HSB
Hochschule Bremen
City University of Applied Sciences

Fakultät 4 – Elektrotechnik und Informatik

Medieninformatik 2

Projekt: Webgame mit Phaser.js

Jump-Adventure: „Possible Game“

Verfasser:	Philipp Moritzer, 5034255 Hannes Lesemann, 5017055 Pascal Seegers, 5051429
Modul:	Medieninformatik 2
Dozent:	Prof. Dr. Volker Paelke
Labor:	Dipl.-Inf. Andreas Lochwitz
Abgabedatum:	28.06.2019

Inhalt

Abbildungsverzeichnis

1 Spielidee

Die Idee des Spiels ist es eine Art „Platformer“ mit nur einer Eingabemöglichkeit zu erstellen. Die Herausforderung hierbei kommt über das perfekte Timing dieser Eingabe und mit dieser Möglichkeit kann man dieses Spiel interaktiv / schwierig machen. Die Eingaben sollen hierbei bewusst gegen den üblichen Rhythmus des Menschen verstoßen. Hier muss die Balance zwischen Frustration und Spielspaß gefunden werden. Die Level sollen fiese Obstacles und Twists enthalten, die das Spielgeschehen interessanter und interaktiv machen. Das Ziel des Spiels soll es sein, einen „Parkour“ bzw. mehrere Level innerhalb von möglichst wenigen Versuchen zu durchdringen. Dabei sollen die Maps / Level jeweils einem bestimmten Theme nachgehen. Gegebenenfalls entsteht durch das Theming weiterhin eine kleinere Story. Das Spiel so primär als Lückenfüller für Desktop / als Spiel nebenbei fürs Handy funktionieren.

2 Handlung

2.1 Story

Ein Astronaut muss sich durch die Level springen, um zurück auf den Mond zu kommen. Das letzte Level ist der Mond, hat er dies bewähltigt, ist er zu Hause angekommen.

2.2 Interaktion

Das Spielen des Hauptspiels definiert sich durch drücken einer Taste oder durch einen einfachen Touch-Input. Der Charakter durchläuft die level automatisch, die Aufgabe des Spielers ist es den Abgründen und den Stacheln auszuweichen und so ins Ziel zu kommen. Im Menü kann man sich durch klick entscheiden ein neues Spiel anzufangen. Hier beginnt man bei Level 1 und arbeitet sich bis Level 4 durch. Ziel ist es in diesem Modus so wenig wie möglich zu sterben. Der Highscore (wenigste Versuche) wird gespeichert. „Stirbt“ man in einem Level, so wird man an den Anfang des Levels zurückgesetzt. Mit der „Space“-Taste kann man das ganze geschehen pausieren. Dort gibt es dann auch eine Möglichkeit zum Menü zurückzukehren. Drückt man erneut „space“, wird das Spiel fortgesetzt.

Neben einfachen Sprungaufgaben wird der Spieler in den Leveln mit umgekehrter Anziehungskraft, Rückwärtslaufen und wenig Anziehungskraft gefordert.

In dem Menü „Select Level“ kann man den Übungsmodus starten, in dem der Spieler die Möglichkeit hat, die Level einzeln zu durchlaufen und zu üben. Diese zählen nicht zu dem lokalen Highscore.

3 Grafische Gestaltung

3.1 Layout und Gestaltungsmerkmale

3.1.1 Hauptmenü

Für Screenshots des Menüs siehe im Anhang Abbildung 23: Desktop-Hauptmenü.

Das spiel startet mit dem Menü. Der Hintergrund des Menüs wird zufällig gewählt und ist jeweils ein Hintergrund eines zufällig ausgewählten Levels. Im Menü ist es möglich sein aktuelles Game fortzufahren, ein neues Spiel zu starten, oder in das Level-Selektions Menü zu gelangen. Außerdem wird der lokale Highscore angezeigt. Außerdem wird die Interaktion mit den jeweiligen Menüelementen implementiert. Als Indikator wird hier ein Sprite neben dem jeweiligen Menüpunkt gezeigt um zu signalisieren, welcher Menüpunkt angeklickt wird.

3.1.2 Level-Selektion

Für Screenshots der Level-Selektion siehe im Anhang Abbildung 24: Desktop-Level-Select.
Die Level-Selektion funktioniert wie das Menü, nur das hier bei einem Klick auf den jeweiligen Menüpunkt, der eine Stage repräsentiert, man direkt in das Spielgeschehen mit dem ausgewählten Level gelangt.

3.1.3 Spiel

Für Screenshots der Game-Seite siehe im Anhang xi. Desktop Game Screenshots.

Das Spiel ist als horizontaler Scroller gestaltet worden. Es wurden 16x16 Tiles in dem jeweiligen Stil der Map aneinander gereiht um eine Plattform zu schaffen, auf der sich der Spieler bewegt. Außerdem wurden auffällige Stacheln als Hindernisse und kleine Dekorationen auf die Plattformen gestellt. Durch den Parallax-Hintegrund wirkt die Bewegung nach rechts, als wenn die Plattformen im Vordergrund deutlich näher dran sind und die Berge / Assets im Hintergrund deutlich weiter weg.

Oben rechts indiziert ein Text in welchem Level man sich befindet und je nach dem ob man sich im Übungsmodus befindet oder in einem durchgehenden Spiel, die Anzahl der benötigten Versuche.

3.2 Parallax

Der Hintergrund der Scenes sind jeweils mit einem Parallaxeffekt hinterlegt. Der Parallax-Effekt arbeitet mit verschiedenen Ebenen, die sich unterschiedlich schnell bewegen. Der Hintergrund bildet dabei am linken und rechten Rand ein endloses Muster. Beispielsweise bewegen sich in Stage 1 die Berge im Hintergrund langsamer als die kleinen Berge davor oder die Landschaft. Es entsteht ein Effekt eines endlosen Hintergrunds, der durch die verschiedenen Bewegungsgeschwindigkeiten der Teile weiter weg oder näher dran wirkt.

3.3 Mobile Ansichten

Für Screenshots der mobilen Ansicht siehe Abbildung 31: Game Screenshot Phone und Abbildung 32: Game Screenshot Tablet

Das Spiel ist ebenfalls auf mobilen Geräten wie Smartphones oder Tablets spielbar und wurde entsprechend skaliert.

3.4 Schriften

Im Spiel werden primär zwei Schriftarten verwendet. Zunächst Arcade Classic um einen Retro/Arcade-Look zu erhalten. Diese Schriftart ist nur für Design-Zwecke. Außerdem wird noch Arial für Texte verwendet, die schnell gelesen werden müssen, wie beispielsweise den Score-Text.

3.5 Tiles / Sprites

Das benutzte Tileset (s. Abbildung 19: Tileset) ist ein Standard Platformer Tilesets mit 16x16 Tiles. Es bietet verschiedene Environments als Pixel-Art, wodurch es möglich ist verschiedene Themes zu erzeugen (Eis, Gras, Wüste, Gestein).

Der Sprite für den Spielercharakter ist ein Astronaut mit verschiedenen Perspektiven für alle Bewegungsrichtungen (s. Abbildung 20: Astronaut Spritesheet).

