Technische Hochschule Lübeck

Dokumentation
Web-Technologie-Projekt 2020

Hall-of-Fame Teilnahmen: Ja

Alonso Essenwanger und Viktoria Ginter

https://webtech.mylab.th-luebeck.de/ss2020/team-01-f/

betreut von
Prof. Dr. Nane Kratzke

17. Juni 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Spielkonzept

Für das diesjährige Webtechnologie Projekt soll ein Spiel mit der Thematik Corona entwickelt werden. Dabei sollen insgesamt neun Komplexitätspunkte erreicht werden.
In dem Spiel „Evade“ soll der Spieler mit Hilfe des Gyrosensors (3 Punkte) auf dem Mobiltelefon oder der Maus auf dem Desktop (3 Punkte) bewegenden Hindernissen ausweichen und so den Abstand wahren, um sich sicher an sein Ziel zu manövrieren. Dabei wird die vom Spieler benötigte Zeit gemessen und entsprechend mit Sternen gewertet. Somit soll es sowohl auf dem Desktop als auch auf dem Browser laufen.
Beim Berühren eines bewegenden Hindernisses wird der Spieler auf die Startposition oder einen Speicherpunkt zurückgesetzt. Zusätzlich wird eine Strafzeit auf die benötigte Zeit aufaddiert. Speicherpunkte sind dabei Flächen, die der Spieler erreicht haben muss, um zu diesen zurückgesetzt werden zu können.
Mit jedem Level soll sich die Schwierigkeit durch eine Variation der Hindernisse erhöhen. Die Level sollen durch die Eingabe von Parametern erstellt werden (1 Punkt), dabei sollen die Elemente eine absolute Position erhalten (2 Punkte).

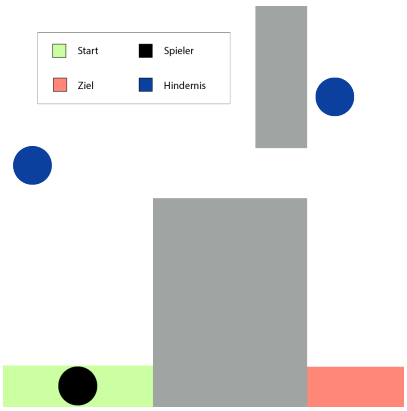


Abbildung 1: Spieler berührt ein Hindernis

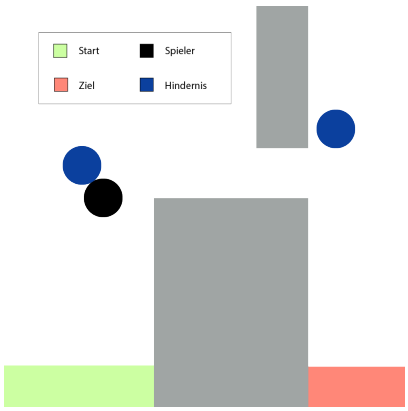


Abbildung 2: Spielbeginn

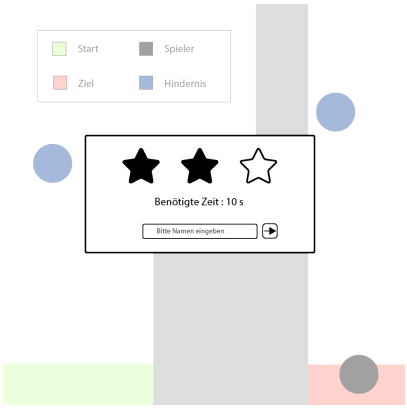


Abbildung 3: Spielende mit Score

Folgende Dinge könnten das Spiel erweitern:

- Aufsammeln von Ressourcen
- Zeitlimit
- Komplexere Verhaltensweisen der Hindernisse

2 Architektur und Implementierung

Die Architektur von `Evade` folgt dem Model-View-Controller Prinzip, sodass sich eine softwaretechnische Gliederung in Klassen ergibt.
Der Controller koordiniert, indem er Modelinteraktionen umsetzt und den Datenfluss durch die Vorgabe eines Takts regelt.
Die GameView nutzt den DOM-Tree, um die im Model errechneten Spielzustände im Browser anzuzeigen. Der Controller in Abschnitt 2.3 nutzt dabei die entsprechenden Methoden der View in Abschnitt 2.2 zur Manipulation des DOM-Trees.
Das Spielkonzept selbst wird im Model abgebildet und gliedert sich in mehrere Klassen, diese werden in Abschnitt 2.1 näher erläutert.

2.1 Model

Aus dem Spielekonzept ergibt sich ein Level (2.1.1) mit einem Character (2.1.9), einer Camera (2.1.4) und mindestens zwei SavePoints (2.1.6) und beliebig vielen Hindernissen aus RectengularObject (2.1.5), MovingObstacle (2.1.8) oder Fiend (2.1.10).

