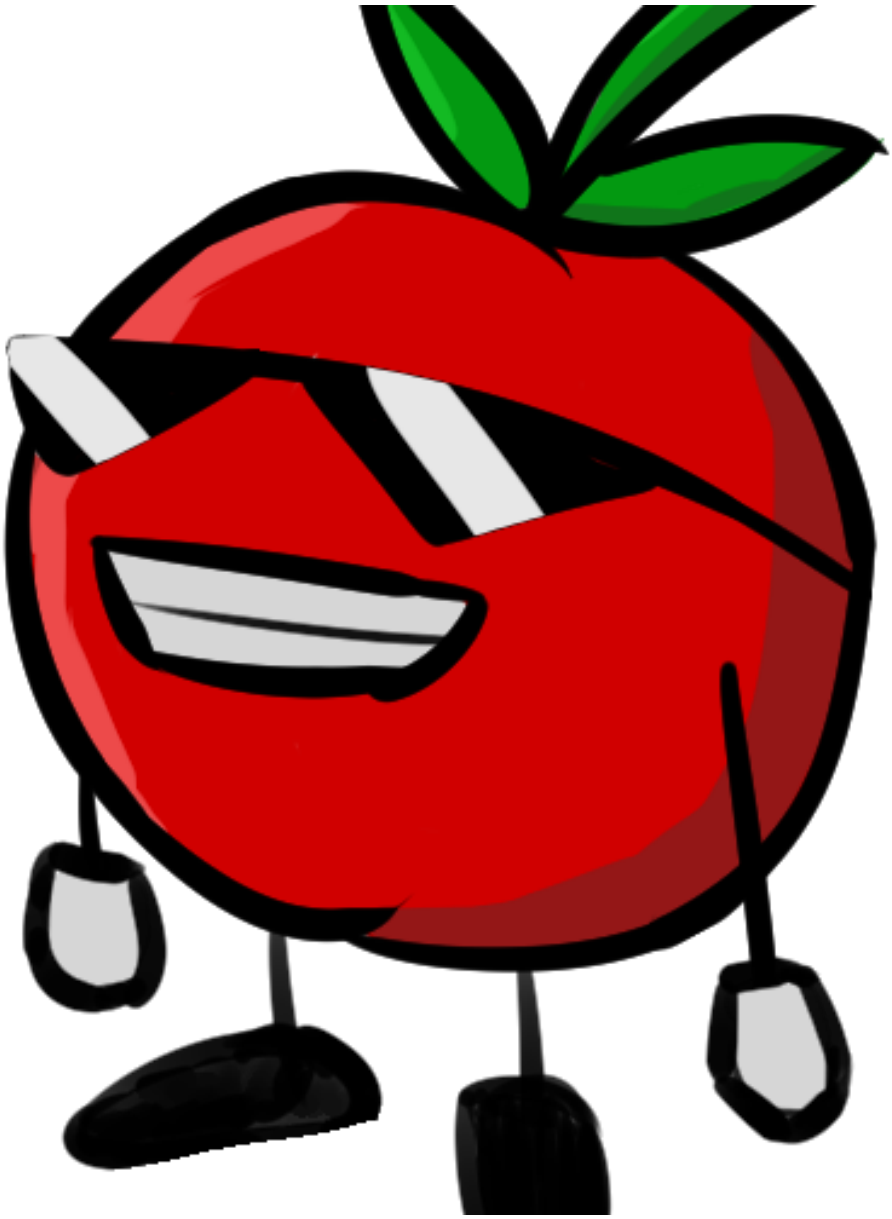
Kwirk-Escape



Art by Frederik Röhr

Dokumentation für das Modul „Webtechnologie Projekt“

von

Nicklas Hummelsheim und Edward Suer

SoSe 2020

Inhaltsverzeichnis

HINWEIS: Dieses Dokument nutzt Querverweise

Abbildungsverzeichnis

Tabellenverzeichnis

1 Anforderungen

Target Device:	Desktop und Browser im Mobile First Ansatz (3 Punkte)
Steuerung:	Tasten und Buttons für die Desktop, Swipe und Touch für die Mobile Version (2Punkte)
Darstellung:	Rasterbasiertes Spiel (1 Punkt)
Levelsystem:	Komplexere Level-definitionen (2 Punkte)
Insgesamt:	8 Punkte / 2 Personen = 4 Punkte / Person

Die expliziten Beschreibungen für die Komplexitätspunkte befinden sich unter 2.3 Ergänzungen und Ansätze, dies wurde hier nur als eine schnelle Übersicht und zur klareren Struktur vorgezogen.

