe Kapitel Module und Namensräume. Die Verwendung von Node.js ist somit notwendig. Node.js ermöglicht den vollständigen Dateisystemzugriff. Der FUDGE-Editor nutzt dies bereits, um Projekte zu speichern und zu laden.

Electron basiert auf Chromium und Node.js. Chromium ist ein quelloffener Webbrowser. Diverse Web-Browser wie Google-Chrome, Opera, Vivaldi, Microsoft-Edge und Brave basieren auf Chromium. Teil von Chromium ist die Javascript-Engine V8 und der HTML-Renderer Blink[5]. In Electron können keine Tabs verwendet werden. Es ist aber möglich mehrere Fenster zu öffnen.

Eine Electron-Anwendung startet zwei Prozesse. Diese sind der Main-Prozess und der Renderer-Prozess[6]. Der Main-Prozess steuert das von Electron erzeugte Fenster. Hier können bspw. Höhe und Breite definiert werden. Außerdem ist eine Verwendung von ‚Node.js' möglich. Im Main-Prozess wird auch bestimmt, welche HTML-Datei geladen werden soll.

---

[4] Dell'Oro-Friedl, „GitHub Wiki - JirkaDellOro/FUDGE: Furtwangen University Didactic Game Editor".
[5] Herbert Braun, „Chrome und Chromium: Was sind eigentlich die Unterschiede?" *heise online*, 07.12.2018, zuletzt geprüft am 30.08.2021, https://www.heise.de/newsticker/meldung/Chrome-und-Chromium-Was-sind-eigentlich-die-Unterschiede-4245456.html.
[6] OpenJS Foundation, „Quick Start | Electron." OpenJS Foundation, zuletzt geprüft am 30.08.2021, https://www.electronjs.org/docs/tutorial/quick-start.

Bei einer Electron-Anwendung kann auch das Anwendungsmenü angepasst werden. Der FUDGE-Editor nutzt dies, um das Laden von UI-Bestandteilen wie PanelGraph und PanelProjekt zu ermöglichen. Menüeinträge können genutzt werden, um den Datei-Öffnen-Dialog bzw. den Datei-Speichern-Dialog des Betriebssystems zu öffnen. Somit ist es möglich FUDGE-Projekte zu speichern und zu laden.

Der Renderer-Prozess steuert die eigentliche Web-Anwendung (Chromium). Das sind HTML-, JavaScript-, und CSS-Dateien sowie sonstige Dateien wie bspw. Bild- und Audiodateien.

Electron beinhaltet das Remote-Modul. Damit ist es möglich im Renderer-Prozess auf Main-Prozess-Module zuzugreifen. Das Remote-Modul wurde aus Sicherheits- und Leistungsgründen inzwischen ersetzt[7]. Die Verwendung ist aber weiterhin möglich. Die Nachfolger sind die Module ‚ipcMain' und ‚ipcRenderer'. ‚ipcMain' ermöglicht die asynchrone Kommunikation vom Main-Prozess zum Renderer-Prozess[8]. ‚ipcRenderer' ermöglicht die Kommunikation des Renderer-Prozesses mit dem Main-Prozess[9].

Das ipcMain-Renderer-Modul wird auch im FUDGE-Editor verwendet. Wird ein Menüeintrag angeklickt, dann kann dies im Renderer-Prozess mithilfe des Electron-Eventsystems abgefangen werden. Je nach ausgewählten Menüeintrag wird die entsprechende Funktion gestartet.
Wird bspw. der Menüeintrag Graph des FUDGE-Editors angeklickt, dann wird mithilfe der Eventsysteme von FUDGE und Electron das Panel mit dem Namen PanelGraph geöffnet. Siehe Kapitel Änderungen am Quellcode des FUDGE-Editors

[7] OpenJS Foundation, „Electron Documentation - Remote." OpenJS Foundation, zuletzt geprüft am 30.08.2021, https://www.electronjs.org/docs/api/remote.
[8] OpenJS Foundation, „Electron Documentation - ipcMain." OpenJS Foundation, zuletzt geprüft am 30.08.2021, https://www.electronjs.org/docs/api/ipc-main.
[9] OpenJS Foundation, „Electron Documentation - ipcRenderer." OpenJS Foundation, zuletzt geprüft am 30.08.2021, https://www.electronjs.org/docs/api/ipc-renderer.

Es kann sinnvoll sein, eine Electron-Anwendung als gepacktes und ausführbares Programm auszuliefern. Der Quellcode der Anwendung wird somit zur Ausführung des Programms nicht benötigt. Die Installation von NPM-Paketen entfällt. Gepackte Anwendungen können mit Tools wie ‚electron-forge', ‚electron-builder' oder ‚electron-packager' erstellt werden.

2.3 Node.js und NPM-Pakete

Node.js ist eine JavaScript-Laufzeitumgebung, die das Ausführen von JavaScript-Dateien außerhalb von Webbrowsern ermöglicht. Mit Node.js können Webserver bzw. Netzwerkanwendungen mit JavaScript programmiert werden[10]. Es können statische oder dynamische Websites ausgeliefert werden. Es kann mit GET und POST gearbeitet werden. Dynamische Websites werden erst erstellt, wenn sie ausgeliefert werden.

2.3.1 Paketmanager - NPM

Paketmanager stellen eine Auswahl an Software bereit, welche heruntergeladen und installiert werden kann.

Die GNU/Linux-Betriebssysteme Debian und Ubuntu beinhalten Softwarepaketquellen. Programme können über eine Art App-Store oder über das Terminal (Bash, ZSH ect.) heruntergeladen und installiert werden. Abhängigkeiten werden automatisch installiert. Mit diesen Softwarepaketquellen ist es möglich das ganze Betriebssystem zu aktualisieren.

Beispiel Debian:

```
sudo apt install chromium
```

Beispiel Debian:

```
sudo apt update und sudo apt upgrade
```

[10] OpenJS Foundation, „About Node.js." OpenJS Foundation, zuletzt geprüft am 30.08.2021, https://nodejs.org/en/about/.

NPM ähnelt dem Paketmanager bzw. genauer dem Paketverwaltungssystem von Debian. Anders als bei Debian sind installierte Pakete i.d.R. nur innerhalb eines Projekts verfügbar. NPM-Pakte werden in einem Unterordner des ausgewählten Projektordners installiert. Eine globale Installation ist aber möglich. Wenn bspw. Typescript global installiert wird, kann der Typescript-Compiler auch dann verwendet werden, wenn Typescript nicht im Projektordner installiert ist. Wird Typescript lokal für ein Projekt installiert, dann können Typescript-Befehle nur mithilfe eines NPM-Scripts oder voranstellen des Schlüsselworts ‚npx' ausgeführt werden.

2.3.2 Package.json

Die Datei ‚package.json' listet alle Abhängigkeiten eines Projekts auf. Der Befehl `npm -init` erzeugt eine ‚package.json' interaktiv. Hier wird der Name der Anwendung sowie weitere Eingaben abgefragt.[11]

Pakete können mit `npm i nameDesPakets bzw. npm install nameDesPakets` installiert werden. In diesem Fall werden Pakete als Dependency (Abhängigkeit) installiert. Installierte Pakete werden in einer JSON-Datei mit dem Namen ‚package.json' aufgelistet. NPM-Befehle beziehen sich auf den ausgewählten Ordner (Shell, CMD). NPM Pakete werden in einem Unterordner namens ‚node_modules' gespeichert. Wird ein Paket installiert, dann werden auch alle benötigten Abhängigkeiten installiert.[12]

Mithilfe der JSON-Datei ‚package.json' kann ein Projekt auf Plattformen wie Github veröffentlicht werden, ohne dass NPM-Pakete mitgeliefert werden müssen. Der Ordner ‚node_modules' kann mit einer Gitignore-Datei von der Veröffentlich ausgeschlossen werden. Das Git-Repository belegt somit weniger Speicher. Mithilfe der JSON-Datei können alle Pakete des Projekts mit dem Befehl `npm i` installiert werden.

[11] Tierney Cyren, „An Absolute Beginner's Guide to Using npm." The NodeSource Blog, zuletzt geprüft am 30.08.2021.
[12] Ebd.

2.3.3 Electron als NPM-Paket

Electron ist ein NPM-Paket, welches diverse Module beinhaltet. Electron benötigt ein Start-Skript und einen Startpunkt. Der Startpunkt ist eine JavaScript-Datei. In dieser wird die Electron-Anwendung definiert (Main-Prozess). Der Schlüssel ist Main und der Wert ist der Pfad zur JavaScript-Datei.
Das Start-Skript wird benötigt, um Electron zu starten. Der Befehl `electron` kann mithilfe eines NPM-Skripts oder ‚npx' ausgeführt werden.

Beispiel: Package.json für Electron (gekürzt):

```
{
  "name": "golden_layout_playground",
  "version": "1.0.0",
  "main": "main.js",
  "scripts": {
    "start": "electron .",
    "build": "tsc"
  },
  "license": "ISC",
  "devDependencies": {
    "@types/node": "^15.0.1",
    "electron": "^12.0.5",
    "typescript": "^4.2.4"
  }
}
```

2.4 Typescript - Allgemein

TypeScript ist eine auf JavaScript basierende statisch typisierte Programmiersprache[13]. Typescript-Dateien werden mithilfe des Typescript-Compilers in Javascript-Dateien kompiliert bzw. umgewandelt (transpiling)[14]. Die zu verwendende JavaScript-Version kann mithilfe einer Konfigurationsdatei bestimmt werden[15]. So kann die neuste JavaScript-Version (ESNEXT) verwendet werden, auch wenn die Ziel-JavaScript-Laufzeitumgebung sie noch nicht unterstützt.

2.4.1 ES6 und neuere Javascript-Versionen

Die meisten Javascript-Laufzeitumgebungen unterstützten ECMAScript-2015 (ES6) und damit ECMA-Script-Module (ESM)[16]. Darüber hinaus ermöglicht ES6 die Verwendung von Let-Variablen, Konstanten (const) und Arrrow-Funktionen. Let-Variablen können nur in dem Code-Block verwendet werden, in dem sie erstellt worden sind. Konstanten (const) haben einen konstanten und unveränderlichen Wert.[17]

Arrow-Funktionen oder Pfeilfunktionen haben eine kürzere Schreibweise als normale Funktionen. Sie können einer Funktion als Parameter übergeben werden[18].

2.4.2 Statische Typisierung

JavaScript ist eine dynamisch typisierte Programmiersprache. Welche Typen verwendet werden ist erst zur Laufzeit bekannt[19]. Einer Funktion die Zahlen addieren soll können bspw. auch Strings als Parameter übergeben werden. In diesem Fall werden die beiden Strings zusammengesetzt. Typen werden implizit bestimmt. Wird eine Variable z.B. mit einer Zahl initialisiert, dann wird zur Laufzeit bestimmt, dass der Typ dieser Variablen Number ist.

---

[13] Microsoft, „The TypeScript Handbook.“ Microsoft, zuletzt geprüft am 30.08.2021, https://www.typescriptlang.org/docs/handbook/intro.html.
[14] Boris Cherny und Jørgen W. Lang, *Programmieren in TypeScript: Skalierbare JavaScript-Applikationen entwickeln*, 1. Auflage (Heidelberg: O'Reilly, 2020), 6–7.
[15] Ebd., 12.
[16] Mozilla Foundation, „JavaScript modules - JavaScript | MDN.“ Zuletzt geprüft am 13.08.2021, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules.
[17] Said Hayani, „JavaScript ES6 — write less, do more.“ *freeCodeCamp.org*, 20.04.2018, zuletzt geprüft am 30.08.2021, https://www.freecodecamp.org/news/write-less-do-more-with-javascript-es6-5fd4a8e50ee2/.
[18] Ebd.
[19] Cherny und Lang, *Programmieren in TypeScript*, 8.

Typescript ist eine statisch typisierte Programmiersprache[20]. D.h. bei Variablen, Parametern, Rückgabewerten und Attributen können die Typen genau bestimmt werden. Soll bspw. einem Konstruktor ein Objekt eines bestimmten Typs als Parameter übergeben werden, so kann das vorgegeben werden.

2.4.3 Objektorientierte Programmierung

In Programmiersprachen wie Java oder C# kann die Sichtbarkeit von Attributen und Methoden definiert werden. Das ist auch mit Typescript möglich. Methoden und Attribute, die mit dem Schlüsselwort ‚private' versehen sind, können nur innerhalb ihrer Klasse verwendet werden. Objekte haben keinen Zugriff auf diese Attribute und Methoden. Der Zugriff kann durch Getter- und Setter-Methoden erfolgen. Mit dem Schlüsselwort ‚Public' ist ein Zugriff auf Attribute und Methoden von jeder Instanz aus möglich. Das Schlüsselwort ‚Protected' ähnelt dem Schlüsselwort ‚Private'. Es ermöglicht aber Subklassen den Zugriff auf Methoden und Attribute der Elternklasse. [21]

Darüber hinaus können Methoden und Attribute mit dem Schlüsselwort ‚Static' versehen werden. Statische Methoden und Attribute haben keinen Objektzusammenhang. Für ihre Verwendung wird kein Objekt benötigt. Das Schlüsselwort ‚static' kann auch in JavaScript verwendet werden. [22]

[20] Microsoft, „The TypeScript Handbook".
[21] Microsoft, „Documentation - Classes." Zuletzt geprüft am 30.08.2021, https://www.typescriptlang.org/docs/handbook/2/classes.html.
[22] Mozilla Foundation, „Statische Methoden - JavaScript | MDN." Zuletzt geprüft am 30.08.2021, https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Classes/static.

