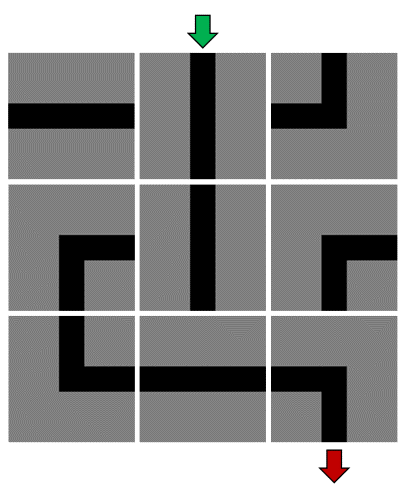
Martin Dolacinski und Benjamin Tamke

Pathpuzzle



Inhaltsverzeichnis

[1. Einleitung 2](#_Toc518251464)

[2. Spielkonzept 3](#_Toc518251465)

[2.1. Anforderungen 3](#_Toc518251466)

[2.2 Spielkonzept 5](#_Toc518251467)

[3 Architektur und Implementierung 6](#_Toc518251468)

[3.1. Modell 7](#_Toc518251469)

[3.1.1. Tile 7](#_Toc518251470)

[3.1.2. Field 8](#_Toc518251471)

[3.2. View 10](#_Toc518251472)

[3.3. Controller 12](#_Toc518251473)

[3.4. Spiel- und Levelparameter 15](#_Toc518251474)

# 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

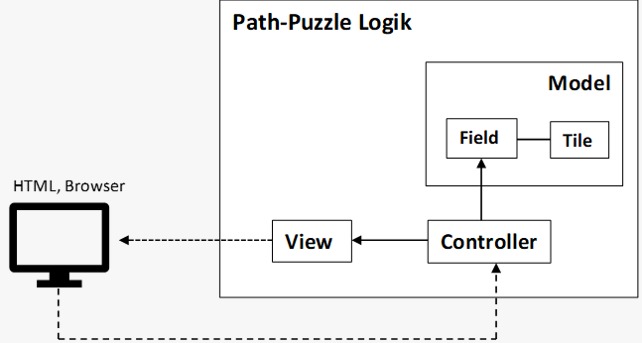
• Hindernisse

• Mehrere Ein und Ausgänge

# Architektur und Implementierung

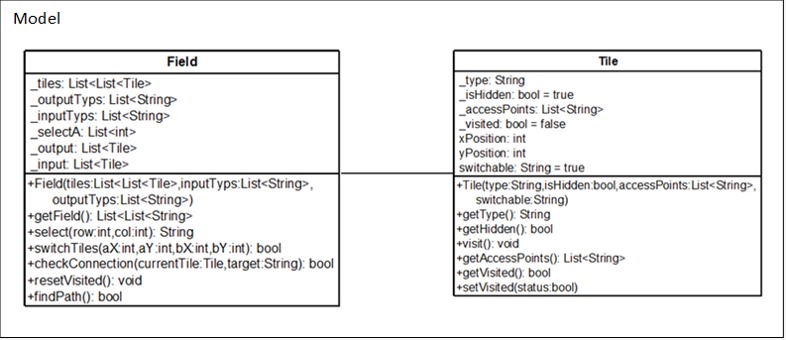
Unsere Architektur ist wie gefordert, gegliedert in einem MVC Konzept. Die Klasse View kapselt die Schnittstelle zum DOM-Tree und dient zur Visualisierung des Models. Sie stellt dem Controller entsprechende Methoden bereit, um die Spieloberfläche zu aktualisieren. Näheres wird im Abschnitt 3.2 erläutert.

Die Klasse Controller ist die Verbindung 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.



