HSB
Hochschule Bremen
City University of Applied Sciences

Fakultät 4 – Elektrotechnik und Informatik

Medieninformatik 1

Projekt: interaktive Webanwendung

Survival-Adventure: „Survive A Month“

Verfasser:	Philipp Moritzer, 5034255 Hannes Lesemann, 5017055 Pascal Seegers, 5051429
Modul:	Medieninformatik 1
Dozent:	Prof. Dr. Volker Paelke
Labor:	Dipl.-Inf. Andreas Lochwitz
Abgabedatum:	28.01.2019

Inhalt

Abbildungsverzeichnis

1 Spielidee

Die Idee des Spiels erklärt sich wie folgt. Es wurde versucht ein Thema zu finden, das für die Autoren eine persönliche Bedeutung hat. So sind sie auf die Idee gekommen, ein Spiel zu entwickeln, das sich um den Alltag verschiedener Studenten mit unterschiedlichen Prioritäten dreht. So lässt sich das allgemeingültige Ziel aller Studenten, nämlich eine erfolgreiche und zufriedenstellende Studienzeit zu haben, in diesem Fall als Ziel den Monat zu überstehen portraitiert, mit den individuellen Bedürfnissen einzelner Studenten kombinieren. Um dies zu ermöglichen, wurden drei Charaktere entworfen, die für die Autoren eine ideale Mischung an Herausforderungen während der Studienzeit bereithalten.

Wie bereits erwähnt, ist das Ziel des Spiels, es mit drei möglichen Aktionen pro Tag, die im Abschnitt 2.2 Interaktion weiter erläutert werden, den Monat, also 30 Tage zu überstehen, indem jeder der drei Balken, die für uns die Grundbedürfnisse eines Durchschnittsstudenten darstellen, durchgehend über null zu halten. Somit soll gewährleistet werden, dass der Spieler kein Bedürfnis zu kurz kommen lässt. Hierfür wird zunächst Zeit benötigt, die personenbezogenen Bedürfnisse des jeweiligen Charakters genauer herauszufinden. Anschließend, wird ein gutes Einschätzungsvermögen verlangt, um alle drei Balken über null zuhalten, was durch eine vierte, finanziell geprägte Komponente deutlich erschwert wird.

2 Handlung

2.1 Story

In dem Spiel geht es um den allgemeinen Studentenalltag. Hierfür werden drei Charaktere entworfen, die in ihrem Leben mit unterschiedlichen Problemen umgehen müssen. So hat Justus, der nach dem Klischee eines BWL-Studenten entworfen ist, eher ein Problem damit glücklich zu werden, was sich anhand eines geringeren Zuwachses des Lifestyle-Wertes bemerkbar macht. Dafür hat er allerdings ein größeres Kapital zur Verfügung, was ihm einen größeren Handlungsspielraum ermöglicht. Außerdem wurden ihm vereinzelt personenbezogene Events zugefügt, die es ihm vereinfachen den Monat zu überstehen. Lisa hingegen ist dem Klischee einer Studentin, die vor dem Studium im Ausland war, nachempfunden, was sich daran widerspiegelt, dass die Herausforderung des Lebens für sie zwar vereinfacht sind, da sich die Balken schneller füllen, jedoch sind ihre finanziellen Mittel äußerst begrenzt, weshalb von Anfang an die Prioritäten erkannt werden müssen. Der letzte Charakter, Sören, ist ein eher ausgewogener Charakter, mit einer Einschränkung. Er konzentriert sich scheinbar zu stark darauf im Leben Spaß zu haben, und vernachlässigt es sich weiterzubilden, was sich im Spiel an einem langsam füllenden Lernbalken bemerkbar macht.

Somit hat jeder Charakter seine eigenen Probleme, die der User im Verlaufe des Spiels bemerken wird.

2.2 Interaktion

Um das Spiel zu starten, wird die Eingabe einer beliebigen Taste benötigt. Anschließend kann der Spieler zwischen den drei Charakteren Justus, Lisa und Sören wählen, wobei ihm offenbart wird, mit welchen Startwerten für Lifestyle, Hunger, Lernstand und auch mit welchem Budget er das Spiel beginnen wird. Nun muss er einen der Spieler wählen und unten rechts auf “Start Game” klicken um das Spiel zu beginnen. Als nächster Schritt nach der Charakterauswahl landet der Spieler beim Homebildschirm, der in diesem Fall dem Wohnort des Charakters entspricht. Hier öffnet sich zunächst einmal das Tutorial, wo auf jede Interaktionsmöglichkeit eingegangen wird. Das Tutorial ist weiterhin

jederzeit über das Zahnradicon am linken Rand zu erreichen. Hier befinden sich zudem weitere Icons, wie die Entscheidungshistorie, der Rucksack sowie der Navigator. Zudem gibt es hier die Möglichkeit, eine Beschreibung der Funktion der einzelnen Icons zu bekommen, indem man mit der Maus über das Icon fährt. Parallel zum Wohnort des Charakters gibt es vier weitere Eventlocations, die über den Navigator erreicht werden können. Auch hier gibt es einen Hover-Effekt, wobei das Bild des Orts für den Moment des Hoverns kurzzeitig verschwindet und lediglich eine Wortbeschreibung des Orts angezeigt wird. Neben dem gestalterischen Unterschied, variieren die Orte auch in den Aktionen die sich dem Spieler bieten. Während am Wohnort des Spielers, sowie auf dem Sportplatz, sowohl Lifestyle und Lern- als auch Hunger-Aktionen angeboten werden, gibt es beim Foodcourt ausschließlich Hunger-Aktionen, in der Hochschule nur Lern-Aktionen und in der City nur Lifestyle-Aktionen.

An jeder der Eventlocations werden dem Spieler drei Aktionen zur Verfügung gestellt, aus denen er dann wählen kann. Alternativ kann er auch per Navigator die Location wechseln, für den Fall, dass die ihm angebotenen Aktionen nicht reizvoll genug sind. Jede der Aktion hat wiederum einen unterschiedlichen Effekt. Auch hier gibt es einen Hover-Effekt, wobei dem Spieler übermittelt wird, wie teuer die Aktion ist und welcher der Balken beeinflusst wird.

Jeden Tag sind maximal drei Aktionen möglich. Sind diese durchgeführt, kann der Spieler links unter den Icons auf “Tag Ende” klicken und der Tag wird beendet.

Eine weitere Option ist, die Entscheidungshistorie, wo dem Spieler jede einzelne seiner vorherigen Aktionen angezeigt wird. Zudem ist dies die einzige Möglichkeit, die genaue Höhe des jeweiligen Effekts einer Aktion einzusehen.

Neben dem Einfluss auf die Balken, beinhalten manche Aktionen auch Nebeneffekte, die in Form von Items übergeben werden. Jedes dieser Items hat einen Einfluss auf den Zuwachs der jeweiligen Balken- während das Autogramm, das man nach einem Konzertbesuch bekommt, dem Spieler für jede Life-Aktion 10% mehr Punkte einbringt, so bringt z.B. eine Lebensmittelvergiftung 10% weniger Punkte für den Hunger-Balken. Ist der Spieler der Meinung, dass er ein Item nicht mehr benötigt, beziehungsweise es loswerden möchte, so kann er das Item “Verkaufen”. Allerdings bringen nur Items mit positivem Effekt dem Spieler Geld ein, negative Items wie eine Lebensmittelvergiftung müssen kompensiert werden. So kostet z.B. eine Lebensmittelvergiftung den Spieler 150 Euro.

3 Grafische Gestaltung

3.1 Layout und Gestaltungsmerkmale

3.1.1 Intro

Für Screenshots der Intro-Seite siehe im Anhang ii. Screenshots - Intro.

Es wird ein zentriertes Layout verwendet, wobei der Fokus auf den Charakteren liegt. Dieser Effekt wird durch eine einfliegende Animation verstärkt. Nach Abschluss dieser Animation hat jeder der Charakter einen kleinen Bewegungs-Effekt, was das Intro lebendiger wirken lässt. Außerdem verlangt die blinkende Schrift nach Aufmerksamkeit, was in diesem Fall sinnvoll ist, da dies die einzige mögliche Funktion ist. Zudem ist die gesamte Ansicht verkleinert, wenn das Spiel auf mobilen Geräten ausgeführt wird.

3.1.2 Charakter-Selektion

Für Screenshots der Charakter-Selektions-Seite siehe im Anhang iii. Screenshots - Charakterauswahl.

Bei diesem Fenster wird die Aufmerksamkeit auf die Mitte des Bildschirms gelegt, da hier der temporär ausgewählte Charakter liegt. Zudem wird für eine bessere Ersichtlichkeit der zentrale Charakter fokussiert. Das Wechseln der Charaktere ist durch einfaches Klicken auf die anliegende Charakter-Karte ebenfalls übersichtlich gehalten. In der mobilen Ansicht wird hier ebenfalls lediglich ein tippen auf die jeweilige Karte benötigt.

3.1.3 Spiel

Für Screenshots der Game-Seite siehe im Anhang iv. Screenshots – Game.

