

Fakultät 4 – Elektrotechnik und Informatik

**Labor: Medieninformatik 1**

**Projekt: interaktive Webanwendung**

Survival-Adventure: „Survive A Month“

**Verfasser:**  Philipp Moritzer, 50002000

Hannes Lesemann, 50002000

Pascal Seegers, 50002000

**Modul:**  Medieninformatik 1

**Dozent:** Paelke

**Labor:** Lochwitz

**Abgabedatum:** 28.01.2019

Inhalt

[1 Spielidee 1](#_Toc534805208)

[2 Handlung 1](#_Toc534805209)

[2.1 Story 1](#_Toc534805210)

[2.2 Interaktion 1](#_Toc534805211)

[3 Grafische Gestaltung 1](#_Toc534805212)

[3.1 Layout 1](#_Toc534805213)

[3.2 Schriften 1](#_Toc534805214)

[3.3 Icons 1](#_Toc534805215)

[3.4 Gestaltungsmerkmale 1](#_Toc534805216)

[4 Verwendete Technologie 1](#_Toc534805217)

[4.1 Bibliotheken 1](#_Toc534805218)

[4.2 Programmiersprache 1](#_Toc534805219)

[5 Architektur 1](#_Toc534805220)

[5.1 Aufbau des Projektes 1](#_Toc534805221)

[5.2 Objektorientierung 1](#_Toc534805222)

[5.3 Kopmonentenbasierte Architektur 1](#_Toc534805223)

[5.4 JSON-Daten 1](#_Toc534805224)

[5.5 Dateien 1](#_Toc534805225)

[5.6 Design Pattern 1](#_Toc534805226)

[6 Projektplanung und Arbeitsteilung 1](#_Toc534805227)

[6.1 Zeitlicher Ablauf 1](#_Toc534805228)

[6.2 Arbeitsteilung 1](#_Toc534805229)

[7 Post Mortem 1](#_Toc534805230)

[7.1 Was hast gut funktioniert 1](#_Toc534805231)

[7.2 Was hat nicht funktioniert 1](#_Toc534805232)

[7.3 Was würden wir anders machen 1](#_Toc534805233)

[7.4 Ausbaustufen des Projekts 1](#_Toc534805234)

[Abbildung 1: Projektstruktur II](#_Toc535070751)

# 1 Spielidee

# 2 Handlung

## Story

## 2.2 Interaktion

# 3 Grafische Gestaltung

## 3.1 Layout

Holy-Grail Layout

Responsive!!!!

## Schriften

Arcade-Classic, Zorque, Arial/Helvetica

## Icons

Flaticons / comic-style

## 3.4 Gestaltungsmerkmale

# Verwendete Technologie

## 4.1 Verwendete Sprachen

Für dieses Projekt werden die drei gängigen Sprachen in der Webentwicklung verwendet: HTML, JavaScript und CSS. Die Sprachen wurden in ihrer Grundform genutzt und es werden keine Abwandlungen, wie TypeScript oder SASS verwendet. Dies hat den Vorteil, einer einsteigerfreundlichen Umgebung, sowie eine allgemeinere Leserlichkeit des Source-Codes.

## 4.2 Bibliotheken

Als einzige Bibliothek wurde jQuery verwendet. Diese Bilbiothek vereinfacht den Umgang mit Selektoren in JavaScript vereinfacht Funktionen zur DOM-Manipulation. Alles was mit jQuery gemacht wird, kann auch in normalen JavaScript programmiert werden.

In diesem Projekt wurde statt dem üblichen „$“-Selector, das Wort jQuery immer ausgeschrieben. Grund hierfür ist, dass jQuery nicht die einzige Bibliothek ist, die vom „$“-Zeichen gebraucht macht.

jQuery("#actionConfirmationCost").text(currentAction.cost);

jQuery("#actionConfirmationActionName").text(currentAction.name);

Abbildung : jQuery-Selektor

So wird Problemen bei Abhängigkeiten, die gegebenenfalls zu einem späteren Zeitpunkt dazu kommen vorgebeugt.

