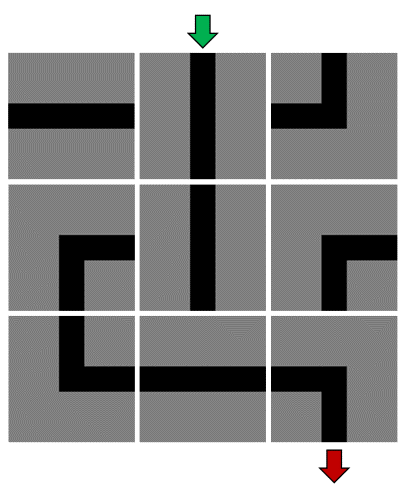
Martin Dolacinski und Benjamin Tamke

Pathpuzzle



Inhaltsverzeichnis

[1. Einleitung 2](#_Toc518133674)

[2. Spielkonzept 3](#_Toc518133675)

[2.1. Anforderungen 3](#_Toc518133676)

[2.2 Spielkonzept 5](#_Toc518133677)

[3 Architektur und Implementierung 6](#_Toc518133678)

[3.1. Modell 7](#_Toc518133679)

[3.1.1. Tile 7](#_Toc518133680)

[3.1.2. Field 8](#_Toc518133681)

[3.2. View 10](#_Toc518133682)

[3.3. Controller 11](#_Toc518133683)

[3.4. Spiel- und Levelparameter 12](#_Toc518133684)

# Einleitung

Im Rahmen des Webtechnologie Projektes haben wir uns für ein klassisches Logik- Strategiespiel entschieden. Ein ähnliches Spielkonzept gibt findet man beifolgendem Spiel im Google Playstore: „Roll the Ball“ von BitMango. Mit mehr als 100 Millionen Downloads (Stand: Juni 2018) zählt es zu einem der meist heruntergeladenen Apps in dem Store.