Die tatsächliche Verteilung der Komplexitätspunkte, finden sie weiter unten

Link zum Spiel:

https://webtech.mylab.th-luebeck.de/ss2020/team-04-e/

QR-Code zum Spiel

QR-Code made with http://goqr.me/

2 Abgeleitetes Spielkonzept

Unser Spiel Kwirk-Escape orientiert sich an dem GAMEBOY Spiel Kwirk, aus dem Jahre 1989. In dem Spiel geht es darum, einen Weg zu finden den Raum, in welchem sich die Spielfigur aufhält, zu verlassen. Dabei stößt er auf mehrere Hindernisse, die zum Rätselspaß einladen. So muss der Spieler Blöcke verschieben, durch Drehkreuze durch oder, mit vorhanden Blöcken, Lücken im Boden ausfüllen.

Kwirk-Escape wird als Einzelspieler-Spiel konzipiert und ist Level-basiert, mit der Idee, dass jedes Folgelevel schwerer ist als sein Vorgänger. Es kann dazu kommen, dass man in einem Level nicht mehr weiterkommt (da seine getroffenen Entscheidungen dies Unmöglich machen), dann kann der Spieler das Level in seinen Ursprung zurücksetzen und von vorne anfangen – Gewonnen hat man, wenn man das Level, bzw. alle Level geschafft hat, verlieren kann man, im traditionellen Sinne, nicht. Ein Level gilt als geschafft, wenn der Spieler es schafft seine Spielfigur zur Treppe zu bringen, mit welcher man den Raum verlässt.

Das Spielkonzept soll anhand der folgenden Abbildungen veranschaulicht werden:

Die Bezeichnungen sind wie folgt zu interpretieren:

- # = nicht interagier barer Teil des Levels, dient dazu verschiedene Formen zu ermöglichen
- S = Spielfigur, die der Spieler bewegt
- G = das Ziel bzw. die Treppe, die der Spieler erreichen muss, um das Level zu beenden. Dieses Objekt ist nicht beweglich
- B = ein Block den der Spieler durch seine Figur verschieben kann. Ein Block kann verschieden geformt sein
- L = Ein Loch im Boden, welches durch einen Block (B) gefüllt werden kann, damit es dem normalen Boden entspricht – der Spieler kann sonst nicht über das Loch hinweg
- D in Verbindung mit A = das Drehkreuz. Das D entspricht einem Gelenk, um welchen sich die anliegenden A's bewegen können, sofern der Spieler eins der A's dreht. Siehe: 2.2 Das Drehkreuz
- nicht ausgefüllte Kasten entsprechen dem normalen Boden, auf dem sich die Spielfigur frei bewegen kann

weiteres:

- Verschieden farbige Buchstaben sollen verdeutlichen, dass es sich um zwei oder mehr verschiedene Entitäten handelt
- Ein Pfeil (► ◄ ▲ ▼) kann, zur Verdeutlichung von Interaktionen, bei den Abbildungen beigefügt sein. Dabei steht die Richtung des Pfeils für die Richtung, in die der Spieler geht/ versucht zu gehen. Der Pfeil befindet sich immer oberhalb der entsprechenden Abbildung.