Bei der Anordnung des tatsächlichen Spiels handelt es sich um ein Holy-Grail Layout. Bei dieser speziellen Art von Layout werden die Hauptinhalte in der Mitte platziert, während die sekundären Spielinhalte am äußeren Bildschirm angeordnet werden. In diesem Fall befinden sich die Aktionen, der Hauptteil des Spiels, in der Mitte, während die Metainformationen, wie der Kontostand, am oberen, die Modal-Dialoge am linken und die Resourcebars am unteren Bildschirmrand platziert werden. Außerdem wurde die Farbe der jeweiligen Resourcebar der Farbe der dazugehörigen Aktion angepasst, damit der Spieler die Aktionen richtig zuordnen kann.

In der mobilen Version werden die Aktionen vertikal statt horizontal aufgelistet, da somit der Platz auf einem Smartphone-Display effizienter genutzt werden kann.

3.1.4 Modal-Dialoge

Für Screenshots von Modal-Dialogen siehe im Anhang v. Screenshots – Tutorial-Modal, viii. Screenshots – Auswahl Location und ix. Screenshots – Aktion durchführen.

Werden die sekundären Aktionen, wie das Tutorial oder der Navigator angewählt, öffnen sich diese in Form von Modal-Windows, beziehungsweise in einer weiteren Ebene über dem Spiel. Hierbei wird der Spiel-Hintergrund leicht ausgegraut, was die Aufmerksamkeit von diesem entzieht und auf das jeweilige Modal lenkt. Jeder dieser Dialoge zwingt den User zu einer Handlung, auch wenn diese lediglich das Schließen des Fensters bedeutet.

3.1.5 Inventar

Für Screenshots des Inventars siehe im Anhang vi. Screenshots – Inventar.

Das Inventar ist in einem Grid System angeordnet. Zudem kommt der Popup an einer Stelle auf dem Bildschirm, wo das allgemeine Spielgeschehen nicht irritiert wird. Außerdem wird somit die Erwartung des Users, dass sich das Fenster an dieser Stelle öffnet, erfüllt. Die Items selbst besitzen einen Hover-Effekt, durch den die wichtigsten Informationen des Items angezeigt werden. Dies ist vor allem für die Entscheidungsfindung, ob man das Item behält oder nicht von Bedeutung.

3.1.6 Button-Effekt

Für Screenshots des Button-Effekts siehe im Anhang vii. Screenshots – Button-Hervorhebung.

Durch einen Bewegungseffekt und eine grafische Hinterlegung wird der Spieler darauf hingewiesen, dass der “Tag Ende” Button nun verfügbar und der Tag somit vorbei ist. Er wird quasi aufgefordert auf diesen Button zu drücken.

3.1.7 Outro

Für Screenshots vom Outro siehe im Anhang x. Screenshots – Outro.

Wie beim Intro werden die Elemente auch beim Outro zentral angeordnet. Somit wird die Aufmerksamkeit auf die Mitte des Bildschirms.

3.2 Schriften

Im Spiel werden vier unterschiedliche Schriftarten verwendet. Diese sind Arcade Classic, Zorque, Arial und Helvetica. Während Arcade Classic einen gewissen nostalgischen Effekt mit sich bringt, sorgt Zorque für einen eher futuristischen Effekt. Beide Fonts haben keinen funktionalen Nutzen, jedoch entsteht durch die Kombination beider Schriften ein Kontrast zwischen einem modernen und einem Retro-Design. Arial und Helvetica hingegen als Standardschrift erfüllen den Nutzen, dass sie besonders leserliche Schriften sind.

3.3 Icons

Die Icons sollen dem Spiel einen einheitlichen Comic-Grafikstil geben und zudem die Texte besser visualisieren. Außerdem ermöglichen sie eine intuitive Bedienung, wenn sie in Form von Buttons auftreten.

3.4 Musik

Um das Spiel authentischer wirken zu lassen, haben einzelne Funktionen einen Klick-Sound. So gibt es bei jedem Bezahlen oder Verkaufen ein Kassengeräusch und beim Öffnen des Rucksacks einen Zip-Sound. An jedem neuen Tag wird eine Musik abgespielt, die einen Sonnenaufgang impliziert. Außerdem besitzt jede Area eine unterschiedliche Hintergrundmusik, die auf die jeweilige Location abgestimmt ist. Je nach Ausgang des Spiels, ertönt eine fröhliche Siegesmusik oder eine traurige Musik.

4 Verwendete Technologien

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. jQuery macht viele Dinge, wie bspw. Animationen, Einspeisung von Markup oder Änderungen von Attributen ein Stück weit leichter und übersichtlicher.
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 werden 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 Safari (iOS) oder dem Standard Android Browser. Es ist kein Support für Internet Explorer oder Microsoft Edge vorgesehen, dementsprechend wurden diese Browser auch nicht zum Testing verwendet.

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, und der Source-Code in der aktuellsten Version zentral verfügbar ist. Des Weiteren können bei einer Entwicklung in die falsche Richtung oder einem unlösbar scheinenden Bug alte Entwicklungsstände zurückgeholt werden.

4.5 Konventionen

Es wurden folgende Naming-Konventionen vor dem Beginn des Projekts besprochen:

CSS-Klassen: Camel-Case, **Dateinamen** (inkl. Bilder und Audio, mit Ausnahme von Klassen): kleingeschrieben, mehrere Wörter mit Bindestich verbunden, **Dateien die nur Klassen sind** und keine Komponenten sind: Camel-Case mit großem Anfangsbuchstaben, **Variablen und Methoden**: Camel-Case.

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 25: 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[1]) umgesetzt. Alle Inhalte des Projektes werden dynamisch geladen und demnach werden nicht nacheinander HTML-Seiten direkt aufgerufen, sondern in die vorhandene Seite eingespeist.

[1] Vgl. https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/ (Stand: 21.01.2019)

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>
```

Abbildung 2: index.html "main"-container

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 in der index.js-Datei (s. Abbildung 27: Ausschnitt index.js), welche 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 (s. Abbildung 26: 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 (s. Abbildung 28: scriptloader.js), 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 (Intro, Character-Selection, Game, Outro), die jeweils nacheinander aufgerufen werden und eine Extra-Seite für die Credits. Die Seiten implementieren verschiedene Komponenten und müssen jeweils Container vorsehen um Komponenten (s. 5.3.2 Komponenten) zu laden. Die Logik innerhalb der JavaScript-Datei für die Seite sieht das laden der Komponenten vor und die Einspeisung der jeweils für die Seite relevanten Daten.

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 (s. Anhang Abbildung 29: ResourceBar-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 3: Einzigartige ID für wiederverwendbare Komponenten am Beispiel der Resource-Bar

Beim Aufruf der jeweiligen Komponente muss dann die `init()`-Methode (s. Abbildung 30: Initialisierung ResourceBar-Komponente) 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 (s. Abbildung 31: Singleton GameStateManager) 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 den Status des Spiels steuern. Es wird das Management der Ressourcen, die Aktionen, die Locations, die Tage, die Gewinn-/Verlierbedingung und alles weitere, was die Logik des Spiels angeht, geregelt.

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 `updateMoney()`-Methode (s. Abbildung 32: UIManager updateMoney-Methode) aus dem Manager aufgerufen.

5.4.3 SoundManager

Der SoundManager ist das Pendant zum UIManager, nur besitzt dieser Methoden zum Abspielen von Sounds. Beispielsweise kann die `playCashSound()`-Methode (s. Abbildung 33: playCashSound()-Methode) verwendet werden, um ein „Kassen“-Sound abzuspielen, wenn Geld abgezogen oder hinzugefügt 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 JSON-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 jeweiligen Objekte ist bereits im Programmcode 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 4: 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. Für ein Beispiel anhand der Aktionen pro Area für die Logik, die dies im GameStateManager zuordnet, siehe im Anhang Abbildung 36: Zuordnung Area -> Action.

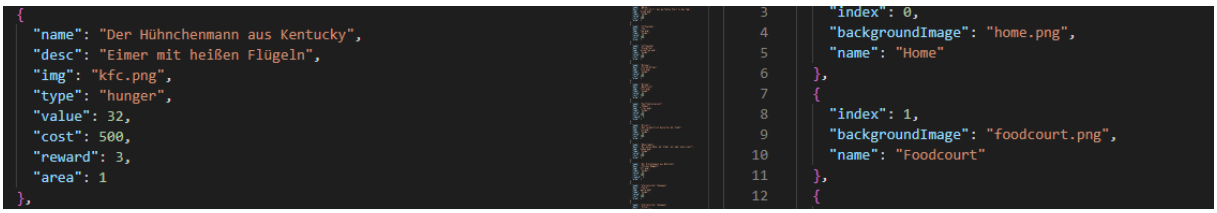

Abbildung 5: Relation zwischen JSON-Objekten

Die Logik dieser Verknüpfung ist im GameStateManager geregelt.

Die JSON-Dateien werden beim Laden des Spiels asynchron geladen und im GameStateManager-Objekt gespeichert. Die `initGameData()`-Methode (s. Abbildung 35: initGameData()-Methode) 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 (s. Abbildung 37: 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, eine Funktion, die einzigartige IDs erstellt und eine Funktion, die eine zufällige Zahl in einem vorgegeben Zahlenbereich zurückgibt.

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, verschiedene Links und andere Konfigurationen.

5.8 Layout-Spezifisches

Für das gesamte Layout wurde durchgehend Flexbox verwendet um Elemente leicht dort anzuordnen, wo man sie haben möchte, sowie ein einfaches Responsive-Design zu erstellen, welches nur an einigen stellen mit Media-Queries angepasst werden musste. Es wurde bewusst auf den Viewport-Tag in der index.html-Datei verzichtet und mit dem CSS-Attribut Zoom für die Einstellung der Größe gearbeitet.