Mit Interfaces ist es möglich die Struktur eines Objekts vorzugeben. Interfaces können Attributs- und Methodendeklarationen beinhalten. Die Initialisierung von Attributen ist nicht möglich. Methoden in Interfaces beinhalten nur die Methodenamen, Parameter und Rückgabewerte. Ein Interface kann als Typescript-Typ verwendet werden. In diesem Fall ist es erforderlich ein Objekt zu erzeugen, welches den Vorgaben des Interfaces entspricht. Das bedeutet, dass es erforderlich ist alle Attribute und Methoden des Interfaces zu implementieren. Interfaces können auch in Klassen implementiert werden. In diesem Fall muss die implementierende Klasse alle Attribute und Methoden des Interfaces implementieren. Interfaces können nur mit Typescript verwendet werden. Sie sind nicht Bestandteil von JavaScript. [23]
Das FUDGE-Projekt nutzt Interfaces, um einigen Klassen Strukturen vorzugeben.

Außerdem ermöglicht Typescript die Verwendung von abstrakten Klassen. Diese können nicht instanziiert werden. Nur Subklassen von abstrakten Klassen können zur Erzeugung von Objekten verwendet werden[24]. Im FUDGE-Editor ist bspw. die View-Klasse abstrakt. Abstrakte Klassen können wie Interfaces nur mit Typescript verwendet werden.

2.4.4 Enums

Mit Enums ist es möglich Werte eines Typs aufzuzählen[25]. Ein Enum kann bspw. die Sprachen Englisch, Spanisch, Deutsch, Französisch als Schlüssel beinhalten. Enums können aus Schlüssel- und Wertpaaren bestehen. Werte sind optional. Der zugewiesene Wert eines Schlüssels kann ein String oder ein Zahlenwert (Number) sein. Das FUDGE-Projekt nutzt Enums. In JavaScript können keine Enums verwendet werden[26].

[23] Cherny und Lang, *Programmieren in TypeScript*, 92–97.
[24] Ebd., 88.
[25] Ebd., 40.
[26] Microsoft, „TypeScript: Handbook - Enums.“ Zuletzt geprüft am 30.08.2021, https://www.typescriptlang.org/docs/handbook/enums.html#numeric-enums.

2.4.5 Tsconf.json

Die JSON-Datei Tsconfig.json kann verwendet werden, um den Typescript-Compiler zu konfigurieren. Darüber hinaus können einzelne Dateien und Ordner von der Kompilierung ausgeschlossen werden. Das explizite inkludieren von Dateien ist ebenfalls möglich. Standardmäßig werden alle Typescript-Dateien des Projekts kompiliert. Kompilierte JavaScript-Dateien und Quellcode-Dateien (Typescript-Dateien) können in separaten Ordnern gespeichert werden. Das können bspw. die Ordner Src und Dist sein. Die Pfade dieser Ordner müssen in der Konfigurationsdatei angegeben werden. In der Konfigurationsdatei kann auch bestimmt werden, ob Node.js verwendet werden soll. Mit dem Befehl `tsc –init` kann eine vordefinierte Konfigurationsdatei erzeugt werden.[27]

Beispiel Tsconfig.json

```
{
  "compilerOptions": {
     "target": "ESNext",
     "module": "esnext",
     "sourceMap": true,
     "outDir": "./dist",
     "rootDir": "./src",
     "strict": true,
     "moduleResolution": "node"
   },
   "types": ["node", "electron"]
}
```

[27] Cherny und Lang, *Programmieren in TypeScript*, 12.

2.4.6 TSLint und ESLint

Mit TSLint bzw. ESLint ist es möglich einen bestimmten Code-Stil bzw. eine bestimmte Core-Formatierung zu erzwingen. So kann bspw. verhindert werden, dass any als Typ angegeben wird. Die Anzahl der Leerzeichen bei einer Einrückung ist einstellbar.
ESLint ist der Standard-Linter in Typescript. TSLint wird nicht mehr weiterentwickelt. Die Entwickler von TSLint unterstützen ESLint. [28]

Linter gibt es nicht nur für Typescript. Für Python gibt es diverse Linter wie bspw. PyLint oder Flake 8. Auch hier kann eine einheitliche Formatierung des Quellcodes vorgegeben werden. Wie bspw. die Anzahl der leeren Zeilen nach einer Funktion. Leerzeichen nach Rechenoperationen und die Anzahl der Leerzeichen bei Einrückungen können vorgeben werden.

2.4.7 Namensräume

Ermöglicht die Organisierung von Programmcode. Dies wird im Kapitel 5 beschrieben.

2.5 WebGL

FUDGE nutzt WebGL zum Rendern von 2D- und 3D-Grafiken.Zur Anzeige wird ein HTML-Canvas-Element verwendet. WebGL (Web Graphics Library) basiert auf OpenGL-ES. WebGL benötigt keine Plug-Ins und kann mit den meisten Webbrowsern verwendet werden. [29]

2.6 GoldenLayout 1

GoldenLayout ist ein mit JavaScript entwickelter Layoutmanager. Siehe Kapitel Layout-Manager

[28] Palantir, „TSLint in 2019 - Palantir Blog.“ *Palantir Blog*, 19.02.2019, zuletzt geprüft am 30.08.2021, https://blog.palantir.com/tslint-in-2019-1a144c2317a9.
[29] Mozilla Foundation, „WebGL - Web API Referenz | MDN.“ Zuletzt geprüft am 30.08.2021, https://developer.mozilla.org/de/docs/Web/API/WebGL_API.

2.6.1 jQuery

GoldenLayout 1 nutzt die JavaScript-Programmbibliothek jQuery. JQuery beinhaltet Funktionen, die das Arbeiten mit der DOM vereinfachen sollen. Die Manipulation von DOM-Elementen ist mit jQuery-Selektoren möglich. Früher unterschieden sich HTML-Renderer und JavaScript-Implementierung stärker voneinander als heute. Die Berücksichtigung der verschiedenen Renderer bzw. Implementierung war zeitaufwändig. JQuery gewährleistet die Funktionsfähigkeit des Quellcodes bei allen verbreiteten Webbrowsern.
Heute gibt es nur wenige HTML-Renderer und Javascript-Implementierungen. Die verbreitetsten Browser bzw. HTML-Renderer sind Chromium, Chrome (Blink), Safari (Webkit) und Firefox (Gecko bzw. Webrenderer)[30].

In vielen Fällen kann auf jQuery verzichtet werden. Da jQuery eine komplexe Bibliothek ist, kann dies einen negativen Einfluss auf die Leistung einer Webanwendung haben. Der Blogger Marco Trombino hat die Leistung von jQuery in einem Versuch untersucht. Dazu sind 10000 DIV-Elemente mit einer Schleife erzeugt worden. Bei jedem Durchlauf der Schleife wurde dem erzeugtem DIV ein Class-Attribut zugewiesen. Die reine JavaScript-Version wurde in 18,99ms geladen. Mit 195,89ms war die Ladezeit der jQuery-Version ungefähr zehnmal länger. [31]

[30] Jörn Brien, „Ranking: Microsoft-Browser Edge verdrängt Firefox von Rang 2.“ *t3n Magazin*, 06.10.2020, zuletzt geprüft am 30.08.2021, https://t3n.de/news/ranking-browser-edge-chrome-1326536/.
[31] Trombino, „You might not need jQuery: A 2018 Performance case study“.

3 Layout-Manager

Ein Layout-Manager ermöglicht die nach Regeln definierte Anordnung von GUI-Elementen[32]. Bei einer pixelgenauen Platzierung ist es erforderlich auf die Fenstergröße bzw. die Auflösung der Anwendung zu achten. Da bei einem Layout-Manager keine pixelgenauen Werte zur Platzierung der Elemente verwendet werden, muss bei der Gestaltung von Benutzerschnittstellen nicht auf die Fenstergröße geachtet werden.

Layout-Manager sind oft Bestandteile eines Software-Development-Kits bzw. eines GUI-Toolkits. Das Java-SDK und das Android-SDK beinhalten jeweils mehrere Layout-Manager, die zur Anordnung von GUI-Elementen verwendet werden können.

Jeder Android-Layout-Manager ordnet GUI-Elemente nach anderen Prinzipien an. Listen mit GUI-Elementen können vertikal oder horizontal angeordnet werden. Darüber hinaus ist eine Anordnung in einem Gitter möglich. [33]

Das Java-SDK beinhaltet 8 Layout-Manager. Der Standard-Layout-Manager in Java ist das Border-Layout Dieser unterteilt das Fenster in die fünf Bereiche West, East, North, South und Center. GUI-Elemente werden bei diesem Layout-Manager einem dieser Bereiche hinzugefügt. [34]

[32] Philip Ackermann und Leo Leowald, *Schrödinger programmiert Java: Das etwas andere Fachbuch; [mit Syntax-Highlighting!; von den Sprachgrundlagen über Multithreading bis zur komplexen GUI-Anwendung; nutze die Schwerter aller Versionen: Generics, New File I/O und Java 8; ideal zum Durchblicken und Hand anlegen, fantastisch illustriert*, 1. Aufl., Schrödinger programmiert (Bonn: Galileo Press, 2014), 589.
[33] Dawn Griffiths und David Griffiths, *Head first Android development*, Second edition (Beijing, Boston, Farnham, Sebastopol, Tokyo: O'Reilly, 2017), 556–57.
[34] Ackermann und Leowald, *Schrödinger programmiert Java*, 590–92.

3.1 GoldenLayout - Allgemein

GoldenLayout ist nicht Bestandteil eines Software-Development-Kits. GoldenLayout ist eine Javascript-Library, die seit Anfang des Jahres 2021 mit Typescript entwickelt wird. GoldenLayout wird als NPM-Modul ausgeliefert. Mithilfe des GoldenLayout-Quellcodes können Bundles erstellt werden. Bundles sind erforderlich, wenn GoldenLayout in einer statischen Web-Anwendung verwendet werden soll. Die mit NPM ausgelieferte Version kann nur zusammen mit einem Framework wie Angular oder einem Bundler wie Webpack verwendet werden. Die Gründe werden im Kapitel 5.8 beschrieben.

GoldenLayout ist ein Layout-Manager der GUI-Elemente bzw. HTML-Elemente nach den Prinzipien Row (Reihe), Column (Spalte) und Stack (Stapel, Registerkarte) anordnet. Die anzuordnenden HTML-Elemente sind Teil einer Component. Eine horizontale Anordnung kann mit einer Row erreicht werden. Columns ordnen Components vertikal an. Stacks sind Registerkarten.

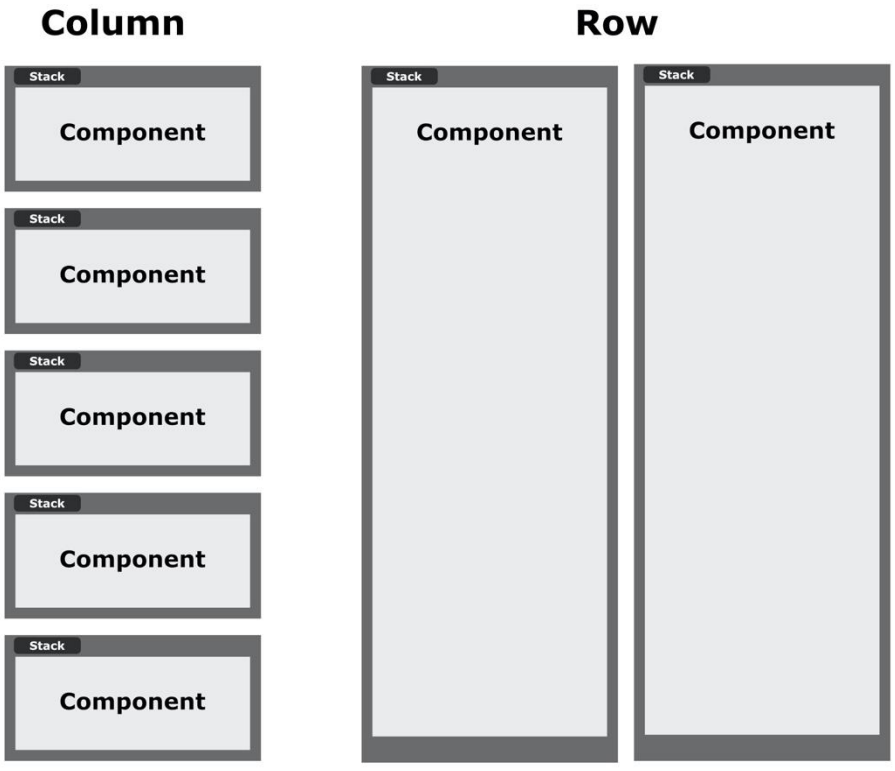

Abbildung 1: Rows und Columns

Ein Stack kann mehre Components beinhalten. Jede Component wird als Registerkarte angezeigt. Gibt es mehrere Registerkarten in einem Stack, wird der Inhalt der fokussierten bzw. geklickten Registerkarte angezeigt. Jeder Stack hat mindestens eine fokussierte Component.

