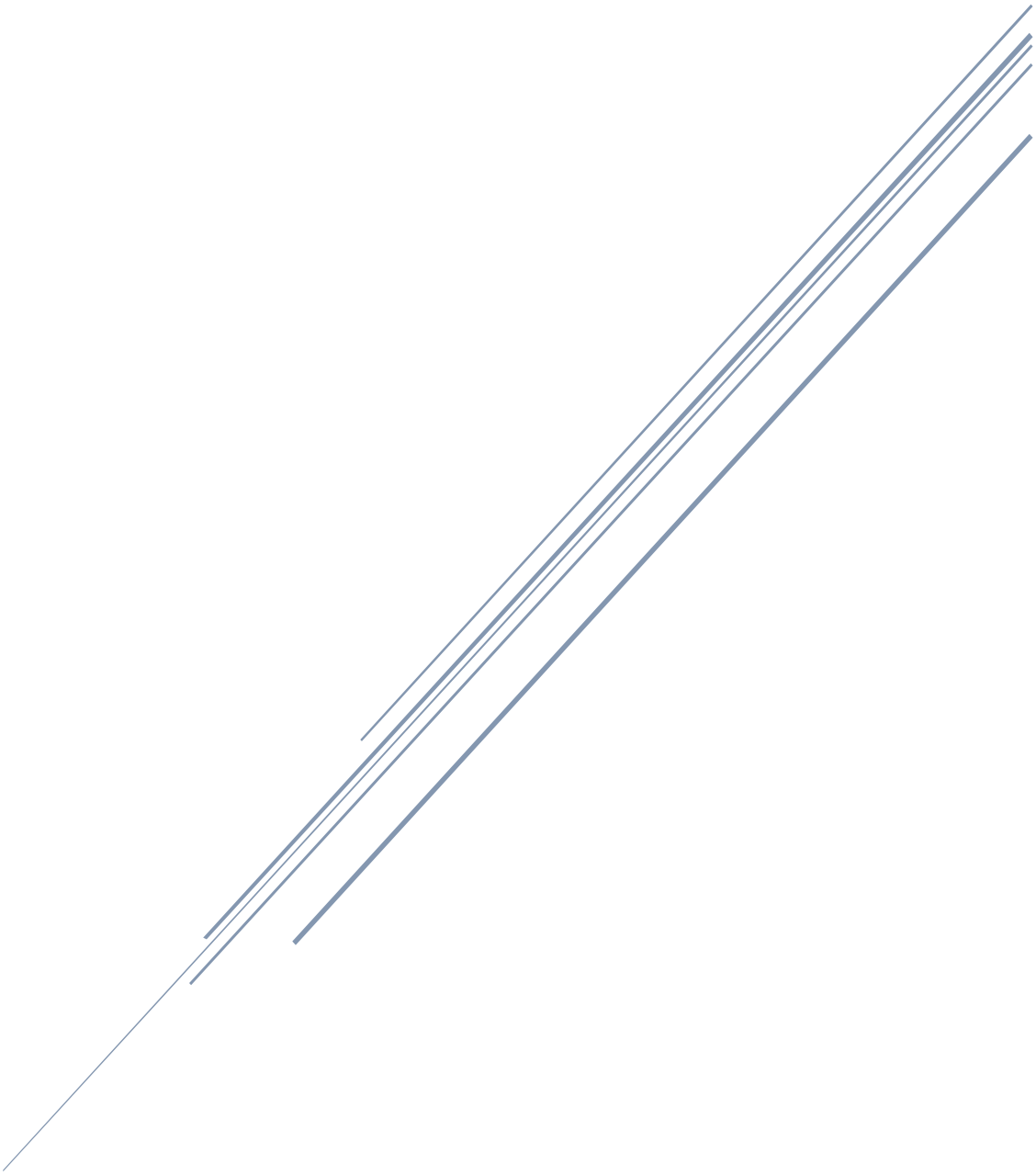
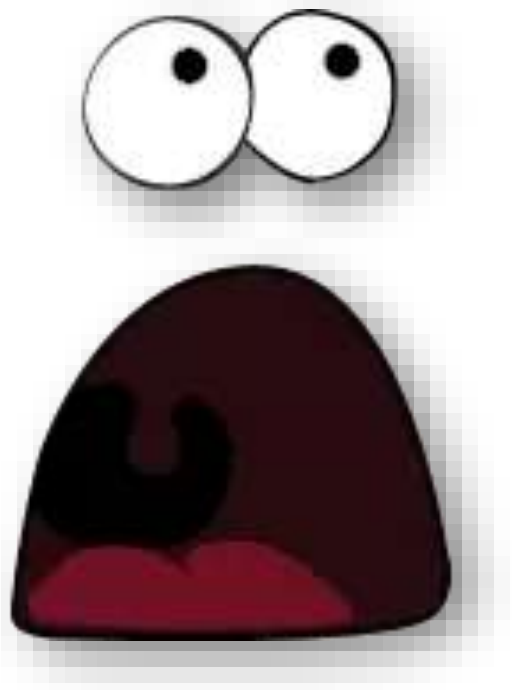
JUMP DX 9001 DELUXE

