

# Australia Simulator

---

# 0 - Verzeichnis

---

## 0.1 Struktur

- 0. Verzeichnis
  - 0.1 Struktur
  - 0.2 Links
- 1. Einleitung
- 2. Anforderung und Konzept
  - 2.1 Spielkonzept: Australia Simulator
  - 2.2 Anforderungen
- 3. Architektur
  - 3.1 Model
  - 3.2 View
  - 3.3 Controller
    - 3.3.1 LevelManager
- 4. Level- und Parametrisierungskonzept
  - 4.1 Levelkonzept
  - 4.2 Parametrisierungskonzept
- 5. Libraries
- 6. Nachweis der Anforderungen
  - 6.1 Nachweis der funktionalen Anforderungen
  - 6.2 Nachweis der Dokumentationsanforderungen
  - 6.3 Nachweis der Einhaltung technischer Randbedingungen
  - 6.4 Verantwortlichkeiten im Projekt

## 0.2 Links

- Programmdokumentation: <https://izedx.github.io/australia-simulator/doc/api/australiasim/australiasim-library.html>

# 1 - Einleitung

---

Australia Simulator ist ein Mobile-First Einzelspieler Webgame von Niklas Kührtmann und Thomas Urner für das Modul "Webtechnologie Projekt" im SoSe 2018 an der Fachhochschule-Lübeck.

Diese Dokumentation soll einen Überblick über das Konzept und die Architektur leisten, wobei genauere Details in der [Programmdokumentation](#) zu finden sind.

Außerdem sind hier die Anforderungen dokumentiert, und wie diese gelöst werden.

## 2 - Anforderungen und Konzept

---

### 2.1 - Anforderungen

<b>Id</b>	<b>Anforderung</b>
AF-1	Single-Player-Game als Single-Page-App
AF-2	Balance zwischen technischer Komplexität und Spielkonzept
AF-3	Target Device: SmartPhone
AF-4	DOM-Tree-basiert
AF-5	Mobile First Prinzip
AF-6	Das Spiel muss schnell und intuitiv erfassbar sein und Spielfreude erzeugen.
AF-7	Das Spiel muss ein Levelkonzept vorsehen
AF-8	Ggf. erforderliche Speicherkonzepte sind Client-seitig zu realisieren
AF-9	Dokumentation

<b>Id</b>	<b>Dokumentationsanforderung</b>
D-1	Dokumentationsvorlage
D-2	Projektdokumentation
D-3	Quelltextdokumentation
D-4	Libraries

<b>Id</b>	<b>Technische Randbedingungen</b>
TF-1	Statische HTML-Single-Page-App
TF-2	Alle Ressourcen relativ adressiert
TF-3	MVC-Architektur
TF-4	DOM-Tree als View / No-Canvas
TF-5	Mobile First
TF-6	SmartPhone Bedienung
TF-8	Level-Parametrisierung per JSON/XML/etc.
TF-9	Client-seitige Speicherung

## 2.2 - Spielkonzept: Australia Simulator

Das Spiel Australia Simulator ist ein top-down Arcade Game, bei welchem der Spieler versuchen muss, Spinnen aus seinem Haus zu verscheuchen.

Dabei sieht der Spieler seinen Character in der Mitte des Bildschirms und kann sich in einem 360° Radius bewegen, abhängig davon, wohin der Spieler mit dem Finger zeigt.

Im Haus laufen abhängig vom derzeitigen Level unterschiedlich viele und schnelle Gegner rum, welche - wenn der Spieler zu nah kommt - versuchen werden vor diesem abzuhaufen. Das Ziel ist es, alle dieser Gegner aus dem Haus zu verscheuchen, bevor sie ihren Wohlfühl-Faktor voll aufgefüllt haben (sich anfangen heimisch zu fühlen). Dabei kann der Spieler den Wohlfühl-Faktor eines Gegner senken, indem er diesen vor sich her scheucht. Hat der Spieler alle Gegner vertrieben, so hat er das Level bestanden und das nächste Level kann geladen werden.