3.6 Musik

Für die passende Untermalung des Menüs und der jeweiligen Stages wurde jeweils unterschiedliche Musik gewählt.

Für das Menü wurde eine relativ repetitive Musik gewählt, die das Menü ein wenig untermalt und lebendiger macht.

Für die 1. Stage, die in den Hügeln/Wald spielt ist ebenfalls ein recht simples Soundbild. Es hört sich sehr einheitlich. Die 1. Stage ist vom Design auch wenig reißerisch, weshalb diese Musik gewählt wurde.

Die 2. Stage ist mit einer relativ minimalistischen Musik untermalt, die sich orientalisch anhört und dementsprechend gut in das Ambiente einer Wüste einpasst.

Die Musik der 3. Stage erinnert durch den Sound an Schneefall und Weihnachten und mischt sich mit diesem Soundbild in das optische der Stage ein.

Für den Mond gibt es eine Musik, die einem Rauschen nachempfunden ist und von der Athmosphäre an ein Orbit erinnert.

Die Sounds für das Springen des Charakters und für das Verlieren eines Spieles sind an klassische Arcade-Sounds angelegt, die meisten Menschen würden also an dem Sound bereits erkennen, was passiert.

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. Der Anteil von HTML und CSS ist sehr gering, da die Elemente über die Game-Engine Phaser geregelt werden, welche es erlaubt die Implementierung fast ausschließlich in JavaScript vorzunehmen. Es muss lediglich ein Container und die Rahmenbedingungen der Spielszene in HTML und CSS umgesetzt werden. Es werden Teile der ES6-Syntax verwendet, beispielsweise Arrow-Funktionen.

4.2 Bibliotheken

Als einzige Bibliothek wurde Phaser verwendet. Phaser ist ein Framework, welches zur Entwicklung von Spielen in HTML5 dient. Als Renderer wird WebGL verwendet.

4.3 Editor und Tools

Als Editor wurde Visual Studio Code mit folgenden Plugins verwendet:

- Live Server
- Prettier
- 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.
Zur Erstellung von Tilemaps wurde das Programm „Tiled“ 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 / Implementierung

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. Die Projektstruktur (s. Anhang TODO) ist folgendermaßen aufgebaut: Die Quelltext-Dateien liegen im Ordner „/src/“, die Assets (Bilder, Audio, Tilemaps) liegen im Ordner „/assets/“, die verwendeten Bibliotheken unter „/lib/“ und die Dokumenation unter „/doc/“.

5.2 Einstiegspunkt

Zunächst wurde in der index.html ein Container erzeugt, in dem das Spielgeschehen ablaufen soll. Über den Script-Tag im Head wird die Phaser-Bibliothek geladen. Außerdem wurden die weiteren benötigten Abhängigkeiten geladen.

```
...
<script src="./lib/phaser.min.js"></script>
...
<body>
    <div id="game-container"></div>
    <script src="./src/managing/game-state.js"></script>
      ......
</body>
...
```

Abbildung 1: index.html Auszug

In der main.js wird zunächst ein config-Objekt (s. Abbildung 14: config-Objekt) initialisiert, welches dann dem Phaser-Game übergeben wird.

```
const game = new Phaser.Game(config);
```

In der Konfiguration werden beispielsweise der Container mitgegeben, die Größe des Spieles und wie das Spiel in verschiedenen Auflösungen skaliert wird. Außerdem wird der „physics“-Mode mitgegeben. Es wurde sich für Arcade entschieden, da die Hauptfunktionalitäten des Spieles um Bewegung und

Collision handelt, welches durch „Arcade“ gewährleistet wird. Die Szenen werden jeweils in einem Array mitgegeben. Bei einer Szene handelt es sich um eine Klasse, die von Phaser.Scene erbt (s. Abbildung 2: Scene-Klasse) und die Methoden `preload()`, `create()` und `update()` implementieren kann.

```
class Stage1 extends Phaser.Scene {
      ...
}
```

Abbildung 2: Scene-Klasse am Beispiel Stage1

5.3 Scenes

Einer Scene wird zunächst im Konstruktor ein „Key“ übergeben, mit dem man auf diese Scene zugreifen kann.

```
constructor() {
    super({ key: "Stage1" });
  }
```

Abbildung 3: Scene-Konstruktor am Beispiel Stage1

Dieses Projekt wurde in drei verschiedene Arten von Scenes aufgeteilt: Preload, Menu und Stages, welche im Folgenden genauer betrachtet werden.

5.3.1 Preload

Die Preload-Szene wird als erste Szene beim Aufrufen der Applikation ausgeführt, da sie im config-Objekt an erster Stelle kommt. Sie implementiert die `preload()`-Methode und dient dafür alle Assets zu laden.

```
preload() {
    //load spritesheet + tileset
    this.load.spritesheet("dude", "assets/sprites/character.png", {
      frameWidth: 48,
      frameHeight: 48
    });
    this.load.image("tileset", "assets/tilesets/Platformer tiles.png");

    //load menu assets
    this.load.image("options_button", "./assets/gui/options_button.png");
    this.load.image("play_button", "./assets/gui/play_button.png");
    ...
}
```

Abbildung 4: GamePreload preload()-Methode Ausschnitt

Die Assets (Bilder, Sounds, Tilemaps) beim Start der Applikation zu laden hat den Vorteil, dass diese während des Spiels nicht nachgeladen werden müssen und dementsprechend die Performance nicht darunter leidet. Dies führt dazu, dass das Spiel zum Starten eine gewisse Zeit benötigt. Damit der Benutzer weiß, dass das Spiel lädt, wurde auch ein Ladebalken implementiert (s. Abbildung 15: LoadingBar()-Methode). Wurde alles geladen, wird die Menu-Scene gestartet.

```
create() {
    this.scene.start("Menu");
  }
```

Abbildung 5: Start Menu-Scene

5.3.2 Menu

Die Menu-Scene implementiert die Methoden `create()` und `update()`. Die update-Methode bewegt hierbei lediglich den Hintergrund. In der create-Methode werden die jeweiligen Menüpunkte als Bild geladen.

```
let playButton = this.add.image(
      this.game.renderer.width / 2,
      this.game.renderer.height * 0.45,
      "play_button"
    );
```

Abbildung 6: Menüpunkt laden am Beispiel vom "Play"-Button

Außerdem wird die Interaktion mit den jeweiligen Menüelementen implementiert. Als Indikator wird hier ein Sprite neben dem jeweiligen Menüpunkt gezeigt um zu signalisieren, welcher Menüpunkt angeklickt wird. Durch „Pointer“-Events ist auch eine Touch-Eingabe möglich.

```
playButton.setInteractive();

playButton.on("pointerover", () => {
      hoverSprite.setVisible(true);
      hoverSprite.play("walk");
      hoverSprite.x = playButton.x - playButton.width;
      hoverSprite.y = playButton.y;
    });

    playButton.on("pointerout", () => {
      hoverSprite.setVisible(false);
    });
    playButton.on("pointerup", () => {
      practiceMode = false;
      this.music.stop();
      this.scene.start("Stage1");
    });
```

Abbildung 7: Interaktion von Menüpunkten am Beispiel vom "Play"-Button

Außerdem wird der gesetzte lokale Highscore, falls gesetzt, aus der localStorage geladen und angezeigt.