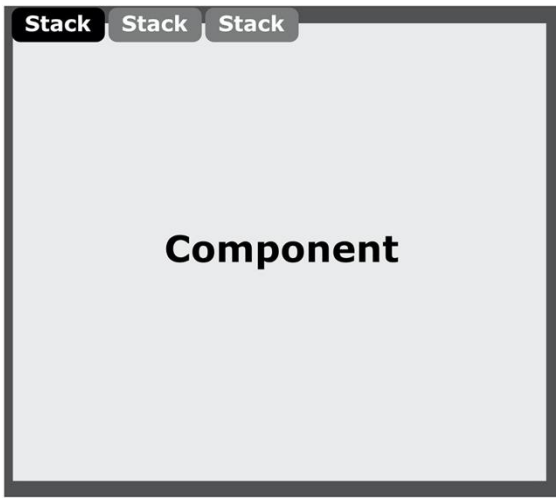

Abbildung 2: Mehrere zusammengefasste Stacks

GoldenLayout unterscheidet sich von den meisten andern Layout-Managern darin, dass es möglich ist, das Layout zur Laufzeit zu verändern. Durch Ziehen und Ablegen können Components bzw. die dazugehörigen Stacks neu platziert werden. Ein Stack kann bspw. von einer Row zu einer anderen Row verschoben werden. Es ist möglich erzeugte Components mithilfe von Trennlinien zu vergrößern oder zu verkleinern. Dafür müssen sie einer Row oder Column zugeordnet sein. Stacks können auch geschlossen oder maximiert werden.

3.1.1 Layouts

GoldenLayout benötigt ein Layout, um Inhalte anzuordnen bzw. anzuzeigen. Ein Layout wird mithilfe einer Variablen des Typs LayoutConfig definiert. Der Aufbau einer LayoutConfig ähnelt einer JSON-Datei. Diese Variable muss der GoldenLayout-Methode `loadLayout()` als Parameter übergeben werden.[35]

Rows, Columns, und Stacks können Components oder weitere Rows und Columns beinhalten. Ein Layout kann auf diese Weise vollständig vordefiniert werden. Components bzw. Items können auch dynamisch hinzugefügt werden Im Kapitel 6 wird die Erstellung eines Layouts beschrieben.

[35] GoldenLayout, „GoldenLayout-Readme“.

4 Benutzerschnittstelle des FUDGE-Editors

In diesem Kapitel wird das Prinzip des User-Interfaces des FUDGE-Editors näher erläutert. Die Einrichtung und Anwendung von GoldenLayout 2 wird im Kapitel 6 beschrieben. Das UI des FUDGE-Editors besteht aus den UI-Elementen View, Panel und Page. Eine Page kann mehre Panels beinhalten. Ein Panel ist ein eigenständiger Teil des UIs. Wenn der FUDGE-Editor gestartet wird, werden die Panels PanelGraph und PanelProject geladen. Panels werden horizontal angeordnet. Ein View ist eine Teilansicht des jeweiligen Panels. Views werden innerhalb des Panels positioniert. Innerhalb eines Panels kann es mehrere Reihen geben. Diese Reihen sind GoldenLayout-Rows. Views werden einer Reihe zugeordnet, in der sie untereinander als GoldenLayout-Column angezeigt werden. Eine View ist bspw. der Renderer. Diese View ist ein Bestandteil des PanelGraphs.

Die statische Methode SetupGoldenLayout() der Page-Klasse erzeugt eine kontrollierende GoldenLayout-Instanz. Die ihr zugeordneten Panels werden als Reihe dargestellt. Jedes Panel erzeugt eine eigene GoldenLayout-Instanz. Das hat zur Folge, dass Views nur innerhalb des ihr zugehörigen Panels verschoben werden können. Ein View des PanelGraphs kann nicht in das PanelProject verschoben werden.

Panels können mithilfe von Menüeinträgen der Electron-Anwendung erzeugt werden. In der vorliegenden Version des FUDGE-Editors wird ein Test-UI geladen. Dies wird im Kapitel 6 näher beschrieben.

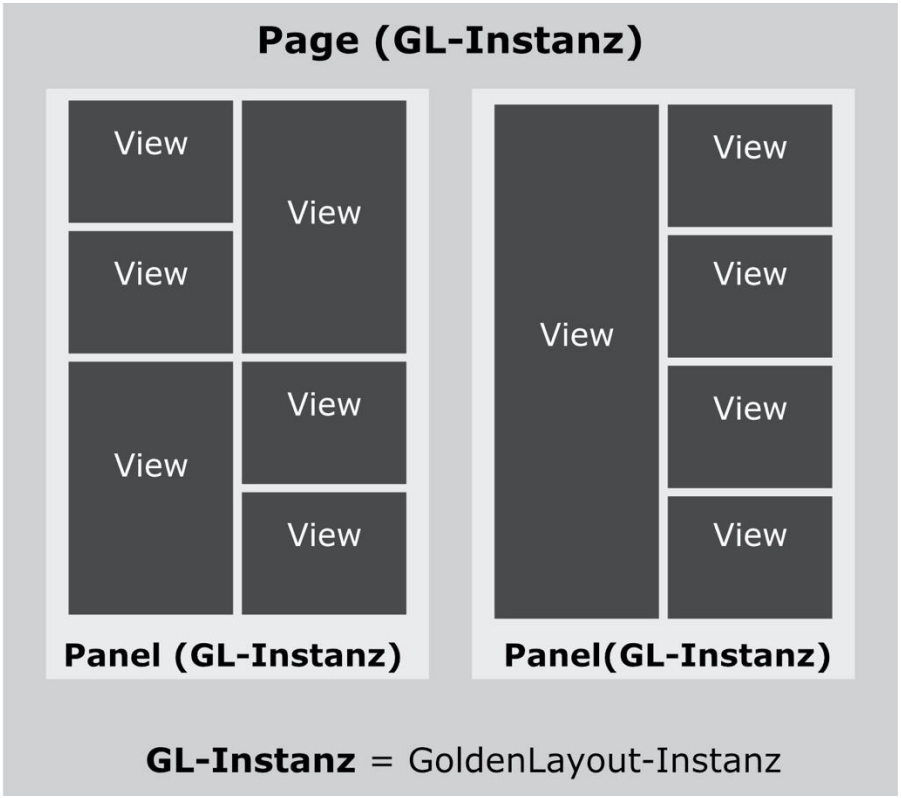

Abbildung 3: Page, Panel, View

4.1 PanelGraph und PanelView

PanelGraph zeigt einen Graph bzw. Renderer an. Ein Graph ist ein FUDGE-Node. Dieser kann Meshes und Materialien beinhalten. Ein Mesh beschreibt die Form eines 3D-Objekts durch Polygone und Vertices[36].

Ein Material beschreibt, wie die Oberfläche eines Meshes gerendert wird. Somit kann die Farbe eines Meshes bestimmt werden. Es ist auch möglich Texturen einzubinden[37].

[36] „Mesh, 3D." In *Encyclopedia of Multimedia* (Springer, Boston, MA, 2006), https://link.springer.com/referenceworkentry/10.1007/0-387-30038-4_126.

[37] Unity Technologies, „Unity - Manual: Meshes, Materials, Shaders and Textures." Zuletzt geprüft am 30.08.2021, https://docs.unity3d.com/2020.2/Documentation/Manual/Shaders.html.

Graphen, Materialien und Meshes können im ‚PanelProject' erzeugt werden. Mithilfe von Ziehen und Ablegen ist es möglich bestimmte FUDGE-Objekte vom ‚PanelProject' in bestimmte Views des ‚PanelGraphs' abzulegen. Das ist nur mit Inhalten möglich. Ganze ‚Views' können nicht in andere Panels verschoben werden. Im Quellcode des FUDGE-Editors wird bestimmt, welche Elemente der ‚Views' in ‚Views' anderer ‚Panels' gezogen werden können. Im ‚ViewInternal' des ‚PanelProject' können ‚Meshes', Materialien und Graphen erzeugt werden. Ein Graph kann im ‚Renderer-View' des ‚PanelGraphen' abgelegt werden. Darüber hinaus können erstellte Graphen auch in bestehende Graphen gezogen werden. Somit entsteht eine Baumstruktur. Bestehende Graphen befinden sich in der ‚Hierachy-View', welche Bestandteil des ‚PanelGraph ist'.

Materialien und Meshes können in die ‚Components-View' gezogen werden. Diese ‚View' zeigt die Inhalte des in der ‚Hierachy-View' ausgewählten Graphen an.