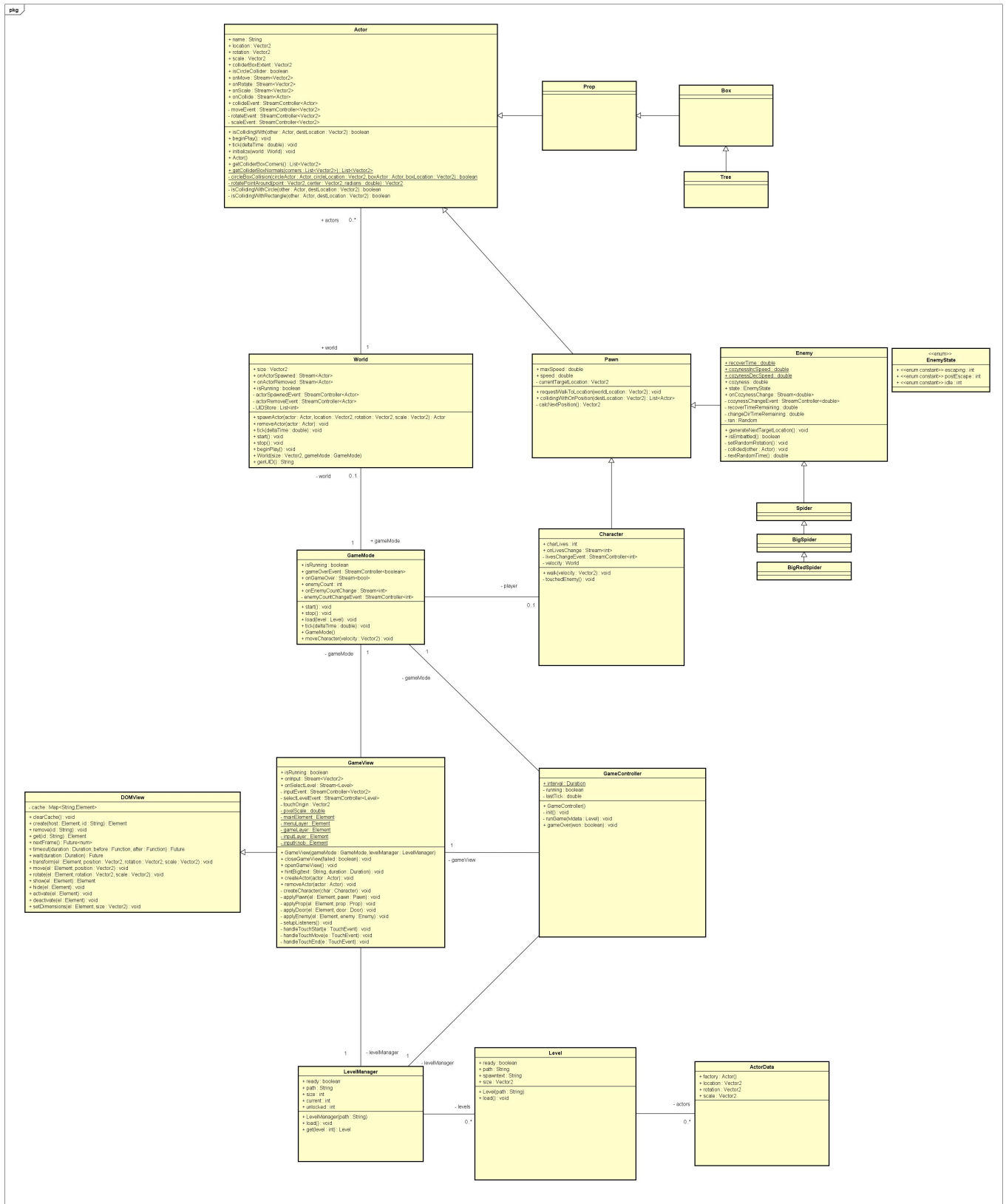
Der Spieler ist allerdings nicht unverwundbar, pro Level hat er zwei Leben. Er verliert ein Leben, wenn er aus Versehen einen Gegner berührt, wobei die Gegner allerdings niemals direkt angreifen werden. Hier kämpft jeder um's Überleben.

Sollte nun entweder ein Gegner seinen Wohlfühl-Faktor voll aufgefüllt haben, oder der Spieler alle seine Leben verlieren, so endet das Spiel und der Spieler hat verloren. Er kann das Level nun erneut versuchen.