## 4.3 Versionsverwaltung

Zur Versionsverwaltung wurde die git-Technologie verwendet. Hier wurde ein Repository auf GitHub (vgl. <https://www.github.com/> ) erstellt. Der Vorteil ist, dass parallel an dem Projekt gearbeitet werden kann, die zentral abgespeichert werden.

# 5 Architektur

## 5.1 Aufbau des Projektes

Das Projekt ist übersichtlich und erweiterbar aufgebaut. die verschiedenen Dateien sind nach verschiedenen Funktionen getrennt und liegen in nach Funktionen sinnvoll benannten Ordnern. Im Folgenden wird die Projektstruktur (s. Abbildung 2: Projektstruktur) beschrieben.

## 5.2 index.html als Einstiegspunkt

Das Projekt ist als „Single Page Application“ (SPA – vergleiche **TODO**). Alle Inhalte des Projektes werden dynamisch geladen und demnach werden nicht nacheinander HTML-Seiten direkt aufgerufen, sondern in die vorhandene Seite eingespeist. Der Einstiegspunkt des Projektes ist die index.html, die im Root-Folder des Projektes liegt. Sie dient als Hülle für den Content der dynamisch in diese Hülle hereingeladen wird.

<div id="main" oncontextmenu="return false;">

<!--Main div where dynamic content is displayed-->

</div>

Es befindet sich ein HTML-Tag mit der id „browserNotSupported“ auf der Hauptseite. Dieser Text wird angezeigt, wenn der Browser nicht unterstützt wird (Internet Explorer oder Edge). Die Logik hierfür befindet sich im index.js, welches gleichzeitig den Startpunkt für das Spiel bildet. Hier wird der erste Content bei Seitenaufruf in die „Hülle“ geladen. Zum Laden der jeweiligen Elemente wird die Methode jQuery().load() verwendet. Diese erlaubt durch Angabe des Pfades einer HTML-Datei den jeweiligen Code einzuspeisen.

Die index.html-Datei referenziert außerdem die styles.css. In dieser Datei werden globale Styles definiert, die für die Seite und gewisse Unterelemente gelten sollen. So wird beispielsweise für html, body festgelegt, dass kein Rand angezeigt werden soll, die Standardschriftart immer Arial ist und dass die Seite ohne Scrollbar auskommen soll.

Des Weiteren werden wichtige Abhängigkeiten geladen. Im Head-Tag der index.html-Datei wird die Bibliothek „jQuery“ geladen, so wie die scriptloader.js-Datei, die alle zunächst wichtigen Abhängigkeiten lädt, so dass diese für den Spielstart initialisiert sind.

## 5.3 Seiten und Komponenten

### 5.3.1 Seiten

Der dynamische Inhalt, der in den Main-Container eingespeist wird beschränkt sich auf Seiten. Eine Seite besteht jeweils aus einer HTML-, einer CSS- und einer JavaScript-Datei.

### 5.3.2 Komponenten

Komponenten bestehen ähnlich wie Seiten auch aus einer HTML-, einer CSS- und einer JavaScript-Datei. Komponenten sind hierbei allerdings nur Unterelemente einer Seite. Eine Seite muss demnach Container für die Komponenten vorsehen und mit der jQuery.load()-Methode laden. Der Vorteil dieser Komponenten ist, dass der Code strukturierter und übersichtlicher wird. Jede Komponente kann gezielt die Sachen ausführen, für die sie gedacht ist und ist für ihren eigenen Status verantwortlich. In dieser Applikation wird zwischen Komponenten unterschieden, die mehrfach verwendbar sind oder nur ein Status behalten müssen.

