Webtechnologie Projekt

1 Einleitung

Das Spiel Attack on Malda welches vorrangig auf Mobilegeräten im Browser laufen soll, ist aus dem Konzept von Star Fox x64 entstanden.
Dabei soll der Spieler über die Gyrosteuerung ein Fadenkreuz und Raumschiff lenken und Gegner abschießen.

Die Gyroausrichtung des Mobilegerätes steuert vorrangig das Fadenkreuz, welches frei auf dem Bildschirm zu bewegen ist. Das Raumschiff („Die Spielfigur“), bleibt dauerhaft am unteren Rand des Gerätes. Dieser bewegt sich nur auf der Horizontalen mit dem Fadenkreuz. Durch das Tippen auf das Display wird ein Schuss losgelassen, der von der aktuellen Position des Spielers gerade nach oben fliegt. Alle Buttons werden ebenfalls durch tippen bedient.

In der Desktopversion wird das Fadenkreuz mit WASD gesteuert und Schüsse werden mit der Leertaste abgeschossen. Die Buttons werden mit der Maustaste/dem Mauszeiger bedient.

Des weitern wird dem Spieler auf einer Minimap angezeigt, wo sich Gegner befinden.

2 Anforderungen und abgeleitetes Spielkonzept

Spielprinzip
Der Spieler wird auf einer 2-Dimensionalen (x*y) Karte erstellt die Gegner werden über eine JSON Datei in das Level geladen und kommen von oben auf den Spieler zu.
Die Aufgabe besteht darin, dass der Spieler alle Gegner zerstören oder ausweichen muss.

Attack on Malada kann auf vielfältige Arten variiert werden.
- Neue Gegner mit eigenem Verhalten.
- Anzahl der Gegner kann variiert werden.
- Leben und Schaden können vom Spieler verändert werden.
- PowerUps können eingebaut werden (unbesiegbar, mehr Schaden, Heilung).

Die nachfolgende Architektur ist so gestaltet, dass solche Variationen nachträglich ergänzt werden könnten.