# Spielkonzept

## Anforderungen

Ihr Spiel soll folgende Anforderungen erfüllen. Sie sollen dabei nachvollziehbar dokumentieren, wie diese Anforderungen durch ihre Realisierung umgesetzt werden. Das Spiel soll folgende in Tabelle [1](#page6) aufgeführten funktionalen Anforderungen, Dokumentationsanforderungen und technischen Randbedingungen erfüllen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Id | Kurztitel | Anforderung | | |
|  |  |  |  |  |  |
|  | AF-1 | Einplayer | Das Spiel soll ein Einplayer Game sein (Mehrplayer Konzepte können als Einplayer Game realisiert werden, wenn Spieler | | |
|  |  | Game | durch “künstliche Intelligenzen” gesteuert werden. Beachten Sie dabei bitte, dass abhängig vom Spielkonzept die | | |
|  |  |  | Komplexität des Spiels erheblich steigt, denken sie bspw. an Schach. Es bietet sich an, sich von alten Arcade Klassikern | | |
|  |  |  | inspieren zu lassen.) | | |
|  |  |  |  |  |  |
|  | AF-2 | 2D Game | Das Spiel soll konzeptionell auf einem 2D-Raster basieren. | | |
|  |  |  |  |  |  |
|  | AF-3 | Levelkonzept | Das Spiel sollte ein Levelkonzept vorsehen. | | |
|  |  |  |  |  |  |
|  | AF-4 | Parametrisier- | Das Spiel sollte ein Parameterisierungskonzept für relevante Spielparameter vorsehen. | | |
|  |  | ungskonzept |  |  |  |
|  |  |  |  |  |  |
|  | AF-5 | REST-basierter | Das Spiel soll auf einen JSON-basierte Storageservice zugreifen können, um Spielzustände (z.B. Highscores und ähnliches) | | |
|  |  | Storage | speichern zu können. Das Spiel muss bei nicht erreichbarem Storage-Service (Gamekey Service) weiterhin spielbar sein. | | |
|  |  |  | Einzig und allein die Speicherfunktionalitäten dürfen natürlich eingeschränkt sein. (Für den Storage-Service | | |
|  |  |  | (Gamekey-Service) gibt es eine Referenz-Implementierung: <https://bitbucket.org/nanekratzke/gamekey>) | | |
|  | AF-6 | Desktop | Das Spiel muss in Desktop Browsern spielbar sein. | | |
|  |  | Browser |  |  |  |
|  |  |  |  |  |  |
|  | AF-7 | Mobile Browser | Das Spiel muss auf SmartPhone Browsern spielbar sein. | | |
|  |  |  |  |  |  |
|  |  |  | Dokumentationsanforderungen | | |
|  |  |  |  |  |  |
|  | D-1 | Dokumenta- | Die Dokumentation soll sich an vorliegender Vorlage orientieren. | | |
|  |  | tionsvorlage |  |  |  |
|  |  |  |  |  |  |
|  | D-2 | Projekt- | Das Spiel muss geeignet dokumentiert sein, so dass es von projektfremden Personen fortgeführt werden könnte. | | |
|  |  | dokumentation |  |  |  |
|  |  |  |  |  |  |
|  | D-3 | Quelltext- | Der Quelltext des Spiels muss geeignet dokumentiert sein und mittels schriftlicher Dokumentation erschließbar und | | |
|  |  | dokumentation | verständlich sein. | | |
|  |  |  |  |  |  |
|  | D-4 | Libraries | Alle verwendeten Libraries sind aufzuführen und deren Notwendigkeit zu begründen. | | |
|  |  |  |  |  |  |
|  |  |  | Technische Randbedingungen | | |
|  |  |  |  |  |  |
|  | TF-1 | No Canvas | Die Darstellung des Spielfeldes sollte ausschließlich mittels DOM-Tree Techniken erfolgen. Die Nutzung von | | |
|  |  |  | Canvas-basierten Darstellungstechniken ist explizit untersagt. | | |
|  |  |  |  |  |  |
|  | TF-2 | Levelformat | Level sollten sich mittels deskriptiver Textdateien definieren lassen (z.B. mittels CSV, JSON, XML, etc.), so dass | | |
|  |  |  | Level-Änderungen ohne Sourcecode-Änderungen des Spiels realisierbar sind. | | |
|  |  |  |  | | |
|  | TF-3 | Parameterformat | Spielparameter sollten sich mittels deskriptiver Textdateien definieren lassen (z.B. mittels CSV, JSON, XML, etc.), so dass | | |
|  |  |  | Parameter-Änderungen ohne Sourcecode-Änderungen des Spiels realisierbar sind. | | |
|  |  |  |  | | |
|  | TF-4 | HTML + CSS | Der View des Spiels darf ausschließlich mittels HTML und CSS realisiert werden. | | |
|  |  |  |  | | |
|  | TF-5 | Gamelogic in | Die Logik des Spiels muss mittels der Programmiersprache Dart realisiert werden. | | |
|  |  | Dart |  |  |  |
|  |  |  |  | | |
|  | TF-6 | Storagelogic in | Der Storageservice muss REST-basiert sein und mittels der Programmiersprache Dart realisiert werden. | | |
|  |  | Dart |  |  |  |
|  |  |  |  | | |
|  | TF-7 | Storage | Das Spiel soll gleichermaßen mit der Referenzimplementierung des Storageservice als auch der Eigenrealisierung des | | |
|  |  | Referenz | Storageservice funktionieren. | | |
|  |  |  |  | | |
|  | TF-8 | Storage | Die Eigenrealisierung des Storageservices muss den Referenztest der Referenzimplementierung implementieren. (Nicht | | |
|  |  | Referenztest | erfolgreiche Testfälle sind explizit auszuweisen, zu dokumentieren und zu begründen!) | | |
|  |  |  |  | | |
|  | TF-9 | Browser | Das Spiel muss im Browser Chromium/Dartium (native Dart Engine) funktionieren. Das Spiel muss ferner in allen anderen | | |
|  |  | Support | Browserrn (JavaScript Engines) ebenfalls in der JavaScript kompilierten Form funktionieren (geprüft wird ggf. mit Safari, | | |
|  |  |  | Chrome und Firefox). | | |
|  |  |  |  | | |
|  | TF-10 | MVC | Das Spiel sollte einer MVC-Architektur folgen. | | |
|  |  | Architektur |  |  |  |
|  |  |  |  | | |
|  | TF-11 | Erlaubte | Erlaubt sind alle dart:\* packages, sowie das Webframework start. | | |
|  |  | Pakete |  |  |  |
|  |  |  |  | | |
|  | TF-12 | Verbotene | Verboten sind Libraries wie Polymer oder Angular. (Sollten Sie Pakete verwenden wollen, die außerhalb der erlaubten | | |
|  |  | Pakete | Pakete liegen, holen Sie sich das Go ab, begründen sie bitte, wieso sie das Paket benötigen). | | |
|  |  |  |  | | |
|  | TF-13 | No Sound | Das Spiel muss keine Sounds unterstützen. | | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Spielkonzept