Komponenten, die mehrfach verwendbar sind, sind als Klassen aufgebaut. Beispielsweise die Resource-Bar-Komponente. Die Komponenten benötigen im Konstruktor eine eindeutige id. Durch die Methode makeElementsUnique() werden mithilfe der jQuery().attr()-Funktion die Ids Elemente der jeweiligen HTML-Datei, die zu der Komponente gehört einzigartig gemacht und sind innerhalb dieser Klasse mit „identifier“ und this.id anzusprechen. Beim Aufruf der jeweiligen Komponente muss dann die init()-Methode im Callback der jQuery().load()-Methode aufgerufen werden. In der init-Methode werden weiterhin Methoden aufgerufen, um die Styles und Attribute von den jeweiligen Elementen zu setzen. Komponentenbasierte Architekturen sind ein moderner Ansatz und fördern die Wiederverwendbarkeit von Code. Beliebte Frameworks/Bibliotheken wie Angular, React oder Vue arbeiten ebenfalls mit einer komponentenbasierten Architektur.

## 5.4 Zentrales „State-Management“

## 5.5 Rohe Daten

## 5.6 Models

## 5.7 Helfer

# 6 Projektplanung und Arbeitsteilung

## 6.1 Zeitlicher Ablauf

## 6.2 Arbeitsteilung

- Versionsverwaltung?

# 7 Post Mortem

## 7.1 Was hast gut funktioniert

//Aufteilung

//Architektur

Sowas halt

## 7.2 Was hat nicht funktioniert

//Probleme Asynchronität

## 7.3 Was würden wir anders machen

## 7.4 Ausbaustufen des Projekts

//Besten-Liste

//Anti-Cheat (Server-Client-Komponente)

//Besten-Liste

//

## 

Anhang

# i. Projektstruktur

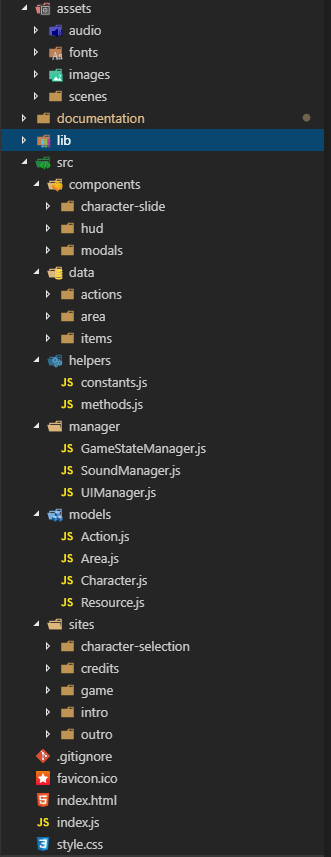


Abbildung : Projektstruktur

/\*\* defining global styles \*\*/

html,

body {

height: 100%;

margin: 0;

padding: 0;

font-family: "Helvetica", "Arial", sans-serif;

overflow: hidden;

user-select: none;

}

#main {

height: 100%;

background: url("/assets/scenes/intro.jpg") no-repeat center center fixed;

background-size: cover;

}

#mainBrowserInfo {

display: none;

}

.modal {

position: absolute;

width: 100%;

height: 100%;

background-color: rgba(0, 0, 0, 0.7);

top: 50%;

left: 50%;

z-index: 800;

transform: translate(-50%, -50%);

display: none;

}

@font-face {

font-family: "ArcadeClassic";

src: url("/assets/fonts/arcadeclassic.ttf");

}

@font-face {

font-family: "zorque";

src: url("/assets/fonts/zorque.ttf");

}

@media only screen and (max-width: 991px) {

#main {

zoom: 2;

}

}

Abbildung : styles.css

$(window).on("load", function() {

var gst = GameStateManager.getInstance();

if (!gst.loaded) {

jQuery("#main").load("./sites/intro/intro.html", () => {});

gst.setLoaded();

gst.initGameData();

}

});

Abbildung : Ausschnitt index.js

/\*\*

\* gets needed scripts.

