

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:** Prof. Dr. Volker Paelke

**Labor:** Dipl.-Inf. Andreas Lochwitz

**Abgabedatum:** 28.01.2019

Inhalt

[1 Spielidee 1](#_Toc535161782)

[2 Handlung 1](#_Toc535161783)

[2.1 Story 1](#_Toc535161784)

[2.2 Interaktion 1](#_Toc535161785)

[3 Grafische Gestaltung 1](#_Toc535161786)

[3.1 Layout 1](#_Toc535161787)

[3.2 Schriften 1](#_Toc535161788)

[3.3 Icons 1](#_Toc535161789)

[3.4 Gestaltungsmerkmale 1](#_Toc535161790)

[4 Verwendete Technologie 1](#_Toc535161791)

[4.1 Verwendete Sprachen 1](#_Toc535161792)

[4.2 Bibliotheken 1](#_Toc535161793)

[4.3 Editor und Tools 2](#_Toc535161794)

[4.4 Versionsverwaltung 2](#_Toc535161795)

[5 Architektur 2](#_Toc535161796)

[5.1 Aufbau des Projektes 2](#_Toc535161797)

[5.2 index.html als Einstiegspunkt 2](#_Toc535161798)

[5.3 Seiten und Komponenten 3](#_Toc535161799)

[5.3.1 Seiten 3](#_Toc535161800)

[5.3.2 Komponenten 3](#_Toc535161801)

[5.4 Zentrales „State-Management“ 3](#_Toc535161802)

[5.4.1 GameStateManager 3](#_Toc535161803)

[5.4.2 UIManager 4](#_Toc535161804)

[5.4.3 SoundManager 4](#_Toc535161805)

[5.5 Rohe Daten 4](#_Toc535161806)

[5.6 Models 5](#_Toc535161807)

[5.7 Helfer 5](#_Toc535161808)

[6 Projektplanung und Arbeitsteilung 5](#_Toc535161809)

[6.1 Zeitlicher Ablauf 5](#_Toc535161810)

[6.2 Arbeitsteilung 5](#_Toc535161811)

[7 Post Mortem 5](#_Toc535161812)

[7.1 Was hast gut funktioniert 5](#_Toc535161813)

[7.2 Was hat nicht funktioniert 5](#_Toc535161814)

[7.3 Was würden wir anders machen 5](#_Toc535161815)

[7.4 Ausbaustufen des Projekts 6](#_Toc535161816)

[i. Projektstruktur II](#_Toc535161817)

**Abbildungsverzeichnis**

[Abbildung 1: jQuery-Selektor 1](#_Toc535162843)

[Abbildung 2: Einzigartige ID für wiederverwendbare Komponenten am Beispiel der Resource-Bar 3](#_Toc535162844)

[Abbildung 3: Beispiel für ein Action-Objekt in JSON 4](#_Toc535162845)

[Abbildung 4: Relation zwischen JSON-Objekten 4](#_Toc535162846)

[Abbildung 5: Projektstruktur II](#_Toc535162847)

[Abbildung 6: styles.css III](#_Toc535162848)

[Abbildung 7: Ausschnitt index.js IV](#_Toc535162849)

[Abbildung 8: scriptloader.js IV](#_Toc535162850)

[Abbildung 9: Resource-Bar-Komponente VI](#_Toc535162851)

[Abbildung 10: Initialisierung Resource-Bar Komponente VII](#_Toc535162852)

[Abbildung 11: Singleton GameStateManager VIII](#_Toc535162853)

[Abbildung 12: UIManager updateMoney-Methode VIII](#_Toc535162854)

[Abbildung 13: playCashSound()-Methode IX](#_Toc535162855)

[Abbildung 14: hunger-actions.json – Ausschnitt IX](#_Toc535162856)

[Abbildung 15: initGameData()-Methode X](#_Toc535162857)

[Abbildung 16: selectCharacter()-Methode XI](#_Toc535162858)

# 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

## Gestaltungsmerkmale

# 4 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. JavaScript wird hauptsächlich in der ES6-Syntax verwendet, was auch der Grund ist, dass der Internet-Explorer-Browser nicht unterstützt wird.