Das Select-Level-Menü ist nach demselben Schema aufgebaut.

5.3.3 Stages

Es gibt vier Stages die jeweils die create() und update()-Methode implementieren. In jeder Stage wird zunächst die Methode initBackgroundLayerArea() aufgerufen um den Hintergrund zu laden, sowie createBaiscLevelSetup(scene) aufgerufen.

```
    createBasicLevelSetup(this);
```

Die Methode createBasicLevelSetup(scene), die in der Datei stage-managing.js implementiert ist, bietet einen Standardaufbau für die jeweiligen Stages. Hier werden die jeweiligen Layer aus Tiled geladen, die bei den Stages gleich sind (Boden, Spikes, Finish, Gravity), die Spielfigur initialisiert, die Animationen, die Texte erstellt und die jeweiligen Inputs definiert. Jede Scene hat dann noch die

Möglichkeit in ihrer create()-Methode für diese Szene spezifische Sachen aufzurufen. Beispielsweise hat die Stage1 noch einen Info-Text, der kurz den Input des Spiels erklärt.

Die update()-Methode der jeweiligen Stage-Scenes ist dafür verantwortlich den Input zu verarbeiten, die Win-/Lose-Condition zu überprüfen und verschiedene Sachen, wie z.B. in Stage 3 die Laufgeschwindigkeit zu verändern.

Außerdem wird der Parallax-Effekt im Hintergrund durch die update()-Methode gesteuert.

```
update(time, delta) {
	...
	moveTilesArea1(this);
	...
	if (this.player.body.blocked.down) {
		playJumpSound(this);
		this.player.setVelocityY(-1100);
	}
}
```

Abbildung 8: update()-Method Auszug

5.4 JavaScript

Um die Scenes nicht übermäßig groß zu machen, Duplizierung von Code zu vermeiden und Quelltext an die sinngemäße Stelle zu schreiben, wurden verschiedene JavaScript-files erstellt und Quelltext in diese ausgelagert.

5.4.1 game-state.js

Das game-state.js File hält ein Objekt, welches den aktuellen Spielstand (aktuelles Level und Versuche hält). Dieser wird in die localStorage geschrieben, damit man das Spiel aufhören und beim erneuten öffnen der Applikation dort weitermachen kann, wo man aufgehört hat. Außerdem hält es ein Objekt, welches benutzt wird, um den Spielstand zurückzusetzen. Des Weiteren ist ein Flag enthalten, welches indiziert, ob man in einem aktuellen Spiel spielt oder ob man den Practice-Mode benutzt.

5.4.2 state-managing.js

Das state-managing.js-File enthält Methoden, die für die jeweilgen Stages interessant sind. So enthält es jeweils Methoden um den Hintergrund einer Stage zu initialiseren oder das Parallax-Movement des Hintergrunds für die Stages. Außerdem hält es die oben genannte createBasicLevelSetup()-Methode und

5.4.3 music-config.js, constants.js, util.js

Die music-config.js, constants.js und util.js sind kurze Helferdateien, die Konstanten wie die Größe des Game-Fensters, die Konfiguration für die Musik und Sounds und Methoden wie die zufällige Generierung einer Zahl in einem bestimmten Zahlenraum enthalten.

5.4 Skalierung

Die Skalierung des Games wird über den von Phaser implementierten ScaleManger erledigt. Im config-Objekt besteht die Möglichkeit verschiedene Parameter mitzugeben. Es wurde sich in diesem Projekt für den Modus Phaser.Scale.FIT entschieden mit der Bildschirmgröße des jeweiligen Geräts als Breite und Höhe des Spiels. So versucht Phaser das Spiel an jeweilige Endgeräte zu skalieren, was für mobile Geräte, Tablets und Desktops in diesem Spiel jeweils gut funktioniert hat.

```
scale: {
    mode: Phaser.Scale.FIT,
    autoCenter: Phaser.Scale.CENTER_BOTH,
    width: GAME_WIDTH,
    height: GAME_HEIGHT
  },
```

Abbildung 9: scale-Property im config-Objekt

5.5 JSON-Tilemaps

Aus Tiled werden die Tilemaps, die für die jeweiligen Stages verwendet wurden als .json-Struktur exportiert (s. Anhang Abbildung 21: Auszug Tilemap .json-Export). So kann man über Phaser auf die jeweilge Struktur zurückgreifen und auf die jeweils mitgegebenen Attribute reagieren. Hat ein tile beispielsweise die property „collides“ kann Phaser davon ein Tile machen, mit dem der Spieler kollidiert.

```
scene.belowLayer.setCollisionByProperty({
    collides: true
  });
...
scene.physics.add.collider(scene.player, scene.belowLayer);
```

Abbildung 10: CollisionByProperty

5.6 Collision-Debugging

Um die Kollisionen zu debuggen, wurde ein jeweiliges Layer in der Farbe verändert. So konnte man sehen mit welchen Objekten der Spieler jeweils kollidieren sollte und so auf Richtigkeit und Abfragegenauigkeit überprüfen.

```
this.spikeLayer.renderDebug(debugGraphics, {
      tileColor: null, // Color of non-colliding tiles
      collidingTileColor: new Phaser.Display.Color(243, 134, 48, 255), // Color
of colliding tiles
      faceColor: new Phaser.Display.Color(40, 39, 37, 255) // Color of colliding
face edges
    });
```

Abbildung 11: Collision-Debugging Implementierung

Für ein Beispiel, wie die Welt mit dem Code aus Abbildung 11 aussieht, siehe im Anhang Abbildung 22: Collision Debugging.

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 der Aufbau des Projekts unter den Gruppenmitglieder besprochen. 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 Umsetzung der Idee und das Arbeiten mit Phaser und Tiled hat gut funktioniert. Die Arcade-Physics und die funktionen der Game-Engine waren sehr gut geeignet für die Umsetzung und das Projekt konnte erfolgreich durchgeführt werden.

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 Probleme und Lösungen

Im Laufe des Projektes gab es häufiger Performance-Probleme. Wurde das Spiel geladen, hat es am Anfang stark geruckelt. Dieses Problem wurde durch das Auslagern der preload()-Methode jeder einzelnen Scene in eine eigenständige Scene, die nur zum vorladen von Assets benötigt wird und einen Ladebalken enthält.

Die Performance-Probleme hielten auf dem Testgerät für den iOS-Browser an. Das Problem legte sich, als bemerkt wurde, dass der Stromsparmodus des Systems eingeschaltet war. Nachdem dieser abgeschaltet wurde, gab es auch keine Probleme mehr mit der Performance auf dem iOS-Browser.

Des Weiteren gab es anfangs Probleme damit, dass die Kollision oft nicht genau war und der Spielcharakter oft durch die Plattformen gefallen ist. In dem Config-Objekt konnte ein TileBias gesetzt werden, der das Problem behoben hast.

7.3 Ausblick

Denkbare Erweiterungen des Spiels sind weitere Stages, die von der Schwierigkeit variieren und andere Themes behandeln. Außerdem wäre es denkbar andere Variationen in das Spiel einzubauen, wie einen Slow-Motion-Modus oder andere interessante Effekte, die das Spielgeschehen erweitern.