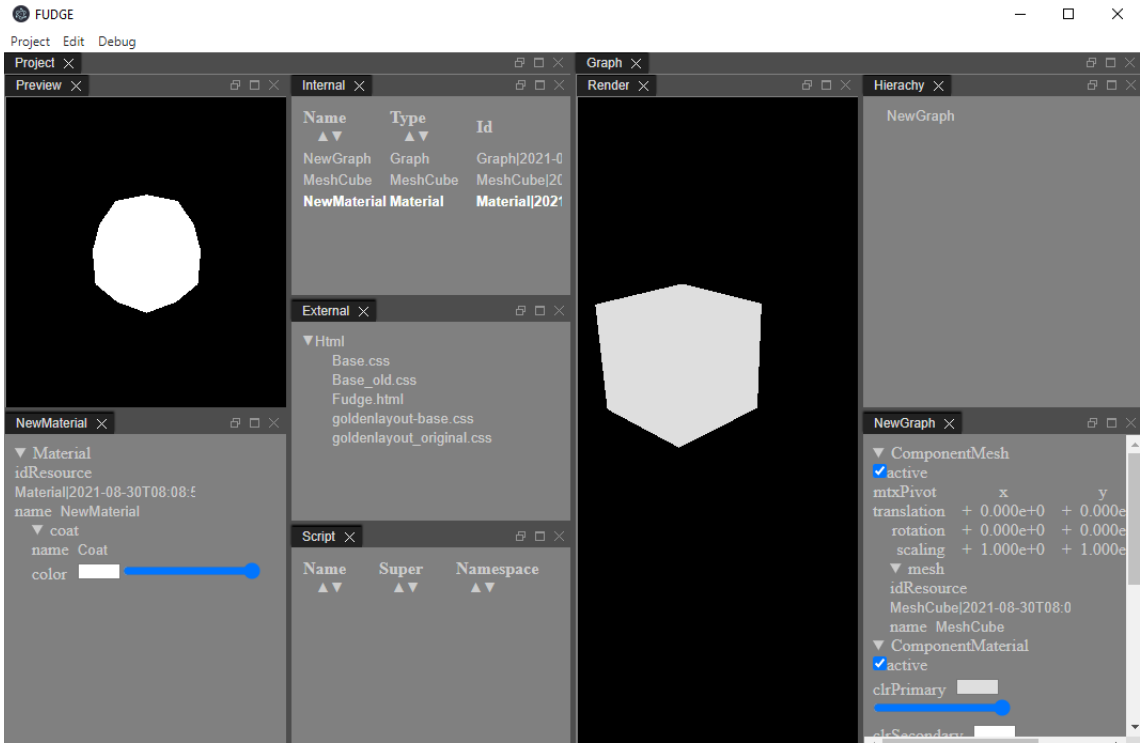

Abbildung 4: Screenshot des Fudge-Editors mit GoldenLayout 2

4.1.1 Aktivitätsdiagramm ‚Drag and Drop'

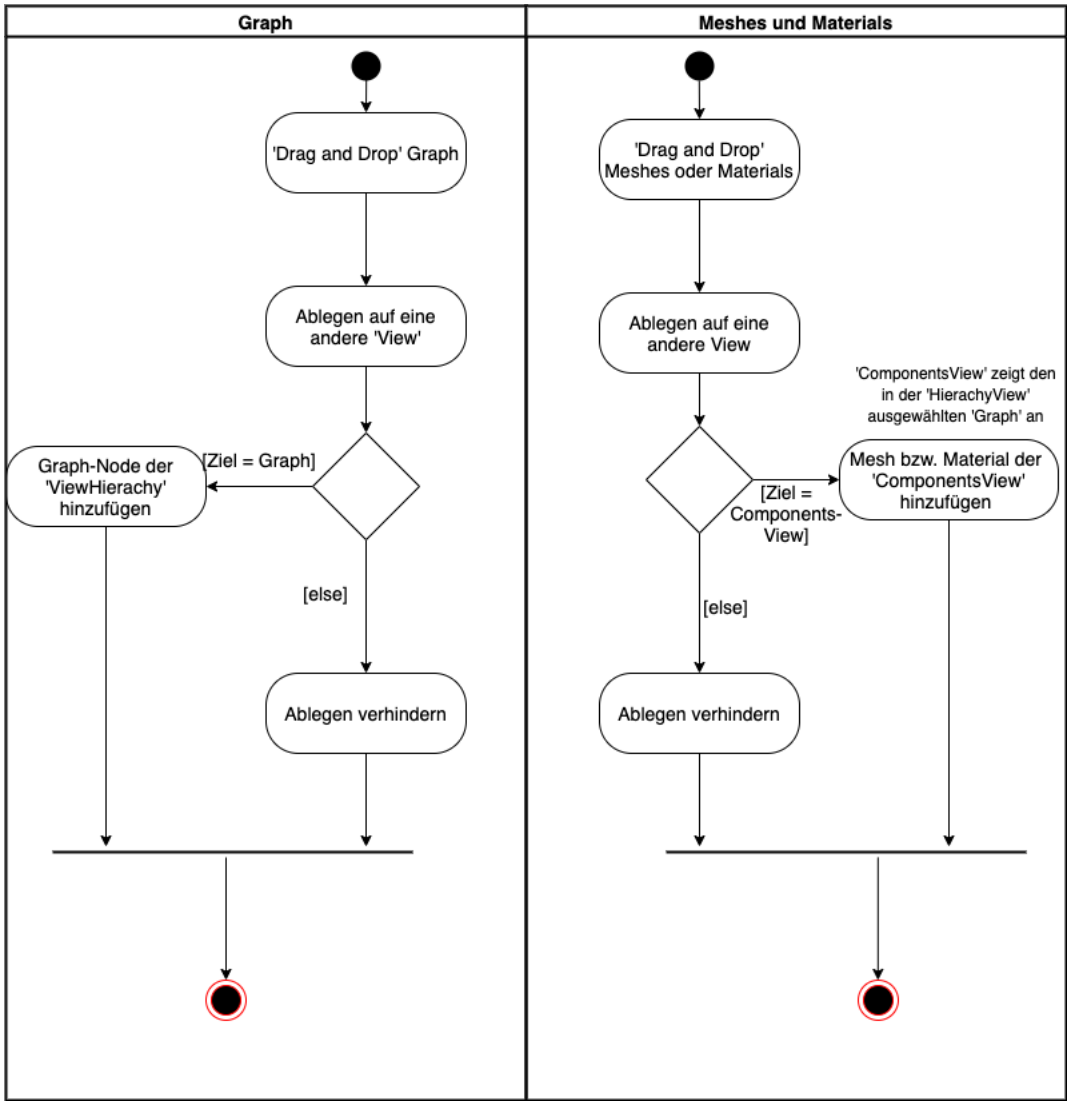

Abbildung 5: Aktivitätsdiagramm „Drag and Drop"

4.1.2 Erstellung eines einfachen Projekts

In diesem Unterkapitel wird beispielhaft gezeigt, wie ein Projekt im FUDGE-Editor erstellt werden kann. Zuerst wird ein Graph im ‚Internal-View' erstellt. Dies erfolgt mithilfe des Kontextmenüs (Rechtsklickmenü) des ‚Views'. Der erstellte Graph wird in den ‚Renderer-View' des ‚PanelGraph' gezogen und abgelegt. Anschließend wird mithilfe des Kontextmenüs des ‚Internal-Views' ein Mesh des Typs ‚MeshCube' und ein Material des Typs ‚ShaderUniColor' erzeugt. Diese werden in den ‚Components-View' des ‚PanelGraph' gezogen.

4.2 Animation

Es gibt Views mit denen es möglich Animationen zu erstellen. Diese Views beziehen sich auf eine alte Version des FUDGE-Editors. Ihre Verwendung ist derzeit nicht möglich.

5 Module und Namensräume

GoldenLayout wird als ESM und CommonJS-Modul ausgeliefert. Der Import von GoldenLayout 2 im FUDGE-Editor war eines der Hauptprobleme dieser Arbeit. Deshalb behandelt dieses Kapitel die Themen Module und Namensräume.

Module können verwendet werden, um JavaScript-Code auf mehrere Dateien aufzuteilen. Eine JavaScript-Datei ist ein Modul, wenn es Klassen oder Funktionen exportiert oder importiert[38]. Nur exportierte Bestandteile eines Moduls können von anderen Modulen importiert werden. Frameworks oder Libraries werden oft als Modul ausgeliefert. GoldenLayout 2 kann als NPM-Paket heruntergeladen werden. Dieses Paket beinhaltet das Projekt als ESM und als CommonJS-Modul. NPM-Pakete können oftmals nur mithilfe eines Build-Systems bzw. Bundlers wie Webpack in Webanwendungen verwendet werden. Bei GoldenLayout 2 ist das auch der Fall. Für JavaScript gibt es mehrere Modul-Systeme. Die verbreitetsten sind CommonJS und ESM.

5.1 ESM (ECMAScript-Module)

ECMAScript-Module sind mit ES6 bzw. ECMAScript-2015 eingeführt worden. Klassen oder Funktionen werden exportiert, wenn ihnen das Wort ‚export' vorangestellt wird. In anderen Modulen können diese Bestandteile mit dem Befehl ‚import' importiert werden. Die zu importierenden Klassen und Funktionen werden in geschweiften Klammern angeben. Abhängigkeiten von anderen Modulen werden auf diese Weise explizit deklariert. Je nach Javascript-Implementierung eines Webbrowsers können Module nur dann geladen werden, wenn bei Importen die Dateiendung ‚JS' angegeben wird. Werden Bundler wie Webpack verwendet, dann kann die Dateiendung entfallen.

Code-Beispiel:

```
export function sayHello(name: string): void {
console.log("Hello " + name);
}

Import {sayHello} from './pfad'
sayHello("Tester");
```

[38] Cherny und Lang, *Programmieren in TypeScript*, 224.

Soll ein ESM in der HTML-Datei eingebunden werden, dann muss es mit einem Script-Tag eingebunden werden. Dabei ist die Angabe des Attributs ‚Type' mit dem Wert ‚Module' erforderlich. Lädt ein eingebundenes ESM Bestandteile anderer Module, dann ist es nicht erforderlich diese Module in der HTML-Datei einzubinden. Jedes Modul lädt seine Importe automatisch. Die Ladereihenfolge ergibt sich aus den Importen. Daraus folgt, dass sichergestellt ist, dass Bestandteile anderer Module immer verfügbar sind, wenn sie verwendet werden sollen. Das ist bspw. bei globalen Variablen nicht der Fall. Diese können zu jedem Zeitpunkt der Laufzeit der Web-Anwendung erstellt oder verändert werden. ES-Module werden von den meisten Webbrowsern und von Node.js unterstützt[39].

5.2 Javascript-Modul-Packer

Die Anzahl der geladenen Module beeinflusst die Ladezeit und Leistungsfähigkeit der Web-Anwendung. Jedes Mal, wenn ein Modul benötigt wird, dann muss die entsprechende JavaScript-Datei geladen werden. Bundler bzw. Javascript-Modul-Packer wie Webpack ermöglichen es mehrere JavaScript-Dateien zu einer einzelnen Datei zu bündeln. Es ist möglich gebündelte Dateien zu vereinfachen bzw. optimieren. Ein Bundle ist in der Regel ein Modul. Die Art des zu erstellenden Moduls ist mithilfe einer Konfigurationsdatei bestimmbar.[40]

Bei Electron-Anwendungen wie dem FUDGE-Editor, werden eingebundene JavaScript-Dateien i.d.R. nicht über das Internet heruntergeladen. Somit können auch größere JavaScript-Dateien schnell geladen werden.

Mithilfe der Webpack-Erweiterung TS-Loader ist es möglich auch Typescript-Dateien zu bündeln. TS-Loader verwendet zur Kompilierung der Typescript-Dateien den Typescript-Compiler. Anschließend erfolgt die Bündelung der kompilierten Dateien. Die Typescript-Konfigurationsdatei ‚tsconfig.json' ist für die Verwendung von Webpack erforderlich.[41]
Darüber hinaus ist es auch erforderlich Webpack mithilfe einer Konfigurationsdatei zu konfigurieren. Ein Beispiel befindet sich im Anhang.

---

[39] Mozilla Foundation, „JavaScript modules - JavaScript | MDN".
[40] Ashish N. Singh, „An intro to Webpack: what it is and how to use it." *freeCodeCamp.org*, 15.01.2019, zuletzt geprüft am 13.08.2021, https://www.freecodecamp.org/news/an-intro-to-webpack-what-it-is-and-how-to-use-it-8304ecdc3c60/.
[41] WebpackJS, „TypeScript and Webpack." Zuletzt geprüft am 30.08.2021, https://webpack.js.org/guides/typescript/.

5.3 CommonJS und AMD

CommonJS ist das Standard-Modulsystem der JavaScript-Laufzeitumgebung Nojde.js[42]. Wie bei anderen Modulsystemen ist es möglich Code zu verkapseln. Der Import von Modulen erfolgt mit der Node.js-Funktion `require()`. JavaScript-Code kann mithilfe des Node.js-Attributs `Module.exports` als Modul exportiert werden.

Import:

```
const {app, BrowserWindow} = require("electron");|
```

Export:

```
module.exports.sayHello = function() {
    //…
}
```

CommonJS-Module können über NPM bezogen werden. NPM ist ein Paket-Manager, der Module über eine Paketequelle zur Verfügung stellt. Das Installationsprogramm von Node.js ermöglicht die Installation von NPM.

NPM-Pakete werden i.d.R. als CommonJS-Modul ausgeliefert. Da CommonJS-Module die Javascript-Laufzeitumgebung Node.js zur Ausführung benötigen, können diese nur dann in anderen JavaScript-Laufzeitumgebungen verwendet werden, wenn Bundler wie Browserify eingesetzt werden[43]. Dabei wird der Eigen- und Fremdcode sowie dessen Abhängigkeiten zu einer Datei gebündelt.
Darüber hinaus können NPM-Pakte mithilfe eines Content-Delivery-Networks wie Skypack geladen werden. Die von Skypack ausgelieferten Pakete sind für die Nutzung im Frontend optimiert. Die Verwendung eines Modul-Packers bzw. Bundlers ist für die Verwendung von Skypack-Paketen nicht erforderlich.[44] .

[42] Cherny und Lang, *Programmieren in TypeScript*, 218.
[43] Ebd., 219.
[44] Skypack, „Introduction: Skypack is a JavaScript Delivery Network for modern web apps.“ Zuletzt geprüft am 30.08.2021, https://docs.skypack.dev/.

Der AMD-Modulstandard (Asynchronus-Module-Definition) ähnelt dem CommmonJS-Modulsystem. AMD-Module können anders als CommonJS-Module asynchron geladen werden. Der AMD-Modulstandard wird vor allem vom Modullader ‚RequireJS' und der JavaScript-Bibliothek ‚DOJO' vorangetrieben. [45]

‚RequireJS' ermöglicht die Verwendung von AMD-Modulen in den Javascript-Laufzeitumgebungen der meisten Webbrowsern[46]. CommonJS-Module können in AMD-Module umgewandelt werden[47].

5.4 Globaler Namensraum

Als es noch keine JavaScript-Modulsysteme gegeben hat, ist es üblich gewesen, Programmcode im globalen Namensraum zu speichern[48]. Globale Variablen, ermöglichen die Aufteilung von Programcode auf mehrere Dateien. Da jede geladene JavaScript-Datei globale Variablen erstellen oder verändern kann, ist es erforderlich auf die Ladereihenfolge zu achten. Bei einer falschen Ladereihenfolge ist es möglich, dass auf nicht deklarierte Variablen zugegriffen wird. Globale Variablen können mit dem Schlüsselwort ‚Var' erzeugt werden. Dafür ist es erforderlich die Variable außerhalb eines Funktionskontexts zu deklarieren. Alternativ können globale Variable auch explizit im globalen Namensraum erstellt werden. Der Zugriff auf globale Variablen erfolgt durch Eingabe des Variablennamens. Die Schlüsselwörter Window und GlobalThis können vorangestellt werden. In JavaScript können Variablen auch Objekte, Klassen und Funktionen beinhalten.

Code-Beispiel - Erstellen von globalen Variablen:

```
var myVariable = „Hello"; // Außerhalb von Funktionen
window.myVariale = „Hello";
globalThis.myVariable = „Hello"
```

[45] Cherny und Lang, *Programmieren in TypeScript*, 219.
[46] RequireJS, „How to get started with RequireJS." Zuletzt geprüft am 30.08.2021, https://requirejs.org/docs/start.html.
[47] RequireJS, „CommonJS Notes." Zuletzt geprüft am 30.08.2021, https://requirejs.org/docs/commonjs.html.
[48] Cherny und Lang, *Programmieren in TypeScript*, 218.