  INFO: diese Anzeige ist kein Teil des Spiels und dient nur dazu, bestimmte Interaktionen zu verdeutlichen
- Die Folgenden Abbildungen dienen zur Veranschaulichung von Spielmechaniken und Interaktionen und besitzen daher nicht unbedingt ein Ziel (G)
- Es wird nur der Block und das Drehkreuz explizit vorgestellt, da die Funktionalitäten der anderen Elemente in den Beispielen erkennbar werden.
- Die Bezeichnungen hier (#, S, G, B usw.) dienen nur zur ersten Visualisierung und entsprechen nicht den Grafiken des Endprodukts

2.1 Der Block

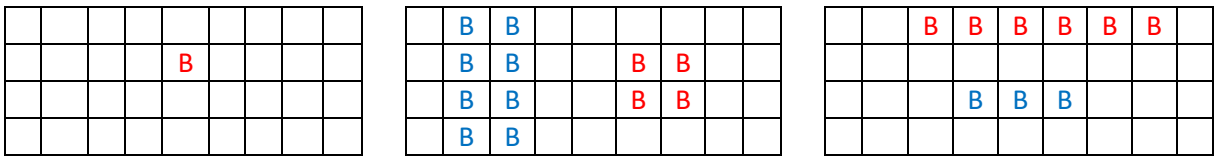


Abbildung 1 Ein Block kann verschieden geformt sein

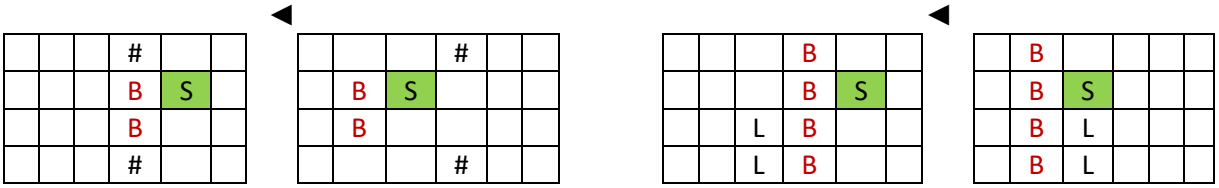


Abbildung 2 Ein Block kann verschoben werden

Auf der linken Seite der Abb. 2 sieht man wie der Spieler den Block nach linksverschiebt um den Weg für sich frei zu machen.

Auf der rechten Seite schiebt der Spieler den Block über eine Lücke hinweg, dabei ist es wichtig zu verstehen, dass ein Block über der Lücke „hängt“, solange einer seiner Bestandteile noch festen Boden unter sich hat. Der Spieler kann den Block nur an den Teilen verschieben, die auch festen Boden unter sich haben.

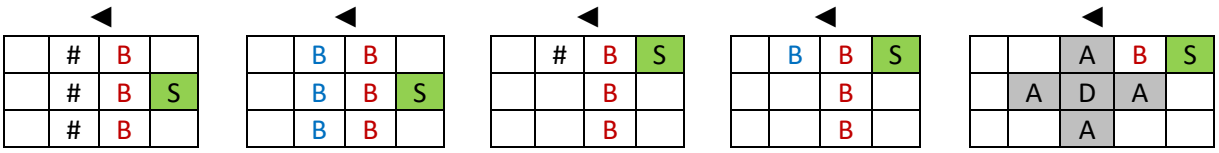


Abbildung 3 Ein Block kann nur verschoben werden, falls Platz ist

Bei allen Beispielen aus Abb. 3 kann der Spieler den roten Block **nicht** nach Links verschieben, da ein anderes Spielelement im Weg ist. Damit ein Block verschoben werden kann müssen alle Blöcke, in der Richtung, in die er geschoben werden soll, frei sein – d.h. entweder normaler Boden oder eine Lücke.

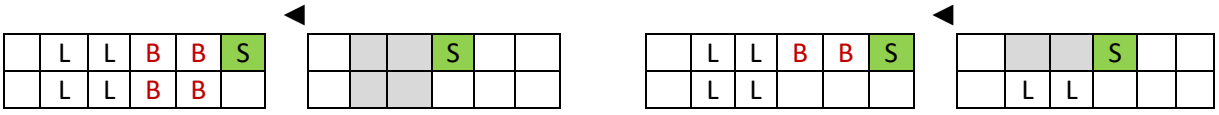


Abbildung 4 Ein Block kann eine Lücke füllen

In Abb. 4 sieht man die Interaktion zwischen einer Lücke und einem Block. Wenn ein Block vollständig über eine Lücke geschoben wird, er also keinen festen Boden mehr unter sich hat, fällt der Block in die Lücke und wird zu einem normalen Boden, auf den der Spieler normal gehen kann. In Abb. 4 ist der neu Entstandene Boden hellgrau eingefärbt.