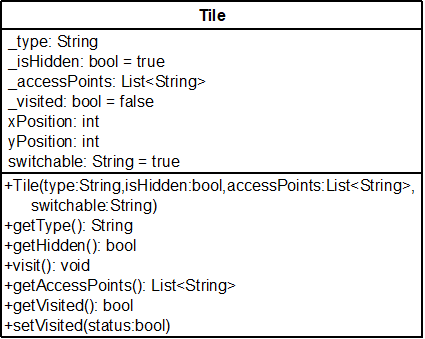
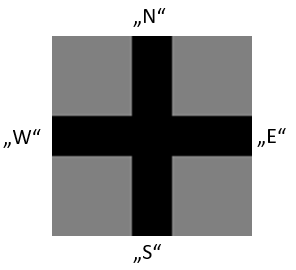
## Modell

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



### Tile

Diese Klasse fungiert als Datencontainer, die sämtliche Informationen der einzelnen Tile hält und über diverse getter und setter für das Field zugänglich macht.



### Field

Das Field hält als aller erstes eine Referenz zu allen Tile die gegenwärtig im Spiel sind, sowie eine Liste mit allen möglichen Typen die einen Eingang oder Ausgang repräsentieren können. Diese bekommt sie beim Aufruf des Konstruktors im Controller. Des Weiteren wird die Position des aktuell ausgewählten Tiles, sowie eine Referenz zu den Ein- bzw. Ausgängen als Datenfeld initialisiert.

**Field (…):**

Dies ist der Basis Konstruktor eines Feldes. Ihm wird das aktuelle Spielfeld vom Controller, sowie je eine Liste der möglichen Ein- und Ausgangstypen übergeben und daraus ermittelt dieser die gesamten Ein- und Ausgänge des aktuellen Levels und speichert sie in den Listen „input“ und „output“.

**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(…)**

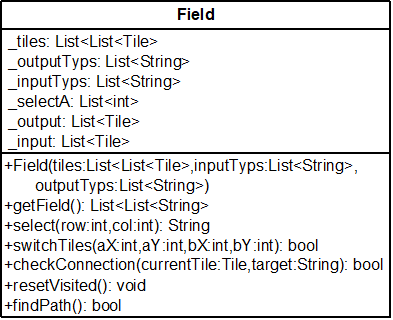
Diese Methode ermittelt, ob der ihr übergebene Tile das zu erreichende Ziel ist. Sollte dies nicht der Fall sein sucht sie rekursiv nach verbundenen Nachbarn und ermittelt ob diese eventuell das gesuchte Ziel sind. Ist das ebenfalls nicht der Fall so liefert sie ein „False“. Außerdem setzt sie den visited-Parameter 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 Ein- und Ausgänge aus und liefert erst wenn alle Eingänge mit allen Ausgängen verbunden sind ein positives Feedback an den Controller.



## View

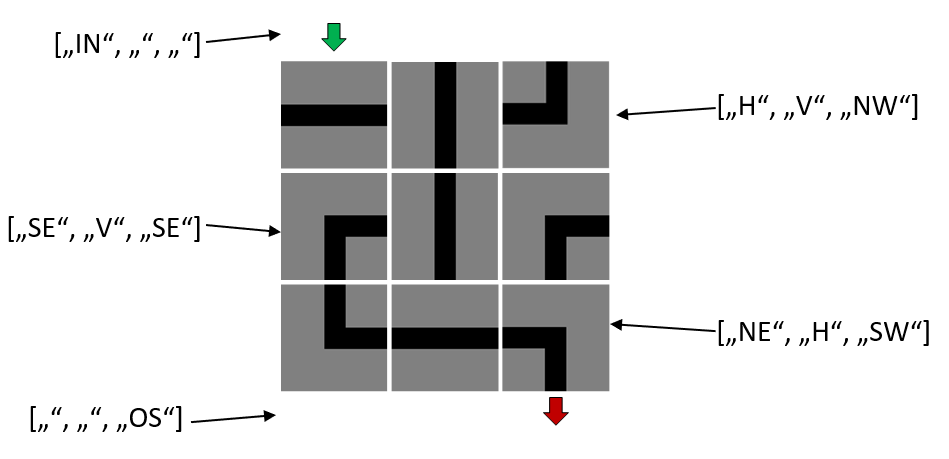
Das erste Datenfeld der View ist eine Liste von Listen von Strings, in der für jeden Tile-Typ der dazugehörige Dateipfad zu dessen Bild aufgeführt ist. Alle anderen Datenfelder ermöglichen einen leichteren Zugriff auf ihr jeweiliges Dom-Tree-Element, sodass man nicht jedes Mal den querySelector aufschreiben muss.