## 4.2 Bibliotheken

Als einzige Bibliothek wurde jQuery verwendet. Diese Bilbiothek 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 1: jQuery-Selektor

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

## 4.3 Editor und Tools

Als Editor wurde Visual Studio Code mit folgenden Plugins verwendet:

* Live Server
* Debugger for Chrome

Hauptsächlich wurde die Applikation auf Google Chrome getestet, von Zeit zu Zeit auf Mozilla Firefox oder Safari (iOS).

## 4.4 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. Die Quelltext-Dateien liegen im Ordner „/src/“, die Assets (Bilder, Audio, Fonts) liegen im Ordner „/assets/“, die verwendeten Bibliotheken unter „/lib/“ und die Dokumentation unter „/documentation/“.

## 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. Es gibt vier Seiten, die jeweils nacheinander

### 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.



Abbildung 2: Einzigartige ID für wiederverwendbare Komponenten am Beispiel der Resource-Bar

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“

### 5.4.1 GameStateManager

Für das Regeln des Status des Games wurde eine zentrale Datei mit dem Namen „GameStateManager.js“ angelegt. Diese beinhaltet eine Klasse, die als Singleton angelegt ist, um sicherzustellen, dass es nicht mehr als eine Instanz dieser Klasse zur Laufzeit der Applikation gibt. Die Instanz der Klasse wird mit dem Laden des Intro-Bildschirms geladen und der variable gst zugeordnet. Von überall kann nun auf die Variable gst zugegriffen werden um Methoden aufzurufen, die das

### 5.4.2 UIManager

Im Gegensatz zum GameStateManager ist der UIManager keine eigene Klasse sondern bietet nur Methoden, um den Status des User-Interfaces zu aktualisieren. Ist eine Aktion ausgeführt worden, kann der GameStateManager auf die Methoden des UIManagers zugreifen und die Änderungen am Status des Spiels sichtbar machen. Ändert sich beispielsweise der Kontostand im Status des Spiels, wird die Mehode const updateMoneyUI = newValue => {}; aus dem Manager aufgerufen.

### 5.4.3 SoundManager

Der SoundManager ist das Pendant zum UI-Manager, nur besitzt dieser Methoden zum Abspielen von Sounds. Beispielsweise kann die Methode playCashSound() verwendet werden, um ein „Kassen“-Sound abzuspielen, wenn Geld abgezogen wird. Außerdem kümmert sich der Manager um das abspielen der Hintergrundmusik und die Methoden legen die Lautstärke, welches Audio-File abgespielt werden soll und die Repetition der jeweiligen Musik fest.

## 5.5 Rohe Daten

Um einheitliche, erweiterbare und strukturierte Daten für die Aktionen, die Belohnungen und Orte zu definieren, die dynamisch in das Spiel geladen werden wurden JSON-Strukturen angelegt. Über diese Dateien kann man nun neue Aktionen, Orte oder Belohnungen hinzufügen, ohne den Programmcode zu ändern. Die Logik für die jeweilgen Objekte ist bereits definiert. Für die Aktionen wurden für jeden Aktionstyp pro Charakter drei JSON-Dateien angelegt und drei JSON-Dateien die charakterübergreifend sind. Des Weiteren wurden JSON-Dateien für die Belohnungen (items.json) und für die Orte (areas.json) angelegt. Die json-Datei enthält jeweils ein Array, dass Objekte vom Typen der Aktion enthält. In dem Action-Objekt kann nun der Name, die Beschreibung, das Bild, der Typ, der Wert, die Kosten, die Belohnung und der Ort angepasst werden.