## 3 - Architektur

---

Australia Simulator folgt der MVC-Architektur, bei der wir Interaktionen und Ausgaben/View vom Model trennen. Im Folgenden werden die Basisklassen erläutert; detailliertere Dokumentation der Implementationsdetails liegen dieser Abgabe als dartdoc bei (./doc/api), bzw. sind unter 0.2 Links zu finden.



### 3.1 - Model

Das Grundgerüst des Models beschränkt sich im Allgemeinen auf drei Klassen; GameMode, World und Actor.

Das GameMode steuert den Spielablauf. Zu jeder Zeit kommuniziert der GameController nur mit dem GameMode, welches das Spiel verwaltet und die Interaktionen der Nutzer an seine Spielfigur weiterleitet. Es implementiert die Spielregeln und leitet den Model-Tick (hot loop; welcher für u.a. Bewegungsberechnungen benutzt wird) von dem GameController an die World weiter. Zu Beginn eines Spieles wird die Welt von dem

GameMode aufgebaut (Props, Gegner und Player spawnen); zum Ende wird ein GameOver Event emittiert auf welches der GameController reagiert

Die World bezeichnet das Objekt, welches die Spielwelt repräsentiert. Sie wird benutzt um jede Art von Actors in das laufende Spiel zu bringen / entfernen und hält eine Liste der aktuell im Level existierenden Actors. Weiterhin gibt sie den Model-Tick an alle Actors weiter.

Actor dient als Basisklasse für alle Objekte, die in einer World existieren (Props, Pawns) und mit der Spielfigur interagieren können. Sie implementiert die abstrakten Basismethoden (tick, initialize, beginPlay), welche in den Childclasses überschrieben werden, die Worldtransforms, sowie Kollisionsabfragen für Box- und Circleprimitives.

Pawn implementiert das Movement, welches die Spielfigur, unter Berücksichtigung von möglichen Kollisionen auf dem Weg, auf dem direkten Weg zu einer Position laufen lässt.

Enemy implementiert ein zufälliges Movement für alle Gegner, sowie die „Fliehfunctionalität“.

Ein Character ist die Spielfigur, die der Spieler steuert. Sie reagiert auf Kollisionen mit Gegnern und beinhaltet die aktuellen Leben des Spielers.

### 3.2 - View

Der View wird zur Darstellung des Models verwendet und vom Controller initiiert. Er erstellt die DOM-Elemente die für das Model benötigt werden und reagiert auf Updates des Models zur Aktualisierung.

Wir verwenden im View ein DOM-Element für die 2D Welt - dem Haus - in welchem wir dann die Actor frei bewegen können.

Das Spiel verwendet kein festes Grid in dem Spielobjekte von Zelle zu Zelle verschoben werden können, sondern stattdessen lose - absolute positionierte - Elemente, die frei in der Spielwelt bewegt und animiert werden können.

Dies hat außerdem zur Folge, dass die HTML Elemente nicht immer komplett geupdated werden müssen, sondern lediglich - sobald benötigt - manipuliert. Mithilfe von **will-change** lassen sich Performance-relevante Elemente (z.B. Spielwelt und Gegner) außerdem in ihrer Position verändern, ohne einen Re-Render der HTML Engine zu erzwingen. Das Positionsupdate wird hierbei von der GPU übernommen.

Im View soll der Character immer in der Mitte des Bildschirms dargestellt werden, hierfür wird er im View fest in der Mitte des Bildschirms erstellt und wenn er sich bewegt wird die Welt im Hintergrund in Relation zum Character bewegt, statt ihm selber.

Um dies auch mit Überblick zu bewerkstelligen basiert der GameView, welcher bestimmt, wie genau das Spiel dargestellt wird, auf dem DOMView, eine Parentklasse, die häufige DOM-Operation abstrahiert und außerdem ein Verzeichnis für die DOM-Elementreferenzen anbietet.

Der GameView reagiert auf ActorSpawn bzw. Remove Events der World und erstellt bzw. entfernt die jeweiligen DOM-Elemente. Weiterhin hört er auf Transform-Events, die von den gespawnten Actors emittiert werden und passt die jeweiligen Transforms im DOM an. Weiterhin stellt er den View der Menüs dar und emittiert Steuerungsevents auf die der GameController reagiert.