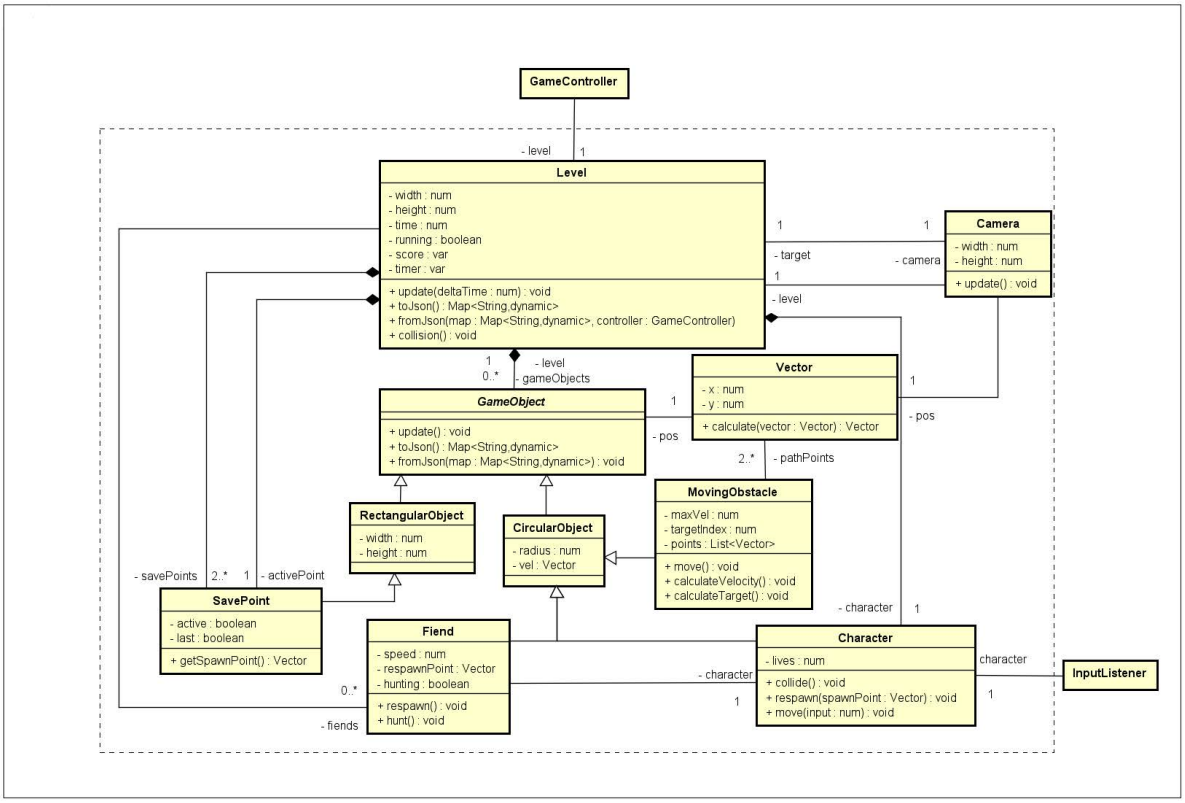


Abbildung 4: Model

2.1.1 Level

Das Level ist das Bindeglied zwischen dem GameController und den darunter liegenden Entities. Durch diese Verbindung ist eine problemlose Erweiterung des Spieles möglich ohne das Veränderungen im Controller vorgenommen werden müssen. Dabei gibt das Spiel dem Controller über folgende Attribute Aufschluss über den Spielstand:

- `width` liefert die Breite des Spielfeldes.
- `height` liefert die Höhe des Spielfeldes.
- `timer` gibt Auskunft über die benötigte Zeit des Spielers.

- `savePoints` gibt eine Liste mit Zwischenspeicherungspunkten an, dabei ist der erste der Startpunkt und der letzte das Ziel.
- `activeSavePoint` gibt den aktuellen Zwischenspeicherungspunkt an.
- `gameObjects` gibt eine Liste mit GameObjects zurück.
- `character` dient als Speicherung der Spieleeingaben.
- `cam` berechnet den dem Spieler sichtbaren Abschnitt.
- `running` gibt an ob das Spiel gerade läuft
- `score` gibt die vom Spieler erreichte Punktzahl an
- `fiends` speichert eine Liste von Fiend Objekten für die Kollision

Ein Level Object

- kann mittels Konstruktor unter Angabe der Breite und der Höhe sowie dem Controller erstellt werden.
- kann mittels `fromJson()` mit Angabe des Controller und einer umgewandelten JSON Datei erstellt werden.
- kann mittels `toJson()` das Level in eine Map zur Umwandelung in JSON umwandeln.
- kann mittels `update()` das Level aktualisieren.
- verwendet `collision()`, um Berührungen zwischen den Objekten zu erfassen.
- kann mittels `getIntersection()` den Schnittpunkt zweier Geraden für die Kollision bestimmen.
- kann mittels `getResultingVel()` die Geschwindigkeit des Charakters nach einer Kollision bestimmen.
- Verwendet `rectangleCollision()`, um CircularObjects von RectangularObjects abzustoßen.
- kann mittels `closestPointCollision()` den nähersten Punkt eines CircularObject zum anderen Object bestimmen.
- verwendet `savePointCollision()` um das Berühren eines Speicherpunktes zu bestimmen.
- nimmt mit `levelTime()` die benötigte Zeit auf.