5.4.1 Javascript-Namespace-Objecte

Da globale Variablen Teil des globalen Namensraums sind, können Variablennamen nur einmal verwendet werden. Selbst definierte Namensräume ermöglichen die Kapselung bzw. Organisierung von Programmcode. Namenskonflikte können auf die Weise vermieden werden. Namensräume können mithilfe von JavaScript-Objekten realisiert werden. Funktionen, Klassen und Variablen werden in diesem Fall als Attribute eines Objekts definiert. Dieses Objekt ist Teil des globalen Namensraums. Somit ist innerhalb eines globalen Namensraums ein dateiübergreifender Zugriff auf JavaScript-Programmcode möglich.

Code-Beispiel

```
var meinNamensraum = {

    halloWelt: function () {

        console.log("Hallo Welt");

    },

    person: class Person {

        name;

        constructor(_name) {

            this.name = _name;

        }

        sayHello() {

            console.log("Hello " + this.name);

        }

    }

};

// Zugriff
let myPerson = new yourNamespace.person("Tester");

myPerson.sayHello();
```

5.5 UMD

Der Modulstandard Universal-Module-Definition kann verwendet werden, um Module zu erzeugen, die in jeder JavaScript-Laufzeitumgebung verwendet werden können[49]. UMD-Module können wie CommonJS- oder AMD-Module importiert werden. Darüber hinaus ist es möglich UMD-Module in HTML-Dateien als Script-Tag einzubinden. In diesem Fall werden Module als globale Variablen deklariert und initialisiert.

5.6 Typescript Namensräume

In Programmiersprachen wie C# oder C++ kann Programmcode in Namensräumen organisiert werden. Somit ist es möglich Quellcode auf mehrere Dateien zu verteilen. Auch werden Namenskonflikte von Variablen, Klassen und Funktionen vermieden. Typescript ermöglicht die Verwendung von Namensräumen. Da dies nicht von JavaScript unterstützt wird, werden Namensräume bei der Kompilierung in JavaScript-Objekte umgewandelt[50]. Diese Struktur ähnelt den im Unterkapitel 5.4.1 beschriebenen Namensraum-Objekten. Der Typescript-Compiler kann Quellcode, der in Namenräumen organisiert ist, in eine Datei bündeln. Dabei ist es erforderlich in jeder Typescript-Datei die Abhängigkeiten von anderen Dateien durch ein Reference-Tag zu kennzeichnen. Werden Dateien nicht gebündelt, dann muss wie bei JavaScript-Namespace-Objekten darauf geachtet werden, dass alle zu ladenden Javascript-Dateien in der richtigen Reihenfolge mittels Script-Tag in einer HTML-Datei eingebunden werden. Das ist gerade bei größeren Web-Anwendungen eine potenzielle Fehlerquelle. Anders als bei Javascript-Namensraum-Objekten können nur exportierte Code-Bestandteile wie bspw. Klassen und Funktionen in anderen Typescript-Dateien verwendet werden[51].

[49] Cherny und Lang, *Programmieren in TypeScript*, 224.
[50] Microsoft, „TypeScript: Documentation - Namespaces and Modules.“ Zuletzt geprüft am 30.08.2021, https://www.typescriptlang.org/docs/handbook/namespaces-and-modules.html.
[51] Ebd.

Code-Beispiel:

```
namespace myNamespace {

    export class TestClass {

        //...

    }

    export function HalloWelt() {

        //...

    }

}
```

5.7 Module und Namensräume im FUDGE-Projekt

Das FUDGE-Projekt nutzt Namensräume zur Organisierung und Aufteilung des Programmcodes. Das Projekt ist aufgeteilt in die Namensräume ‚FudgeCore', ‚FudgeAid', ‚FudgeUserInterface' und ‚Fudge'. Für diese Arbeit ist insbesondere der Namensraum ‚Fudge' von Bedeutung, da in diesem der Quellcode des Editors organisiert ist. Innerhalb dieses Namensraum ist der Zugriff auf die anderen Namensräume möglich. Dies geschieht im Falle des FUDGE-Editors mithilfe von Alias-Importen. Aliase sind Repräsentation eines Namensraums, deren Name frei gewählt werden kann. In der Typescript-Datei ‚page.ts' wird bspw. der Namensraum ‚FudgeCore' mit dem Alias ƒ importiert. In diesem Fall kann durch Voranstellen des Alias-Namens auf Inhalte des Namensraums zugegriffen werden.

Das FUDGE-Projekt nutzt die Typescript-Compiler-Option ‚outFile', um Dateien zu bündeln. Aus jedem Namensraum wird eine entsprechende Javascript-Bündel-Datei erzeugt. Diese werden in der HTML-Datei des FUDGE-Editors mithilfe von HTML-Script-Tags eingebunden.

Jeder Teilbereich des Projects beinhaltet eine eigene Tsconfig-Datei. Das betrifft auch die Electron-Anwendung und den FUDGE-Editor. Electron ist ein CommonJS-Modul. Deshalb wird es mit der Node.js-Funktion ‚Require()' importiert. Die Electron-Anwendung lädt die HTML-Datei des FUDGE-Editors, welche alle gebündelten Javascript-Dateien des FUDGE-Projects einbindet. Die Ausführung der Electron-Anwendung erfolgt durch ein NPM-Skript.

5.8 Gemeinsame Verwendung von ESM und Typescript-Namensräumen

Wenn ECMAScript-Module importiert werden, dann können Typescript-Namensräume nur eingeschränkt verwendet werden, da ECMA-Script-Modul-Importe dazu führen, dass die importierende Datei selbst zu einem Modul wird. Namensräume können in diesem Fall nur dann von anderen Modulen bzw. Typescript-Dateien verwendet werden, wenn diese exportiert werden. Andere Module können dann mit ESM-Importen auf diese Namensräume zugreifen.

Darüber hinaus sind nicht alle Zielmodulsystem des Typscript-Compilers sind mit ESM-Importen kompatibel. Das voreingestellte Modulsystem ist CommonJS. In diesem Fall findet eine Umwandlung von ESM-Importen in CommonJS-Modulimporte statt. In der JSON-Konfigurationsdatei ‚tsconfig.json' kann auch ESM als Zielmodulsystem definiert werden. Der Typescript-Compiler kann in beiden Fällen nicht mehr verwendet werden, um gebündelte Javascript-Dateien zu erzeugen. Die Bündelung mithilfe des Typescript-Compilers ist nur bei den Zielsystemen ‚AMD' und ‚System' möglich Eine Bündelung ist auch dann möglich, wenn auf ein Modulsystem verzichtet wird. Das entspricht dem Zielmodulsystem ‚None'. [52]

5.8.1 Folgen für den Import von GoldenLayout 2 im FUDGE-Editor

Das mit NPM ausgelieferte ESM-Modul von GoldenLayout beeinträchtigt die Bündel-Funktion des Typescript-Compilers. Die Verwendung des ESM-Moduls würde mit umfangreichen Änderungen des gesamten FUDGE-Projekts einhergehen. Statt Namensräumen müssten Module und Bündler zur Organisierung und Bündelung des Programmcodes verwendet werden. Namespaces müssten in diesem Fall wie andere Codebestandteile explizit importiert und exportiert werden. Alternativ kann auf die Verwendung von Typescript-Namensräumen verzichtet werden. Die ESM-Version von GoldenLayout 2 ist auf mehrere Dateien verteilt. Die von GoldenLayout intern verwendeten Importpfade beinhalten keine Dateiendung. Somit ist eine Verwendung nur mit einem Bundler wie Webpack möglich. Die CommonJS-Version kann nicht ohne weiteres in Javascript-Laufzeitumgebungen von Webbrowsern verwendet werden. Dafür ist ebenfalls ein Bundler erforderlich. FUDGE verwendet bereits den Typescript-Compiler als Bundler.

[52] Microsoft, „TypeScript: Documentation - Namespaces and Modules".

Am 04.06.21 ist das GoldenLayout-Projekt mit Scripts erweitert worden, die das Erstellen von ESM- und UMD-Bündel ermöglichen[53]. Diese Bündel sind nicht Bestandteil des NPM-Pakets. Die Erstellung der Bündel ist nur mit dem Quellcode möglich. Diese Option ist zu Beginn der Bearbeitungszeit nicht verfügbar gewesen. Bis zu diesem Zeitpunkt wurde mit einem selbsterstellten ESM-Bundle gearbeitet. Dieses ist ebenfalls mit Webpack erzeugt worden.

Die Entwickler von GoldenLayout empfehlen die Verwendung der mit NPM ausgelieferten ESM-Version. In dieser Version ist der Programmcode auf mehrere Dateien verteilt. Selbsterstellte Javascript- bzw. Typescript-Dateien sollten in diesem Fall mit den GoldenLayout-ESM-Dateien gebündelt werden. Dieses Bündel muss dann mithilfe eines Script-Tags als Modul in einer HTML-Datei eingebunden werden.

[53] GoldenLayout, „Add single file bundle by martin31821 · Pull Request #665 · golden-layout/golden-layout.“ Github, zuletzt geprüft am 30.08.2021, https://github.com/golden-layout/golden-layout/pull/665.

6 Einrichtung und Verwendung von GoldenLayout 2

In diesem Kapitel geht es um den Import und die Verwendung von GoldenLayout 2 im Game-Editor FUDGE. Dies beschreibt den praktische Teil dieser Arbeit. Im Kapitel Änderungen am Quellcode des FUDGE-Editors wird beschrieben, wie der bestehende Code des FUDGE-Editors angepasst worden ist.

Die UMD-Version von GoldenLayout ist nicht Bestandteil des NPM-Pakets. Die Erzeugung der Bundles ist nur mit dem Quellcode von GoldenLayout möglich. Dazu wird das gesamte Projekt von Github heruntergeladen bzw. geklont. Die Installation aller NPM-Pakte ist erforderlich, um das Build-System zu verwenden. Die Erzeugung der UMD- und ESM-Bündel erfolgt mithilfe von Webpack. [54]

Befehle zur Erstellung der Bundles:

```
git clone https://github.com/golden-layout/golden-layout
npm install
npm run build
npm run build:bundles
```

Im Unterordner Dist befindet sich ein weiterer Unterordner mit dem Namen Bundle. In diesem befinden sich die ESM- und UMD-Bündel von GoldenLayout.

6.1 Import des UMD-Moduls von GoldenLayout

Nach der Erzeugung der Bundles wird der Ordner ‚Dist' aus dem Ordner des GoldenLayout-Projeks in den Ordner des FUDGE-Editors kopiert. Das UMD-Bundle wird mithilfe eines HTML-Script-Tags eingebunden. Die HTML-Datei des FUDGE-Editors wird entsprechend angepasst. Der Zugriff auf GoldenLayout ist nun mithilfe der globalen Variable ‚`GlobalThis.goldenLayout`' möglich. Auch die Einbindung der zu GoldenLayout gehörenden CSS-Dateien in der HTMl-Datei des FUDGE-Editors ist erforderlich.

[54] GoldenLayout, „GoldenLayout-Readme".

UMD-Module laden keine Typescript-Definition-Dateien. Das hat zur Folge, dass die von GoldenLayout definierten Typen nicht bekannt sind. Das schränkt die statische Typisierung ein. Auch die Code-Vervollständigung von Code-Editoren ist somit eingeschränkt. Beim Import des ESM-Moduls von GoldenLayout werden die Typ-Definitionen von Code-Editoren wie Visual-Studio-Code automatisch erkannt und verwendet.

Die Typ-Definitionen können auch mit UMD-Modulen verwendet werden. Dazu ist es erforderlich die Datei ‚index.d.ts' von GoldenLayout zu verändern. Das Schlüsselwort ‚export' führt bei der Verwendung des UMD-Moduls zu Problemen. Deshalb ist es erforderlich das Schlüsselwort ‚export' von allen Bestandteilen der Typ-Definitions-Datei zu entfernen. Das kann bspw. mithilfe der Suchfunktion von Visual-Studio-Code erreicht werden. Die Type-Definition wird mithilfe einer Tripple-Slash-Directive importiert. In der Datei ‚tsconfig.json' muss der Ordner des GoldenLayout-Bundles von der Kompilierung ausgeschlossen werden.

Code-Beispiel Tripple-Slash-Reference für GoldenLayout-Types:

```
/// <reference types="./goldenLayoutBundle/bundle/esm/golden-layout" />
```

Code-Beispiel - Zugriff auf die globale Variable von GoldenLayout:

```
export class Page {
    public static goldenLayoutModule = (globalThis as any).goldenLayout;
}
```

Dieses statische Attribut wird verwendet, um auf die globale Variable des UMD-Moduls von GoldenLayout zuzugreifen.

6.2 Alternative mit ESM-Modulen

Alternativ ist es möglich auf Typescript-Namensräume zu verzichten. In diesem Fall müsste der Programmcode des gesamten FUDGE-Projekts angepasst werden. Die Namensräume und Tripple-Slash-Direktiven müssten entfernt und Webpack zur Bündelung eingesetzt werden. NPM-Skripts ermöglichen die Ausführung von Webpack und das Starten des Typescript-Compilers. Die Webpack-Erweiterung ‚TS-Loader' kann eingesetzt werden, um Typescript-Dateien zu Bündeln. In diesem Fall startet ‚TS-Loader' den Typscript-Compiler.