```
@media only screen and (max-width: 991px) {
  #main {
    zoom: 2;
  }}
```

Abbildung 6: Responsive-Design durch zoom-Attribut

6 Projektplanung und Arbeitsteilung

6.1 Zeitlicher Ablauf und Arbeitsteilung

Der zeitliche Ablauf wurde in im Anhang unter i. Projektplanung mithilfe eines Gantt-Diagramms dargestellt. Die Arbeitsteilung erfolgte zunächst nach dem was derjenige sich zutraute. So wurde die Architektur des Programmes an den Entwickler gegeben, der das meiste Vorwissen hatte, allerdings immer in Absprache mit den Gruppenmitgliedern. Nach und nach konnten die Aufgaben besser verteilt werden und in fester Absprache konnten einzelne Teilpakete des Projektes getrennt bearbeitet werden. Am Ende zum „Zusammenfügen“ der Puzzleteile und Testen wurde die Arbeit wieder gemeinsam erledigt. Das Projekt wurde in erfolgreicher Zusammenarbeit abgeschlossen.

7 Post Mortem

7.1 Was hast gut funktioniert

Die Komponentenbasierte Architektur wurde sehr ordentlich durchgeplant und hat letztendlich auch den Rahmen des Projektes getragen. Durch den strukturierten Aufbau war es leicht neue Komponenten hinzuzufügen und die Arbeit aufzuteilen.

Die Versionsverwaltung sorgte ebenfalls für ein gutes teamorientiertes Arbeiten und einen Austausch von Entwicklungsergebnissen, ohne dass man nebeneinandersitzen musste. Dadurch hat die Aufteilung der Aufgaben auch gut funktioniert und die Arbeit gleichmäßig aufgeteilt.

Die Versionsverwaltung sorgte auch einmal dafür, dass Projekt zu „retten“, als die Applikation sich immer in einer Endlosschleife befand, und nicht herausgefunden werden konnte, woran das Problem lag. Hier wurde das Projekt auf den vorherigen Commit zurückgespielt.