\*/

const getScripts = () => {

jQuery.getScript("./models/Resource.js");

jQuery.getScript("./models/Character.js");

jQuery.getScript("./models/Action.js");

jQuery.getScript("./helpers/methods.js");

jQuery.getScript("./helpers/constants.js");

jQuery.getScript("./manager/SoundManager.js");

jQuery.getScript("./manager/UIManager.js");

jQuery.getScript(

"./components/modals/decision-tree-modal/decision-tree-modal-item/decision-tree-modal-item.js"

);

jQuery.getScript(

"./components/modals/tutorial-modal/tutorial-element/tutorial-element.js"

);

jQuery.getScript("./components/hud/resource-bar/resource-bar.js");

};

getScripts();

Abbildung : scriptloader.js

class ResourceBar {

constructor(id, type, color1, color2, color3, image, progress) {

this.id = id;

this.type = type;

this.color1 = color1;

this.color2 = color2;

this.color3 = color3;

this.image = image;

this.progress = progress;

}

makeElementUnique() {

jQuery("#resourceBar").attr(

"id",

jQuery("#resourceBar").attr("id") + this.id

);

jQuery("#resourceIcon").attr(

"id",

jQuery("#resourceIcon").attr("id") + this.id

);

jQuery("#numbersBar").attr(

"id",

jQuery("#numbersBar").attr("id") + this.id

);

jQuery("#progress").attr("id", jQuery("#progress").attr("id") + this.id);

jQuery("#resourceBarText").attr(

"id",

jQuery("#resourceBarText").attr("id") + this.id

);

}

setAttributes() {

const IMAGE\_PATH = "./../assets/images/";

jQuery("#resourceBar" + this.id).css({ "background-color": this.color1 });

jQuery("#numbersBar" + this.id).css({

"background-image":

"linear-gradient(" + this.color2 + "," + this.color3 + ")"

});

jQuery("#resourceIcon" + this.id).attr("src", IMAGE\_PATH + this.image);

jQuery("#resourceBarText" + this.id).text(this.progress + "%");

jQuery("#progress" + this.id).animate(

{ width: this.progress + "%" },

"slow"

);

jQuery("#progress" + this.id).css({

"background-color": this.color1

});

}

updateState(value) {

jQuery("#progress" + this.id).css({ "box-shadow": "inset 0 0 15px white" });

jQuery("#progress" + this.id).animate(

{ width: value + "%" },

"slow",

() => {

jQuery("#progress" + this.id).css({

"box-shadow": "inset 0 0 15px black"

});

}

);

jQuery("#resourceBarText" + this.id).text(value + "%");

}

init() {

this.makeElementUnique();

this.setAttributes();

this.updateState(this.progress);

}

}

Abbildung : Resource-Bar-Komponente

const loadResourceBars = (resourceBar, identifier, htmlpage) => {

jQuery(document).ready(() => {

jQuery(identifier).innerHTML = jQuery(identifier).load(htmlpage, () => {

resourceBar.init();

});

});

};

const resource\_bar1 = new ResourceBar(

0,

RESOURCE\_BARS[0].type,

RESOURCE\_BARS[0].color1,

RESOURCE\_BARS[0].color2,

RESOURCE\_BARS[0].color3,

RESOURCE\_BARS[0].image,

gst.character.hunger.value

);

const resource\_bar2 = new ResourceBar(

1,

RESOURCE\_BARS[1].type,

RESOURCE\_BARS[1].color1,

RESOURCE\_BARS[1].color2,

RESOURCE\_BARS[1].color3,

RESOURCE\_BARS[1].image,

gst.character.life.value

);

const resource\_bar3 = new ResourceBar(

2,

RESOURCE\_BARS[2].type,

RESOURCE\_BARS[2].color1,

RESOURCE\_BARS[2].color2,

RESOURCE\_BARS[2].color3,

RESOURCE\_BARS[2].image,

gst.character.learn.value

);

activeResourceBars.push(resource\_bar1);

activeResourceBars.push(resource\_bar2);

activeResourceBars.push(resource\_bar3);

loadResourceBars(resource\_bar1, "#res1", RESOURCE\_BAR\_PAGE);

loadResourceBars(resource\_bar2, "#res2", RESOURCE\_BAR\_PAGE);

loadResourceBars(resource\_bar3, "#res3", RESOURCE\_BAR\_PAGE);

Abbildung : Initialisierung Resource-Bar Komponente