Path Puzzle ist ein Strategiespiel, welches auf einem n x m Spielfeld basiert. Dieses ist in ein 2D Raster gegliedert. Der Spieler blickt aus der Vogelperspektive auf das Spielfeld herab. Ziel des Spieles ist es, einen Weg vom Eingangspunkt zum Ausgangspunkt mit den gegebenen Spielsteinen zu bauen. Dies gelingt ihm, indem er die vorgegebenen Puzzelstücke vertauscht, durch tippen auf die entsprechenden Felder. Die Schwierigkeit besteht darin, dass er wahlweise lediglich eine feste Anzahl hat, je nach Level, um die Stücke zu bewegen, oder einen Timer, der eine Zeitspanne vorgibt, in welcher der Weg gebaut werden muss.

Um die Level anzupassen, sind folgende Erschwerungen vorstellbar:

• Feste, nicht veränderbare Teile

• Puzzelstücke sind zu Beginn verdeckt

• Vergrößerung des Spielfelds

• Nur die genaue Anzahl an Zügen, die es für den am kürzesten zu bauenden Weg braucht

• Hindernisse

• Mehrere Ein und Ausgänge

# Architektur und Implementierung

Unsere Architektur ist wie gefordert, gegliedert in einem MVC Konzept. Die Klasse View kapselt die Schnittstelle zum DOM-Tree und dient zur Visualisierung des Models und stellt dem Controller entsprechende Methoden bereit, um die Spieloberfläche zu aktualisieren. Näheres wird im Abschnitt 3.2 erläutert.

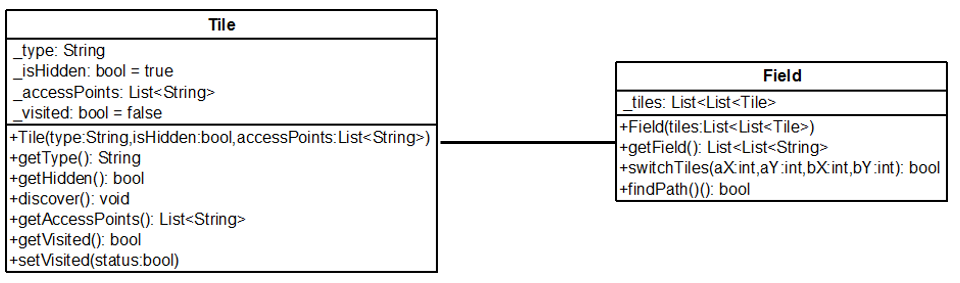
Die Klasse Controller ist die Schnittstelle zwischen dem Model und der View. Der Controller kontrolliert das Model und fordert die View auf sich zu aktualisieren. Näheres wird im Abschnitt 3.3 erläutert.

Das Model selbst ist komplex und kann nicht in einer einzelnen Klasse erfasst werden, viel mehr besteht das Model aus einem Zusammenspiel vieler einzelner Entitäten, die Komplexität überschaubar machen. Näheres wird im Abschnitt 3.2 erläutert.

BILD ZUSAMMENFASSUNG UML

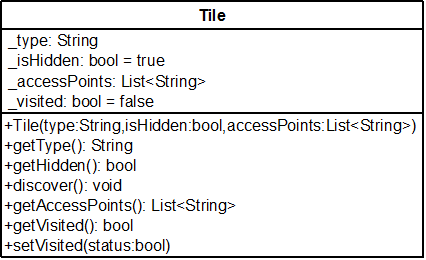
## Modell

Unser Modell besteht im Wesentlichen aus den beiden Klassen „Field“ und „Tile“.



### Tile

In Tile werden alle atomaren Eigenschaften gespeichert.