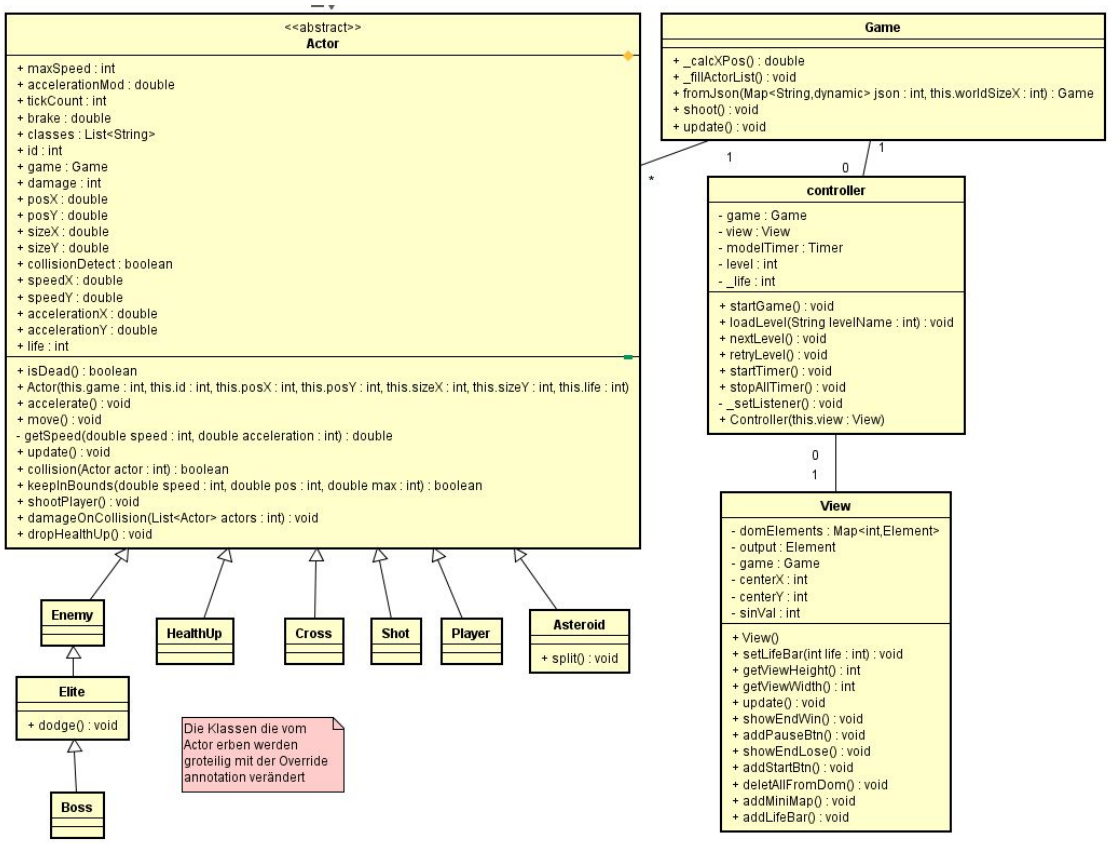

Abbildung 1 zeigt die Architektur von Attack on Malada im Überblick. Die Architektur folgt dem bewährten Model-View-Controller Prinzip. Softwaretechnisch gliedert sich die Spiellogik so in mehrere Komponenten (Klassen) mit spezifischen funktionalen Verantwortlichkeiten.

3 Architektur und Implementierung

3.1 Model

Im Groben lässt sich das Spiel in vier Entities aufteilen (Player, Cross, Enemy, Game), allerdings wird dieses Prinzip zugunsten der Erweiterbarkeit ausgebaut. Es existieren etwa auch abstrakte Klassen, welche das spätere Abbilden erleichtern.

3.1.1 Actor-Klassen

Actor-Klassen sind in Attack on Malada alle 4 Gegnertypen (Asteroiden, Enemy/Casual, Elite und Boss) sowie der Spieler, das Fadenkreuz, die Schüsse und PowerUps.

- Actor haben generell eine Geschwindigkeit, Beschleunigung, Id, Schaden, X- und Y-Koordinaten und eine Höhe und Breite
- Des weiteren haben sie Funktionen wie accelerate, move, update, collision, shoot, damageOnCollision und dropHealthUp
- Diese sind je nachdem in welcher Klasse man sich befindet manchmal überschrieben, um spezielleres Verhalten zu ermöglichen

3.1.2 Constants

Constants sind lediglich zwei festgelegte Konstanten, die für den Spielfluss benötigt werden, diese wären:

- Tick - Ist die Wiederholungsrate des Timers in unserem Spiel
- Growth - Ist die Wachstumsrate für Gegner (Hierdurch erweiterbar für andere Klassen, wobei das gerade nur beim Asteroiden angewendet wird)

3.2 View

In unserer View haben wir alle Funktionen, die Elemente auf dem Bildschirm erzeugen. Beispielsweise unsere Endscreens, Startscreen, aber auch die Minimap, die einzig und alleine hier erstellt und behandelt wird. Außerdem können wir hierüber alle Elemente aus dem Dom-Tree löschen und die View updaten (Positionen, Gegner entfernen etc.)

3.3 Controller

Eine zentrale Rolle für die Spielsteuerung hat der Controller.
Der Controller kann:

- Nutzerinteraktionen erkennen und verarbeiten
- Level laden und erstellen
- Zeitsteuerung, um regelmäßig die Anzeige zu erneuern und ein Gegnerverhalten darzustellen
- Das Spiel starten und die Steuerung richtig umsetzen
- Anweisungen an die View geben, damit die Ausgabe dem aktuellen Spielstand entspricht

4 Level und Parametrisierungskonzept

4.1 Levelkonzept

Das Levelkonzept gibt folgendes für das Spiel vor. Wir haben jedes Level gleich aufgebaut mit folgenden Attributen und Inhalten:

- Fortschritt - Ist ein Integer, der dem aktuellen Level entspricht
- Name - Ist der Name des Levels
- worldSizeY - Ist die Höhe unseres Levels, diese geht über das sichtbare Spielfeld hinaus, damit die Gegner von oben reinfliegen können und man sie vorher bereits auf der Minimap erkennen kann