Des Weiteren könnte der lokale Highscore gegen eine Bestenliste erweitert werden. Diese könnte gegebenfalls auch auf einem Webserver gespeichert werden, so dass die Rekorde nicht nur lokal sind.

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

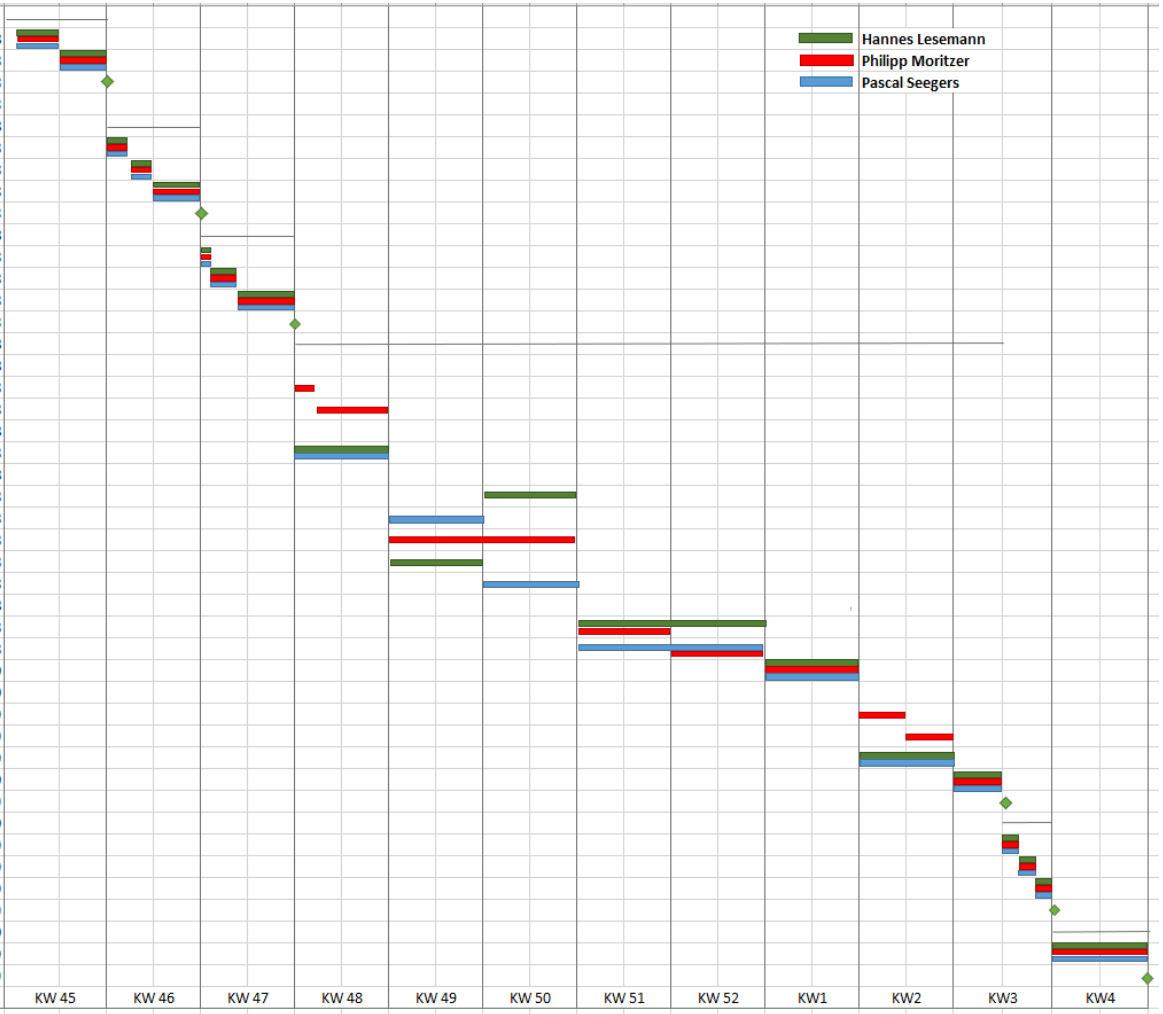

Abbildung 12: GANTT-Diagramm Planung

ii. Projektstruktur

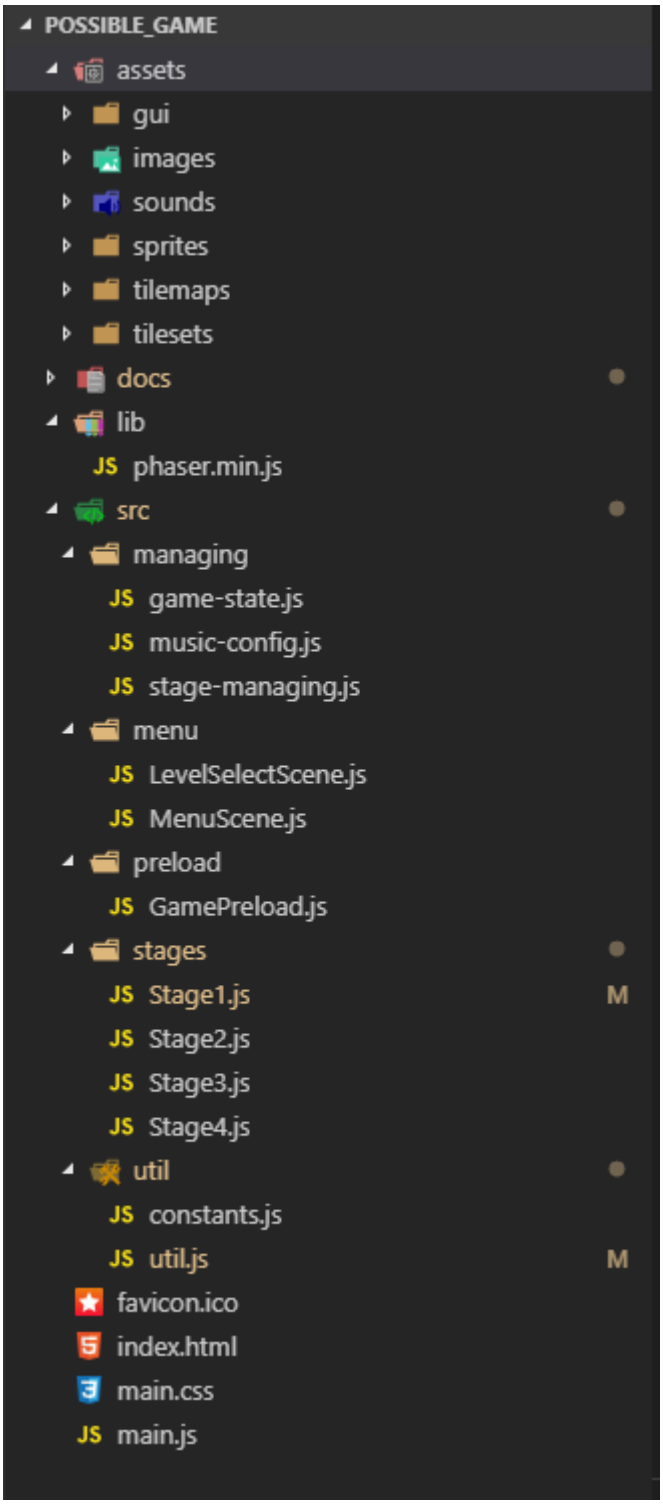

Abbildung 13: Projektstruktur

iii. Config-Objekt

```
const config = {

  type: Phaser.AUTO,
  scale: {
    mode: Phaser.Scale.FIT,
    autoCenter: Phaser.Scale.CENTER_BOTH,
    width: GAME_WIDTH,
    height: GAME_HEIGHT
  },
  parent: "game-container",
  backgroundColor: "#E4F6F8",
  zoom: 1,
  physics: {
    default: "arcade",
    arcade: {
      tileBias: 32,
      gravity: { y: 5000 }
    }
  },
  scene: [
    GamePreload,
    MenuScene,
    LevelSelectScene,
    Stage1,
    Stage2,
    Stage3,
    Stage4
  ]
};
```