Dies ist nur in einem Test-Projekt getestet worden. Für den FUDGE-Editor wird das UMD-Modul verwendet.

6.2.1 Barrel-Dateien

Jedes zu importierende Modul erfordert eine eigene Importzeile. Importiert ein Modul z.B. zehn andere Module, dann führt dies zu genauso vielen Importzeilen. Je mehr Module importiert werden, desto mehr Platz nehmen die Importzeilen ein. Mit Barrel-Dateien (Mantel) ist es möglich die Anzahl der Importzeilen zu reduzieren.

Ein Barrel ist ein Modul, welches andere Module exportiert. Dazu müssen die Pfade der zu exportierenden Module in den Exportzeilen der Barrel-Datei angegeben werden. Jedes so exportierte Modul kann somit auch über den Umweg des Barrel-Moduls geladen werden. Somit können alle Module, die von einem Barrel-File exportiert werden, mit nur eine Codezeile von anderen Modulen importiert werden. [55]

Code-Beispiel einer Barrel-Datei

```
export * from "./typescriptexport1";
export * from "./typescriptexport2";
```

Das FUDGE-Projekt beinhaltet mehrere Namensräume. Jeder organisiert einen Teil des FUDGE-Projekts. Im Namensraum des FUDGE-Editors ist der Import der anderen Nebenräume des FUDGE-Projekts möglich. Das ist z.B. in der Datei page.ts der Fall.

[55] Basarat A. Syed, „Barrel." Zuletzt geprüft am 30.08.2021, https://basarat.gitbook.io/typescript/main-1/barrel.

Ein Import von allen Modulen des FudgeCore könnte dann bspw. so aussehen:

```
import * as fCore from "../FudgeCore/FudgeCoreBarrel"
let myNode: fCore.Node = new fCore.Node();
```

Durch diesen Import entsteht eine Variable, die den Zugriff auf alle exportierten Module der Barrel-Datei ermöglicht. Einzelne Bestandteile von Modulen können mithilfe von geschweiften Klammern importiert werden. In diesem Fall wird keine Variable erstellt. Das Folgende Beispiel zeigt den expliziten Import von Modulbestandteilen:

```
Import {Node} from ",/FudgeCore/FudgeCoreBarrel"
let myNode: Node = new fCore.Node();
```

6.3 Kommunikation mit den Entwicklern

Die neue GoldenLayout-Version ist umfangreich überarbeitet worden. Eine Dokumentation gibt es nicht. Eine Reademe-Datei erläutert die wichtigsten Änderungen. Eine Demo-Anwendung demonstriert einen Teil der GoldenLayout-Funktionalitäten. Einige Detailfragen beantworten die GoldenLayout-Entwickler in den Gitub-Issues des Projekts. Auch Im Rahmen dieser Arbeit sind Fragen an die Entwickler gestellt worden. Es ist erfragt worden, wie GoldenLayout 2 ohne Webpack oder Frameworks wie z.B. Angular importiert werden kann. Außerdem sind Fragen zur dynamischen Erstellung von Layout-Inhalten beantwortet worden. Dynamisch hinzugefügte Components und Items werden nicht mit einem vordefinierten Layout platziert. Hierbei erfolgt die Positionierung mithilfe von GoldenLayout-Methoden.

6.4 Rows, Columns, Stacks, Components und Items

Wie im Kapitel 3,1 beschrieben, können auch mit GoldenLayout 2 Layout-Elemente in Rows, Columns und Stacks angeordnet werden. Layout-Elemente werden innerhalb von Rows horizontal und innerhalb von Colums vertikal angeordnet. Stacks sind Registerkarten.

6.4.1 Components und Items

Components beinhalten die HTML-Elemente, welche mit GoldenLayout angeordnet werden. Jede Component erzeugt einen Stack. Components können einem Layout hinzugefügt werden. Ein Layout kann entweder vollständig über einer Variablen bestimmt werden oder mithilfe von Items nacheinander und dynamisch aufgebaut werden. Items können verwendet werden, um Components anzuordnen. Ein Item beinhaltet Components und Layout-Konfigurationen. Components beinhalten Container. Diese können verwendet werden, um auf die HTML-Elemente der Components zuzugreifen.

Ein Item könnte bspw. so gestaltet werden, dass es mehrere Components jeweils vertikal anordnet. Dieses Item wird dann einem vordefinierten Layout hinzugefügt. Dieses Layout würde dann bspw. die von ihm gesteuerten Components horizontal anordnen. Das hinzugefügte Item wird dann neben den bestehenden Components angeordnet. Die Inhalte des Items werden untereinander angeordnet.

Items können mithilfe von Konfigurationsvariablen der GoldenLayout-Typen ‚RowOrColumnItemConfig', ‚StackItemConfig' und ‚ComponentItemConfig' erstellt werden.

Jede Component kann geschlossen und maximiert werden. Außerdem ist es möglich Components in ein eigenes Fenster zu verschieben. Diese Browserfenster beinhalten eigene DOM-Kontexte (Document Object Model) und GoldenLayout-Instanzen. Für jede Component kann bestimmt werden, ob sie schließbar ist. Dies erfolgt in den Item-Konfigurationsvariablen oder in der Layout-Config-Variable. Diese wird im nächsten Unterkapitel näher beschrieben.

6.5 Layouts erstellen

Dem Konstruktor von GoldenLayout kann ein HTML-Element übergeben werden. Innerhalb dieses Elementes wird GoldenLayout platziert. Ohne Übergabe eines HTML-Elements wird GoldenLayout innerhalb des Body-Tags platziert.

Jede GoldenLayout-Instanz muss mithilfe der GoldenLayout-Methode `loadLayout()` eine Variable des GoldenLayout-Typs ‚LayoutConfig' laden. In der alten GoldenLayout-Version muss die Layout-Config-Variable dem Konstruktor-Aufruf übergeben werden. Eine Variable des Typs 'LayoutConfig' hat folgenden Aufbau:

```
const mainConfig: LayoutConfig = {
    root: {
        type: "row",
        isClosable: true,
        content: [
            {
                type: "component",
                componentType: "PanelLeft",
                isClosable: true,
                componentState: "Panel-Test-String"
            },
            {
                // weitere Component, Row, Column oder Stack
            },
        ]
    }
}
```

Die Layout-Config-Variable beinhaltet das Attribut ‚Root'. In diesem wird die Item-Konfiguration des Root-Elements bestimmt. Diese wird zur Erstellung des ersten Items verwendet. Das Root-Attribut muss angegeben werden. In der alten GoldenLayout-Version ist das nicht der Fall. Das Content-Array beinhaltet Components oder weitere Rows, Columns und Stacks. Der ComponentType ist der Name der Component. [56]

6.5.1 Registrieren von Components

GoldenLayout kann nur registrierte Components anordnen. Die Registrierung erfolgt mit den GoldenLayout-Methoden `RegisterComponentConstructor()` und `RegisterFactoryFunction()`. In der alten GoldenLayout-Version muss die Methode `RegisterComponent()` verwendet werden. Diese Methoden können von einer GoldenLayout-Instanz aufgerufen werden.

Der Methode ‚`RegisterComponentFactoryFunction()`' kann verwendet werden, um eine Component mithilfe einer Callback-Funktion zu registrieren. In diesem Fall wird der ‚`RegisterComponentFactroyFunction()`' eine anonyme Funktion als Parameter übergeben. Dabei werden dem Container der zu registrierenden Component HTML-Elemente hinzugefügt. Ein String wird verwendet, um zu bestimmen welche Component registriert werden soll.

Code-Beispiel:

```
this.goldenLayout.registerComponentFactoryFunction("PanelTest",
(container, state) => {
    let newDiv = document.createElement("div");
    newDiv.style.height = "100%";
    newDiv.style.width = "100%";
    newDiv.style.color = "white";
    newDiv.innerHTML = "<h2>Test</h2>";
    container.element.appendChild(newDiv);
});
```

[56] GoldenLayout, „GoldenLayout-Readme".

Die GoldenLayout-Methode ‚RegistierComponentConstructor()' wird ebenfalls ein String als Parameter übergeben. Mit diesem wird auch hier die zu registrierende Component bestimmt. Anders als bei der GoldenLayout-Methode ‚RegisterComponentFactroyFunction(,) wird hier keine Callback-Funktion verwendet. Die Registrierung der Components erfolgt mithilfe eines Konstruktors. Dafür wird eine Klasse erstellt. Der Konstruktor dieser Klasse wird anstelle einer Callback-Funktion verwendet.

Code-Beispiel:

```
export class PanelTest {
    constructor(_container?: ComponentContainer,
    state?: JsonValue | undefined) {
        this.mainDiv = document.createElement("div")
        this.mainDiv.style.height = "100%";
        this.mainDiv.style.width = "100%";
        this.mainDiv.style.color = "white";
        this.mainDiv.innerHTML = "<h2>" + state "</h2>";
        container!.element.appendChild(this.mainDiv);
    }
}
```

6.5.2 ComponentState

Jede Component beinhaltet neben einem Container auch einen ComponentState. Zur Bestimmung von ComponenStates können die Konfigurationsvariablen für Items, Components oder Layouts verwendet werden. Siehe Codebeispiel der LayoutConfig. ComponentStates haben den GoldenLayout-Typ ‚JsonValue'. Zulässige Typen für GoldenLayout-JsonValues sind String, Number, Boolean, Null, Json, Object und JsonValueArray. Die beiden GoldenLayout-Component-Registrierungsmethoden können auf die ComponentStates der von ihnen registrierten Components zugreifen.

6.5.3 ContentItems und ComponentItems

Außerhalb einer GoldenLayout-Registriermethode werden alle Layout-Bestandteile als ContenItem typisiert. Jedes ContentItem-Objekt beinhaltet ein ContentItem-Array als Attribut. Das erste Item einer GoldenLayout-Instanz ist das RootItem. Eine Getter-Methode ermöglicht den Zugriff auf die ContenItem-Arrays der ContentItem-Objekte. Somit entsteht ein verschachtelter Aufbau der ContentItems.

Code-Beispiel:

```
this.goldenLayoyut.rootItem.contentItems[0].contentItems[1].cont
enItems[2] //
```

Der Container einer Component ermöglicht ebenfalls den Zugriff auf den ComponentState der Component. Dies ist mithilfe einer Getter-Methode des ComponentContainers möglich. ComponentContainer sind Attribute von ComponentItems. Nicht jedes ContentItem ist eine Component. ContentItems können auch Rows, Column oder Stacks sein. Ist ein ContenItem eine Component, dann ist es möglich dieses ContenItem in den Typ ComponentItem zu casten. Das bedeutet, dass eine Datentypkonvertierung eines ContenItems in ein ComponentItem möglich ist, wenn sichergestellt wird, dass das zu konvertierende ContentItem eine Component ist. Der Typ eines ContentItems kann mithilfe des ContentItem-Getters mit dem Namen ‚Type' oder der ContentItem-Attribute isComponent, isRow, isColumn und isStack ermittelt werden.

Code-Beispiel:

```
let myComponent: ComponentItem =
this.goldenLayoyut.rootItem.contentItems[0].contentItems[1] as
ComponentItem;

//Zugriff auf ComponentState
console.log(myComponent.container.initialState);
```

6.6 Dynamisches Hinzufügen von Items und Components

Das Layout einer GoldenLayout-Instanz kann auch dynamisch aufgebaut werden. Das bedeutet, dass die von GoldenLayout geladene LayoutConfig-Variable nicht oder nur teilweise zur Anordnung von Components verwendet wird. Im FUDGE-Editor werden bei allen GoldenLayout-Instanzen die Content-Arrays der LayoutConfig-Variablen leergelassen. Die Components dieser GoldenLayout-Instanzen werden mithilfe von Item-Konfigurationen dem Layout hinzugefügt.

6.6.1 ItemConfigs

ItemConfigs werden zur Erzeugung von Items verwendet. Sie ähneln dem Typ ‚LayoutConfig', jedoch beinhalten sie kein Root-Attribut. In der alten GoldenLayout-Version gibt es nur den Datentyp ‚ItemConfig'. Die neue GoldenLayout-Version führt die neuen Datentypen ‚RowOrColumnItemConfig', ‚ComponentItemConfig' und ‚StackItemConfig' ein.

Beispiel aus dem FUDGE-Editor:

```
const hierachyAndComponents: RowOrColumnItemConfig = {
        type: "column",
        isClosable: true,
        content: [
          {
            type: "component",
            componentType: VIEW.HIERARCHY,
            componentState: _state
            title: "Hierachy"

          },
          {
            type: "component",
            componentType: VIEW.COMPONENTS,
            componentState: _state, // Konstruktor-Parameter
            title: "Components"
          }
        ]
      }
```

ComponentItemConfig ist der einfachste Typ einer Item-Konfiguration. ComponentItemConfigs beschreiben Components und beinhalten keine Rows, Columns oder Stacks. StackItemConfigs können zum Hinzufügen von Stacks verwendet werden. RowOrColumnItemConfigs können zum Hinzufügen von Rows und Columns verwendet werden.