### Field

Das Field hält als aller erstes eine Referenz zu allen Tile die gegenwärtig im Spiel sind, diese bekommt sie beim Aufruf des Konstruktors im Controller. Des Weiteren wird die Position des aktuell ausgewählten Tiles, sowie eine Referenz zu den Ausgängen als Datenfeld initialisiert.

**Field (…):**

Dies ist der Basis Konstruktor eines Feldes. Ihm wird das aktuelle Spielfeld vom Controller übergeben und daraus ermittelt dieser die gesamten Ausgänge des aktuellen Levels und speichert sie in der Liste „out“.

**getField (…)**

Diese Methode liefert eine Liste von Listen von Strings, in denen die Typen der einzelnen Tiles gespeichert sind, damit diese vom Controller bzw. von der View für die Ausgabe verarbeitet werden können.

**select (…)**

Diese Methode erhält sowohl Zeile, als auch Spalte, des vom Spieler aktuell angetippten Tile und speichert diese. Wird die Methode erneut für ein gültigen Tile – Das heißt keine Rand oder blockierten Tile – aufgerufen ruft sie die switchTiles() -Methode auf. Für die einzelnen Ergebnisse die in der select – Methode auftreten können, liefert die Methode eine Antwort in Form eines Strings den der Controller für eine visuelle Darstellung verarbeiten kann.

**switchTiles()**

Diese Methode tauscht die ausgewählten Tile und sorgt dafür das sie alle richtige Position-Parameter haben.

**checkConnection(…)**

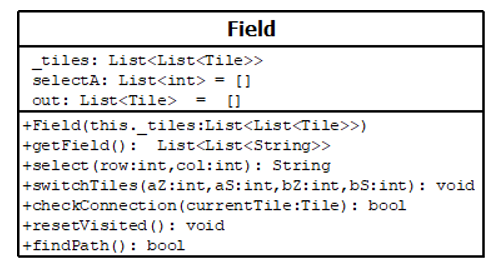
Diese Methode ermittelt, ob der ihr übergebene Tile ein Level-Eingang ist. Sollte dies nicht der Fall sein sucht sie rekursiv nach verbundenen Nachbarn und ermittelt ob diese eventuell ein Level-Eingang sind. Ist das ebenfalls nicht der Fall so liefert sie ein „False“. Außerdem setzt sie den visited-Parameter des aktuellen Tiles auf „True“ um eventuelle Zyklen im Algorithmus zu vermeiden.

**resetVisited(…)**

Diese Methode setzt den visited-Parameter aller Tile auf „False“ zurück.

**findPath(…)**

Diese Methode führt die checkConnection() Methode für alle Ausgänge aus und liefert erst wenn alle mit dem Eingang verbunden sind ein positives Feedback an den Controller.



## View

Unser Spiel wird durch das HTML Dokument erreicht, welches den Kern der View darstellt. Die Anordnung der Elemente, also die Oberfläche, wird durch die Klasse View und durch die CSS Datei vorgenommen.

### Index

Im folgenden ist die Basis Darstellung des HTML Dokuments zu sehen, bevor es durch die FroggerView manipuliert wurde. Wahrend¨ der Spielzeit wird dauerhaft der DOM Tree manipuliert und die Tabelle zu erstellen und zu aktualisieren. Dafur¨ ist die Tabelle mit der ID ’gameplayField’ vorgesehen, hierin wird das Spielfeld erstellt. Ebenfalls wird die Tablle mit der ID ’infoTable’ immer wieder aktualisiert, da diese die Spielubersicht¨ darstellt. Hier sind Informationen zu Le-vel, Punkte, Leben und wie viele LadyFrog’s man eingesammelt hat ersichtlich.

Die DIVs mit den IDs ’menuArea’ und ’flashMessageContainer’ haben beson-dere Bedeutung in dem Sinne, dass hier wahrend¨ der Laufzeit Buttons und Nachrichten erzeugt werden und ggf. ein- oder ausgeblendet werden.

## Controller

## Spiel- und Levelparameter

Die Spielfeldparameter werden in der JSON Datei „Field“ gehalten. Sie enthält wichtige Parameter wie der Spielfeldgröße und die Festlegungen eindeutiger Kennzeichnungen der Level.