2.2 Das Drehkreuz

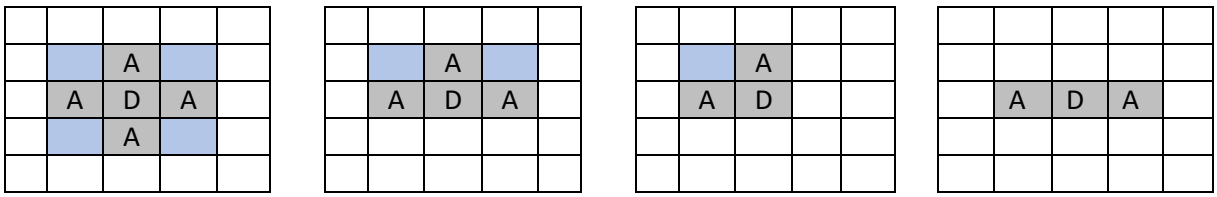


Abbildung 5 Formen des Drehkreuzes

Das Drehkreuz ist ein spezielles Element, man kann die Elemente (A) um das Gelenk (D) in der Mitte drehen. Das Drehkreuz an sich kann nicht (wie der Block) verschoben werden – das Gelenk sitzt fest. Das Kreuz besteht aus den grau eingefärbten Elementen – die blauen Ecken nennen sich Winkel-Eck und haben eine besondere Eigenschaft, welche im nächsten Abschnitt erklärt wird.

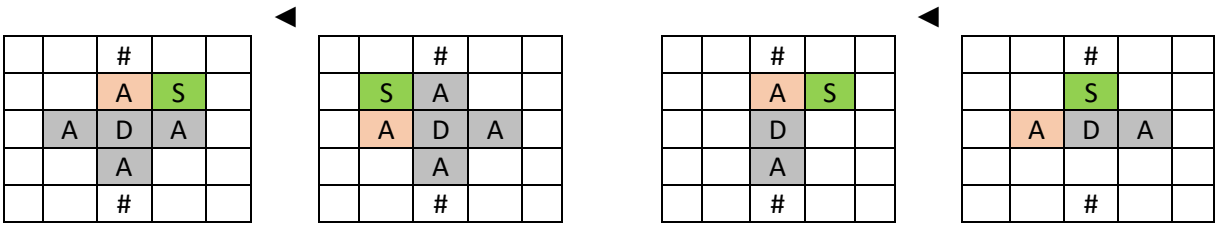


Abbildung 6 So funktioniert das Drehkreuz

Um zu verdeutlichen wie sich das Drehkreuz dreht, ist der Block, den der Spieler anschiebt, eingefärbt.

Auf der linken Seite der Abb. 6 schiebt der Spieler den oberen Teil das Kreuzes nach links, das Kreuz dreht sich um 90° und der Spieler wird um zwei Felder in die Richtung bewegt, in die er geschoben hat. Auf der rechten Seite dreht er ein Gerades Kreuz nach links und wird um ein Feld bewegt. Beide Abbildungen entsprechen einem Eingabekommando des Spielers.

Wenn der Spieler sich im Winkel-Eck eines Drehkreuzes befinden sollte und das Kreuz dreht, bewegt er sich immer um zwei Felder. Bei den anderen Drehungen bewegt er sich nur ein Feld.