Milan Stuhlsatz | Marius Mauritz



Technische Hochschule Lübeck
Web-Technologien (SoSe 19)



Inhaltsverzeichnis

Einleitung

Im Rahmen der Vorgaben des Web-Technologien-Projekts, im Sommersemester 2019, haben wir ein Spielkonzept implementiert. Das Spiel wurde in der Programmiersprache Dart entwickelt und läuft als Singlepage-App auf dem Client, lediglich für den Zugriff auf weitere Ressourcen, wie Level oder Grafiken wird dabei auf den Server zugegriffen. Insbesondere findet keine Server oder Clientseitige Speicherung von Informationen, durch das Spiel, statt. Das Spiel wurde nach einem „Mobile First“-Ansatz entwickelt, ist aber auch auf Desktopsystemen intuitiv spiel- und steuerbar.

Auf Grund der Vorgabe eines „Mobile First“-Ansatzes haben wir uns entschieden, uns an vorhandenen mobilen Spielkonzepten zu orientieren. Wir haben dabei von Anfang an auch die Machbarkeit im Rahmen des gegebenen Projektumfangs und weitere gegebene Vorgaben berücksichtigt. Die Entscheidung viel schlussendlich auf eine freie Interpretation des Konzeptes von „Doodle Jump“.

Das gewählte Spielkonzept hat es uns ermöglicht alle gegebenen Anforderungen zu berücksichtigen. Die Steuerung per Bewegungssensor etwa folgt als logische Konsequenz dem grundlegenden Konzept des Spiels. Auch andere Anforderungen, wie die Nutzung des DOM-Trees, der Ausschluss von Canvas und externen Libraries, ein Levelkonzept und die Intuitive Bedienung erschienen und waren unproblematisch.

Anforderungen und abgeleitetes Spielkonzept

Nachfolgend werden wir beschreiben, wie wir die einzelnen Anforderungen des Projekts in unserem Spieldesign berücksichtigt haben.

AF-1: Single-Player-Game als Single-Page-App

Die gesamte Logik des Spiels wurde in der Sprache Dart realisiert, welche in kompilierter Form als JavaScript in die index.html Datei eingebunden wird. Auf einen Server wird hier nur zum Abruf weiterer Ressourcen wie Grafiken oder Dateien im JSON Format zur Darstellung weiterer Level und des Stylesheets zurückgegriffen.

Da das Spiel hauptsächlich die Geschicklichkeit des Spielers testet, ist das Vorhandensein eines Mit- oder Gegenspielers nicht erforderlich. Obwohl nicht implementiert wäre es jedoch möglich, weitere menschliche Spieler mit weiteren Spielfiguren in einem Level unterzubringen. Da das Spieldesign eine Interaktion jedoch nicht vorsieht, bestünde der Reiz hier lediglich im Wettbewerb um den höheren Punktestand oder das schnellere Absolvieren des Levels. Beides setzt jedoch nicht voraus, dass sich die Spieler in derselben Instanz des Spiels befinden.