{

"name": "Der Hühnchenmann aus Kentucky",

"desc": "Eimer mit heißen Flügeln",

"img": "kfc.png",

"type": "hunger",

"value": 32,

"cost": 500,

"reward": 3,

"area": 1

},

Abbildung 3: Beispiel für ein Action-Objekt in JSON

Die Attribute „area“ und „reward“ referenzieren hier Ids von den Dateien „areas.json“ und „items.json“. Die ID referenziert ein anderes Objekt und kann so dem Objekt zugeordnet werden. Die Aktion mit dem Namen „Der Hünchenmann aus Kentucky“ kann nur in dem Ort „Foodcourt“ stattfinden und bei erfüllen der Aktion bekommt man das Item mit der Id 3.

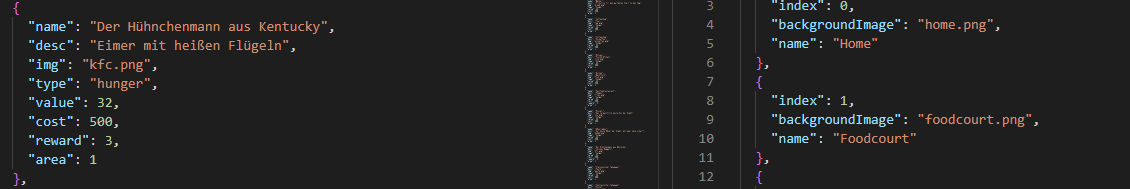


Abbildung 4: Relation zwischen JSON-Objekten

Die Logik dieser Verknüpfung ist im GameStateManager geregelt.

Die JSON Dateien werden beim Laden des Spiels asynchron geladen und im GameStateManager-Objekt gespeichert. Die Methode initGameData() gibt einen Promise-Objekt zurück, welches sicherstellt, dass die Dateien geladen sind, wenn auf sie zugegriffen werden muss. Erst dann geht das Spiel weiter.

## 5.6 Models

Für festgelegte Objekte wurde keine JSON-Struktur erstellt sondern Model-Klassen vorgesehen. Dies ist für die Charaktere und deren Ressourcen der Fall. Die Charaktere werden in der selectCharacter()- Methode initialisiert und enthalten alle Eigenschaften die für den Charakter und das Spiel notwendig sind (Name, Bild Geld, Ressourcen). Die Ressourcen sind ein eigenes Objekt, die den Wert, den Verlust pro Tag und einen Faktor enthalten, der bestimmt, wie schnell eine Ressource ansteigt.

## 5.7 Helfer

Es werden zwei Helfer Dateien verwendet, die thematisch den anderen Komponenten der Applikation nicht zugeordnet werden können und allgemeingültige Funktionen enthalten. So ist methods.js eine Datei die eine Funktion enthält um eine Fließkommazahl zu runden und eine Funktion, die einzigartige IDs erstellt.

Die andere Datei ist die constants.js-Datei, die konstante Werte enthält. Darunter zählen die Start-Ressourcen, die Aktionen pro Ort, die Anzahl der Züge pro Tag und verschiedene Links und Konfigurationen.

# 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

Two-Way-Data-Binding

## 7.4 Ausbaustufen des Projekts

//Besten-Liste

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

//Besten-Liste

//

## 

Anhang

# i. Projektstruktur

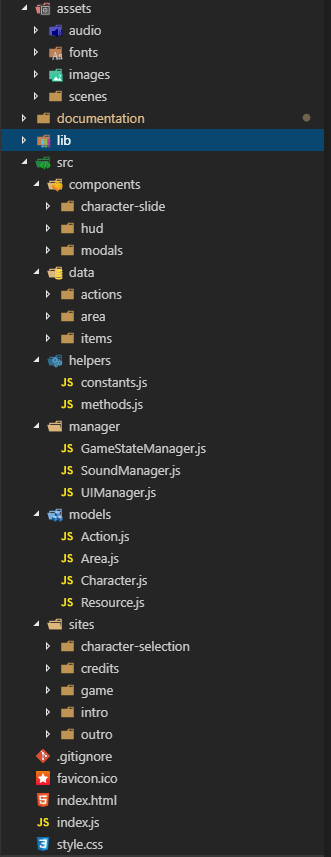


Abbildung 5: Projektstruktur

/\*\* 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 6: 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 7: 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 8: 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 9: 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 10: Initialisierung Resource-Bar Komponente

class GameStateManager {

constructor() {

if (this.instance) {

throw new Error(

"Constructor call is private, please use GameStateManager.getInstance()"