- actorList - Ist die Liste aller Gegnertypen. Hier stehen nur die Bezeichnungen drin für den jeweiligen Gegnertyp
- posXList - Ist die Liste der Faktoren für die X-Positionen der Gegner. Hier werden nur Faktoren zwischen 0.0 und 1.0 verwendet, da diese später mit der variablen Bildschirmbreite multipliziert werden, um bei jedem Nutzer die Bildschirmbreite voll zu nutzen
- posYList - Ist die Liste der Y-Positionen. Diese sind fest und werden so aus der JSON übernommen
- sizeXList - Ist die Liste, die die Breiten der Gegner enthält
- sizeYList - Ist die Liste, die die Höhen der Gegner enthält
- healthList - Ist die Liste, die das Leben der Gegner vorgibt
- heavyList - Ist die Liste, die vorgibt, ob ein Gegner das Heavy-Attribut bekommt oder nicht (Heavy bewirkt Schaden und Leben sind verdoppelt)
- damageList - Ist die Liste, die den Schaden der Gegner vorgibt

Aus diesen Daten bauen wir das Level, wobei aus jeder Liste der Eintrag an Stelle x genommen wird, um daraus einen Gegner zu bauen. Außerdem kommen bis Level 3 jedes Level 2 Gegner hinzu und die Levelhöhe wird um 250 erhöht. Ab Level 3 passiert dies nur noch alle 2 Level.

Die steigende Schwierigkeit besteht darin, dass im Laufe des Spiels mehr Gegner auftauchen und dabei die Anzahl der stärkeren Gegner zunimmt. In Level 10 trifft das Ganze auf seinen Höhepunkt, in dem noch ein Boss erscheint. Zudem wird das Spiel mit jedem hinzukommenden Gegnertyp schwerer, da diese immer mehr eigenes Verhalten und Gefahr für den Spieler mit sich bringen. Zuletzt lassen sich die vorhandenen Gegner immer noch mit dem Heavy-Attribut buffen, um es dem Spieler noch schwieriger zu machen.

Ein Beispiel für ein solches Level würde im JSON Format so aussehen:

```
{
  "fortschritt": 1,
  "name": "Level 1",
  "worldSizeY": 1500,
  "actorList":
  [
    "asteroid","asteroid","asteroid","asteroid",
    "asteroid","asteroid","asteroid","asteroid"
  ],
  "posXList": [
    0.2,0.9,0.1,0.32,
    0.88,0.76,0.5,0.25
  ],
  "posYList" : [
    1500,1350,1150,1000,
    900,1450,1100,980
  ],
  "sizeXList" : [
    32,60,64,32,
    32,20,60,32
  ],
  "sizeYList" : [
    32,60,64,32,
    32,20,60,32
  ],
  "healthList" : [
    1,1,1,1,
    1,1,1,1
  ],
  "heavyList": [
    false,false,false,false,
    false,false,false,false
  ],
  "damageList" : [
    1,1,1,1,
    1,1,1,1
  ]
}
```

4.2 Parametrisierungskonzept

Im Controller (lib/controller/controller.dart) werden diese Festlegungen über das Game (lib/model/game.dart) eingelesen. Siehe 4.1 für eine Beschreibung der Parameter.

Das Einlesen erfolgt mit folgender Funktion:

```
Game.fromJson(Map<String, dynamic> json, this.worldSizeX) {
  fortschritt = json['fortschritt'];
  name = json['name'];
  worldSizeY = json['worldSizeY'];

  cross = Cross(this, currentEntityID++, 300, 100);
  player = Player(this, currentEntityID++, cross.posX, 20);
  player.cross = cross;
  _fillActorList(
      json['actorList'].cast<String>(),
      json['posXList'].cast<double>(),
      json['posYList'].cast<double>(),
      json['sizeXList'].cast<double>(),
      json['sizeYList'].cast<double>(),
      json['healthList'].cast<int>(),
      json['heavyList'].cast<bool>(),
      json['damageList'].cast<int>());
  actors.addAll([player, cross]);
}
```

