

# 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 Input
```

## 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 Kühnmann 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.

## 2 - Anforderungen und Konzept

### 2.1 - Spielkonzept: Australia Simulator

Das Spiel Australia Simulator ist ein top-down Arcade Game, bei welchem der Spieler versuchen muss, die wilden Tiere aus seinem Haus zu verscheuchen, bevor diese sich einnisten und die Zeit abläuft oder der Spieler von denen verletzt wird.

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 vertreiben, bevor sie ihren "Coziness"-Balken auffüllen (sich anfangen heimisch zu fühlen). Hat der Spieler alle Gegner vertreiben, so hat er das Level bestanden und das nächste Level kann geladen werden.

### 2.2 - Anforderungen

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

*Australia Simulator* ist ein Singleplayer-Spiel als Single-Page-App, das 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

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 im Haus.

Mit der Zeit erhöht sich der Wohlfühl-Faktor der Gegner, der Spieler kann diesen aber reduzieren, indem er die Gegner scheucht. Erreicht ein Gegner seinen vollen Wohlfühl-Faktor, so hat der Spieler verloren.

Schafft es der Spieler aber vorher alle Gegner aus dem Haus zu verscheuchen, so hat der Spieler gewonnen und das nächste Level kann geladen werden.

Die gesamte Gamelogik läuft in Echtzeit mit pixel-genauen Positionen und einer zentrierten Kamera über dem Spieler.

## AF-3: DOM-Tree basiert

*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.

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. Genauere Informationen zur Verwendung von **will-change** in unserem View finden sich 3.2

## AF-4: Target device: Smartphone

*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.

Diese Eingabemethode ähnelt einem virtuellen Joy-/Analogstick auf dem Bildschirm.

## AF-5: Mobile First Prinzip

Für erweiterte UI Interaktionen, wie z.B. das wegdrücken von Modals am Anfang einer Runde bieten sich neben klassischen Eingabearten (Close-Button obere rechte Ecke vom Widget) außerdem Touchscreen Interaktionen an, wie das Wegswipen vom Widget. Da wo es angebracht ist, wollen wir die Interaktionen für Touchscreens vereinfachen.

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

Das gesamte Spiel kann mit nur einer einzigen Eingabemethode gespielt werden und ist entsprechend intuitiv aufzugreifen. Wie viel Spielspaß dieses Spiel am Ende tatsächlich erzeugen kann und wie lange das Spiel diesen aufrecht erhalten kann, muss in einem späteren Schritt balanciert werden.

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

*Australia Simulator* wird mindestens sieben im Schwierigkeitsgrad aufsteigende Level anbieten, welche extra konfiguriert werden können, wobei die jeweilige Karte in der Größe festgelegt werden kann. Weitere Einstellungsmöglichkeiten für die Level beinhalten: Anzahl der Gegner, Art der Gegner, Geschwindigkeit der Gegner und dem Wohlfühl-Faktor der Gegner, sowie der zugehörigen Steigungsrate.

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

Es ist vorgesehen den Levelfortschritt im localStorage abzuspeichern, damit der Spieler dort weitermachen kann, wo er aufgehört hat. Weitere Daten, wie Einstellungen für das Spiel etc. können ebenfalls im localStorage abgespeichert werden.

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

Gegeben.

### **3 - Architektur**

Australia Simulator folgt der MVC-Architektur, bei der wir Interaktionen und Ausgaben/View vom Model trennen. Entscheidende Rolle spielt hierbei der GameController, da er View und Model erstellt und auf User-Interaktionen und Timern horcht und diese an das Model weiterleitet.

Der GameView ist für DOM-Manipulation vorgesehen, deswegen erstellt er das Spielfeld und horcht auf Model-Events um den DOM-Tree up-to-date zu halten.

Das Model des Spiels besteht aus mehreren Schichten an Klassen und orientiert sich an bekannten Spiele-Engines wie z.B. der Unreal Engine, allerdings sehr stark auf die Bedürfnisse für dieses Projekt zugeschnitten.

So wird zu Beginn eines Spieles die Welt von dem Gamemode aufgebaut (Architektur / Props, Gegner, Player spawnen). Zu jeder Zeit kommuniziert der GameController nur mit dem GameMode, welches das Spiel

verwaltet und die Interaktionen der Nutzer an seine Spielfigur weiterleitet.

Während das Spiel läuft werden alle beweglichen Objekte (Spielfiguren) "getickt" (hot loop) um die nötigen Berechnungen zu machen, die für das Spiel benötigt werden (positionsupdate, collision, ...).

Zu einem Levelwechsel wird zuerst das aktuelle Level gecleared und dann durch eine neue World ersetzt.

## 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 dem gespielt wird - in welchem wir dann die Actor frei bewegen können.

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

## 3.3 - Controller

Der Controller ist das zentrale Nervensystem des Spiels, er erstellt das Model und den View und verbindet diese miteinander. Er horcht auf die Eingaben des Spielers und updated das Model entsprechend.

### 3.3.1 Input

Im Controller horchen wir auf den Input und leiten diesen dann an das Model weiter, die entsprechende Eingabeposition die weitergeleitet wird ist relativ zur Position des Characters, also relativ zur Mitte des Bildschirms.

```
onInput(onInput(Vector2 worldPos), onInputStop()) {
  relay(TouchEvent e) {
    e.preventDefault();
    onInput(new Vector2(
      e.touches[0].page.x - view.world.offset.left,
      e.touches[0].page.y - view.world.offset.top
    ));
  }

  view.input.onTouchStart.listen((e) {
    relay(e);
  });

  view.input.onTouchMove.listen((e) {
    relay(e);
  });

  view.input.onTouchEnd.listen((e) {
    e.preventDefault();
    onInputStop();
  });
}
```

```
    });  
}
```

Events vom Model an das View muss der Controller nicht selbst weiterleiten, da der View direkt auf den Events des Models horchen kann und sich so up-to-date halten kann. Allerdings ist der Controller dafür verantwortlich weiteren User-Input zu verarbeiten, wie z.B. das Starten eines Spiels und das entsprechende Setup des Views und Models.