### 3.3 - Controller

Der GameController ist das zentrale Nervensystem des Spiels, er erstellt das Model und den View und verbindet diese miteinander. Der Controller steuert den groben Ablauf des Spieles (Wechsel zwischen Menü und Gameplay), initialisiert das GameMode der aktuellen Session und tickt das Model. Er horcht auf die Eingaben des Spielers zur Steuerung der Spielfigur und gibt sie entsprechend an das Model weiter.

### 3.3.1 LevelManager

Der LevelManager, sowie seine Hilfsklassen Level und ActorData werden verwendet um als JSON vorliegende Levels zu parsen und ihre Informationen dem GameView und GameController zur Verfügung zu stellen.

## 4 - Level- und Parametrisierungskonzept

---

### 4.1 - Levelkonzept

Das Levelkonzept von Australia Simulator ist - wie in Punkt 3.3.1 beschrieben - komplett in die `level.dart` abstrahiert. Dort ist der LevelManager zu finden, welcher eine JSON-Datei einliest mit eine Liste von Leveln und Verweisen auf deren JSON-Dateien. Für jedes Level in der Liste lädt er dann ebenfalls die JSON-Datei, und fügt sie dann zur Levelliste hinzu.

Ein Auszug aus der `levels.json`:

```
[
  {
    "path": "./assets/data/levels/0-welcome-home.json"
  },
  {
    "path": "./assets/data/levels/1-this-is-my-house.json"
  },
  ...
]
```

Jedes Level enthält dann einen Text, welcher beim Start angezeigt werden soll (`spawnText`), eine Größe, wie groß das Level sein soll (`size`) und eine Liste mit Actors, welche erstellt werden sollen (`actors`). Jeder Actor kann dabei durch seinen `type` bestimmt werden und mit ``

Hier noch ein Auszug aus einem Level:

```
{
  "spawnText": "Welcome Home!",
  "size": [1200.0, 900.0],
  "actors": [
    {
      "type": "spider",
      "location": [600.0, 700.0]
    },
    {
      "type": "table",
      "location": [300.0, 300.0],
    }
  ]
}
```



```

        "rotation": [0.1, 0.9]
      },
      {
        "type": "flower",
        "location": [300.0, 250.0],
        "rotation": [0.05, 0.95]
      }
    ]
  }

```

## 4.2 - Parametrisierungskonzept

Es gibt keine externe/datei-basierte Konfigurationsmöglichkeit für Australia-Simulator, bis auf die Level-Daten welche im vorigen Punkt erläutert werden.

Allerdings gibt es zwei relevante Konstanten die im Programmcode bearbeitet werden können:

- In `gamecontroller.dart`: `interval` beschreibt den tick interval, indem updates im Model ausgeführt werden sollen.
- In `gameview.dart`: `pixelScale` beschreibt den Skalierungsfaktor, mit dem zwischen den Einheiten im Model und Pixeln auf dem Screen umgewandelt wird.

## 5 - Libraries

---

Wir verwenden für Australia Simulator zwei externe Dependencies: `vector_math` und `rxdart`, wobei `vector_math` die `Vector2`-Klasse zur Verfügung stellt, welche wir im gesamten Projekt verwenden.

`rxdart` allerdings bietet uns erweiterte Dart-Streams mit denen wir - insbesondere was zeitliche Operationen angehen - mehr Möglichkeiten haben.

Zur Veranschaulichung ein Beispiel aus der `model/pawns/character.dart`, wie `rxdart` in Australia Simulator zum Einsatz kommt.

```

new Observable(this.onCollide)
  .where( (Actor a) => a is Enemy )
  .throttle( new Duration(seconds: 1) )
  .listen( (Actor a) => this._touchedEnemy());

```

## 6 - Nachweis der Anforderungen

### 6.1 Nachweis der funktionalen Anforderungen

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

*Australia Simulator* ist ein Singleplayer-Spiel als Single-Page-App, welches in Dart entwickelt wurde. Alle Assets sind relativ adressiert und es wird kein Backend benötigt. Jeglicher Fortschritt wird direkt im Browser des Users gespeichert und somit kann *Australia Simulator* statisch vertrieben werden.