AF-2: Balance zwischen technischer Komplexität und Spielkonzept

Das Spielkonzept berücksichtigt und erfüllt die Anforderungen an die Komplexität, da es zwar verschiedene physikalische Fragestellungen berücksichtigt, so etwa die Kollision der Spielfigur mit anderen Elementen und die Physik des darauffolgenden Sprungs, sowie den Einfluss von Gravitation auf die Spielfigur. Auch wird überprüft ob die Spielfigur in der Lage ist, den aktuellen Sturz zu überleben (basiert auf ihrer Geschwindigkeit). Es wurden jedoch keine Konzepte wie etwa eine KI implementiert.

Auch die grafische Darstellung stellt einen geeigneten Kompromiss zwischen Komplexität und Machbarkeit dar. So verlassen Objekte den Viewport und betreten ihn, Objekte verändern ihre Position oder ihr Aussehen. Als Beispiel sei hier die Spielfigur genannt, welche basierend auf ihrer Bewegungsrichtung ihre Textur wechselt. Wir haben jedoch z.B. von der Manipulation von Vektorgrafiken (auch aus Performance-Überlegungen bezüglich leistungsschwacher Mobilgeräte) abgesehen.

AF-3: DOM-Tree-basiert

Das Spiel wurde konsequent nach dem MVC Prinzip umgesetzt. Das Model und seine zugehörigen Klassen beinhalten alle Informationen und Zustände über den Verlauf des Spiels und die zugehörige Logik. Die View hingegen ist das Bindeglied zwischen der internen Logik des Models und der Repräsentation der Elemente im DOM-Tree des Browsers. Aufgrund der Isolation beider Elemente wäre es Möglich durch einen Austausch der View etwa von einem DOM-Tree basierten Konzept auf ein Canvas Konzept oder eine völlig andere Darstellung umzustellen. Auch verwenden Model und View jeweils eigene Aktualisierungsraten. Es ist so möglich, das Spiel mit 60fps im Browser darzustellen aber z.B. 83 mal pro Sekunde das Model zu aktualisieren. Da das Model keine Kenntnisse über die Eigenschaften der Grafischen Darstellung hat, verwendet es ein internes System zur Positionierung der Objekte, die View übersetzt diese dann auf den Maßstab des verwendeten Devices. Die Darstellung ist so auch nicht an feste Seitenverhältnisse gebunden.

AF-4: Target Device: Smartphone und Gyroskop-Steuerung

Da das Spiel vorrangig für die Nutzung auf Smartphones konzipiert wurde, ist Steuerung über die Neigung des Geräts die primäre Kontrolle der Spielfigur. Auch auf die Verwendung von z.B. MouseOver Elementen zur Steuerung des Menüs oder Slider wurde verzichtet, da diese auf Touchscreens nur schwer zu bedienen sind. Auch wurde darauf geachtet, dass das Spiel auch auf kleinen Displays spiel- und darstellbar bleibt. Von der Verwendung auf Smartwatches raten wir jedoch ab.

AF-5: Mobile First Prinzip

Das gesamte Spiel lässt sich über das Tappen des Displays und Neigen des Geräts bedienen. Dies ist die primäre Eingabemethode, jedoch lässt sich das Spiel auch am Desktop nutzen. Die Maus ersetzt hierbei die Toucheingaben und die Pfeiltasten der Tastatur substituieren die Neigung des Geräts

AF-6: Intuitive Spielfreude

Das Simple Spielkonzept und die Steuerung der Spielfigur sollten für den Spieler, sofern er mit der Nutzung moderner Smartphone Applikationen vertraut ist einfach zu erfassen sein. Insbesondere, da auf jede Eingabe ein direktes Feedback erfolgt. Auch wird der Status des Spiels, wie etwa ob das Spiel pausiert oder die Spielfigur verstorben ist durch verschiedene textuell signalisiert. Eine Übersicht der wichtigsten Spielelemente ist in allen Menüs vorhanden und dem Spieler somit jederzeit präsent.