7.2 Was hat nicht funktioniert

Durch das dynamische Laden der Daten gab es etwas größere Schwierigkeiten mit der Asynchronität. Es musste sich Wissen über „Promises“ angeeignet werden, welches im Nachhinein schwierig war in das Projekt einzupflegen. So kommt es sehr selten immer noch vor, dass das Projekt nicht startet, da die Daten nicht rechtzeitig geladen werden. Ein einfacher Reload löst das Problem allerdings meistens schon.

Des Weiteren hat die Eigengestaltung von Grafiken nicht funktioniert, da zeitlich wenig Platz und generell viel Misserfolg in der Kreation stattfand. So wurde entschieden auf fertige Icons, Bilder und Musik aus dem Internet zuzugreifen und diese zu verwenden und die Autoren zu kreditieren. Das macht das Spiel nicht ganz „stimmig“ auch wenn auf identischen Stil der Icons geachtet wurde.

Des Weiteren gab es kleinere Probleme mit dem Balancing. Dadurch dass die Aktionen zufällig geladen werden, ist es möglich, dass ein Spiel unschaffbar ist.

7.3 Rückblick

Obwohl die Architektur dieses Projekts gut getragen hat und gut geplant und strukturiert wurde, würde nächstes Mal mehr Wert auf Data-Binding gelegt werden. Jede Änderung muss über den UI-Manager aufgerufen werden. Einen Weg die Oberfläche zu aktualisieren, wenn sich die Daten ändern, würde die Menge an Code deutlich schmälern und das Projekt leichter erweiterbar machen.

Des Weiteren muss beim nächsten Projekt darauf geachtet werden, Probleme von Asynchronität und großen Dateien (Stichworte Lazy-Loading, Pre-Loading, Promises) mit einzuplanen und die Architektur und das Gesamtprodukt stabiler zu machen und zu verbessern.