2.1.2 GameObject

Das GameObject ist eine abstrakte Klasse, die mit dem `pos` Attribut, welches ein Vector ist, für die Position und der `update()` Methode als Vorlage für diverse andere Klassen dient. Zudem verfügt sie über eine `toJson()` Methode, die eine Map für die Erstellung von JSON Dateien herausgibt und eine `fromJson()` Methode die eine umgewandelt JSON Datei, also eine Map, auslesen und implementieren kann. GameObject ist abstrakt und kann nicht erstellt werden.

2.1.3 Vector

Die Vector-Klasse dient als Abwandlung der Point-Klasse in Dart, dabei sind die Koordinaten nicht final und können geändert werden. Ein Vektor kann unter Angabe von zwei Zahlen im Konstruktor erstellt werden. Das Vector Objekt verfügt über diverse arithmetische Methoden.

2.1.4 Camera

Die Klasse Camera verfügt über eine Position und eine `updatePos()` Methode, mit der sie die Position vom Ziel abhängig neu berechnen kann.
Ein weiteres Attribut der Camera ist

- `target`, welches das zu verfolgende Ziel angibt.
- `width`, die Breite der Camera, welche der Bildschirmbreite entspricht.
- `height`, die Höhe der Camera, die der Bildschirmhöhe entspricht.
- `pos`, das die Position der Camera angibt.
- `level`, für die Berechnung der Positionierung.

Die Camera kann unter Angabe eines target Attributes im Konstruktor erstellt werden.

2.1.5 RectangularObject

Das RectangularObject realisiert das GameObject. Diese Klasse dient zur Erstellung von rechteckigen Hindernissen, die sich nicht bewegen sollen.
Diese Klasse verfügt über folgende Attribute:

- `pos` gibt die Position des RectangularObject vor.
- `width` gibt die Breite des Objektes an.
- `height` gibt die Höhe des Objektes an.

Ein RectangularObject

- kann unter Angabe der Breite und der Höhe sowie der Position mit Hilfe des Konstruktors erstellt werden.
- kann mittels `update()` pos aktualisieren.
- kann mittels `toJson()` das StaticObject in eine Map zur Umwandelung in JSON umwandeln.
- kann mit Hilfe einer `Map()` ein Objekt erstellen und ein erstelltes in eine Map umwandeln.

2.1.6 SavePoint

Die SavePoint-Klasse erbt vom RectangularObject und verfügt somit über die gleichen Datenfelder und Methoden. Zusätzlich verfügt sie jedoch über eine `getSpawnPoint()` Methode, die Positionierung des Spielers beim Auftauchen berechnet, einen last bool zur Bestimmung des Ziels und einen active bool, der bestimmt, ob er der aktive, also wohin der Charakter zurückkehrt, ist.

2.1.7 CircularObject

Die Klasse CircularObject implementiert die GameObject Klasse. Die Klasse dient zur Erstellung eines runden Objektes.
Sie verfügt über die Attribute:

- `pos`, die Position des Objekts.
- `radius`, den Radius des Objekts.
- `vel`, der Geschwindigkeit des CircularObject, der Standardwert ist 0.

Ein CircularObject

- kann unter Angabe der Position und des Radius mittels Konstruktor erstellt werden.
- kann mittels `fromJson()` mit einer Map erstellt werden.
- kann eine Map zur Umwandelung in JSON erzeugen.
- kann mittels `update()` die Position aktualisieren.

2.1.8 MovingObstacle

Die Klasse MovingObstacle erbt vom CircularObject und hat somit die gleichen Methoden und Attribute.
Weitere Zustände des MovingObstacle, sind:

- `maxVel`, welche die maximale Geschwindigkeit des MovingObstacle angibt.
- `points`, eine Liste des Typs Vector, die die Punkte zwischen denen sich das Objekt bewegen soll angibt.
- `targetIndex`, gibt den anzusteuernden Punkt aus der points Liste an.

Für die Erstellung eines MovingObstacle müssen dem Konstruktor alle Attribute übergeben werden.
Die Methode `move()` überprüft, ob ein Punkt erreicht oder überschritten wurde und bewegt das Objekt dessen in Richtung.
`CalculateVelocity()` berechnet die Geschwindigkeit bis zum nächsten Punkt, um eine gleichmäßige Bewegung zu ermöglichen.
`CalculateTarget()` ändert den Zielpunkt des Objektes.

2.1.9 Character

Die Klasse Character erbt von der Klasse CircularObject und verfügt über ein zusätzliches Attribut lives, welches die Menge der Leben angibt.
Ein Character Model

- bekommt Input vom InputListener, der in der `move()` Methode verarbeitet wird.
- setzt mit `respawn()` den Charakter auf einen SavePoint zurück.