AF-7: Das Spiel muss ein Levelkonzept vorsehen

Da zwischenzeitlich die Realisierung eines externen Leveleditors vorgesehen war, wurde das ursprüngliche Konzept, die Level basierend auf verschiedenen Parametern zur Laufzeit zu generieren, verworfen. Es wurde daher ein Konzept mit „hardgecodeten“-Element umgesetzt, die Beschreibung eines Levels beinhaltet somit auch die exakte Position aller Elemente. Zusätzlich verfügt jedes Level über verschiedene zu erreichende Ziele.

AF-8: Ggf. erforderliche Speicherkonzepte sind Client-seitig zu realisieren

Es werden keine Speicherkonzepte verwendet.

AF-9: Basic Libraries

Es werden nur Libraries verwendet, die zum Standard-Sprachumfang von Dart(2.2) gehören.

AF-10: Keine Spielereien

Auf alle unter „Spielereien“ genannten Elemente wird verzichtet.

AF-11: Dokumentation

Der Quellcode wurde gemäß den Vorgaben und Beispielen kommentiert und ein Wiki in GitLab-Pages angelegt. Diese Dokumentation umfasst Überlegungen und Erläuterungen zum Spielkonzept, der Realisierung sowie das Level- und Parametrisierungskonzept. Der Aufwand der Dokumentation wurde, wie auch der der Implementierung, zu gleichen Teilen verteilt

Architektur und Implementierung

Nachfolgend werden wir die Funktionalität der verschiedenen Programm Bestandteile erklären.

Controller:

Der Controller bringt Bewegung ins Spiel, dafür besitzt er 2 Timer die zeitgesteuert die Berechnung des Models und die Berechnung der View anstoßen. Des Weiteren lauscht der Controller auf Eingaben des Nutzers, ob nun über den Gyrosensor bei einem Smartphone/Tablet oder über die Tastatur bei einem PC/Laptop. So führt eine Neigung des Smartphones zu einer Bewegung des Spielers, die Bewegung kann aber auch mit der Tastatur über die Pfeiltasten erreicht werden.

Die Timer können durch die Funktionen, pauseGame() und startGame() gestoppt oder gestartet werden. Für die Erstellung von Leveln haben wir zwei weitere EventListener hinzugefügt, die lediglich mit einer Tastatur funktionieren. Ein Druck auf die Taste „N“ pausiert das Spiel und ein Druck auf die Taste „M“ startet es wieder. Man sollte nicht mehrmals auf „M“ drücken, dadurch werden weitere Timer erstellt und das Spiel läuft schneller ab. Da es lustig ist haben wir dieses Verhalten aber beibehalten.

Ferner implementiert der Controller diverse EventHandler die für die Knöpfe im Menu zuständig sind.

Aufgrund der Anzahl dieser EventHandler werden nun beispielhaft einige genannt.

```
/// eventhandler to draw level selection
view.levelSel.onClick.listen((_) {
  view.drawLevelMenu();
});

/// eventhandler to draw credits
view.creditsBtn.onClick.listen((_) {
  view.drawCredits();
});

/// eventhandler to draw tutorial
view.howtoBtn.onClick.listen((_) {
  view.drawHowToScreen();
});
```

Nun zur Gyrosteuerung, da unser Model beim Beschleunigen des Spielers Werte von negativ Eins bis positiv Eins erwartet muss die Ausgabe des Gyrosensors angepasst werden, hier erhalten wir Werte von -180 bis 180 entsprechend der Neigung des Handys, zusätzlich wollen wir nicht die ganze Rotation nutzen, da man sich sonst verrenken müsste um spielen zu können. Auch ist ein Bereich indem nichts passiert ganz angenehm.

```
window.onDeviceOrientation.listen((ev) {…};
```

Die maximale Drehung des Smartphones wird durch „range“ definiert, unsere Reichweite ist 30 respektive 30°im echten Leben, d.h. 30° Linksdrehung und 30° Rechtsdrehung sind möglich.

Die Zone, in der nichts passiert heißt „DEADZONE“, sie ist 2° in beide Richtungen.