);

}

}

static getInstance() {

if (!this.instance) {

this.instance = new GameStateManager();

}

return this.instance;

}

}

Abbildung 11: Singleton GameStateManager

/\*\*

\* updates the money

\*/

const updateMoneyUI = newValue => {

playCashSound();

jQuery(".gameMoneyText").animate(

{

opacity: "0"

},

100

);

jQuery("#gameMoney").text(newValue);

jQuery("#gameMoney").animate(

{

opacity: "1"

},

1000

);

};

Abbildung 12: UIManager updateMoney-Methode

/\*\*

\* call this method to play a cash sound

\* (used at updating money)

\*/

const playCashSound = () => {

var soundAudio;

soundAudio = new Audio("../assets/audio/cash-register.mp3");

soundAudio.loop = false;

soundAudio.volume = 0.3;

const playPromise = soundAudio.play();

if (playPromise !== null) {

playPromise.catch(() => {

soundAudio.play();

});

}

};

Abbildung 13: playCashSound()-Methode

{

"name": "XXL Burgerhaus- Die mit dem großen M",

"desc": "Ein Menü bestehend aus Burger, Pommes und einem Getränk",

"img": "burger.png",

"type": "hunger",

"value": 34,

"cost": 500,

"reward": 3

},

{

"name": "Der Hühnchenmann aus Kentucky",

"desc": "Eimer mit heißen Flügeln",

"img": "kfc.png",

"type": "hunger",

"value": 32,

"cost": 500,

"reward": 3,

"area": 1

},

{

"name": "Bäcker",

"desc": "Frühstück für den perfekten Start in den Tag",

"img": "bread.png",

"type": "hunger",

"value": 32,

"cost": 500,

"area": 1

},

Abbildung 14: hunger-actions.json – Ausschnitt

/\*\*

\* inits the game data stored as json in data-folder

\* asynchronous request -> returns a promise

\*/

initGameData() {

let initPromise = new Promise((resolve, reject) => {

let items = [];

jQuery.getJSON("./data/items/items.json", data => {

items = data;

GameStateManager.getInstance().items = items;

});

let hungerActions = [];

jQuery.getJSON("./data/actions/all/hunger-actions.json", data => {

hungerActions = data;

GameStateManager.getInstance().hungerActions = hungerActions;

});

let lifeActions = [];

jQuery.getJSON("./data/actions/all/life-actions.json", data => {

lifeActions = data;

GameStateManager.getInstance().lifeActions = lifeActions;

});

let learnActions = [];

jQuery.getJSON("./data/actions/all/learn-actions.json", data => {

learnActions = data;

GameStateManager.getInstance().learnActions = learnActions;

});

const actions = [hungerActions, lifeActions, learnActions];

let areas = [];

jQuery.getJSON("./data/area/areas.json", data => {

areas = data;

GameStateManager.getInstance().areas = areas;

resolve();

});

});

this.initPromise = initPromise;

}

Abbildung 15: initGameData()-Methode

/\*\*

\* Call this Method from CharacterSlide model

\* The character gets injected when clicked the character's slide

\* CharacterSlide is a ViewHelper Object.

\* Selects the character

\* @param {Slide} slide

\*/

const selectCharacter = id => {

let name = NAME\_INDEX\_PAIR[id].name;

let startingValues = STARTING\_VALUES[id];

const character = new Character(

id,

name,

NAME\_INDEX\_PAIR[id].portrait,

startingValues[0],

startingValues[1],

startingValues[2],

startingValues[3]

);

jQuery("#characterSelectionGameStartButton").attr("disabled", false);

GameStateManager.getInstance().setCharacter(character);

};

Abbildung 16: selectCharacter()-Methode