2.1.10 Fiend

Die Klasse Fiend erbt von dem CircularObject, zudem besitzt es die Attribute speed, also die Geschwindigkeit dieses Objektes, ein respawnPoint, die Position auf die das Objekt zurückkehrt, wenn der Charakter auf einen Speicherpunkt zurück geschickt wird und char, also den Character, zur Überwachung seiner Position. Zudem verfügt es über den bool hunting, was eine spätere Verfolgung ermöglicht.

Das Fiend Model

- nutzt `hunt()`, um die Richtung und Geschwindigkeit Richtung Character anzupassen.
- Kehrt mir `respawn()` an den respawnPoint zurück.

2.2 View

Die Darstellung des Spiels lässt sich mit Hilfe der View anpassen. Der Quelltext enthält keine Programmlogik, sondern dient ausschließlich zur Darstellung von `Evade`. Der Spieler steuert das Spiel mittels Gyrosensor oder Maus.

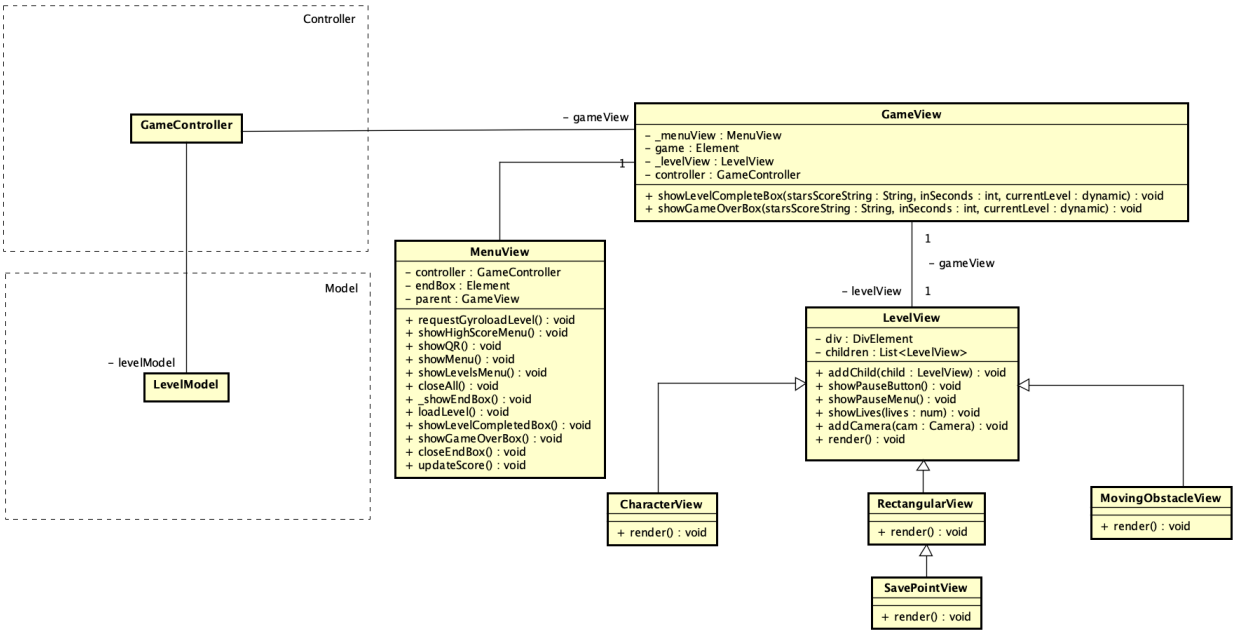


Abbildung 5: View

2.2.1 HTML

Das folgende HTML-Dokument dient zur initialen Erzeugung der View. Für die Anzeige des Spielzustands und das Ermöglichen der Nutzerinteraktionen ist die Klasse `gameController` verantwortlich. Diese wird durch `main.dart.js` (siehe Abbildung 6) geladen. Ebenfalls wird in das HTML-Dokument die CSS Datei `style.css` geladen, die für das Style von HTML zuständig ist.

```html
<!DOCTYPE html>

<html lang="de">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="scaffolded-by" content="https://github.com/google/stagehand">
    <title>Team 1-F</title>
    <link rel="stylesheet" href="styles.css">
    <link rel="icon" href="favicon.ico">
    <script defer src="main.dart.js"></script>
</head>

<body>
    <div id="output"></div>
</body>
</html>
```

Abbildung 6: HTML Datei

2.2.2 Die View-Klassen

Die `gameView` ist unterteilt in zwei Klassen, die das Spiel aufbauen, die `menuView` und `levelView`. Bei der `menuView` ist der Spieler in der Lage die Levels auszuwählen, den QR Code anzuzeigen und die jeweiligen Highscores pro Level anzusehen.
Bei der `levelView` wird mithilfe von Unterklassen das Level aufgebaut.

- Die `characterView` ist dafür zuständig den Character aufzubauen und darzustellen.
- Die `rectangularView` baut mittels dem RectangularModel Rechtecke im Spielfeld, die die Parkour Karte darstellen sollen.
- Die `movingObstacleView` platziert bewegende Hindernisse mittels der MovingObstacle Klasse, denen der Spieler ausweichen soll, um das Level zu gewinnen.