Außerdem würde, wenn dieses Projekt erneut durchgeführt werden würde, mehr Wert auf Grafik gelegt werden, um eine stimmigere Atmosphäre und ein besser abgestimmtes Produkt zu erstellen.

Um das Balancing zu verbessern, ist es gegebenenfalls sinnvoll den Aktionen eine Häufigkeit zu geben, damit mehr Kontrolle vorhanden ist, die Aktionen und Belohnungen besser ins Gleichgewicht zu bringen und das Spiel auch durch mehr Strategie zu gewinnen.

7.4 Ausblick

Denkbare Erweiterungen des Spiels sind weitere Charaktere, die unterschiedliche Schwierigkeitsstufen beinhalten. Diese Charaktere können wiederum mit weiteren Aktionen, die charakterspezifisch sind erweitert werden.

Außerdem können die Locations und die Items erweitert werden, um mehr Möglichkeiten im Spiel zu haben und das Universum zu erweitern.

Denkbar ist auch, eine neue Art von Ressourcenleiste hinzuzufügen, um die Komplexität des Spiels zu erweitern und interessanter und ggf. schwieriger oder leichter zu machen.

Anhang

i. Projektplanung

Vorgangsname	Anfang	Ende
> Planung	**Di. 06.11.18**	**So. 11.11.18**
Ideenfindung	Di. 06.11.18	Do. 08.11.18
Aufstellung Konzept	Fr. 09.11.18	So. 11.11.18
Meilenstein 1: Abschluss Planung		So. 11.11.18
> Visualisierung	**Mo. 12.11.18**	**So. 18.11.18**
Entwicklung von Wireframes	Mo. 12.11.18	Di. 13.11.18
Game-Design-Dokument	Mi. 14.11.18	Do. 15.11.18
Prototypische Implementierung der Spiele-Navigation	Do. 15.11.18	So. 18.11.18
Meilenstein 2: Abschluss Visualisierung		So. 18.11.18
> Entwurf	**Mo. 19.11.18**	**So. 25.11.18**
Auswahl der Technologie	Mo. 19.11.18	Mo. 19.11.18
Erweiterter Entwurf der Oberfläche	Di. 20.11.18	Mi . 20.11.18