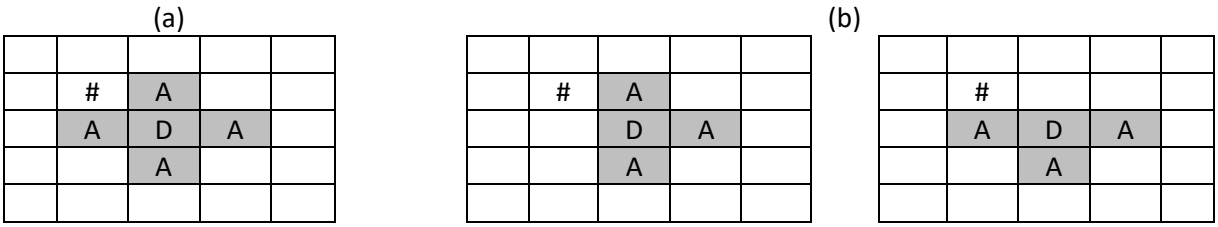


Abbildung 7 Restriktionen beim Drehkreuz

Vorweg: es macht keinen Unterschied ob das Drehkreuz durch ein # oder ein B blockiert wird.

Abb. 7 (a) zeigt ein Drehkreuz welches komplett Bewegungsunfähig ist. Da das Kreuz sogar von einem # blockiert wird, wird es sich nie drehen können, da der Spieler nicht mit dem # interagieren kann. Dieses Beispiel dient nur zur Veranschaulichung, wie ein Drehkreuz blockiert sein kann und sollte **nie** in einem Level genutzt werden.

Abb. 7 (b) Dieses Drehkreuz ist in seinen Interaktionsmöglichkeiten eingeschränkt. So kann der Spieler auf der linken Seite (von b) das Kreuz nur im Uhrzeigersinnbewegen, rechts nur gegen. In beiden Fällen ist die eine Seite, das Ergebnis einer erfolgreichen Drehung, der anderen.

2.3 Ergänzungen und Ansätze

Die Level sind Raster basierend (m x n) wobei der Rand der Level durch # markiert sind. Zudem dienen die # auch als Objekte innerhalb des Levels, damit es nicht immer eine Rechteckige Form hat. Ein Level muss mindestes aus einem Startpunkt und dem Ziel bestehen. Außerdem sollte noch weiter Elemente enthalten sein, damit es ein Rätsel gibt, welches man lösen kann. Der Startpunkt ist der Punkt, an dem die Spielfigur zu Beginn das Levels steht.

Die Level sollen in Form von JSON gespeichert werden und von der Applikation interpretiert werden können. Dadurch lassen sich Level einfach ergänzen, ohne dass man neu Programmieren muss. Aufgrund dessen, dass das Ziel des Spiels ist, Rätsel zu lösen, und die Level auch immer schaffbar sein müssen, werden die Level nicht über Parameter generiert, sondern von uns erstellt.

Die Steuerung des Spiels ist beschränkt auf die vier Bewegungsrichtungen (oben, unten, links, rechts), dem Levelneustart und dem Aufrufen des Menüs.

Am Desktop ließe sich dies über die Pfeiltasten und oder WASD, dem ESC-Button für das Menü und einer weiteren Taste für den Reset (sofern nicht als Option im Menü) verwirklichen.

Für Mobile-Geräte könnte man einzelne Bewegungen des Spielers über Swipes in die jeweiligen Richtungen abbilden. Das Menü könnte man als Double Tap Funktionalität umsetzen, so kann man den Reset-Button auch erreichen.

Dem Entsprechend ist die Steuerung intuitiv nutzbar und braucht keine komplizierten Einleitungen, die Mechaniken des Spiels kann man über Level spielerisch erklären bzw. dem Spieler näherbringen ohne eine Art Tutorial.

Dem Original ließen sich neue interagierbare Objekte hinzufügen, wie z.B. eine Art Laser, welcher einen Weg blockiert (dies würde ein GameOver hinzufügen), oder OneWayGates welche der Spieler nur einseitig passieren kann. Außerdem wäre ein Leveleditor denkbar.