Wie bereits erwähnt dient die View ausschließlich zur Darstellung des Spiels. Der `gameController` ist für die Erzeugung und Aktualisierung der View Elemente verantwortlich, aber der Controller selbst manipuliert nicht den DOM-Tree dafür sind Methoden in der View zuständig. Die Methode `render()` wird im Controller aufgerufen und erzeugt die Elemente für die Darstellung des Spiels und die Methode `update()` aktualisiert das Spiel.

2.3 Controller

Der Controller dient als eine Art Vermittlungsschicht zwischen Model und View und ist für die Ablaufsteuerung und das Laden des Spiels verantwortlich. Hiermit lassen sich Events und die Interaktionen des jeweiligen Spielers verarbeiten. Der Spieler steuert das Spiel mittels Gyrosensor oder Maus.

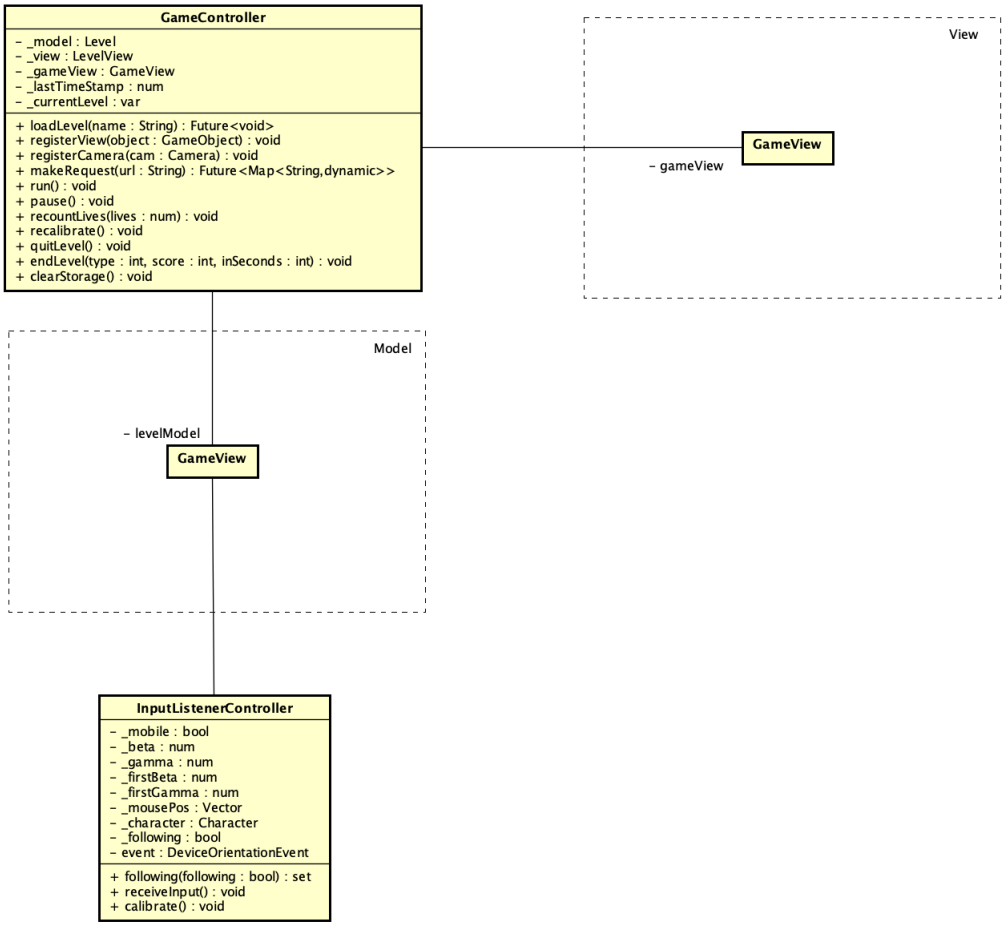


Abbildung 7: Controller UML

2.3.1 JSON

Das Laden des Spiels erfolgt über eine JSON Datei. Zuerst wird die JSON Datei im Controller ausgelesen und ins Model übergeben. Das Model erstellt dann ein GameModel Array mit allen Objekten des Spiels. Der Controller wertet sie aus und unterteilt sie in die jeweiligen Views, um das Spiel darzustellen.

2.3.2 InputListener

Der InputListener wertet die Interaktionen der Spielers mit dem Level aus. Die Interaktionen können sowohl mit dem Gyrosensor als auch mit der Maus erfolgen. Danach überträgt er die erhaltenen Daten dem Character, die er für die Weiterverarbeitung nutzt.