Der folgende Auszug aus dem Quellcode implementiert dieses Verhalten. Bei Werten größer als 30, oder kleiner als -30, werden die Grenzwerte genutzt, 30 respektive -30.

```
final dx = (gamma.abs() > DEADZONE)
    ? ((gamma.abs() > range)
        ? ((gamma.isNegative) ? -range : range)
        : gamma)
    : 0;
```

Nun ist dx aber definiert als Zahl mit Werten zwischen -30 und 30, das passt nicht zu der beschleunigen Funktion, deswegen teilen wir das noch durch die Range, so kommt ein Wert von -1 bis 1 heraus.

View

Die View ist aufgeteilt in zwei Teile, die View an sich und die Kamera.
Während die View primär für das Darstellen von Menus ist, ist die Kamera zum Darstellen des Spielgeschehens.

Es gibt hier zahlreiche querySelectors, da wir Knöpfe im Menu brauchen, diese Selektoren machen den DOM aus der Index.html für uns zugängig. Wie schon beim Controller nenne ich 3 Beispiele.

```
// querySelector for menu
final menu = querySelector("#menu");

//querySelector for resumeBtn
final resumeBtn = querySelector("#resume");

// querySelector for levelSelector 1
final levelSel = querySelector("#levelSel");
```

Zusätzlich gibt es Funktionen die den DOM Manipulieren um bestimmte Elemente anzuzeigen oder auszublenden. Auch hiervon gibt es mehrere, als Beispiel nennen wir nun eine.

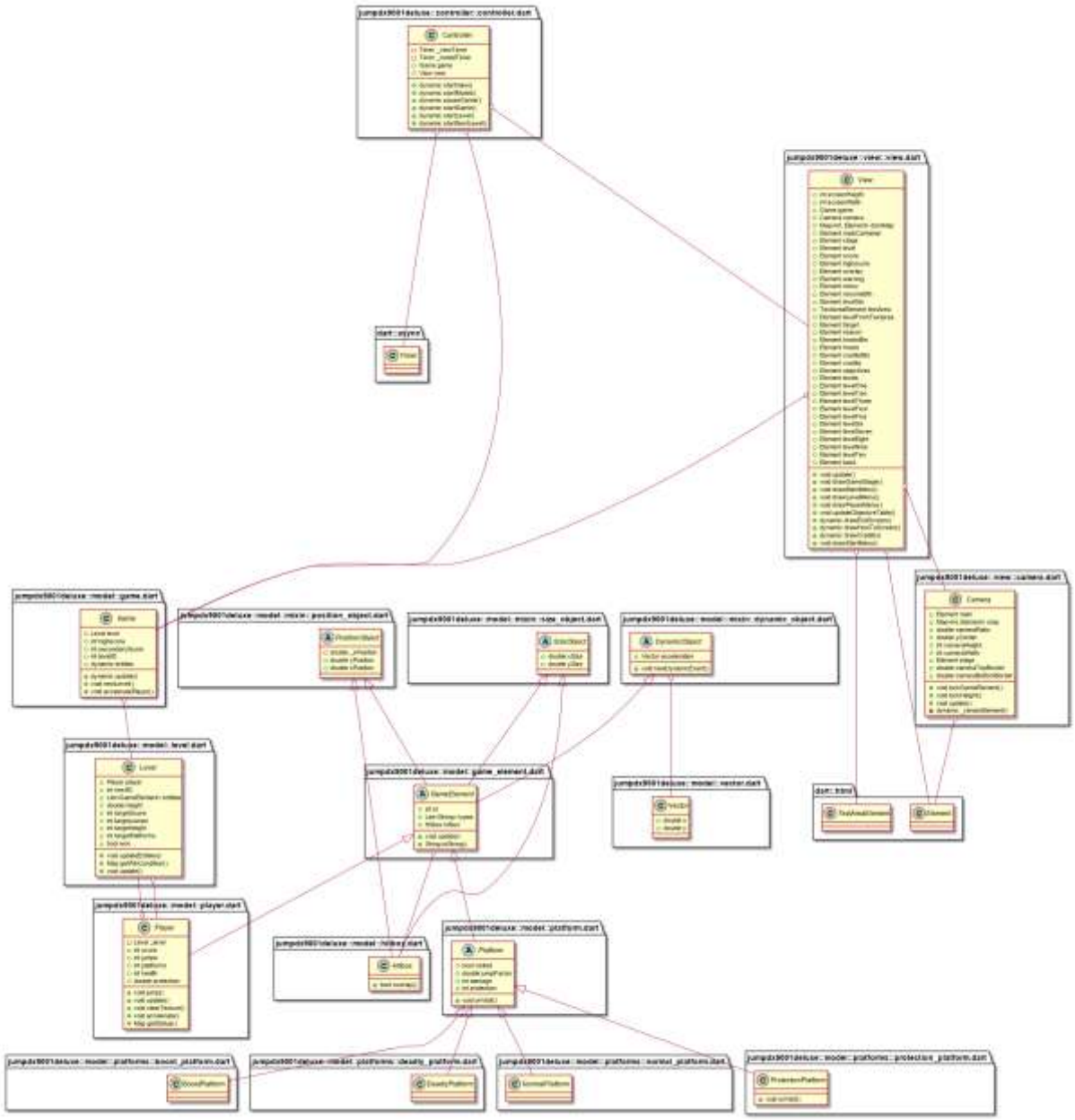
```
/// modifies the DOM to display the level Selection

void drawLevelMenu() {

  levels.style.display = "block";

  menu.style.display = "none";

  stage.style.display = "none";

  overlay.style.display = "none";

}
```