2.4 Beispiele

#	#	#	#	#	#	#	#	#	#	#	#
#				#			B	B			#
#		G					B	B		S	#
#				#			B	B			#
#	#	#	#	#	#	#	#	#	#	#	#

#	#	#	#	#	#	#	#	#	#	#	#
#				#	B	B	B	B			#
#		G					S				#
#				#		B	B				#
#	#	#	#	#	#	#	#	#	#	#	#

Abbildung 8 Beispiel für ein Level

Abbildung 8 zeigt uns ein mögliches Leveldesign, der Spieler muss den Weg frei machen, damit er zum Ziel kommt. Dazu verschiebt er erst den roten Block zweimal nach links, danach den grünen ein, oder zweimal nach links und schlussendlich den blauen Block einmal nach oben. Jetzt kann er ins Ziel gehen.

#	#	#	#	#	#	#	#	#	#	#	#
#				#			B	B			#
#		G					B	B		S	#
#				#			B	B			#
#	#	#	#	#	#	#	#	#	#	#	#

#	#	#	#	#	#	#	#	#	#	#	#
#				#		B	B				#
#		G	B	B	S						#
#				#		B	B				#
#	#	#	#	#	#	#	#	#	#	#	#

Abbildung 9 Ein Levelzustand in dem man neustarten muss

Abbildung 9 zeigt uns ein Level, welches in dem aktuellen Zustand nicht mehr schaffbar ist. Der Spieler hat leider den blauen Block zweimal nach links verschoben, das Ziel verhindert jetzt, dass der Block weiterhin verschoben werden kann und es gibt keinen Weg mehr das Level noch zu schaffen. Der Spieler muss das Level leider neustarten.

#	#	#	#	#					#	#	#	#	#	#						
#				#					#					#						
#		G		#					#	A	D	A		#						
#				#					#					#						
#	#	L	#	#					#	#	#	#	#	#						
	#		#			#	#	#	#		B			#	#	#	#	#	
	#		#	#	#	#													#	
	#						B	B									S		#	
	#			#	#	#													#	
	#	#	#	#		#	#	#	#	#	#	#	#	#	#	#	#	#	#	

Abbildung 10 Beispiel eines nicht Rechteckigen Levels

Hier ist noch ein Link zu einem Video, welches das Gameplay des Originals zeigt.

3 Architektur und Implementierung

Eine PDF der gesamten (und komplett volständigen) Architektur finden Sie: hier

3.1 MVC

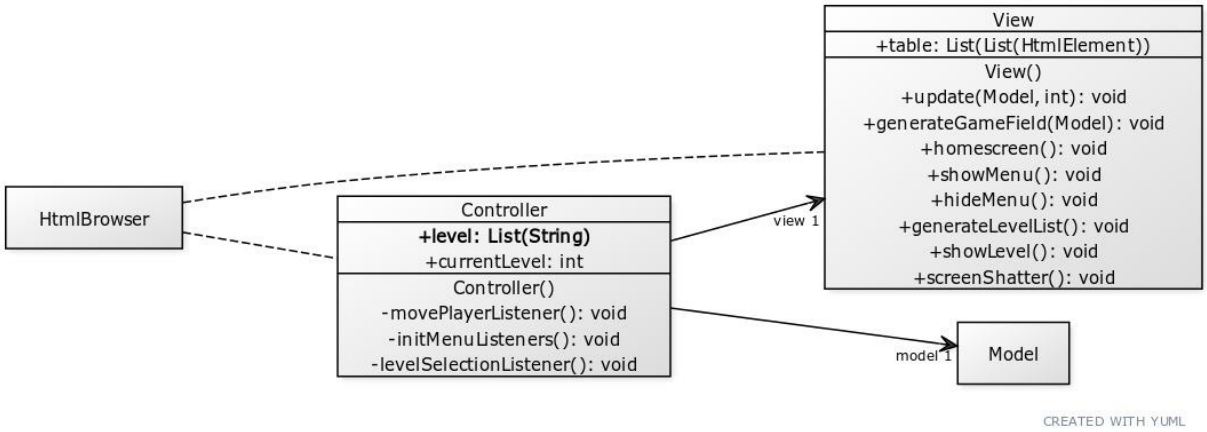


Abbildung 11 das MVC-Pattern

HINWEIS:	- das Tool unterstützt eckige Klammern <> nicht, daher haben wir runde () genommen - querySelector-Datenfelder der View werden nicht mit angezeigt

Abb. 11 zeigt uns die Architektur von Kwirk Escape. Es ist erkennbar, dass die Architektur auf dem MVC-Pattern basiert. So diktiert die View den HTML Browser, welcher Eingaben das Nutzers and den Controller weitergibt. Der Controller wiederum sagt der View, was Sie zu tun hat, unter der Berücksichtigung des Models.