2.3.3 LocalStorage

3 Alphaversion

In der Alphaversion von Evade wurden Level (2.1.1), Character (2.1.9), der grüne Punkt, SavePoint, violette Flächen, (2.1.6) und RectengularObject, türkise Flächen, (2.1.5) umgesetzt. Die MovingObstacles (2.1.8) wurden repräsentativ durch CircularObjects, rote Punkt, (2.1.7) ersetzt. Die Version ermöglicht die Auswahl eines SavePoints und die Rückkehr zu diesem, es wurde noch kein Ende und HighScore implementiert. Die Kollision mit RectangularObjects führt außerdem zu demselben Ergebnis wie die Kollision mit den Circularobject, was sich in späteren Versionen ändern soll.

pubspec.yaml

```yaml
name: demo
description: An absolute bare-bones web app.
# version: 1.0.0
#homepage: https://www.example.com
#author: david <email@example.com>

environment:
  sdk: '>=2.1.0 <3.0.0'

#dependencies:
#  path: ^1.4.1

dev_dependencies:
  build_runner: ^1.1.2
  build_web_compilers: '>=2.6.1 <3.0.0'
  pedantic: ^1.0.0
```

Abbildung 8: pubspec.yaml

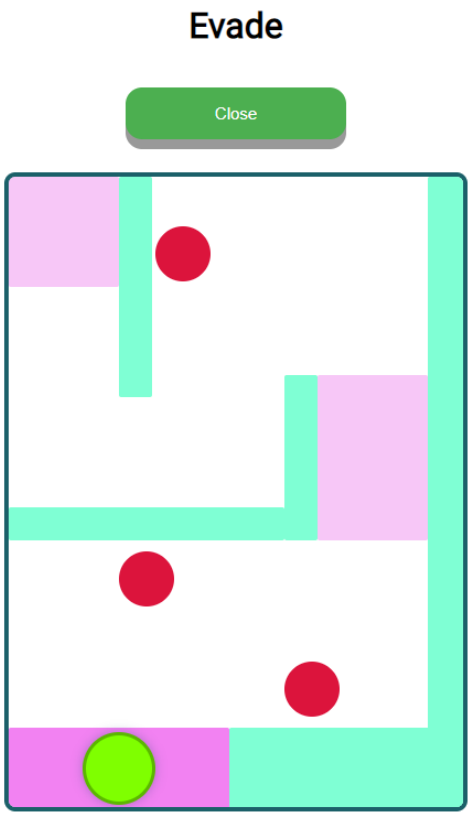


Abbildung 9: Evade Alpha

4 Betaversion

Die Betaversion wurde um die Klasse MovingObstacle (2.1.8) erweitert. Zudem wurde die Kollision des Charakters mit den RectangularObjects ausgearbeitet, sodass dieser sich entlang der Wände bewegen kann. Eine Kollision mit mehreren Objekten gleichzeitig ist für die nächste Version geplant. Der Charakter verfügt in dieser Version über Leben, die er durch das Berühren eines MovingObstacles verliert. Der Verlust aller Leben führt zu einem GameOver. Außerdem können Level nun mit dem Erreichen des letzten Checkpoints abgeschlossen werden. Ebenfalls wurde ein Datenspeicherung (LocalStorage) eingeführt. Die gespeicherten Daten werden für die Auflistung in der Rangliste genutzt.

5 Gammaversion

In dieser Version wurde die Kollision verbessert, sodass es möglich ist bei gemäßigten Geschwindigkeiten mit mehreren Objekten gleichzeitig zu kollidieren. Es wurde eine Camera (2.1.4) implementiert, die es ermöglich größere Level zu bauen, die über die Bildschirmgröße hinaus gehen und gespielt werden können. Der Start und das Ziel unterscheiden sich jetzt von den anderen Speicherpunkten und es gibt am Anfang des Levels eine kurze Kalibrierung, diese setzt jedoch manchmal aus. Eine Verbesserung ist bis zur nächsten Version geplant.

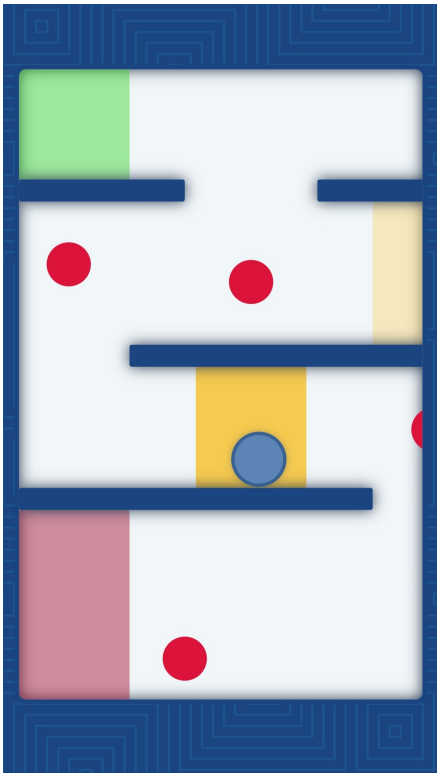


Abbildung 10: Evade Gamma

6 Prefinal-Version

In der Prefinal-Version wurde ein Pausenmenü implementiert, in dem es möglich ist sein mobiles Gerät neu auszurichten oder das Level zu verlassen. Zudem ist es möglich nach Erreichen der Ziellinie das nächste Level zu starten ohne das Spiel zu verlassen. Die Leben des Spielers sowie die beste Zeit wird im Highscore angezeigt. Das Spiel ist nun auch im Fullscreen verfügbar.