**View(…)**

Das ist der simple Konstruktor der View, dem die Bildpfad-Liste übergeben wird.

**toHtmlTable(…)**

Diese Methode formt aus dem übergebenen Feld von Tile-Typen eine vernünftige Visualisierung. Dh.



**updateField(…)**

Diese Methode aktualisiert das Feld, wenn z.B. zwei Tile getauscht wurden.

**removeSelection(…)**

Diese Methode entfernt mögliche Selektierungs-Marker von allen -Tile

**loadField(…)**

Diese Methode lädt das Feld initial und fügt der Tabelle einen Validierer hinzu damit „row“ und „col“ als Attribute der Tabellen-Elemente erkannt werden.

**updateLevelCatalog**

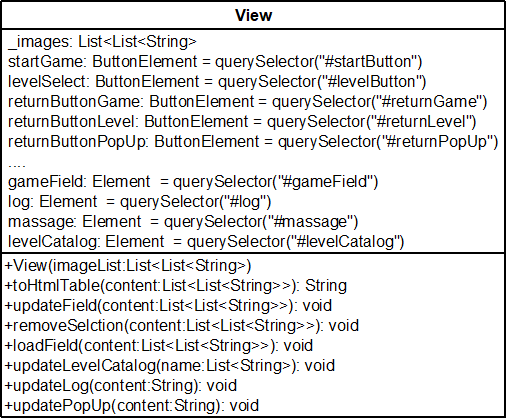
Diese Methode erstellt eine Visualisierung der Level-Liste und fügt einen Validierer hinzu damit „content“ als Attribut anerkannt wird und so ein leichterer Rückschluss darauf geführt werden kann welches Level gewählt wurde.

**updateLog()**

Diese Methode aktualisiert das Log, zB. mit dem aktuellen Stand des Rundenzählers.

**updatePopUp()**

Diese Methode aktualisiert den PopUp.



## Controller

Der Controller hält je eine Referens zu dem Model („\_field“) und der View („\_view“). Anschließend haben wir global je ein Datenfeld für den Inhalt der Tiles.json und der Level.json, damit man im gesamten Controller leicht mit den Daten arbeiten kann ohne die JSON jedes Mal neu auslesen zu müssen. Aus einem ähnlichen Grund gibt es eine globale Liste für die möglichen Eingangs- und Ausgangstypen, sowie eine Liste um die Keys der Level, sprich die Levelbezeichnung zu speichern. Die letzten beiden Attribute sind für den Start eines Levels und für einen sauberen Übergang zum nächsten Level wichtig. Zum einen ein Datenfeld, dass den vom Spieler gewünschten Spielmodus beinhaltet und zum anderen der Name des aktuellen Levels, damit dieser in der Level-Map leichter gefunden werden kann.

**loadTileData()**

Diese Methode liest die Tile.json aus und sendet den erhaltenen String an prepareView(...).

**prepareView(…)**

Diese Methode decodiert den ihr übergebenen String zu einer Map und speichert diese in der Tiledata-Map. Außerdem ermittelt sie für jedes in der JSON aufgeführte Teil den Typen, den Dateipfad zu dem jeweiligen Bild und ob der Tile als Eingang, oder Ausgang vorgesehen wurde. Mit der Liste der Bildpfade initialisiert sie anschließend die View und lässt das Programm, mittels loadLevelData(), mit dem einlesen der Level.json fortfahren.

**loadLevelData()**

Diese Methode liest wiederum die Level.json aus und sendet den erhaltenen String an die Methode loadLevel(…).