Model

Aus Zeitgründen nur eine Übersicht.

Der Controller hat ein Game, das Game ein Level und Dieses Level ist mit Game Enteties gefüllt.

Siehe hierzu auch UML



Level- und Parametrisierungskonzept

Levelkonzept

Es werden JSON files geladen.

Parametrisierungskonzept

Da wir ein Levelkonzept implementiert haben, mussten wir auch Parameter nutzen.

Wir laden unserer Level aus einem jsonFile, was einer Map entspricht.

```
"winCondition": [  ],
```

Beinhaltet 4 Werte die unsereZiele darstellen

```
  "player": {  },
```

Beinhaltet 3 Werte, 2 davon sind für die Position und einer für die Anzahl der Leben:

```
"xPos": 0.0,

"yPos": 1.0,

"lifes": 1
```

Bei den Plattformen gibt es durchwegs die gleichen Parameter.

Die Plattformen:

```
  "normalPlatform": [  ],

  "boostPlatform": [  ],

  "deadlyPlatform": [  ],

  "protectionPlatform": [  ]
```

Die xPos gibt an wo sich das Objekt auf der x - Achse befindet.
Die yPos gibt an wo sich das Objekt auf der y - Achse befindet.
Die xDim gibt an wie breit das Objekt ist.
Die yDim gibt an wie hoch das Objekt ist.

Bsp.:

```
"xPos": 0.0,

"yPos": 0.0,

"xDim": 0.1,

"yDim": 0.01
```

Nachweis der Anforderungen

Nachfolgend wird erläutert wie die jeweilig im Semester definierten funktionalen Anforderungen, die Dokumentationsanforderungen und die technischen Randbedingungen erfüllt bzw. eingehalten werden.

Nachweis der funktionalen Anforderungen

ID	Kurztitel	Erfüllt	Teilw.	Nicht	Erläuterung
AF-1	Single-Page-App	X			Siehe: Anforderungen und abgeleitetes Spielkonzept
AF-2	Technische Komplexität	X			Siehe: Anforderungen und abgeleitetes Spielkonzept
AF-3	DOM-Tree	X			Siehe: Anforderungen und abgeleitetes Spielkonzept
AF-4	Gyroskop-Steuerung	X			Siehe: Anforderungen und abgeleitetes Spielkonzept
AF-5	Mobile First	X			Siehe: Anforderungen und abgeleitetes Spielkonzept
AF-6	Intuitive Spielfreude	X			Siehe: Anforderungen und abgeleitetes Spielkonzept
AF-7	Levelkonzept	X			Siehe: Anforderungen und abgeleitetes Spielkonzept
AF-8	Speicherkonzepte	X			Sind nicht vorhanden
AF-9	Basic Libraries	X			Es wurden ausschließlich Libraries aus dem Umfang von Dart 2.2 verwendet
AF-10	Keine Spielereien	X			Keine der genannten Elemente wurden verwendet
AF-11	Dokumentation		X		Unvollständig