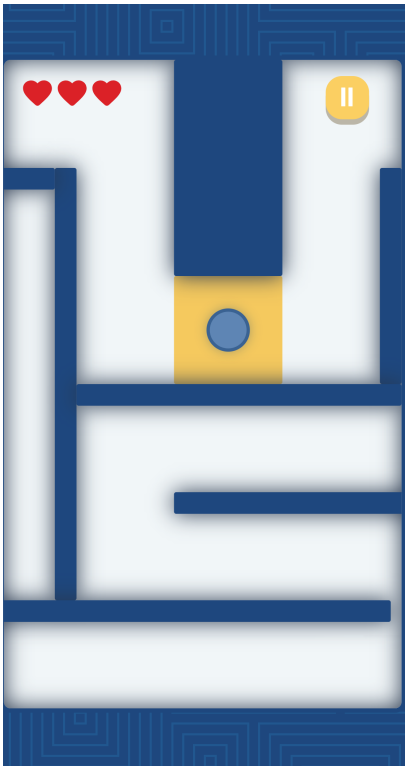


Abbildung 11: Evade Prefinal

7 Final-Version

Die Final-Version wurde um die Klasse Fiend (2.1.10) erweitert, diese bewegt sich in Richtung des Characters und versucht ihn zu fangen. Bei Berührung werden beide auf eine bestimmte Position zurückgesetzt. Außerdem wurden kleinere Bugs an der Camera behoben.

8 Level- und Parametrisierungskonzept

8.1 Levelkonzept

Die Erstellung eines Levels erfolgt in einer JSON Datei, deren Namen gleichzeitig als ID dient. Die Größe des Levels wird zu Anfang durch Angabe der Höhe und der Breite bestimmt, dabei kann das Level dank der Kameraverfolgung beliebig groß in die Höhe oder Breite gebaut werden. Die RectangularObjects dienen als Wände und sind somit das Grundgerüst des Levels, die Position und die Größe können dabei frei bestimmt werden. Für ein vollwertiges Level müssen mindestens zwei SavePoints angegeben werden, unter Bestimmung der Größe und der Position, dabei bildet der erste den Anfang und der letzte das Ende eines Levels. Da die Bewertung des Spielers durch Zeit erfolgen soll, kann die geschickte Platzierung von Wänden und die Vergrößerung des Levels die Schwierigkeit eines Levels erhöhen, gleichzeitig kann durch das Festsetzen des Radius des Charakters, dieser in eine vorteilhaftere Position gebracht werden. Die Aufgabe der Schwierigkeitserhöhung liegt aber vor allem bei den Fiend und MovingObstacle, durch Berührung dieser wird der Spieler an den zuletzt berührten SavePoint zurückgesetzt und verliert somit Zeit. Das Fiend Objekt bewegt sich in Richtung des Spielers und versucht diesem zu folgen, dabei ist die Geschwindigkeit frei wählbar. Die Variabilität der Schwierigkeit des MovingObstacle liegt auf dem frei definierbaren Pfad, der unter Angabe von Positionspunkten berechnet wird und der Bestimmung ihres Radius und der Geschwindigkeit.
Ein gutes Beispiel für die Erhöhung der Schwierigkeit mit MovingObstacles ist level3.json. Durch Platzierung der Wände in Bereichen ohne MovingObstacle ist der Spieler gezwungen diesen auszuweichen. Dabei wird das Geschick des Spielers durch deren unterschiedliche Pfade und unterschiedliche Geschwindigkeit erprobt.

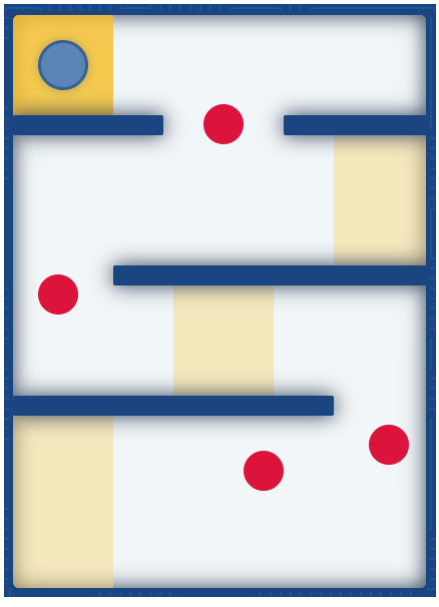
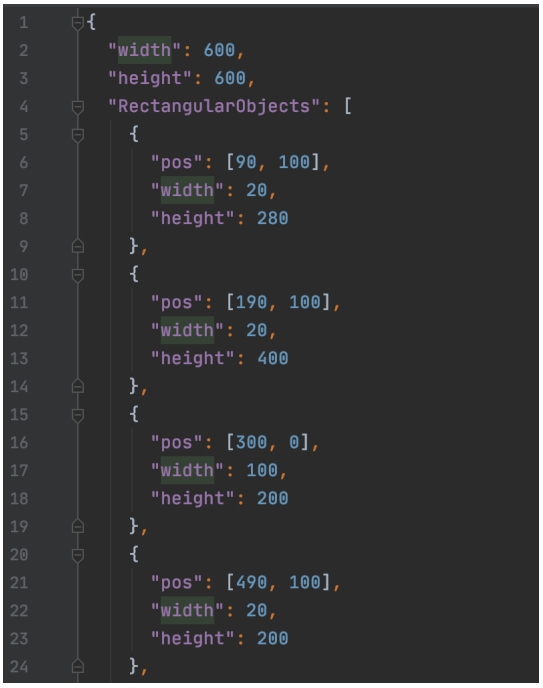