**loadLevel(…)**

Diese Methode decodiert den erhaltenen String und speichert die entstandene Map in dem global dafür vorgesehenen Datenfeld. Außerdem ermittelt sie die Keys aller Level und speichert diese in der Levels-Liste. Von hier aus fährt das Programm mit der Initialisierung des Start Menüs fort.

**switchMenu(…)**

Dies Methode ermöglicht das leichte umschalten zwischen zwei Menüs (zB. vom Hauptmenü in die Level-Auswahl).

**genarateLevel(…)**

Diese Methode ermittelt anhand des ihr übergebenen Level-Keys die Daten des aktuellen Levels und baut dieses auf. Dabei achtet das Programm auf diverse Spieleigenschaften, so werden Eingänge und Ausgänge standartmäßig offen dargestellt auch wenn durch die Level-Information alle andern Tile verdeckt werden. Außerdem prüft diese Methode ob die Typ-Bezeichnung jedes Tiles in den Levelinformationen auch valide ist, damit nicht unbekannte Teile die Spiellogik zerstören können.

**startMenu()**

In dieser Methode achtet der Controller auf Nutzereingaben im Hauptmenü.

**game()**

In dieser Methode achtet der Controller auf Nutzereingaben im während des Spiels, also unteranderem, wenn Tile ausgewählt werden und liefert mithilfe der View ein visuelles Feedback. Außerdem werden hier die Gewinn-/Niederlagebedingungen überwacht, D.h. wurde ein valider Pfad vom Spieler gefunden oder hat er eventuell die maximale Zuganzahl überschritten?

**levelselect()**

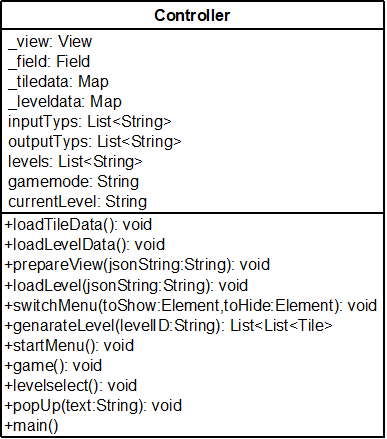
Diese Methode verarbeitet die Nutzereingaben in dem Level-Auswahl-Menü und ermöglicht es dem Spieler ein gewünschtes Level zu spielen.

**popUp()**

Diese Methode liefert dem Nutzer ein Feedback, wenn er eine Gewinn-/Niederlagebedingungen erfüllt hat und ermöglicht ihm dann das Level zu wiederholen, oder zum nächsten voranzuschreiten

**Main()**

Diese Methode startet den gesamten Ablauf des Programmes



## 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.

Ein Level ist nach folgendem Konzept aufgebaut:

**Beschreibung des Feldes**

(als zweidimensionale Liste, mithilfe der Tile-Typen)

**Zug-Limit**

Gibt an wie viele Züge der Spieler maximal zum lösen des Levels hat

**Levelbezeichnung**

**Startzustand des Feldes**

Gibt an ob die Tile des Feldes verdeckt starten

"Level 1": {

"Field": [["IN","",""],

["V","SW", "V"],

["SW","H", "NW"],

["NE","NE","NW"],

["","","OS"]],

"Hidden": false,

"Counter": 10

},

**Parameterisierungskonzept**

Bei unserem Spiel ist der Inhalt der Tiles wohl der größte Punkt der ausgelagert werden kann. Daher haben wir folgendes Konzept entwickelt:

**Eingang/Ausgang**

Gibt an ob dieses Tile später als Eingang oder Ausgang gewertet werden soll

**Dateipfad**

Gibt den Pfad zur Bilddatei an, die diesen Tile repräsentiert

**Zugangspunkte**

Gibt nach Vorbild eines Kompasses die Seiten an, an denen dieser Tile einen Übergang hat

**Bewegbarkeit**

Gibt an ob dieser Tile im Spiel bewegbar ist

"H":{

"accessPoints": ["W","E"],

"switchable": "true",

"Path": "Path\_horizontal.png",

"Input": false,

"Output": false

},

**Tile-Typ**