Das Model beinhaltet die gesamte Spiellogik, daher ist es in mehrere Entities unterteilt die alle unter 3.2 Das Model erklärt werden.

Die View kapselt den Dom-Tree und besitzt mehrere Methoden, die dafür sorgen, dass der Browser, die sich verändernden Zustände richtig da stellt. Die View wird unter 3.3 Die View näher erläutert.

Der Controller ist die zentrale Klasse, wenn es um die Verarbeitung von Nutzereingaben geht. Er nutzt diese, um das Model anzutreiben und die View zu update. Details zum Controller finden Sie hier: 3.4 Der Controller.

Unsere Klassen-Struktur erfüllt insofern das MVC-Pattern, das das Model komplett gekapselt funktioniert und die anderen Klassen nicht benötigt. So kann auch eine andere Kombination von View und Controller, das Spiel (Model) darstellen. Wie üblich im MVC ist unser Controller komplett von allen anderen Klassen abhängig. Unsere View nutzt für ihre `update` und `generateGameField` Methoden das Model – daher ist unsere View nicht kapselbar.

3.2 Das Model

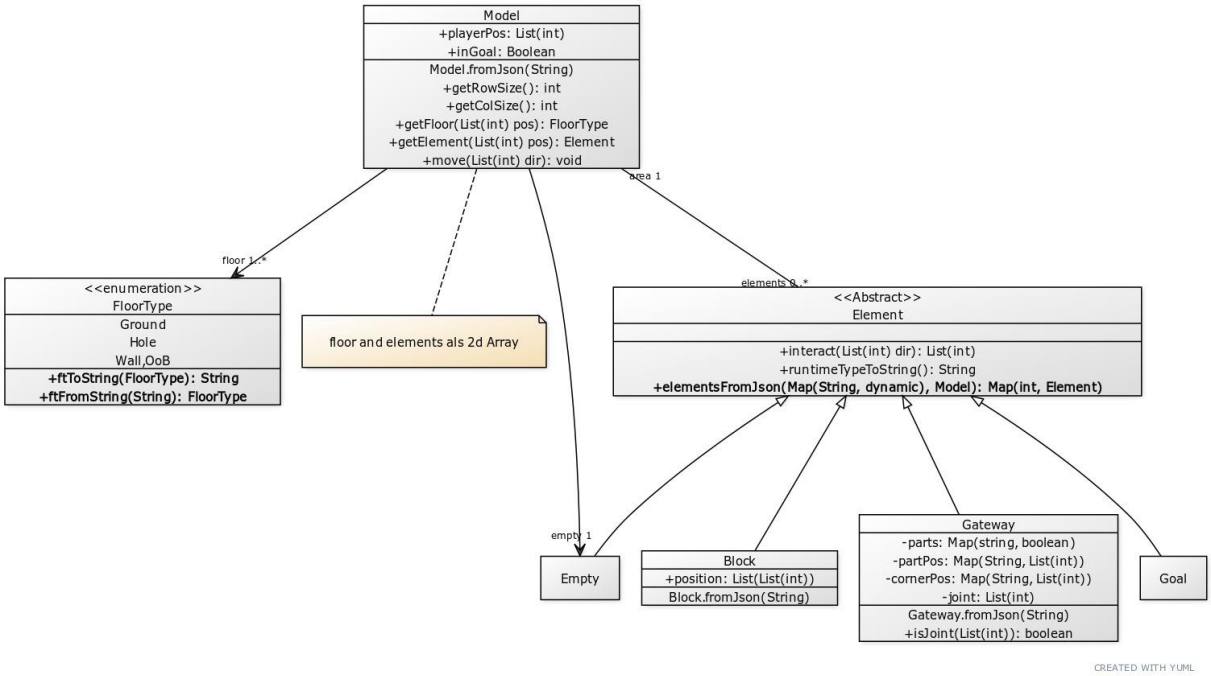


Abbildung 12 Das Model

Hinweis:	private Methoden sind, für eine bessere Übersicht, in dieser Darstellung **nicht** abgebildet, da es sich bei diesen nur um unterstützende Methoden handelt. Außerdem sind die in Bold geschriebenen Stelle static Elemente