Abbildung 14: config-Objekt

iv. LoadingBar()-Methode

```
function loadingBar(scene) {
  let loadingBar = scene.add.graphics({
    fillStyle: {
      color: 0x000000 // white;
    }
  });
  scene.load.on("progress", percent => {
    loadingBar.fillRect(
      0,
      scene.game.renderer.height / 2,
      scene.game.renderer.width * percent,
      50
    );
  });
}
```

Abbildung 15: LoadingBar()-Methode

v. Tiled Map Building

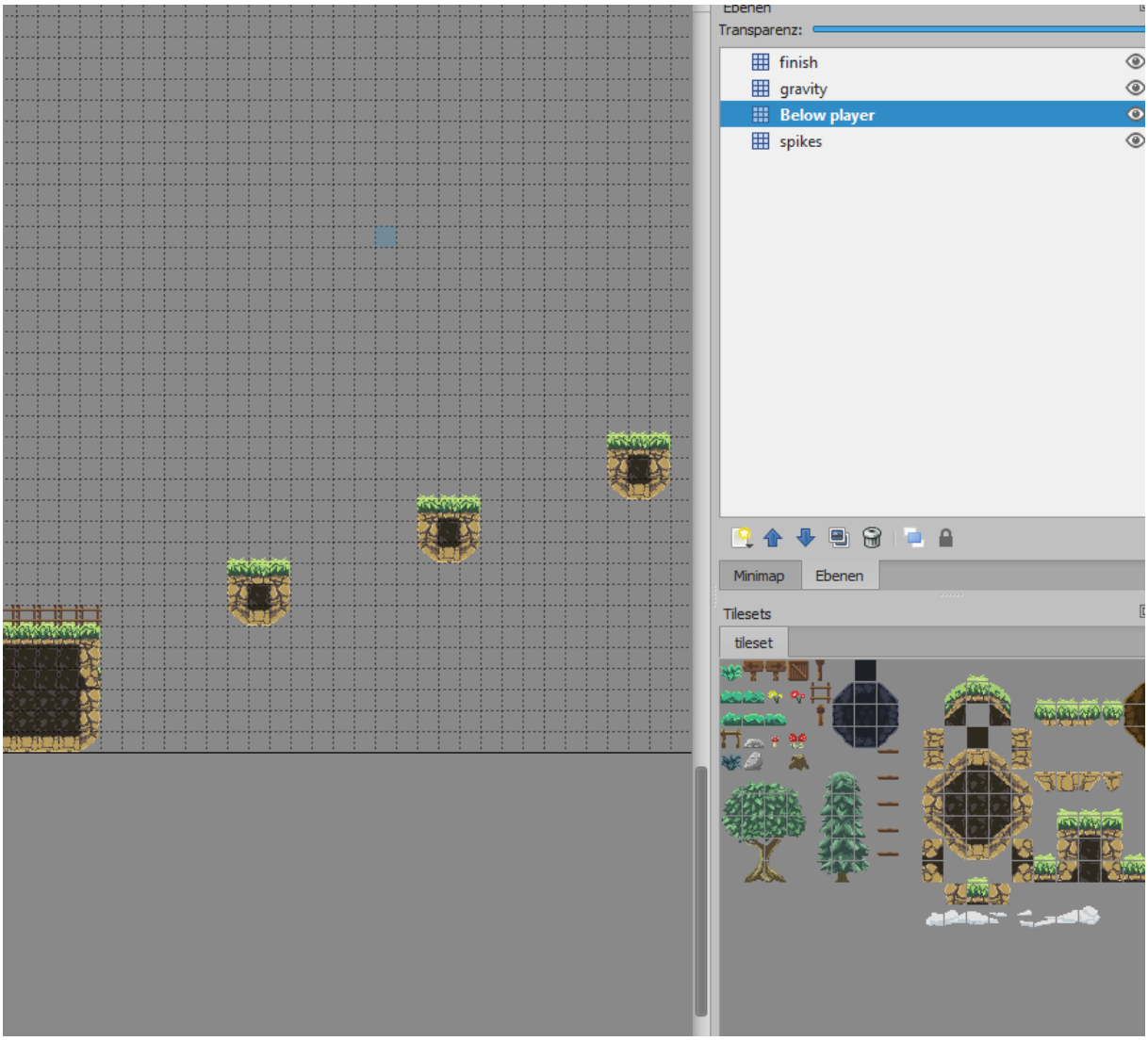

Abbildung 16: Erstellen von Tiledmaps

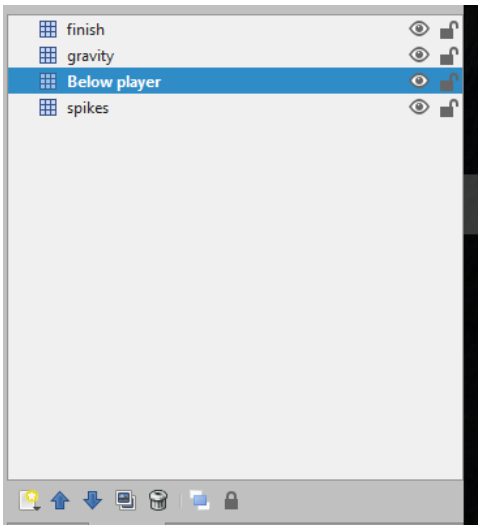

Abbildung 17: Tiled layers

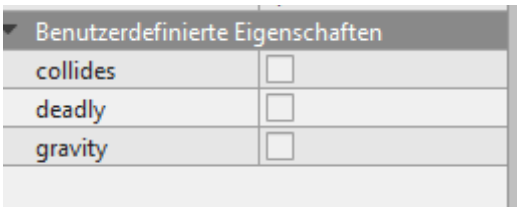

Abbildung 18: Tiled Attribute

vi. Tileset

Abbildung 19: Tileset

vii. Spritesheet

Abbildung 20: Astronaut Spritesheet

viii. Auszug Tilemap als .json

```
{ "height":150,
 "infinite":false,
 "layers":[
        {
         "data":[154, 155, 155, 3221225627, 155, 155, 3221225627, 155, 155,
3221225627, 155, 155, 3221225627, 155, 155, 3221225627, 155, 155, 3221225627
```

...

```
"height":150,
        "id":16,
        "name":"gravity",
        "opacity":1,
        "type":"tilelayer",
        "visible":true,
        "width":550,
        "x":0,
        "y":0
```

...

```
                "id":35,
                "properties":[
                        {
                         "name":"collides",
                         "type":"bool",
                         "value":false
                        }]
                },
```

...

Abbildung 21: Auszug Tilemap .json-Export

ix. Collision debugging

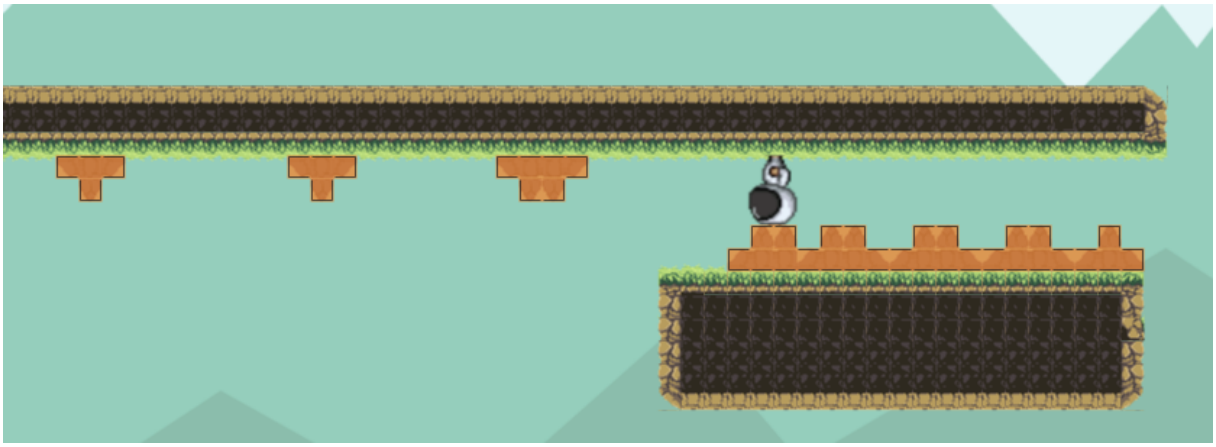

Abbildung 22: Collision Debugging

x. Desktop Menu Screenshots

Abbildung 23: Desktop-Hauptmenü

Abbildung 24: Desktop-Level-Select

xi. Desktop Game Screenshots

Abbildung 25: Stage 1 – Screenshot

Abbildung 26: Stage 2 – Screenshot

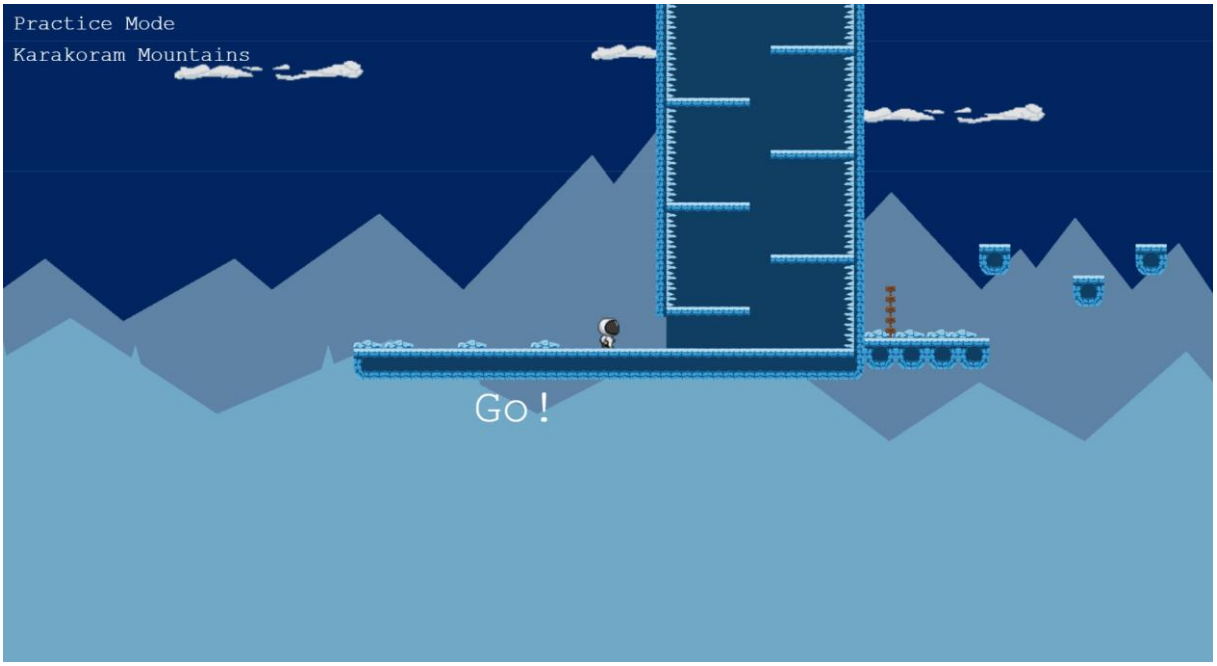

Abbildung 27: Stage 3 – Screenshot

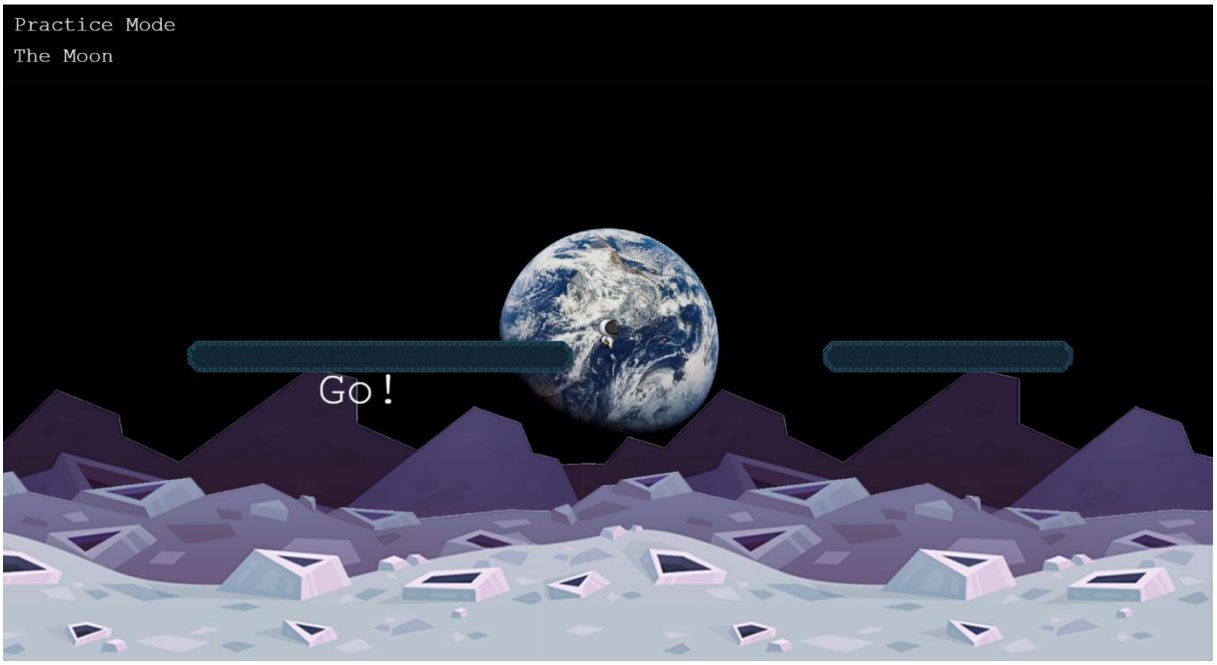

Abbildung 28: Stage 4 – Screenshot

xii. Mobile Ansichten

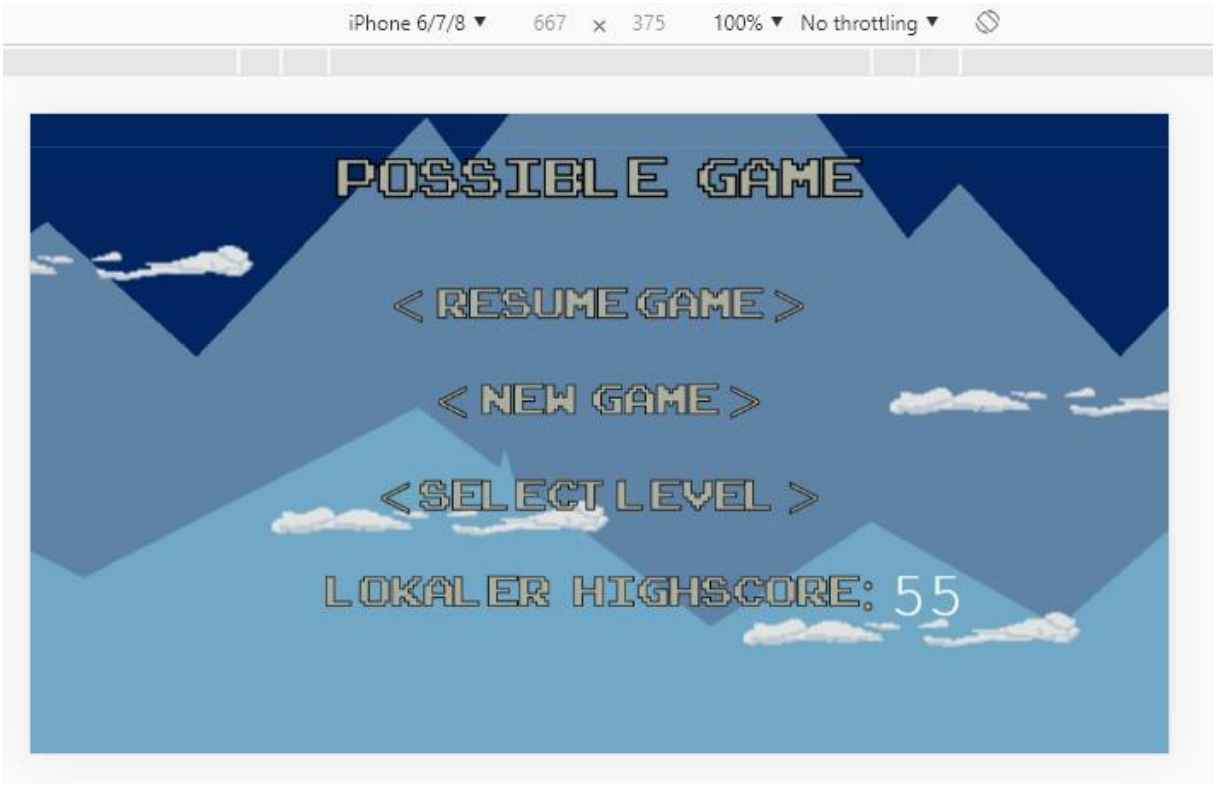

Abbildung 29: Menu-Screenshot Phone

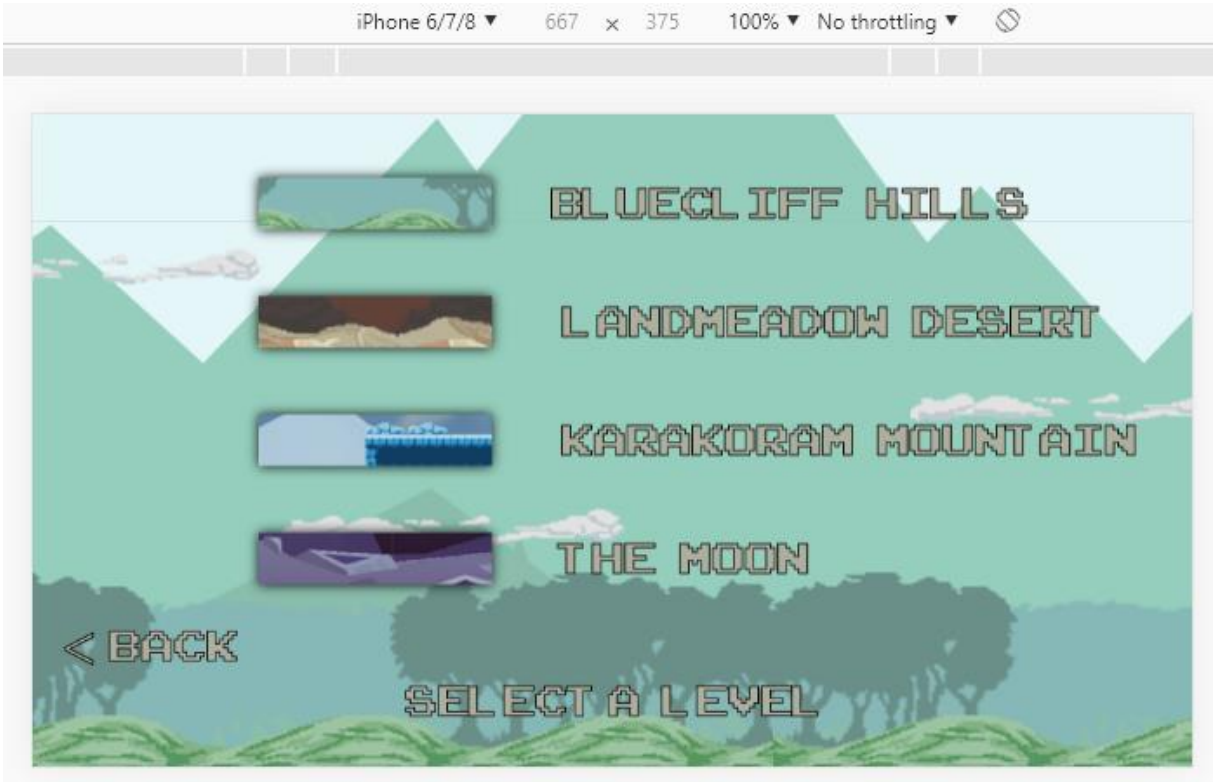

Abbildung 30: Select-Level Screenshot Phone

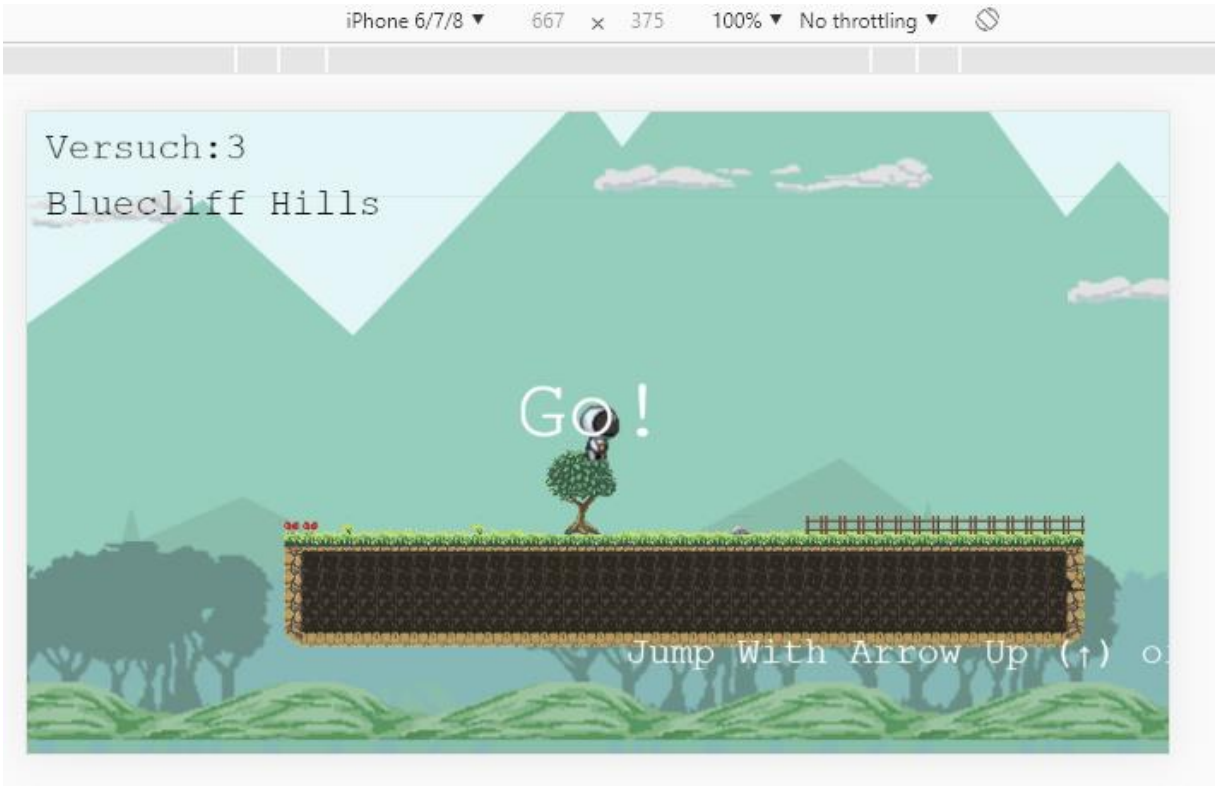

Abbildung 31: Game Screenshot Phone

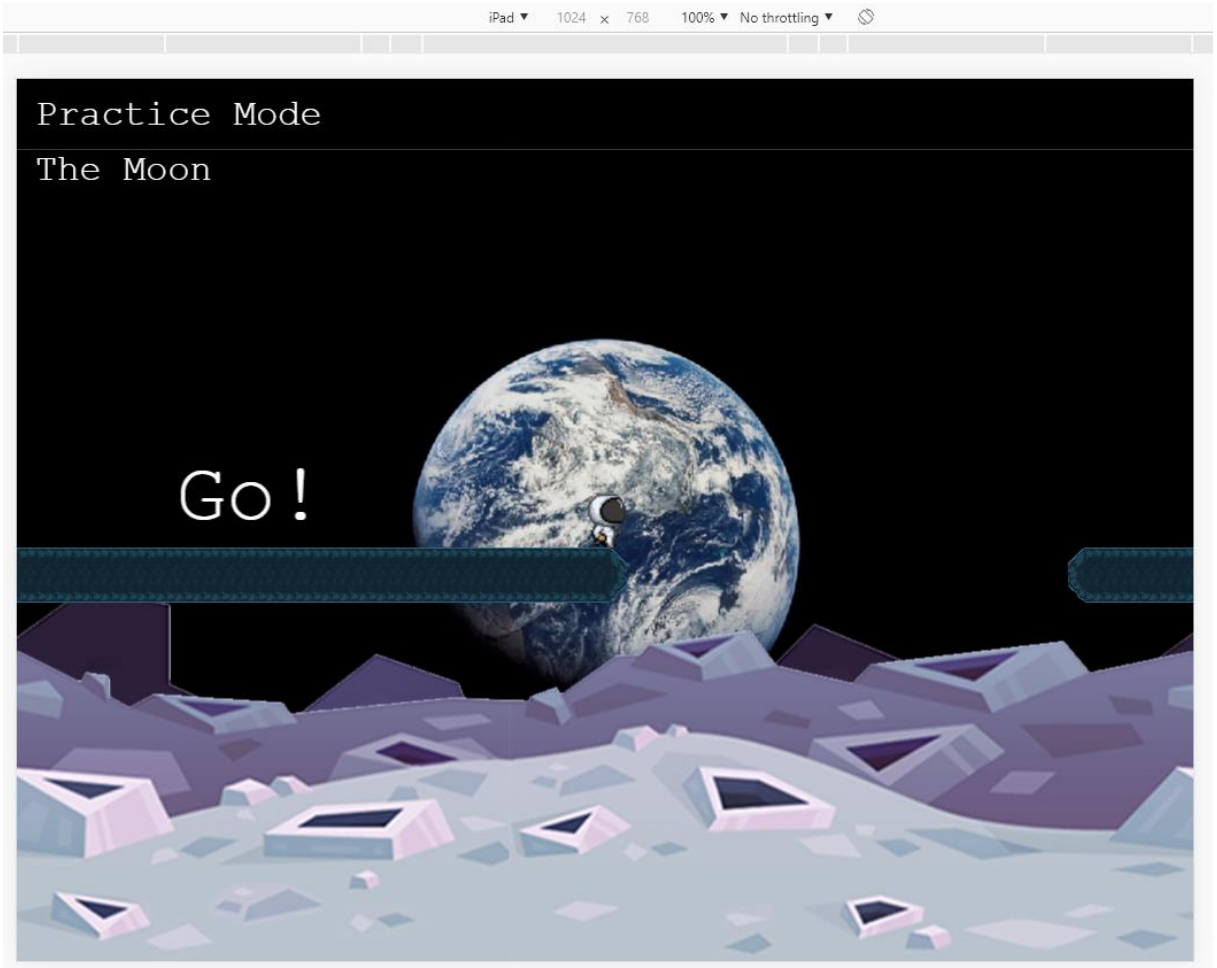

Abbildung 32: Game Screenshot Tablet