Nachweis der Dokumentationsanforderungen

Da eine Auflistung nicht bekannt ist orientieren wir uns hier an SnakeDart

ID	Kurztitel	Erfüllt	Teilw.	Nicht	Erläuterung
D-1	Projektdokumentation		X		Vorliegende Dokumentation erläutert die übergeordneten Prinzipien und verweist an geeigneten Stellen auf die Quelltextdokumentation. Ist jedoch unvollständig.
D-2	Quelltextdokumentation	X			Es wurden alle Methoden und Datenfelder, Konstanten durch Inline-Kommentare erläutert.
D-3	Libraries	X			Alle genutzten Libraries werden in der pubspec.yaml der Implementierung aufgeführt. (Annahme: da aus Barebones übernommen) Da nur die zugelassenen Pakete genutzt wurden, sind darüber hinaus keine weiteren Erläuterungen, warum welche Pakete genutzt wurden, erforderlich.

Nachweis der Einhaltung technischer Randbedingungen

Da eine Auflistung nicht bekannt ist orientieren wir uns hier an SnakeDart

ID	Kurztitel	Erfüllt	Teilw.	Nicht	Erläuterung
TF-1	No Canvas	X			Es werden keinerlei Canvas basierte Elemente genutzt
TF-2	Levelformat	X			Die einzelnen Level werden aus vorliegenden JSON Dateien geladen
TF-3	Parameterformat	X			Parameter werden in Constants.dart zusammengefasst und durch Levelspezifische Einträge in den Level-Dateien ergänzt
TF-4	HTML + CSS	X			Der View des Spiels beruht ausschließlich auf HTML und CSS (vgl. web/index.html). Es werden PNG Grafiken eingebunden.
TF-5	Gamelogic in Dart	X			Die Logik des Spiels ist mittels der Programmiersprache Dart realisiert worden.
TF-6	Browser Support	X			Alle erforderlichen Browser werden unterstützt, auf Browser spezifische Elemente (etwa CSS Funktionalitäten wurde verzichtet)
TF-7	MVC-Architektur	X			Das Spiel folgt durch Ableitung mehrerer Modell-Klassen (vgl. lib/model/), zweier View Klassen (vgl. lib/view/view.dart) und dem zentralen Controller (vgl. lib/controller/controller.dart) einer MVC-Architektur. Der View greift zudem auf das Model nur lesend und nicht manipulierend zu.
TF-8	Erlaubte Pakete	X			Es sind nur dart:* packages verwendet worden.
TF-9	Verbotene Pakete	X			Es wurden ausschließlich Libraries aus dem Umfang von Dart 2.2 verwendet
TF-10	No Sound	X			Das Spiel hat keine Soundeffekte.

f

Verantwortlichkeiten im Projekt

Komponente	**Detail**	**Assets**	**Milan Stuhlsatz**	**Marius Mauritz**	**Anmerkung**
Model	vollständig	/lib/model/	V		beinhaltelt Inline Dokumentation
View	vollständig	/lib/view/	U	V	beinhaltelt Inline Dokumentation
Controller	vollständig	/lib/controller		V	beinhaltelt Inline Dokumentation
CSS+HTML	Zu gleichen teilen, häufig kooperativ		U	U	Zu gleichen teilen, häufig kooperativ
Grafiken			V	U	
Dokumentation			V	U	unvollständig

V = verantwortlich (hauptdurchführend, kann nur einmal pro Zeile vergeben werden)
U = unterstützend (Übernahme von Teilaufgaben)