Ausarbeitung der Architektur	Do. 21.11.18	So. 25.11.18
Meilenstein 3: Abschluss Entwurf		So. 25.11.18
> Implementierung	**Mo. 26.11.18**	**Do. 17.01.18**
> Aufbau der Architektur	**Mo. 26.11.18**	**So. 02.12.18**
Implementierung der Ordnerstruktur	Mo. 26.11.18	Di. 27.11.18
Implementierung der Komponentenbauweise	Mi. 28.11.18	So. 02.12.18
> Implementierung der Spiellogik	**Mo. 26.11.18**	**So. 02.12.18**
Implementierung wichtiger funktionen für den "GameState"	Mo. 26.11.18	So. 02.12.18
> Implementierung der Seiten	**Mo. 03.12.18**	**So. 16.12.18**
Intro	Mo. 10.12.18	So. 16.12.18
Character-Selection	Mo. 03.12.18	So. 09.12.18
Spiel	Mo. 03.12.18	So. 16.12.18
Outro	Mo. 03.12.18	So. 09.12.18
Credits	Mo. 10.12.18	So. 16.12.18
> Implementierung verschiedener Komponenten	**Mo. 17.12.18**	**So. 23.12.18**
HUD-Komponenten	Mo. 17.12.18	So. 30.12.18
Modals	Mo. 17.12.18	So. 30.12.18
> Zusammenfügen der Komponenten und "Flick"-Arbeit	**Mo. 31.12.18**	**So. 06.01.19**
> Erstellung der Spieldaten	**Mo. 07.01.19**	**So. 13.01.19**
Areas	Mo. 07.01.19	Do. 10.01.19
Items	Do. 10.01.19	So. 13.01.19
Actions	Mo. 07.01.19	So. 13.01.19
> Recherche für Grafiken und Sounds	**Mo. 14.01.19**	**Do. 17.01.19**
Meilenstein 4: Abschluss Implementierung		Do. 17.01.19
> Test	**Do. 17.01.19**	**So. 20.01.19**
Entwickler-Test	Do. 17.01.19	Fr. 18.01.19
Test durch Komilitonen und Bekannten	Fr. 18.01.19	Sa. 19.01.19
Behebung gefundener Fehler	Sa. 19.01.19	So. 20.01.19
Meilenstein 5: Abschluss Test		So. 20.01.19
> Dokumentation	**Mo. 21.01.19**	**So. 27.01.19**
Erstellung der Projektdokumentation	Mo. 21.01.19	So. 27.01.19
Meilenstein 6: Abschluss Dokumentation		So. 27.01.19

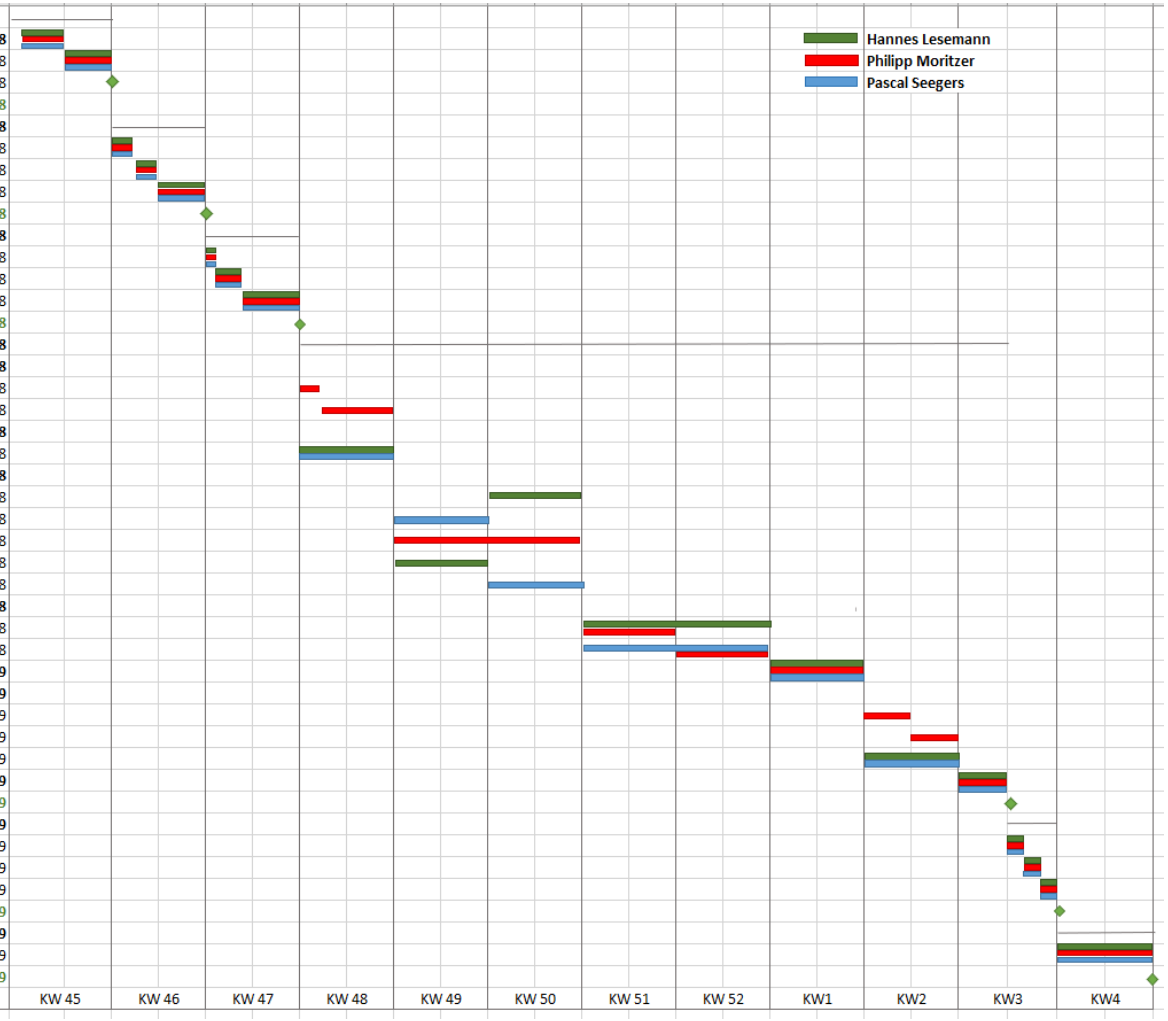