Items werden mit der alten GoldenLayout-Version an eine bestimmte Stelle eines Content-Arrays hinzugefügt. Dies erfolgt mit der ContentItem-Methode `addChild()`. Dieser Methode wird als Parameter eine Itemkonfiguration vom GoldenLayout-Typ ItemConfig übergeben.

Mit der neuen GoldenLayout-Version verändert sich das Hinzufügen von Items und Components. Die ContentItem-Methode addChild() ist noch enthalten, aber die Entwickler von GoldenLayout empfehlen die neuen Layoutmanager-Methoden addItem(), addItemAtLocation(), addComponen()t, addComponentAtLocation(), newItem(), newComponent(), newItemAtLocation() und new ComponentAtLocation()

Im Rahmen dieser Arbeit ist erfragt worden, wie diese Methoden verwendet werden können. Die neuen Methoden die zum Hinzufügen von Items bzw. Components verwendet werden können basieren auf der LayoutManager-Methode addItemAtLocation().[57]

[57] GoldenLayout, „Dynamically adding components and items, (rows, columns, stacks).“ Github, zuletzt geprüft am 30.08.2021, https://github.com/golden-layout/golden-layout/issues/692.

6.6.2 AddItemAtLocation()

Im FUDGE-Editor wird nun ausschließlich die LayoutManager-Methode addItemAtLocation verwendet, da diese die Verwendung von LocationSelectors ermöglicht. LocationSelctors erlauben eine eingeschränkte Kontrolle darüber, wo Items bzw. Components im Layout platziert werden sollen. Die Verwendung von LocationSelectros ist nicht optional. Methoden die Items bzw. Components hinzufügen, nutzen entweder ein Standard-LocationSelector-Array oder ermöglichen es ein selbstdefiniertes Array von LocationSelectors zu übergeben. In allen Fällen wird der erste gültige Eintrag des verwendeten Arrays zur Platzierung von Items und Components verwendet[58]. Folgende Selektoren können gewählt werden:

- FocusedItem
- FocusedStack
- FirstStack
- FirstRowOrColumn
- FirstRow
- FirstColumn
- Empty
- Root

Code-Beispiel eines LocationSelector-Arrays:

```
let selector: LayoutManager.LocationSelector[] = [

{typeId:LayoutManager.LocationSelector.TypeId.FirstRowOrColumn,
index: undefined },

{typeId:LayoutManager.LocationSelector.TypeId.Root,index:
undefined },

];
```

[58] GoldenLayout, „Dynamically adding components and items, (rows, columns, stacks)“.

Jedes ContentItem beinhaltet ein Attribut des GoldenLayout-Typs LayoutManager. Dieses Attribut ist ein Objekt, welches die oben genannten Methoden, die zum Hinzufügen von Items und Components verwendet werden, beinhaltet. Die LocationSelectors der Methoden beziehen sich bei ContentItems auf das Root-Item. Somit kann nicht genau bestimmt werden, an welche Stelle des Layouts ein Item oder eine Component hinzugefügt wird. Dazu ist erforderlich ein ContenItem in den GoldenLayout-Typ RowOrColumn zu casten. Diese beinhalten ebenfalls Methoden, die das Hinzufügen von Items und Components ermöglichen. Laut den GoldenLayout-Entwicklern können auf diese Weise Components und Items an bestimmte Rows oder Columns hinzugefügt werden[59]. Dieser Ansatz ist erfolgreich in einer Test-Anwendung getestet worden. Die Test-Anwendung befindet sich im Anhang.

Der FUDGE-Editor erzeugt das Layout mithilfe der ContentItem-Layoutmanager-Methoden.

Code-Beispiel aus dem FUDGE-Editor:

```
this.goldenLayout.rootItem.layoutManager.addItemAtLocation(panel
Config, [{ typeId: LayoutManager.LocationSelector.TypeId.Root
}]);
```

Im FUDGE-Editor werden auf diese Weise Views den Panels und Panels der Page hinzugefügt. Das ist nur dann möglich, wenn alle hinzuzufügenden Components bereits registriert sind.
Die LayoutManager-Methode ‚addItem()' verwendet das Default-LocationSelector-Array Ein selbstdefiniertes LocationSelctors-Array kann nicht als Parameter übergeben werden. RowOrColumn-Objekten beinhalten auch eine addItem()-Methode.

[59] GoldenLayout, „Dynamically adding components and items, (rows, columns, stacks)".

6.6.3 addComponent() und addComponentAtLocation()

Mit der LayoutManager-Methode addComponen() können Components mithilfe von ComponentItemConfigs dem Layout hinzugefügt werden. Diese Components sind nicht Inhalte von Content-Arrays, die von Stacks, Columns oder Rows erzeugt werden. Eine so hinzugefügte Component erzeugt immer einen Stack. Dieser Stack wird mithilfe der LocationSelectors einem andern Stack hinzugefügt. Das ist laut Aussagen der Entwickler ein Bug[60]. Somit werden zwei Registerkarten nebeneinander angezeigt. Der Inhalt kann durch Klicken auf die entsprechende Registerkarte gewechselt werden. Dieses Verhalten kann nicht geändert werden. Auf diese Weise ist es nicht möglich Components innerhalb eines Columns oder einer Row zu erzeugen.

addComponent() istf auch Bestandteil von RowOrColumn- bzw. Stack-Objekten. Die LayoutManager-Methoden addComponentAtLocation() und addComponent() werden im FUDGE-Editor nicht verwendet.

[60] GoldenLayout, „Github Issue 686 - Cannot remove first tab added.“ Zuletzt geprüft am 30.08.2021, https://github.com/golden-layout/golden-layout/issues/686.

7 Änderungen am Quellcode des FUDGE-Editors

Der FUDGE-Editor ist so überarbeitet worden, dass die neue GoldenLayout-Version verwendet werden kann. Dazu wurden Konfigurationsvariablen angepasst oder neue hinzugefügt. Das dynamische Hinzufügen von Items ist mit den neuen GoldenLayout-Methoden umgesetzt worden.

Der FUDGE-EDITOR erzeugt auch mit der neuen GoldenLayout Version drei GoldenLayout-Instanzen. Somit kann sichergestellt werden, dass Views nur innerhalb ihres zugehörigen Panels verschoben werden können. Panels sind weiterhin GoldenLayout-Instanzen die als ‚Row' angeordnet werden. Die äußere GoldenLayout-Instanz registriert die Components ‚PanelGraph' und ‚PanelProjekt' mithilfe ihrer RegisterComponentConstructor-Methode. Dadurch werden die Konstruktoren der PanelGraph- und PanelProjekt-Klasse aufgerufen. Diese erzeugen jeweils eine eigene GoldenLayout-Instanz und registrieren die zu ihnen gehörenden Views. Innerhalb der Konstruktoren der Panels werden Konstanten des neuen GoldenLayout-Typs ‚RowOrColumnItemConfig' erzeugt. Die Root-Items der Panels rufen jeweils die neue LayoutManager-Methode addItemAtLocation() auf. Mithilfe der Konstanten und einem LocationSelectors-Array werden Items dem Layout hinzugefügt. Diese Items ordnen die Views bzw. Components entsprechend ihrer Konfiguration an. Views verwenden die bereits bestehende Methoden, zum Abruf und zur Veränderung ihrer Inhalte.

Die Panels sind als Components der äußeren GoldenLayout-Instanz registriert. Somit ist es möglich, diese mithilfe von Konstanten des neuen GoldenLayout-Typs ‚RowOrColumnItemConfig' dem Layout der äußeren Instanz hinzuzufügen. Die statische Methode add() der Page-Klasse ermöglicht dies. Electron-Menüeinträge lösen bei ihrer Verwendung Electron-Events aus. Event-Listener werden verwendet, um bei Auslösung dieser Events die entsprechenden Panels mithilfe der Klassenmethode add() hinzuzufügen. Derzeit werden die Events beim Start des FUDGE-Editors direkt ausgeführt.

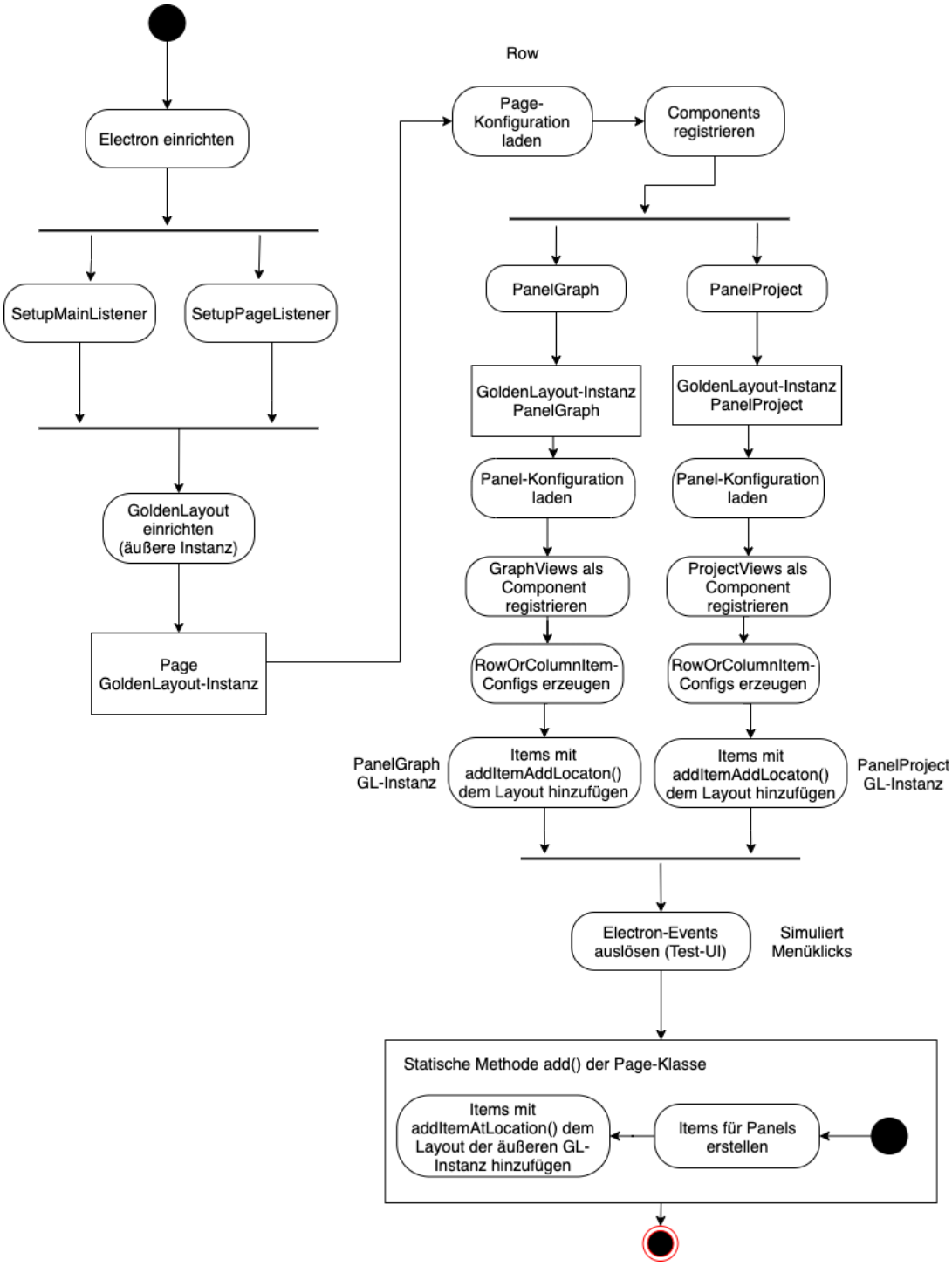

Abbildung 6: Aktivitätsdiagramm „Erstellen des FUDGE-Editor-Layouts“

Wenn Panels nicht auf diese Weise hinzugefügt werden, dann kann die äußere GoldenLayout-Instanz die inneren Instanzen nicht kontrollieren. Die Panels werden in diesem Fall nicht angezeigt. Mithilfe von CSS können die Panels und ihre Views sichtbar gemacht werden. Dazu kann ein GoldenLayout-Resize-Event verwendet werden. Das GoldenLayout-Resize-Event wird ausgeführt, wenn das Browserfenster vergrößert oder verkleinert wird. Ein Eventlistener führt bei Auslösung dieses Events eine Funktion bzw. Programmcode aus. Die Größe des LayoutManagers wird in diesem Fall auf die Größe des Panel-Containers angepasst. Die Views des Panels werden somit angezeigt. Dies ist nicht im FUDGE-Editor umgesetzt worden. Die Erprobung dieses Ansatzes erfolgte in einer Test-Anwendung. Diese befindet sich im Anhang dieser Arbeit.