Abbildung 12: Level 2

9 Parametrisierungskonzept

Die Parametrisierung eines Levels erfolgt in einer JSON Datei. Dabei wird die Höhe und Breite des Levels angegeben, sowie die Position und der Radius des Characters (2.1.9). Zudem werden mehrere Listen von Spieleobjekten in spezifischer Form benötigt. Darunter fallen RectangularObjects (2.1.5) und SavePoints (2.1.6), deren Höhe, Breite und Position zu bestimmen ist und die MovingObstacles (2.1.8), denen die Position, der Radius, die Pfadpunkte und die maximale Geschwindigkeit übergeben werden. Auch gehören Fiend (2.1.10) Objekte dazu, die unter Angabe der Position, des Radius und der Geschwindigkeit erstellt werden.



```
{
  "width": 600,
  "height": 600,
  "RectangularObjects": [
    {
      "pos": [90, 100],
      "width": 20,
      "height": 280
    },
    {
      "pos": [190, 100],
      "width": 20,
      "height": 400
    },
    {
      "pos": [300, 0],
      "width": 100,
      "height": 200
    },
    {
      "pos": [490, 100],
      "width": 20,
      "height": 200
    },
```

Abbildung 13: Parametrisierungskonzept

10 Nachweis und Anforderungen

ID	Kurztitel	Erfüllt	Teilw. erfüllt	Nicht erfüllt	Erläuterung
AF-1	Single-Page-App	×			Evade ist ein Einspieler Game, was sich auch im InputController widerspiegelt. Siehe Punkt 2.3.2
AF-2	Balance Spielkozept - Komplexität	×			Evade ist ein einfaches Spiel, dessen Komplexitätspunkte vor allem in der Umsetzung der MovingObstacles (2.1.8) und der Kollision im Level (2.1.1) liegen.

AF-3	DOM-Tree-basiert	×			Evade nutzt Html zur Darstellung der Spielobjekte. Siehe Punkt 2.2.1
AF-4	Target Device: Mobile + Desktop	×			Das Spiel ist sowohl mobil als auch auf dem Desktop spielbar.
AF-5	Mobile First	×			Das Spiel ist für den Gyrosensor ausgelegt und bietet nur eine zusätzliche Steuerung für den Desktop. Siehe Abschnitt 2.3.2
AF-6	Intuitive Spielfreude	×			Das Spiel ist auf Mobile Devices intuitiv erfassbar und bietet Herausforderungen in Form eines Parcours. Siehe Abbildung 9
AF-7	Levelkonzept	×			Das Spiel besitzt 10 Level, die mithilfe von JSON eingelesen werden.
AF-8	Speicherkonzept	×			Es können Nutzerdaten wie der Score gespeichert werden.
AF-9	Basic Libraries	×			Es werden keine zusätzlichen Bibliotheken verwendet, wie man in Abbildung. 8 sieht.
AF-10	Keine Spielereien	×			Evade besitzt weder Ton noch zusätzliche Bibliotheken.
AF-11	Dokumentation	×			Die Dokumentation ist fertig geschrieben.

11 Verantwortlichkeiten im Projekt

Komponente	Detail	Asset	Essenwanger A.	Ginter V.	Anmerkungen
Controller	gameController	web/Controller/ gameController.dart	V		LocalStorage wurde hier implementiert
	inputController	web/Controller/ inputController.dart		V	
Model	camera	web/Model/camera.dart		V	
	character	web/Model/character.dart	U	V	
	circularObject	web/Model/ circularObject.dart		V	
	fiend	web/Model/fiend.dart		V	
	gameObject	web/Model/ gameObject.dart		V	
	level	web/Model/level.dart	U	V	
	movingObstacle	web/Model/ movingObstacle.dart		V	
	rectangularObject	web/Model/ rectangularObject.dart		V	
	savePoint	web/Model/savePoint.dart		V	
	vector	web/Model/vector.dart		V	
View	HTML Dokument	Index.html	V		
	Gestaltung	Style.css	V	U	
	characterView	web/View/ characterView.dart		V	
	circularView	web/View/circularView.dart	V		
	gameView	web/View/gameView.dart	V		
	levelView	web/View/levelView.dart	V	U	
	menuView	web/View/menuView.dart	V		
	rectangularView	web/View/ rectangularView.dart	V		
	savePointView	web/View/ savePointView.dart		V	
Dokumentation	WebTech_Team-01-F	doc/WebTech_Team-01-F.pdf	V	U	Ginter: Model & Änderungen Alonso: View Controller,Layout