Abbildung 7: GANTT-Diagramm Planung

ii. Screenshots - Intro

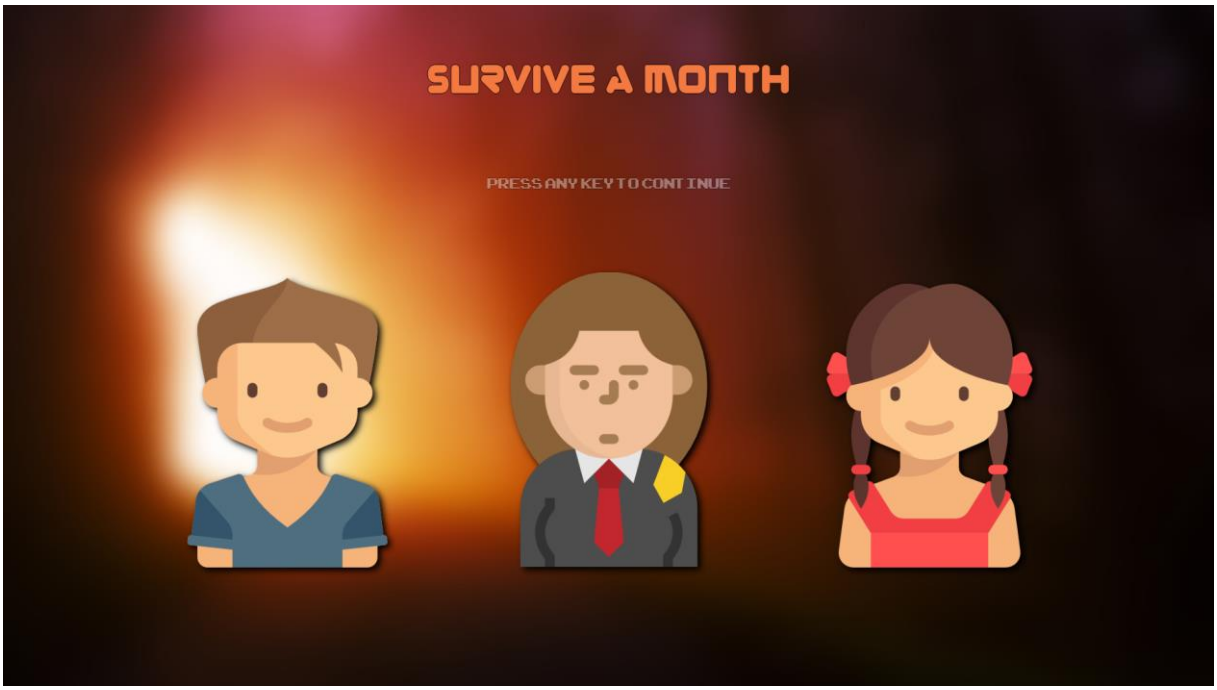

Abbildung 8: Intro

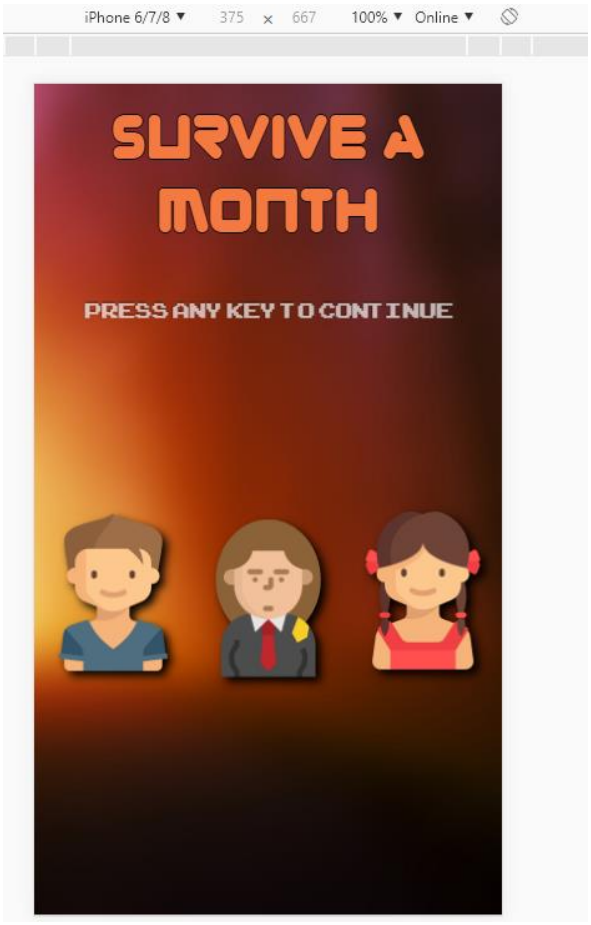

Abbildung 9: Intro Mobile

iii. Screenshots - Charakterauswahl

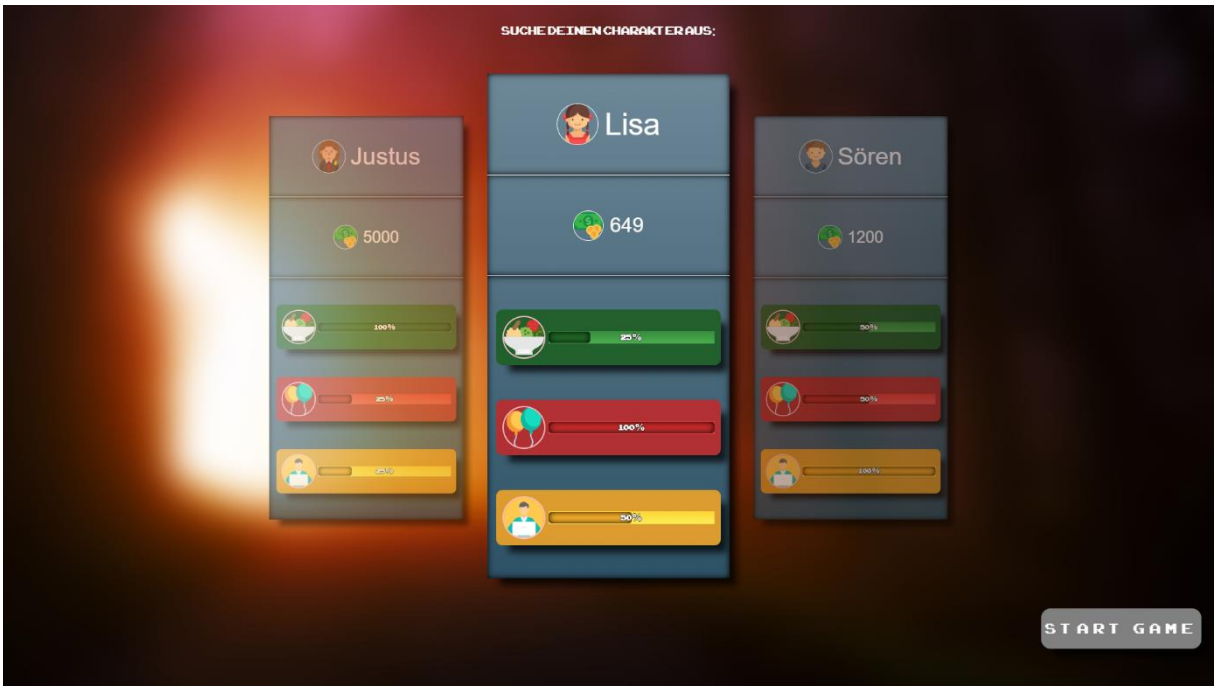

Abbildung 10: Charakterauswahl

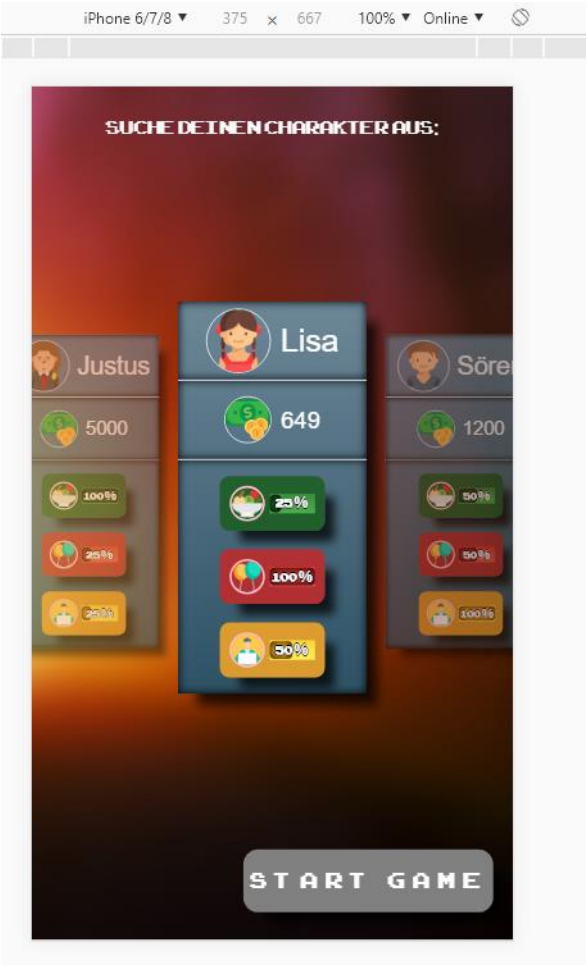

Abbildung 11: Charakterauswahl Mobile

iv. Screenshots – Game

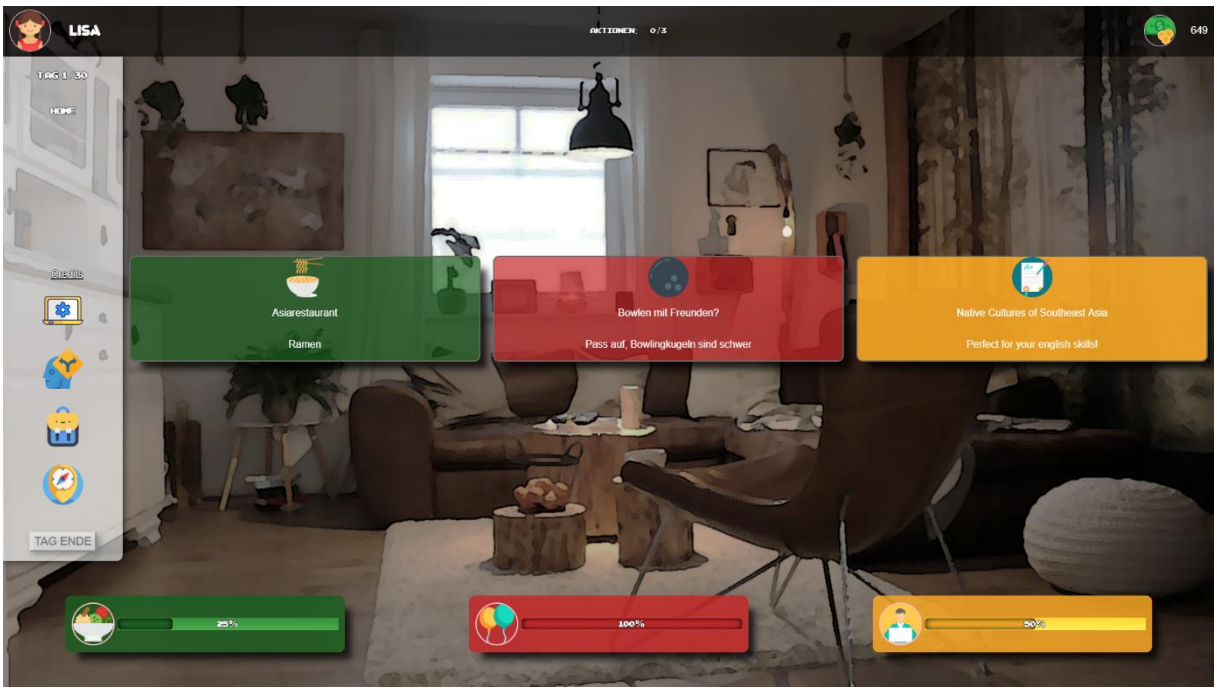

Abbildung 12: Spiel

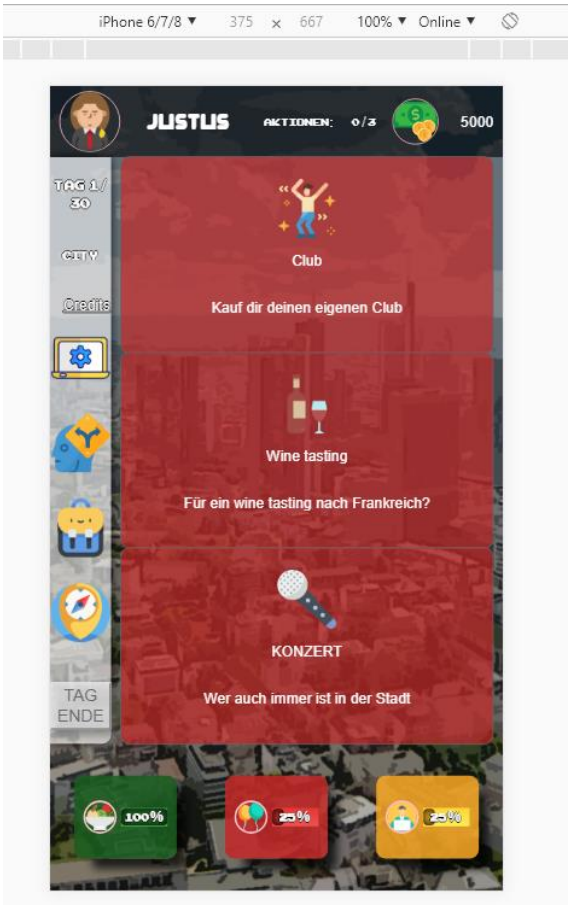

Abbildung 13: Spiel Mobile (Handy)

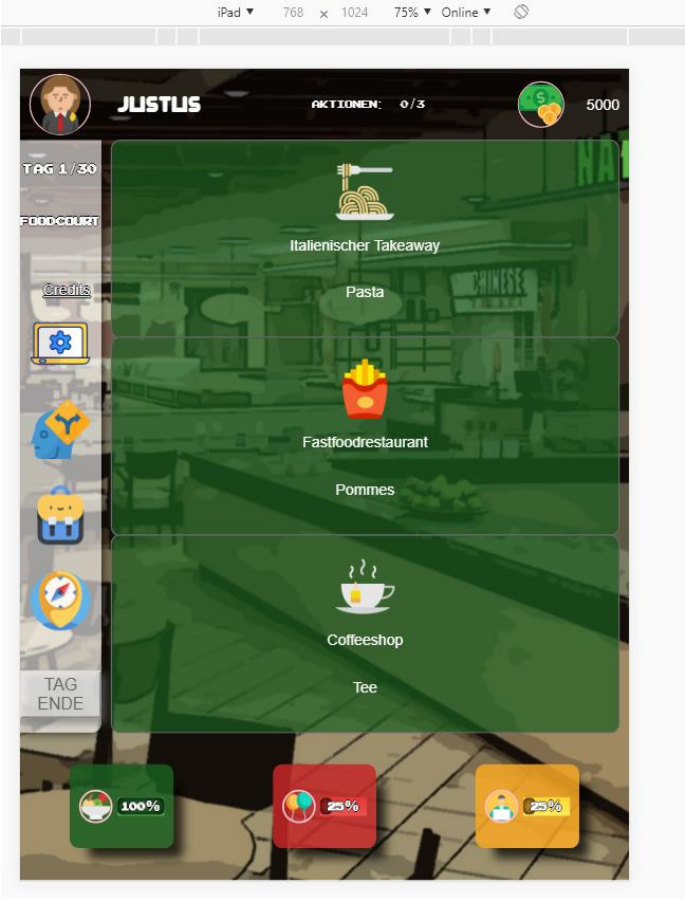

Abbildung 14: Spiel Mobile (Tablet)

v. Screenshots – Tutorial-Modal

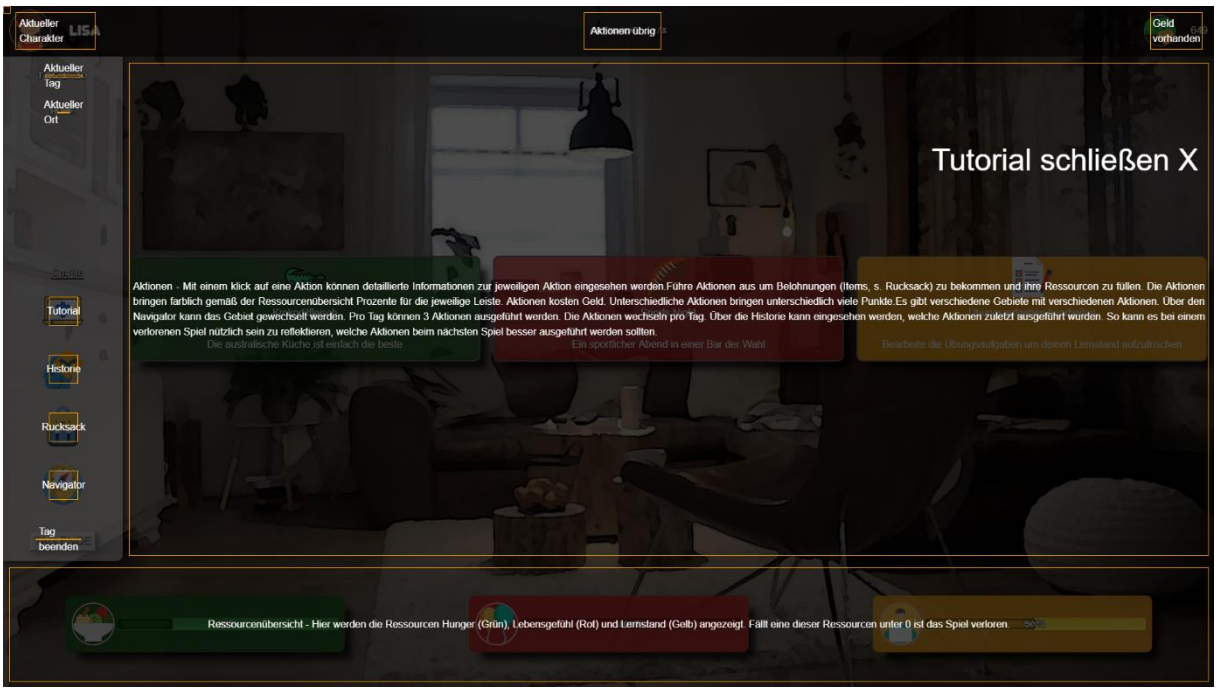

Abbildung 15: Tutorial-Modal

Abbildung 16: Tutorial Modal (Handy)

vi. Screenshots – Inventar

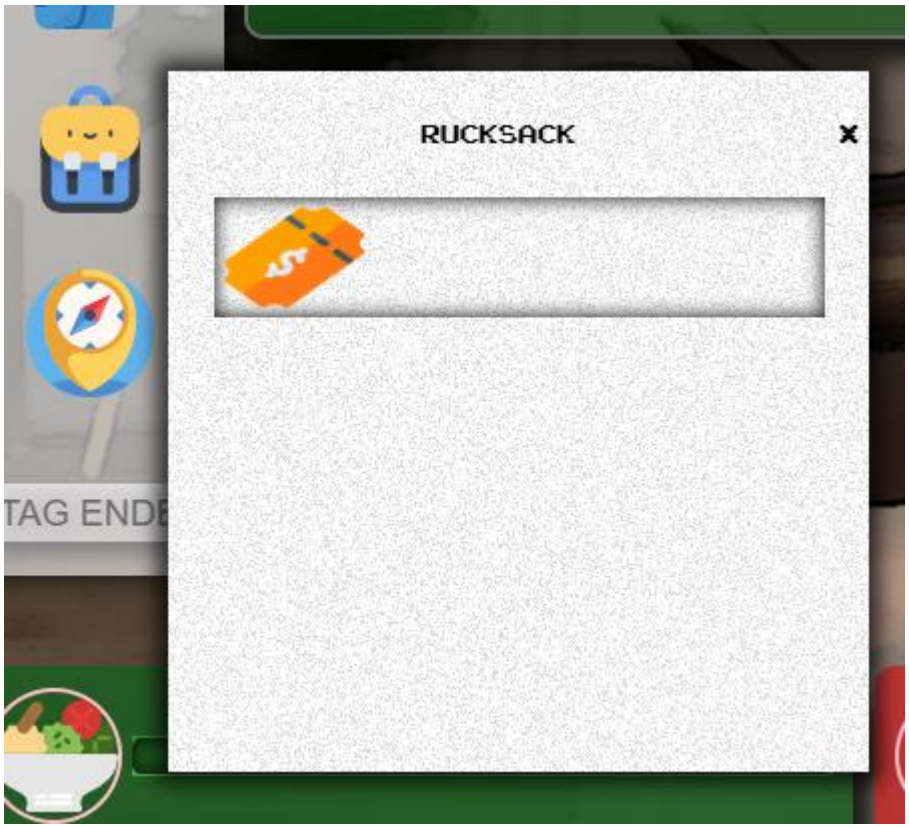

Abbildung 17: Inventar

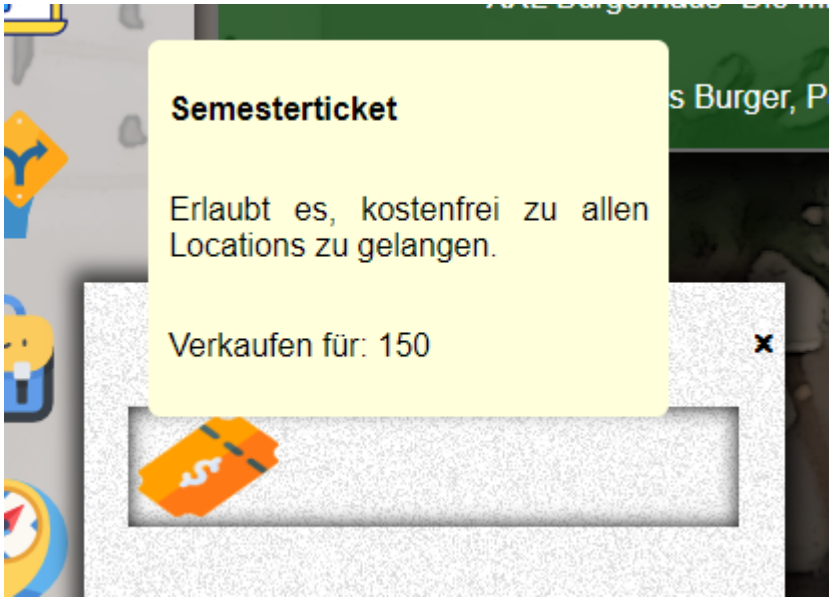

Abbildung 18: Inventar Hover-Effekt

vii. Screenshots – Button-Hervorhebung

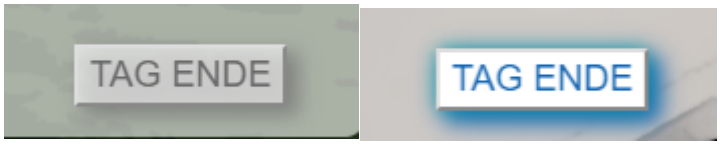

Abbildung 19: Button-Hervorhebung

viii. Screenshots – Auswahl Location

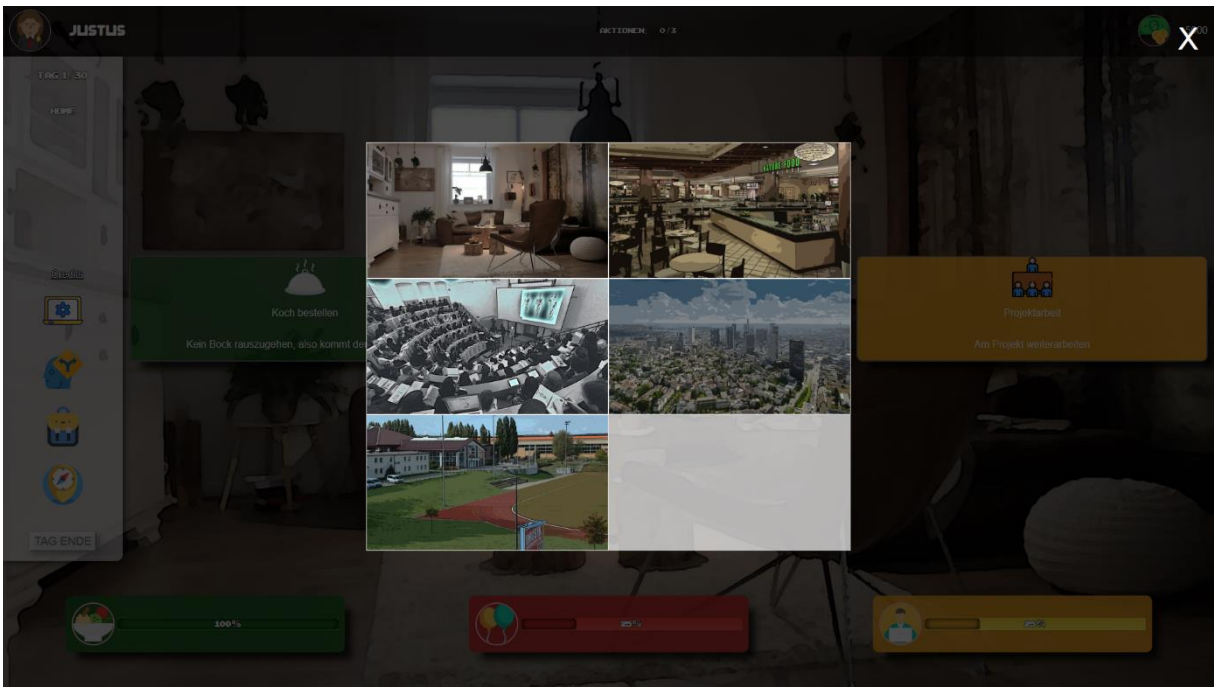

Abbildung 20: Auswahl Location

ix. Screenshots – Aktion durchführen

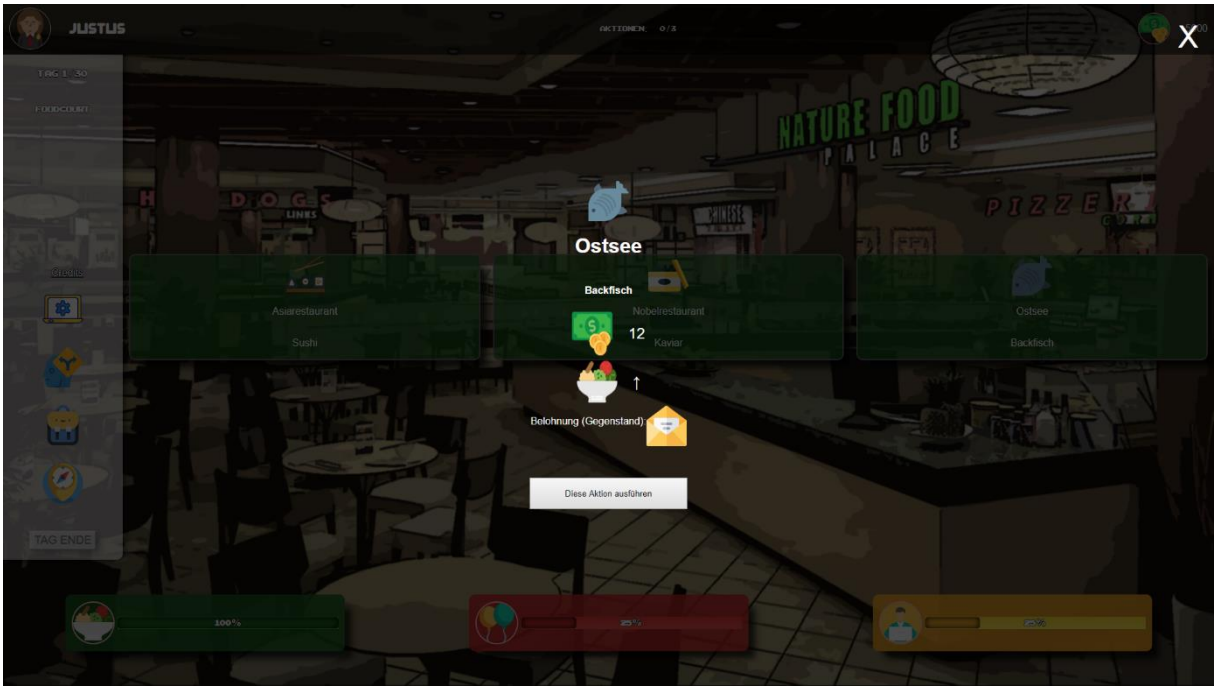

Abbildung 21: Aktion durchführen

x. Screenshots – Outro

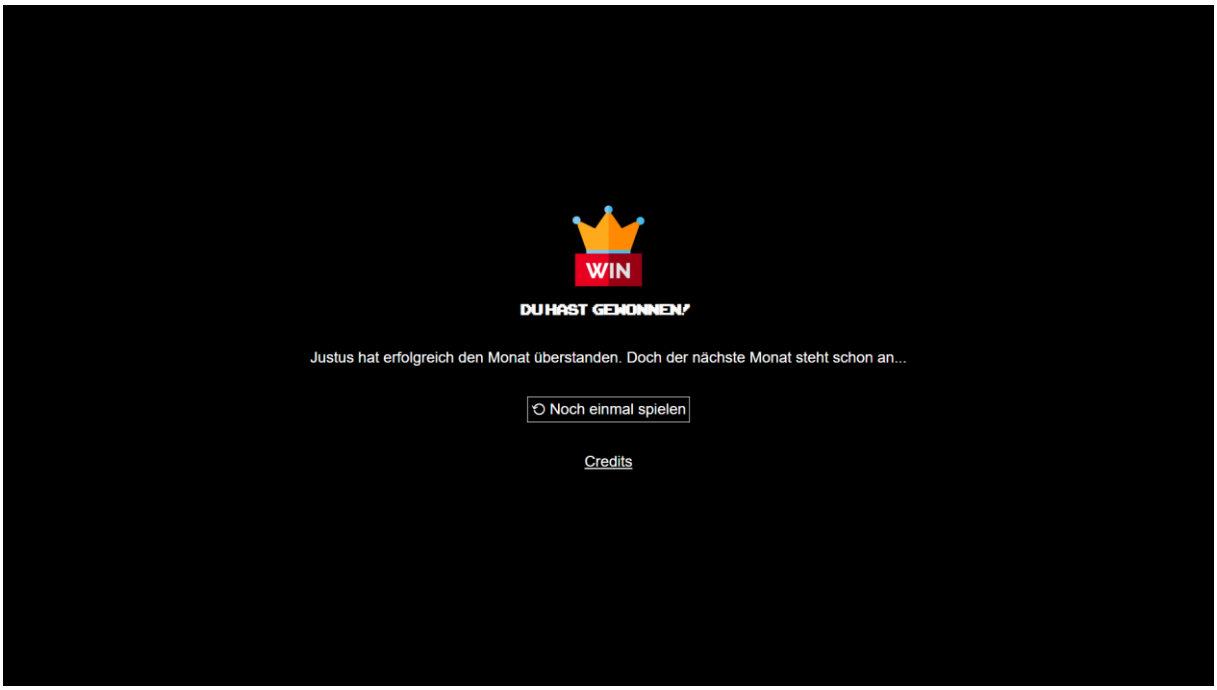

Abbildung 22: Outro Sieg

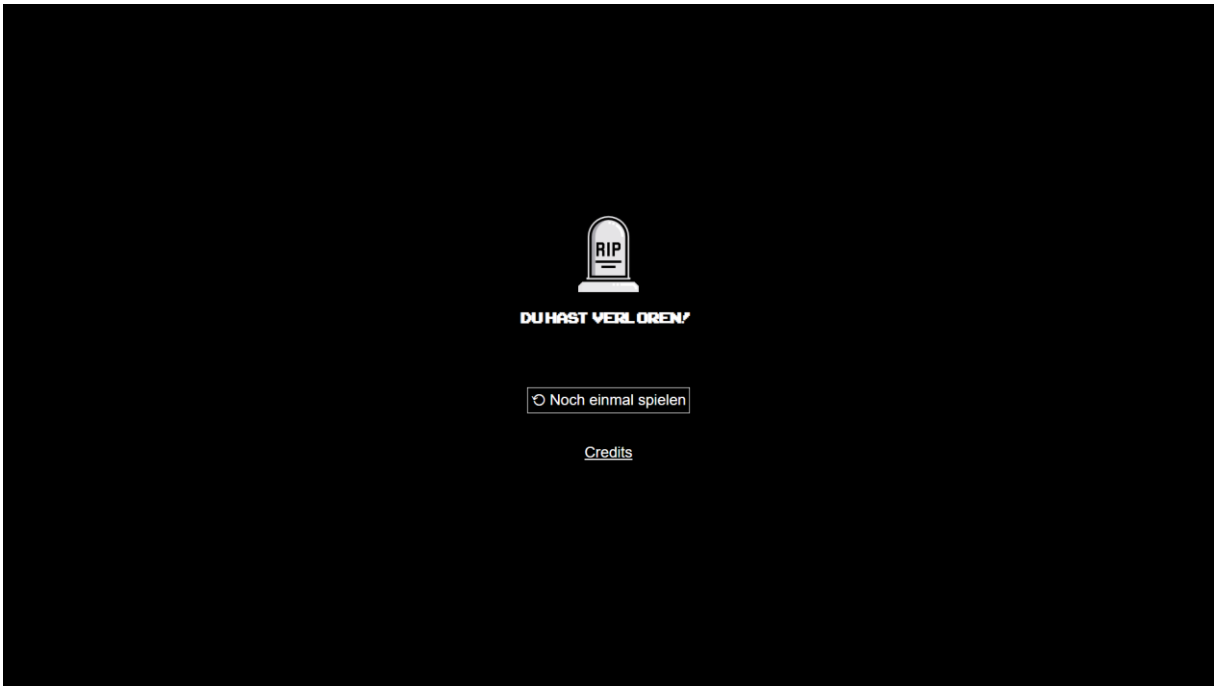

Abbildung 23: Outro Niederlage

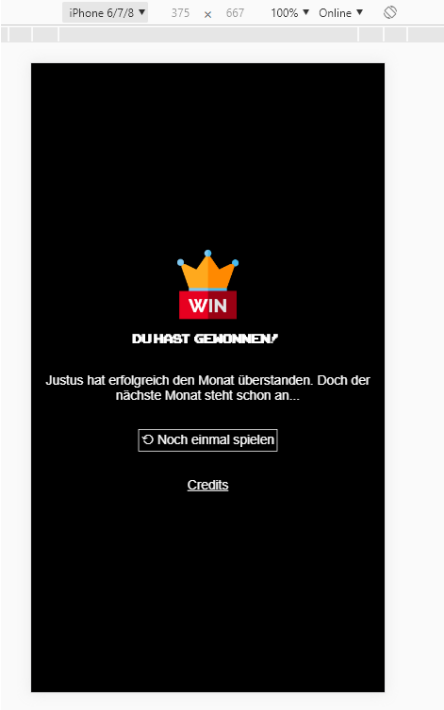

Abbildung 24: Outro Sieg Mobile (Handy)

xi. Projektstruktur

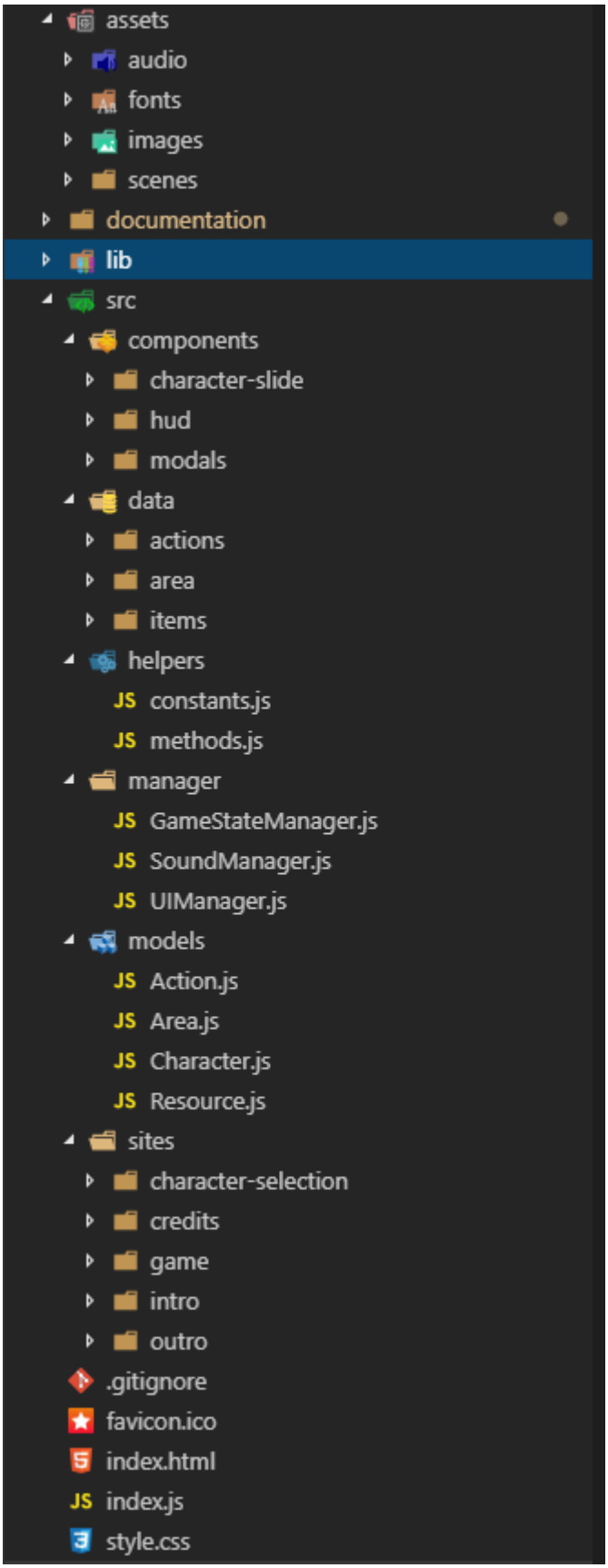

Abbildung 25: Projektstruktur

xii. styles.css