Code-Beispiel:

```
_container?.on("resize", () => {
    let width = this.goldenLayout.container.clientWidth;
    let height = this.goldenLayout.container.clientHeight;
    this.goldenLayout.setSize(width, height);
})
```

7.1 Weitere Veränderungen im Programmcode des FUDGE-Editors

GoldenLayout nutzt ein eigenes Eventsystem. Die Namen der Events sind mit der neuen Version verändert worden. Der FUDG-Editor ist entsprechend angepasst worden. Das Event mit dem Namen componentCreatet kann mit der neuen GoldenLayout-Version nicht mehr verwendet werden. Die Panel-Methode addViewComponent() wird bei auslösen dieses Events ausgeführt. Das GoldenLayout-Event ItemCreated wird nun verwendet. Dieses wird bei der Erstellung von Items und Components ausgelöst. Das alte Event wird nur bei der Erstellung von Components ausgelöst. Der Programmcode der Methode ist entsprechend angepasst worden.

ContentItems sind in der neuen GoldenLayout-Version nicht nur Components. Die Objektmethode addViewComponent() der Panel-Klasse ist aufgrund dieser Änderung angepasst worden. Die Methode addViewComponent fügt dem Attribut Views der Panel-Klasse eine View hinzu. Das Attribut Views ist ein Array. Mit einer If-Anweisung ist sichergestellt worden, dass nur Components dem Attribut Views hinzugefügt werden.

```
private addViewComponent = (_event: EventEmitter.BubblingEvent):
void => {

    let target: ComponentItem = _event.target as ComponentItem;

    if (target instanceof Page.goldenLayoutModule.ComponentItem)
     {

      this.views.push(<View>target.component);

    }

}
```

7.1.1 ComponentState

Wie bereits beschrieben beinhalten Components jeweils einen ComponentState. Der FUDGE-Editor hat dies genutzt, um einen FUDGE-Node mit einem Weltkoordinatensystem als ComponentState des PanelGraphs festzulegen. Dies ist genutzt worden, um bereits beim Start des FUDGE-Editors ein Weltkoordinatensystem anzuzeigen.

Der ComponentState wird mit der statischen Methode add() der Page-Klasse festgelegt. Wird diese statische Methode ausgeführt, wenn die neue GoldenLayout-Version verwendet wird, dann führt dies zu einem Fehler, wenn ein Objekt des Typs FUDGE-Node als ComponentState festgelegt wird. Die Fehlermeldung lautet je nach Laufzeitumgebung „Maximum call stack size exceeded“ oder „ too much recursion“. Das bedeutet, dass zu viele Funktionen aufgerufen werden. Dieser Fehler konnte außerhalb des FUDGE-Editors nicht reproduziert werden.

Das Problem ist umgangen worden, in dem bei add()-Aufrufen, die ein PanelGraph hinzufügen , Null‘ als ComponentState festgelegt worden ist. Das hat zur Folge, dass beim Start des Editors kein Weltkoordinatensystem-Node dem Graphen zugewiesen wird. Die Funktionalität des Editors wird dadurch nicht eingeschränkt. Nodes können weiterhin im PanelProject erzeugt und mithilfe von Ziehen und Ablegen dem Graphen zugewiesen werden.

7.2 CSS

Damit GoldenLayout 2 verwendet werden kann, müssen in der HTML-Datei des FUDGE-Editors mehrere CSS-Dateien geladen werden. Die Einbindung der CSS-Datei goldenlayout-base.css ist erforderlich. Darüber hinaus ist ein Theme als CSS-Datei eingebunden worden. Ein ‚Theme‘ verändert die Farben des Layouts. Hintergründe und Kontrollelemente können somit eingefärbt werden.

Der FUDGE-Editor hat ein eigenes ‚Theme‘ eingebunden. Dieses hat das Layout in Grautönen eingefärbt. Dieses ‚Theme‘ kann mit GoldenLayout 2 nicht mehr verwendet werden. Die HTML-Datei lädt deshalb CSS-Dateien der alten und neuen GoldenLayout-Version. Die ursprünglich vom FUDGE-Editor geladene CSS-Datei Base.css wird so überarbeitet, dass sie nicht mehr alle HTML-Elemente einfärbt. Es werden nur noch der GoldenLayout-Hintergrund und die Hintergründe Components eingefärbt. Somit erfolgt die Einfärbung teilweise mit der Original-Dark-Theme-Datei von GoldenLayout 2 und teilweise mit der selbstdefinierten CSS-Datei Base.css.

8 Fazit

Im Rahmen dieser Arbeit ist der Layout-Manager des FUDGE-Editors aktualisiert worden. Die neue GoldenLayout-Version hat Änderungen am Quellcodes des FUDGE-Editors erfordert. Neue GoldenLayout-Methoden und GoldenLayout-Typen sind verwendet worden. Außerdem wurde die HTML-Datei des FUDGE-Editors so überarbeitet, dass sie zusätzlich die neuen CSS-Dateien von GoldenLayout einbindet.

Dem ComponentState des PanelGraphs konnte kein Objekt des FUDGE-Typs Node zugewiesen werden. Das Problem konnte außerhalb von FUGE nicht reproduziert werden.

Da es für GoldenLayout 2 keine Dokumentation gibt, sind den Entwicklern Detailfragen gestellt worden. Darüber hinaus sind Test-Projekte erstellt worden. Mit ihnen sind die neuen Methoden erprobt worden. Die Test-Projekte befinden sich im Anhang.

Das ESM-Modul von GoldenLayout 2 konnte nicht verwendet werden, da ESM-Importe die Funktionalität von Typescript-Namensräumen einschränken. Die Recherche über Module und Bundling ist deshalb für diese Arbeit von Bedeutung. Darüber hinaus werden in dieser Arbeit Alternativen zu Namensräumen vorgestellt.

Panels, die in früheren FUDGE-Versionen verwendet worden sind, konnten nicht wiederhergestellt werden. Die Animation-Views sind nicht wieder eingesetzt und weitere Views nicht erstellt worden.

Literaturverzeichnis

Ackermann, Philip und Leo Leowald. *Schrödinger programmiert Java: Das etwas andere Fachbuch; [mit Syntax-Highlighting!; von den Sprachgrundlagen über Multithreading bis zur komplexen GUI-Anwendung; nutze die Schwerter aller Versionen: Generics, New File I/O und Java 8; ideal zum Durchblicken und Hand anlegen, fantastisch illustriert*. 1. Aufl. Schrödinger programmiert. Bonn: Galileo Press, 2014.

Braun, Herbert. „Chrome und Chromium: Was sind eigentlich die Unterschiede?“ *heise online*, 07.12.2018. Zuletzt geprüft am 30.08.2021. https://www.heise.de/newsticker/meldung/Chrome-und-Chromium-Was-sind-eigentlich-die-Unterschiede-4245456.html.

Brien, Jörn. „Ranking: Microsoft-Browser Edge verdrängt Firefox von Rang 2.“ *t3n Magazin*, 06.10.2020. Zuletzt geprüft am 30.08.2021. https://t3n.de/news/ranking-browser-edge-chrome-1326536/.

Cherny, Boris und Jørgen W. Lang. *Programmieren in TypeScript: Skalierbare JavaScript-Applikationen entwickeln*. 1. Auflage. Heidelberg: O'Reilly, 2020.

Cyren, Tierney. „An Absolute Beginner's Guide to Using npm.“ The NodeSource Blog. Zuletzt geprüft am 30.08.2021.

Dell'Oro-Friedl, Jirka. „GitHub Wiki - JirkaDellOro/FUDGE: Furtwangen University Didactic Game Editor.“ Zuletzt geprüft am 30.08.2021. https://github.com/JirkaDellOro/FUDGE/wiki.

GoldenLayout. „Add single file bundle by martin31821 · Pull Request #665 · golden-layout/golden-layout.“ Github. Zuletzt geprüft am 30.08.2021. https://github.com/golden-layout/golden-layout/pull/665.

———„Dynamically adding components and items, (rows, columns, stacks).“ Github. Zuletzt geprüft am 30.08.2021. https://github.com/golden-layout/golden-layout/issues/692.

———„Github Issue 686 - Cannot remove first tab added.“ Zuletzt geprüft am 30.08.2021. https://github.com/golden-layout/golden-layout/issues/686.

———„GoldenLayout-Readme.“ GoldenLayout. Zuletzt geprüft am 30.08.2021. https://github.com/golden-layout/golden-layout/blob/master/README.md.

Griffiths, Dawn und David Griffiths. *Head first Android development*. Second edition. Beijing, Boston, Farnham, Sebastopol, Tokyo: O'Reilly, 2017.

Hayani, Said. „JavaScript ES6 — write less, do more.“ *freeCodeCamp.org*, 20.04.2018. Zuletzt geprüft am 30.08.2021. https://www.freecodecamp.org/news/write-less-do-more-with-javascript-es6-5fd4a8e50ee2/.

„Mesh, 3D.“ In *Encyclopedia of Multimedia*, 406–7. Springer, Boston, MA, 2006. doi:10.1007/0-387-30038-4_126.

Microsoft. „TypeScript: Documentation - Namespaces and Modules.“ Zuletzt geprüft am 30.08.2021. https://www.typescriptlang.org/docs/handbook/namespaces-and-modules.html.

———„TypeScript: Handbook - Enums.“ Zuletzt geprüft am 30.08.2021. https://www.typescriptlang.org/docs/handbook/enums.html#numeric-enums.

———„Documentation - Classes.“ Zuletzt geprüft am 30.08.2021. https://www.typescriptlang.org/docs/handbook/2/classes.html.

———„The TypeScript Handbook.“ Microsoft. Zuletzt geprüft am 30.08.2021. https://www.typescriptlang.org/docs/handbook/intro.html.

Mozilla Foundation. „WebGL - Web API Referenz | MDN.“ Zuletzt geprüft am 30.08.2021. https://developer.mozilla.org/de/docs/Web/API/WebGL_API.

———„JavaScript modules - JavaScript | MDN.“ Zuletzt geprüft am 13.08.2021. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules.

———„Statische Methoden - JavaScript | MDN.“ Zuletzt geprüft am 30.08.2021. https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Classes/static.

OpenJS Foundation. „About Node.js.“ OpenJS Foundation. Zuletzt geprüft am 30.08.2021. https://nodejs.org/en/about/.

———„Electron Documentation - ipcMain.“ OpenJS Foundation. Zuletzt geprüft am 30.08.2021. https://www.electronjs.org/docs/api/ipc-main.

———„Electron Documentation - ipcRenderer.“ OpenJS Foundation. Zuletzt geprüft am 30.08.2021. https://www.electronjs.org/docs/api/ipc-renderer.

———„Electron Documentation - Remote.“ OpenJS Foundation. Zuletzt geprüft am 30.08.2021. https://www.electronjs.org/docs/api/remote.

———„Quick Start | Electron.“ OpenJS Foundation. Zuletzt geprüft am 30.08.2021. https://www.electronjs.org/docs/tutorial/quick-start.

Palantir. „TSLint in 2019 - Palantir Blog.“ *Palantir Blog*, 19.02.2019. Zuletzt geprüft am 30.08.2021. https://blog.palantir.com/tslint-in-2019-1a144c2317a9.

RequireJS. „CommonJS Notes.“ Zuletzt geprüft am 30.08.2021. https://requirejs.org/docs/commonjs.html.

———„How to get started with RequireJS.“ Zuletzt geprüft am 30.08.2021. https://requirejs.org/docs/start.html.

Singh, Ashish N. „An intro to Webpack: what it is and how to use it.“ *freeCodeCamp.org*, 15.01.2019. Zuletzt geprüft am 13.08.2021. https://www.freecodecamp.org/news/an-intro-to-webpack-what-it-is-and-how-to-use-it-8304ecdc3c60/.

Skypack. „Introduction: Skypack is a JavaScript Delivery Network for modern web apps.“ Zuletzt geprüft am 30.08.2021. https://docs.skypack.dev/.

Syed, Basarat A. „Barrel.“ Zuletzt geprüft am 30.08.2021. https://basarat.gitbook.io/typescript/main-1/barrel.

Trombino, Marco. „You might not need jQuery: A 2018 Performance case study.“ Zuletzt geprüft am 13.08.2021. https://medium.com/@trombino.marco/you-might-not-need-jquery-a-2018-performance-case-study-aa6531d0b0c3.

Unity Technologies. „Unity - Manual: Meshes, Materials, Shaders and Textures.“ Zuletzt geprüft am 30.08.2021. https://docs.unity3d.com/2020.2/Documentation/Manual/Shaders.html.

WebpackJS. „TypeScript and Webpack.“ Zuletzt geprüft am 30.08.2021. https://webpack.js.org/guides/typescript/.

Anhang auf USB-Stick

- FUDGE-Editor mit GoldenLayout 2
- Thesis als PDF
- Test-Anwendung mit Skypack
- Test-Anwendung mit Webpack
- Test-Anwendung über Barrel-Files
- Test-Anwendung über das Hinzufügen von Items an eine Column
- Test-Anwendung mit importiertem UMD-Modul von GoldenLayout

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Thesis selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel, sind angegeben.

Furtwangen, den 31.08.2021

Unterschrift: