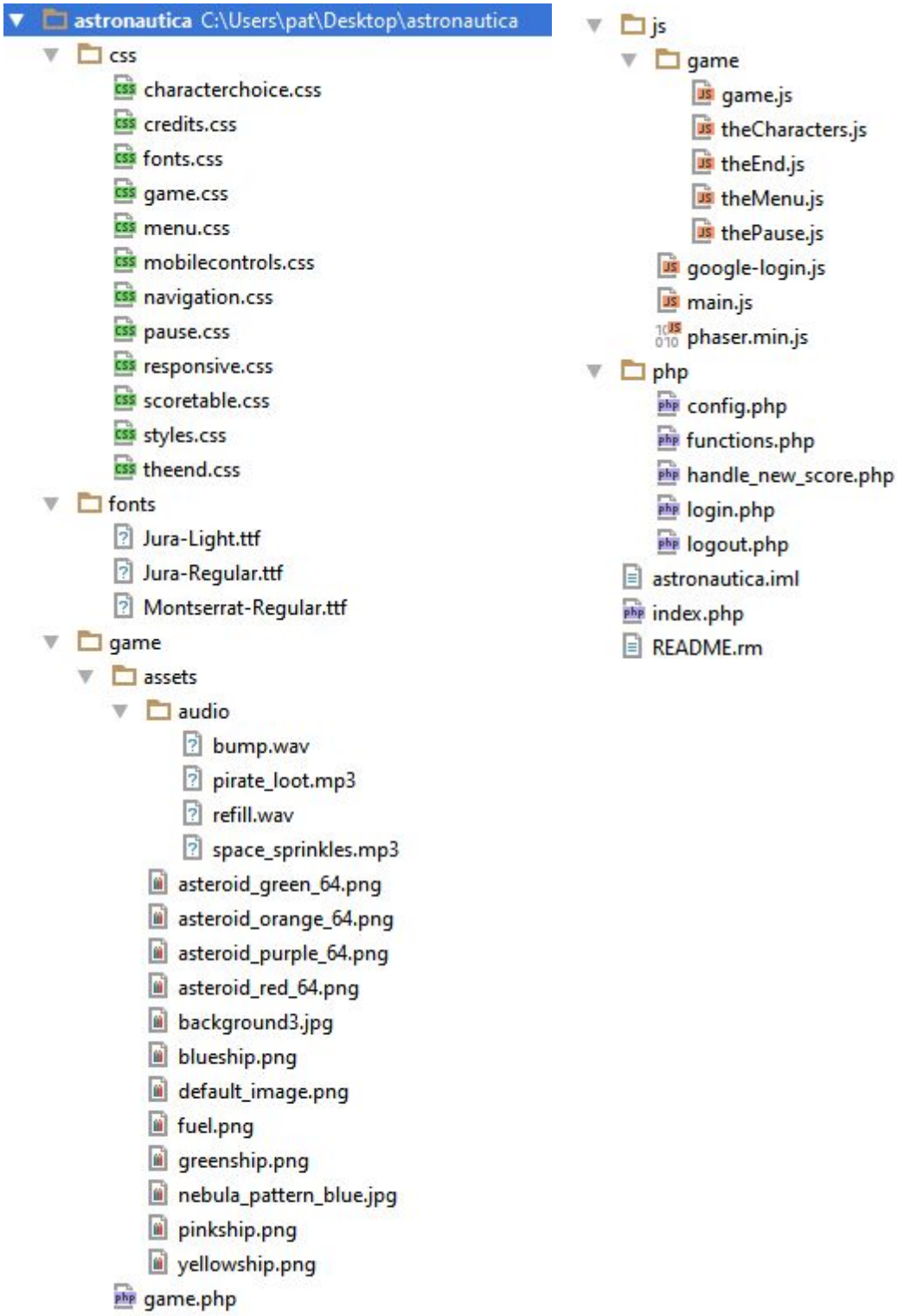
ASTRONAUTICA

Technische Umsetzung



ORDNERSTRUKTUR

CSS

- CSS (Hauptseite)
    - credits.css
        - steht in Verbindung zu Credits Abspann
    - navigation.css
        - steht in Verbindung zur Navigation
    - responsive.css
        - steht in Verbindung zur Responsiveness (3 Stufen)
    - scoretable.css
        - steht in Verbindung zum scoreboard
    - styles.css
        - allgemeines Styling
- CSS (Allgemein)
    - fonts.css
        - bindet Google Fonts eins
- CSS (Game)
    - game.css
        - steht in Verbindung zu allgemeinen Gameelementen
    - characterchoice.css
        - steht in Verbindung zu Charakterauswahlmenü
    - menu.css
        - steht in Verbindung zum Menüstate
    - mobilecontrols.css
        - kümmert sich um die Bedienung am Smartphone bzw. Touchdisplay
    - pause.css
        - steht in Verbindung zum Pausemenü
    - theend.css
        - steht in Verbindung zum Endstate

FONTS

- Allgemein
    - Beinhaltet alle benötigen Google Fonts als .ttf Format

GAME

- game.php
    - gesamte Website die sich auf das Game bezieht.
- assets
    - beinhaltet alle notwendigen Grafik assets
    - audio
        - Beinhaltet Audio assets

JS

- Google-login.js
    - Behandelt login & logout der Applikation, leitet per Ajax Daten (Google Profil) an login.php weiter. Genauso erfolgt hier die Anzeige des Profils (HTML,CSS) auf index.php
- Main.js
    - Kontrolliert folgende Hauptseitenelemente: Fadeffekt des Titels, Musik, Credits, Profilfading, Go-top-button, Scroll-buttons, Navigation-Events
- Game
    - Game.js
        - Herzstück des Spiels, vereint alle assets, Gamestates, Funktionen, Preload
    - theCharacters.js
        - Erstellt Charaktermenü als HTML Gerüst und legt es als Overlay über das Spiel
    - theEnd.js
        - Erstellt Endscreen und schickt bei erfolgreicher Paramterübergabe Daten an handle_new_score.php
    - theMenu.js
        - Erstellt das Hauptmenü
    - thePause.js
        - Erstellt das Pausemenü während des Spiels

PHP

- Config.php
    - Stellt Verbindung zum FH Server bzw. Localhost her
- Functions.php
    - Startet Session und erstellt das $dbh Element um mit der DB kommunizieren zu können
- Handle_new_score.php
    - Filtert schadhafte Einträge und speichert Spielergebnisse in die Datenbank
- Login.php
    - Prüft ob Sessionvariablen gesetzt sind und entscheidet ob ein neues Profil für den Google Account erstellt werden muss oder nicht
- Logout.php
    - Vernichtet Cookies und beendet die Session

INDEX.php

- Herzstück der Hauptseite: Abspann, Credits, Scoreboard Anzeige, Loginelemente

#Wie erfolgt der LOGIN?#

Ich habe mich diesbezüglich für die Google Login API entschieden, einerseits weil diese einen interessanten Eindruck vermittelt hat (passender Schwierigkeitsgrad, guter Leitfaden) sowie das Unwissen wie man damit arbeitet. Um die API zu verwenden hab ich ein Google Skript (benötigt für u.a. Buttonrendering, Funktionen, etc.) sowie den meta-tag zum Verknüpfung meiner APP mit der Google API eingebunden.

> <script src="https://apis.google.com/js/platform.js" async defer></script>
> <meta name="google-signin-client_id"
> content="YOUR_CLIENT_ID.apps.googleusercontent.com">

Der Login Button öffnet einen Link zu Google wo sich der User einloggen kann. Sobald dieses erfolgreich ist, wird die Funktion onSignIn mit dem Google-User-Profil-Objekt aufgerufen. Auf dieses Objekte kann ich bereits für mich relevante Daten auslesen wie z.b. ID, Name, ImageURL. Die

```
var onSignIn = function (googleUser) {
    var profile = googleUser.getBasicProfile();
    //var id_token = googleUser.getAuthResponse().id_token;
    var auth1 = gapi.auth2.getAuthInstance();

    if (auth1.isSignedIn.get()) { //google-logged-in
        google_user_id = profile.getId();
        first_name = profile.getGivenName();
        last_name = profile.getFamilyName();
        email = profile.getEmail();
        profile_image_url = profile.getImageUrl();
        display_user_information(profile);
    }

    //log_user_information(profile);

    $.ajax({
            type: 'POST',
            url: 'php/login.php',
            data: {
                'google-id': google_user_id,
                'firstname': first_name,
                'lastname': last_name,
                'username': "astronautica_player",
                'email': email
            }
        }).done(function () {
            if(!isLoggedIn) {
                location.reload();
            }
    });
```

Was passiert noch. Auch wird ein AJAX-Paket an die Login.php weitergeleitet, welche die functions.php mit sich bringt → sobald diese geöffnet wird, wird eine Session gestartet → Setzen der USER & ID Cookies in der Session beim Login.

Um unabhängig von der Google API zu wissen ob man in der Session eingeloggt ist, befindet sich in der index.php eine globale Variable welche prüft ob z.b. $_SESSION[‘USER’] gesetzt worden ist → liefert true oder false zurück. Dadurch kann ich onSignIn auch prüfen ob meine eigenen Cookie Variablen gesetzt sind oder nicht.

#Wie erfolgt die erstrige Erstellung eines Profils in der Datenbank?#

Sobald die onSignIn(profile) ausgeführt wird eine AJAX-Paket mit POST an login.php verschickt. Zunächst wird eine Query abfrage gemacht ob die mitübergebene ID (von Google) bereits in der Datenbank vorkommt. Wenn nicht liefert das Ergebnis der Rückgabe “false” → Wenn als (!result) dann speichere mit prepared statements in meine Datenbank den User ein.

```php
$sth = $dbh->query("SELECT * FROM profile WHERE id = '$id'");
$results = $sth->fetchAll();

if (!$results) { //not first time play

    $sth = $dbh->prepare("INSERT INTO profile(id, firstname, lastname, username, email) VALUES (?,?,?,?,?)");

    $id = strip_tags($_POST['google-id']);
    $firstname = strip_tags($_POST['firstname']);
    $lastname = strip_tags($_POST['lastname']);
    $username = strip_tags($_POST['username']);
    $email = strip_tags($_POST['email']);

    $secure_execute = $sth->execute(
        array(
            $id,
            $firstname,
            $lastname,
            $username,
            $email
        )
    );
}
```

#Wie funktioniert die Kollision im Spiel?#

Im Spiel werden P2-Physics verwendet. Die Arcade Version welche lediglich eine rechteckige Form als Boundary box zeichnet, war für mein Spiel und meinen persönlichen Ansprüche zu wenig. P2 Vorteil: komplexere Polgyone als nur Rechtecke → genauer. Nachteil: Perfomance!

Nachdem die sprites/images in der preload Funktion erfolgreich übertragen worden sind, wird das Physics System aktiviert.

```javascript
initGamePhysics: function () {
    game.physics.startSystem(Phaser.Physics.P2JS);
    game.physics.p2.setImpactEvents(true);
    game.physics.p2.gravity.y = 480;
    game.physics.p2.restitution = 0.15;
},
```

Zuerst starten wir die P2-Engine. Danach erlauben wir callbacks bei aktivierter Kollision. Die Welt besitzt eine Standard Gravity von 480. Die restitution beschreibt die “Rückstoßkraft”.

Daraufhin habe ich drei Kollisionsgruppen erstellt: “asteroidCollisionGroup”,”fuelCollisionGroup” und die “playerCollisionGroup” denenich allen die P2-Physics mitübergebe. Zusätzlich müssen die “bodies” enabled sein, um auf die Eigenschaften zugreifen zu können.

```
//ASTEROID SETTINGS
asteroidCollisionGroup = game.physics.p2.createCollisionGroup();
asteroids = game.add.group();
asteroids.enableBody = true;
asteroids.physicsBodyType = Phaser.Physics.P2JS;
```

Nun da die Grundfesten der Gruppen erstellt wurden, müssen diesen Sprites bzw. Objekte zugewiesen werden, wie hier z.b. An Hand der Asteroiden:

```
createNewAsteroid: function (x, y, asteroidSprite, speed) {
    var asteroid = asteroids.create(x, y, asteroidSprite);
    asteroid.scale.setTo(1.3);
    asteroid.body.setCollisionGroup(asteroidCollisionGroup);
    asteroid.body.static = true;
    asteroid.body.collides([asteroidCollisionGroup, playerCollisionGroup]);
    asteroid.body.velocity.y = speed;

    asteroid.checkWorldBounds = true;
    asteroid.outOfBoundsKill = true;
    asteroid.body.collideWorldBounds = false;
},
```

Hier wird ein neues meteor Objekt erstellt und der Kollisionsgruppe “asteroidCollisionGroup” zugewiesen. Der body wird auf static gesetzt, da er nicht Beschleunigen noch “verschiebbar” sein soll. Dann muss noch definiert werden mit wem der Asteroid kollidieren darf (hier mit der playerCollisiongroup). Je nach Eigenschaften wiederholen sich Codezeilen fürSpieler und Fuel Objekten.

#Technischer Aufbau des Charaktermenüs#

Zunächst habe ich eine Liste als HTML Gerüst gebaut und mir vorgestellt einem List-Item eine Klasse zuzuweisen der dieses Element anzeigen lässt. Dazu muss ich wissen welches Item zurzeit angezeigt wird → getCurrentIndex(). Diese Methode holde ein Array von ListItems und iterriert mit der JQuery Methode .each() über alle Elemente. Ist bei einem

Element die gesuchte Klasse gefunden wird der index returned.

```
//GET CURRENT INDEX
var getCurrentIndex = function num_i() {
    var elements = $('#slide-list li');//JQuery collection
    var current_show_index;
    elements.each(function (index) {
        if (($(this).attr('class')) === 'show') {
            current_show_index = index;
        }
    });
    return current_show_index;
};
```

Jetzt muss nur noch die Klasse immer richtig gesetzt werden (je nach index, auf dem ich mich befinde). Um die Klasse richtige zu setzen habe ich eine setClassOnIndex(index) Funktion geschrieben.

```
//SET IMAGE INDEX
var setClassOnIndex = function (index) {
    var elements = $('#slide-list li');
    elements.each(function (element_i) {
        if (element_i === index) {
            $(this).attr('class', 'show');
        } else {
            $(this).attr('class', '');
        }
    });
};
```

Bei Betätigung des linken Pfeils wird der index dekrementiert und beim rechten vice versa. Um dem Spiel mitzuteilen, dass er eine andere Grafik verwenden soll gibt es eine setAstronautSprite(index) Methode die genau dieses übernimmt.

```
//CHANGES URL/NAME OF USED CHARACTER GRAPHIC
var setAstronautSprite = function (index) {
    if (index == 0) {
        graphicAssets.astronautName = "astronaut";
        graphicAssets.astronautURL = "assets/pinkship.png";
    }

    if (index == 1) {
        graphicAssets.astronautName = graphicAssets.astronautBlueName;
        graphicAssets.astronautURL = graphicAssets.astronautBlueURL;
    }

    if (index == 2) {
        graphicAssets.astronautName = graphicAssets.astronautGreenName;
        graphicAssets.astronautURL = graphicAssets.astronautGreenURL;
    }

    if (index == 3) {
        graphicAssets.astronautName = graphicAssets.astronautYellowName;
        graphicAssets.astronautURL = graphicAssets.astronautYellowURL;
    }
    console.log("Current Index:" + index);
    console.log(graphicAssets.astronautURL);
};
```

Standardmäßig wird der index auf 0 gesetzt. Also auf das pinkship.png.

#Wie werden die Menü's im Spiel erstellt?#

Eine gute Mischung aus JS, JQuery und vor-allem CSS ermöglichen es gut positioniert HTML Elemente über das Spiel zu legen. Ich habe mich deshalb für diese Variante entschieden, sprich gegen Phaser Buttons/Textfelder etc., da ich mir JS/JQuery im allgemein besser anschauen wollte und laut Rückmeldungen Phaser Handhabung auf diese Dinge bezogen nicht so "benutzerfreundlich" sei (→ möchte ich mir td. bei Gelegenheit nochmal anschauen).

```
var enterMenu = function (playGame, initCharacterChoice,goFullScreen) {
    $game_canvas.append("<ul id='play-astronautica-menu'></ul>");
    $('#play-astronautica-menu').css('list-style', 'none');

    $('#play-astronautica-menu').append("<li id='play-welcome-button' style='margin-bottom: 20px;'>PLAY</li>");
    $('#play-astronautica-menu').append("<li id='introduction-welcome-button'>INTRODUCTION</li>");
    $('#play-astronautica-menu').append("<li id='character-welcome-button'>CHARACTERS</li>");
    $('#play-astronautica-menu').append("<li id='fullscreen-welcome-button'>FULLSCREEN</li>");
    $('#play-astronautica-menu').append("<li id='home-welcome-button'>HOME</li>");


    $('#home-welcome-button').css('margin-top', '30px');

    $('#fullscreen-welcome-button').click(function () {
        goFullScreen();
    });

    $('#introduction-welcome-button').click(function () {
        window.open('../index.php#introduction',"_self","","");
    });

    $('#play-welcome-button').click(function () {
        playGame();
    });

    $('#character-welcome-button').click(function () {
        initCharacterChoice();
    });

    $('#home-welcome-button').click(function () {
        window.open('../index.php',"_self","","");
    });
};
```

Durchgehend habe ich diese menü-holder, hier "play-astronautica-menu" mit absoluter positionierung in der Mitte zentriert. Top: 50%, Left: 50% und mit einer Transformierung um -50% auf beiden Achsen richtig zentriert.

#Wie erfolgt die Generierung der Asteroiden, Asteroidereihen (Lücke)?#

Da es mit der JQuery Methode "setTimeOut()" nicht korrekt funktionierte, zeitlich Aktionen zu aktivieren (die timer wurden im zweiten, dritten game wieder neu gestartet → AsteroidMadness), musste eine andere Lösung her. PHASER!

```
//SPAWN ASTEROIDS
game.time.events.add(Phaser.Timer.SECOND * 0.2, this.generateAsteroidRow, this);
game.time.events.add(Phaser.Timer.SECOND * 8, this.generateFuelItem, this);
game.time.events.add(Phaser.Timer.SECOND * 16, this.generateRedAsteroid, this);
game.time.events.add(Phaser.Timer.SECOND * 25, this.generateGreenAsteroid, this);
game.time.events.add(Phaser.Timer.SECOND * 50, this.generatePurpleAsteroid, this);
```

Diese Events werden je nach ersteren Parameter nach einer bestimmten Zeit getriggert. Asteroidenreihen werden mit der Property “generateAsteroidRow” folgendermaßen erstellt:

```
generateAsteroidRow: function () {
    game.time.events.loop(2000, function () {
        var count = 5;
        open = Math.floor((Math.random() * count));

        for (var i = 0; i < count; i++) {
            if ((i != open)) {
                this.createNewAsteroid(i * 83 + 25, 0, graphicAssets.asteroidOrangeName, 200);
            }
        }
        current_score += 50;
    }, this);
},
```

Der in der Phaser-Loop stehende Code wird alle 2 Sekunden ausgeführt und spawnt so dieses Reihen. In der Count-Variable lege ich die Anzahl an Asteroiden fest. Mit einer Zufallszahlgenerierung erstelle ich mir einen freien spot, der in “open” gespeichert wird. Solange der Öffnungsspot nicht gleich dem Schleifenspot ist, erstelle ich einen Asteroiden, bis die Schleifenbedingung nicht mehr erfüllt ist.

#Wie erfolgt die Punkte/Spielübertragung ins Backend?#

Der Score wird per AJAX (POST) übertragen, sobald sich der Spieler im “EndState” befindet. Das heißt, sobald der Spieler verloren wird die handle_new_score.php mit zwei Parameter aufgerufen. Dort wiederum wird schadhafter Code rausgefiltert. Bei erstrigen Spiel triggert die erste IF-Condition wodurch eine neue Reihe eingetragen wird. Erspielt man nun einen höheren Score, wird der vorherige Stand mit dem “Update”-PostgreSQL Statement überschrieben. Erzielt man einen schlechteren Score als wie der in der Tabelle eingetragene eingetragene passiert “nichts”.

```
var sendScore = function () {
    $.ajax({
        type: 'POST',
        url: '../php/handle_new_score.php',
        data: {
            '_score': play_score,
            '_this-id': id
        }
    });
};

sendScore();
```

```php
<?php
/** ASTRONAUTICA '16 ...*/

    include "functions.php";

    if (isset($_POST["_score"]) && (isset($_POST["_this-id"])) && (isset($_SESSION['ID']))&& (isset($_SESSION['USER']))) {

        $id = $_POST["_this-id"];
        $myScore = $_POST["_score"];

        $sth = $dbh->prepare("SELECT
                                game.score
                              FROM game
                              WHERE game.played_by_id = ?
                              ");
        $sth->execute(array($id));
        $score_object = $sth->fetch();

        if ($score_object === false) {
            $sth = $dbh->prepare("INSERT INTO game(played_by_id, score) VALUES (?, ?)");
            $update_went_ok = $sth->execute(
                array(
                    $id,
                    $myScore
                )
            );
        }

        else if($score_object->score < $myScore){
                $sth = $dbh->prepare("UPDATE game SET score=? WHERE played_by_id=?");
                $update_went_ok = $sth->execute(
                    array(
                        $myScore,
                        $id,
                    )
                );
        }
    }
?>
```

#Wie wurde der Abspann umgesetzt?#

Verwendet wurde eine Keyframeanimation mit CSS-Code. Das “closing-credits-content”-div bewegt sich über das transformierte Elternelement und erzeugt so einen Abspannt.

```css
closing-credits {
    display: none;
    height: 400px;
    font-size: 25px;
    text-align: center;
    overflow: hidden;
    transform: perspective(400px) rotateX(35deg); /*z-distance and X-axe rotation*/
    padding-left: 20px;
    padding-right: 20px;


closing-credits-content {
    position: relative;
    top: 100%;
    animation: scroll 50s linear;


keyframes scroll {
    0% {
        top: 100%;
    }

    100% {
        top: -300%;
    }
```

#Wie funktioniert die Erkennung für Touchbildschirme?#

Zunächst füge ich meinem document einen Eventlistener hinzu bei touch eine Funktion aufruft die in den FullScreenModus geht sowie einen isMobile Boolean auf true → soll nur einmal gesetzt werden und nicht getoggelt werden, da während dem Spiele/Interagieren weiters getoucht wird. Sei das Spiel noch nicht gestartet, wird in eine IF-Abfrage gegangen die das Spiel unpaused. Bild 1: Listener, Bild 2: FullScreen Funktion

```
document.addEventListener("touchstart", function () {
    goFullScreen();

    if (!isMobile) {
        isMobile = true;
    }

    if (!isMobileStart) {
        game.paused = false;
    }
});
```

```
var goFullScreen = function () {

    $game_canvas_real = $('#astronautica > canvas');
    $game_canvas_real.css('display', 'inline-block');

    game.scale.fullScreenScaleMode = Phaser.ScaleManager.SHOW_ALL;

    if (game_div.webkitRequestFullscreen) {
        game_div.webkitRequestFullscreen(Element.ALLOW_KEYBOARD_INPUT);
    }

    else if (game_div.mozRequestFullScreen) {
        game_div.mozRequestFullScreen();
    }

    else if (game_div.msRequestFullscreen) {
        game_div.msRequestFullscreen();
    }

    else if (game_div.requestFullscreen) {
        game_div.requestFullscreen();
    }
};
```

Die Vorbereitung ist geschafft nun wird in der initNewGame() Funktion geprüft ob sich der User im mobilen Modus befindet, wenn ja, dann führt dieser initMobileControls() aus. Diese erstellt mit JS/JQuery zwei transparente Buttons am unteren Rand des Spiels.

```
var initMobileControls = function () {
    $game_canvas.append("<div id='mobile-controls'></div>");
    $mobile_controls = $('#mobile-controls');

    $mobile_controls.append("<button id='go-left'></button>");
    $mobile_controls.append("<button id='go-right'></button>");

    var left_button = document.getElementById("go-left");
    var right_button = document.getElementById("go-right");

    left_button.addEventListener('touchstart', function () {
        astronaut.body.moveLeft(400);
    });

    right_button.addEventListener('touchstart', function () {
        astronaut.body.moveRight(400);
    });

    game_div.addEventListener('touchstart', function () {
        GameState.boost();
    });
};
```

Um also auf Touchgeräte eine Skalierung zu vollbringen bzw. Die Buttons einzublenden ist es Pflicht einmal zu touchen um dieses beiden Dinge zu initialisieren.

#Wie wurde die Tankleiste erstellt?#

Die Tankleiste besteht aus einem blockbildenden Elternelement und einem nicht-blockbildenden Kindelement.

```
//FUEL BAR
$hud_holder.append("<div id=fuel-bar></div>");
$fuel_bar = $('#fuel-bar');

$fuel_bar.append("<span id=fuel-bar-content></span>");
$fuel_bar_content = $('#fuel-bar-content');
```

Um einen “fancy” Fill-Up Effekte zu erstellen habe ich mich wiedereinmal der CSS Transition bemächtigt, die getriggert wird sobald sich die Breite verändert:

```
#fuel-bar-content {
    height: 22px;
    display: block;
    background: rgba(5, 37, 163, 1);
    background: -moz-linear-gradient(left, rgba(5, 37, 163, 1) 0%, rgba(255, 255, 255, 1) 53%, rgba(199, 8, 138, 1) 100%);
    background: -webkit-linear-gradient(left, rgba(5, 37, 163, 1) 0%, rgba(255, 255, 255, 1) 53%, rgba(199, 8, 138, 1) 100%);
    background: -o-linear-gradient(left, rgba(5, 37, 163, 1) 0%, rgba(255, 255, 255, 1) 53%, rgba(199, 8, 138, 1) 100%);
    background: -ms-linear-gradient(left, rgba(5, 37, 163, 1) 0%, rgba(255, 255, 255, 1) 53%, rgba(199, 8, 138, 1) 100%);
    background: linear-gradient(to right, rgba(5, 37, 163, 1) 0%, rgba(255, 255, 255, 1) 53%, rgba(199, 8, 138, 1) 100%);
    border-radius: 3px;
    transition: width 400ms linear;
}
```

Der Farbübergang wurde mit dem einem Gradient-Generator für alle Browser erstellt: http://www.cssmatic.com/.

In der Update Funktion wird dann die Breite des Kindelements folgendermaßen angepasst:

```
$fuel_bar_content.width(current_fuel + '%');
```