```
/** 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 26: styles.css

xiii. Ausschnitt index.js

```
$(window).on("load", function() {
    var gst = GameStateManager.getInstance();
    if (!gst.loaded) {
      jQuery("#main").load("./sites/intro/intro.html", () => {});
      gst.setLoaded();
      gst.initGameData();
    }
  });
```

Abbildung 27: Ausschnitt index.js

xiv. scriptloader.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 28: scriptloader.js

xv. ResourceBar-Komponente

```
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 29: ResourceBar-Komponente

xvi. Initialisierung ResourceBar-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 30: Initialisierung ResourceBar-Komponente

xvii. Singleton GameStateManager

```
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 31: Singleton GameStateManager

xviii. UIManager updateMoney-Methode

```
/**
 * updates the money
 */
const updateMoneyUI = newValue => {
  playCashSound();
  jQuery(".gameMoneyText").animate(
    {
      opacity: "0"
    },
    100
  );
  jQuery("#gameMoney").text(newValue);
  jQuery("#gameMoney").animate(
    {
      opacity: "1"
    },
    1000
  );
};
```

Abbildung 32: UIManager updateMoney-Methode

xix. playCashSound()-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 33: playCashSound()-Methode

xx. hunger-actions.json - Ausschnitt

```
{
    "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 34: hunger-actions.json – Ausschnitt

xxi. initGameData()-Methode

```
/**
   * 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 35: initGameData()-Methode

xxii. Zuordnungslogik im GameStateManager

```
changeAreaTasks() {
    for (let area of this.areas) {
      area.actions = [];
      switch (area.name) {
        ...
        case "Foodcourt":
          for (let i = 0; i < ACTIONS_PER_AREA; i++) {
            let foodcourtAction = this.hungerActions[
              Math.floor(Math.random() * this.hungerActions.length)
            ];

            area.actions.push(
              this.evaluateAreaActionFit(
                foodcourtAction,
                this.hungerActions,
                area
              )
            );
          }
          break;
        case "Hochschule":
          for (let i = 0; i < ACTIONS_PER_AREA; i++) {
            let hochschuleAction = this.learnActions[
              Math.floor(Math.random() * this.learnActions.length)
            ];
            area.actions.push(
              this.evaluateAreaActionFit(
                hochschuleAction,
                this.learnActions,
                area
              )
            );
          }
          break;
      }
    }
    return this.areas;
  }

/**
   *  makes sure the right action is displayed only in the right area
   * called in method changeAreaTasks();
   *
   * @param action -initial action
   * @param actions wanted actionArray (hunger, life or learn)
   * @param area - the area where the action should be checked for
   */
  evaluateAreaActionFit(action, actions, area) {
```

```
    while (action.area !== area.index) {
      action = actions[Math.floor(Math.random() * actions.length)];
      if (!action.area) {
        if (action.area !== 0) {
          break;
        }
      }
    }
    return action;
  }
```

Abbildung 36: Zuordnung Area -> Action

xxiii. selectCharacter()-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 37: selectCharacter()-Methode