**AF-2: Balance zwischen technischer Komplexität und Spielkonzept - ERFÜLLT**

In *Australia Simulator* muss der Spieler versuchen verschiedene gefährliche Tiere aus seinem Haus zu verscheuchen, ohne dabei die Tiere zu berühren, ansonsten verliert der Spieler ein Leben und hat dieser keine Leben mehr, so hat er verloren. Die Gegner fliehen also entsprechend vor dem Spieler oder bewegen sich zufällig rotationsbasiert im Haus (s. *Enemy.dart*); es wird keine komplexe KI für eine vergleichbar gute Spielerfahrung benötigt. Die Bedienung erfolgt intuitiv mittels eines simulierten Analogsticks, welcher bei einer Berührung des Bildschirms hervorgehoben wird.

**AF-3: DOM-Tree basiert - TEILWEISE ERFÜLLT (s. Begründung)**

*Australia Simulator* verwendet ein MVC Model, bei dem der DOM-Tree das View darstellt und entsprechend das Spiel rendert und auf Events im Model reagiert (vgl. 3.; UML).

Bei dem LevelManager wird von dem MVC Pattern abgewichen (*gameview.dart*, *gamecontroller.dart*), weil sich eine Anbindung an den Controller und den View analog zum Model sehr gut eignet ohne unnötige Umwege für die View Klasse zu nehmen (vgl. 3.; UML), jedoch betrachten wir ihn als Controller Klasse, da Input in Form der vorliegenden Level-JSONs stattfindet. Zur Initialisierung verwendet der GameController die Informationen aus dem LevelManager um das GameMode mit den bereitgestellten Levelinformationen zu starten (vgl. *\_runGame()* in *gamecontroller.dart*).

**AF-4: Target device: Smartphone ERFÜLLT**

*Australia Simulator* ist eine mobile-first Single-Page-App und somit auf iOS und Android, sowie in modernen HTML5 Browsern gleich funktionieren. Als device-agnostic Eingabemethode haben wir uns eine Touch-and-Move Steuerung ausgedacht, bei der der Spieler mit der Maus oder per Touchscreen auf dem Bildschirm ziehen kann und der Character bewegt sich entsprechend. Sollte der Spieler wieder los lassen bleibt der Character stehen.

**AF-5: Mobile First Prinzip ERFÜLLT**

Die Optik, Menüs und die Steuerung (vgl. AF-2) sind bewusst so entworfen, dass sie sich gut zum lageunabhängigen Spielen mit einem Smartphone eignen. Auf einem Desktop PC ist eine Eingabe mit der Maus, analog zum Touchinput, möglich.

**AF-6: Das Spiel muss schnell und intuitiv erfassbar sein und Spielfreude erzeugen - ERFÜLLT**

Das gesamte Spiel kann mit nur einer einzigen Eingabemethode gespielt werden (vgl. AF-2) und ist entsprechend intuitiv aufzugreifen. Spätestens nach einer verlorenen Runde der jeweiligen Lose-Bedingungen (Ein Gegner erreicht vollen Wohlfühlfaktor bzw. Spieler verliert alle Leben) sollte das Spielekonzept offensichtlich sein. Wie viel Spielspaß dieses Spiel am Ende tatsächlich erzeugen kann und wie lange das Spiel diesen aufrecht erhalten kann, muss in einem späteren Schritt eventuell nachbalanciert werden, bzw. hängt von den Präferenzen, der Spieler ab.

**AF-7: Das Spiel muss ein Levelkonzept vorsehen - ERFÜLLT**

*Australia Simulator* bietet mehr als sieben im Schwierigkeitsgrad aufsteigende Levels an, welche extra in JSON vorliegen und konfiguriert werden können, wobei das jeweilige Level in der Größe festgelegt werden kann.

Weitere Einstellungsmöglichkeiten für die Level beinhalten: Anzahl der Gegner, Art der Gegner und die zu dem Level gehörigen Props (s. level.dart, gamecontroller.dart).

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

Der Stand der freigeschalteten Levels wird im HTML5 Web Storage (localStorage) des Clients abgespeichert (s. level.dart).

#### **AF-9: Dokumentation**

Gegeben.

##### 6.2 Nachweis der Dokumentationsanforderungen

TODO

##### 6.3 Nachweis der Einhaltung technischer Randbedingungen

TODO

##### 6.4 Verantwortlichkeiten im Projekt

TODO