5 Nachweis der Anforderung

5.1 Nachweis der funktionalen Anforderungen

ID	Titel	Erfüllt	Teilw. erfüllt	Nicht erfüllt	Erläuterung
AF-1	Singleplayer	**X**			Alles wurde relativ gestaltet und läuft als Single Page App
AF-2	Balance	**X**			Spielefeatures außerhalb unseres Spektrums wurden ausgelassen/ wieder verworfen.
AF-3	Dom-Tree	**X**			DOM-Tree wurde von uns nur als View genutzt, Innenleben liegt nirgendwo im DOM herum oder Ähnliches. Werte, wie Koordinaten werden intern abgespeichert und nicht anhand eines Elements ausgelesen.
AF-4	Smartphone	**X**			Gyroskop-basierte Steuerung wurde sinnvoll eingebracht (aktives Feedback, statt Panzersteuerung)
AF-5	Mobile First	**X**			Unsere Software läuft sowohl auf dem Smartpone, als auch auf dem PC. Steuerung wurde angemessen angepasst
AF-6	Intuitiv		**X**		Das Fadenkreuz erzeugt im ersten Moment die Illusion, dass man Gegner auch nur in diesem treffen könnte.
AF-7	Levelkonzept	**X**			Schwierigkeit steigt jedes Level an, neue Konzepte werden eingeführt
AF-8	Nichts Globales	**X**			Das einzige, was gespeichert wird, ist der Fortschritt zwischen den Levels und das wird lokal geregelt
AF-9	Basic Libraries	**X**			Zuerst gab es die Überlegung, eine Vektorklasse zu importieren, was dann verworfen wurde, da wir unsere eigene schrieben
AF-10	Simpel	**X**			Zeitlich hätte das schon nicht gepasst
AF-11	Dokumentation		**X**		Vermutlich zu spät angefangen und auch wenn die SnakeGame Doku als gutes Beispiel dient, ist es doch schwerer, jede Entity zu beschreiben, als man zuerst denkt.

5.2 Nachweis der Dokumentationsanforderungen

ID	Kurztitel	Erfüllt	Teilw. erfüllt	Nicht erfüllt	Erläuterung
D-1	Dokumentationsvorlage	**X**			Es wurde sich an die vorliegende Dokumentation gehalten
D-2	Projektdokumentation		**X**		Es wurde bestmöglich zu allen vorliegenden Punkten mindestens ein Satz erläutert.
D-3	Quelltextdokumentation		**X**		Alles an Funktionen wurde erklärt
D-4	Libraries	**X**			Die genutzten Libraries wurden in der *pubspec.yaml* aufgeführt

5.3 Nachweis der Einhaltung technischer Randbedingung

ID	Kurztitel	Erfüllt	Teilw. erfüllt	Nicht erfüllt	Erläuterung
TF-1	No Canvas	**X**			Erstaunlicherweise haben wir das komplette Spiel programmiert und allesamt immernoch keine Ahnung, was ein Canvas sein soll. Diesen Punkt haben wir also erfüllt.
TF-2	Levelformat	**X**			Die Level werden nicht zufällig erstellt.
TF-3	Parameterformat	**X**			Eine JSON Datei wird eingelesen und kann anhand dieser ein Level generieren.
TF-4	HTML+CSS	**X**			Anhand von der CSS Eigenschaften “bottom” und “left” konnten wir alle Elemente angemessen positionieren.
TF-5	Gamelogic in Dart	**X**			Gamelogic wurde komplett in Dart realisiert.
TF-9	Browser Support	**X**			Das Spiel läuft in allen getesteten Browsern (Opera, Firefox, Chrome).
TF-10	MVC Architektur	**X**			Unser View kennt zwar das Model, ändert diese jedoch nicht und wir wollten uns auf diese Weise extra Zugriffe sparen. Änderungen geschehen weiterhin nur über den Controller.
TF-11	Erlaubte Pakete	**X**			Es werden nur die dart:* packages genutzt
TF-12	Verbotene Pakete	**X**			Es wurden keine Weiteren Pakete verwendet, außer sie wurden selbst programmiert.
TF-13	No Sound	**X**			Es wird kein Sound genutzt

5.4 Verantwortlichkeiten im Projekt

Komponente	**Detail**	**Marcel Azadi**	**Laurenz Schindler**	**Daniel Waage**
Model	Grundgerüst	U	U	V
	World (wurde später zu Game)			V
	Einlesen von Levels		V	
	Aufbau der Level	V	U	
	Gegner	V		
	Schuss	U	V	
Controller	Weltenlogik	U	V	U
	MiniMap			V
View	Gerüst	V		
	Grafiken	V		U
	Spieleranimation			V
Documentation	MaLaDa Report	U	U	V
	Codekommentare	V	U	U

V = Verantwortlich
U = Unterstützend