Nach dem Spielkonzept aus: 2 Abgeleitetes Spielkonzept, sind folgende Klassen entstanden:

- Model: Das Model des Spiels, enthält die Spieler Position und verwaltet das Spielfeld. Dadurch muss (und kann) der Controller nur mit dem Model kommunizieren.

  - `playerPos`: Position des Spielers
  - `inGoal`: Variable um zu schauen ob der Spieler das Level (erfolgreich) beendet hat
  - `elements`: 2-dimensionals Array zur Angabe der Element Positionen
  - `floor`: 2-dimensionals Array zur Angabe der FloorTypes
  - `empty`: Da die Logik von Empty unabhängig von dessen Position ist, wird nur ein Empty Element erzeugt und immer wieder verwendet. Dazu wird es hier im Model gesichert
  - `getRowSize`: Gibt an wie viele Rows das Spielfeld hat
  - `getColSize`: Gibt an wie viele Cols das Spielfeld hat
  - `getFloor`: Gibt den Floor der angegebenen Position wieder

- `getElement`: Gibt das Element der angegebenen Position wieder

- `move`: Schaut ob sich der Spieler in die angegeben Richtung bewegen kann und falls möglich, interagiert der Spieler dann mit dem Element (diese Interaktion kann auch ein nicht Bewegen des Spielers sein)

- FloorType: Die verschiedenen Böden werden hier als Enum dargestellt, um Sie einfach in einem Array speichern zu können und weil Sie keine eigene Funktionalität besitzen.

  - Ground: Entspricht den vorher nicht ausgefüllten Kasten der verschiedenen Abbildungen. Auf dem Ground-Floor kann der Spieler sich bewegen.
  - Hole: Wurde vorher als Lücke bezeichnet. Dieses Feld kann nicht von Spieler betreten werden und hat eine spezielle Interaktion mit dem Block. Vrgl. Abbildung 4 Ein Block kann eine Lücke füllen
  - Wall: Die Wall wurde bisher als nicht interagier barer Teil des Levels, oder als # bezeichnet. Sie dient als Levelbegrenzung und zum Formen des Levels
  - OoB: Vorher nicht genannt - dieser dient dazu dem Level auch wirklich eine Form geben zu können (in dem er nicht visualisiert wird), da (grafisch) das Spielfeld sonst immer echt mxn wäre

- Element: Das Element ist die Oberklasse von den vier Objekten, mit den der Spieler interagieren kann.

  - `interact`: Implementiert das Bewegungsverhalten des Elements

  - `runtimeTypeToString`: Gibt an um welchen Typ von Element es sich handelt

  - `elementsFromJson`: Ruft die Konstruktoren aller Elemente auf, um so eine Map zu erstellen mit (ID, Element). Die ID wird genutzt um die Position der Elemente auf das Spielfeld richtig zu mappen

  - Block: Ist der Block (B), Interaktionsmöglichkeiten nach: 2.1 Der Block

  - Gateway: Das Gateway ist das Drehkreuz. Interaktionen nach: 2.2 Das Drehkreuz

    - `isJoint`: schaut ob die angegebene Position das joint Element ist

  - Empty: Dieses Feld existierte vorher nicht. Es dient dazu, dass man im Programm nicht mit null pointern umgehen muss. Wenn ein Element Empty ist heißt das einfach, dass da kein Element ist (aus Spielsicht). Interaktion ist mit dem Element immer möglich. Wie oben bereits erwähnt ist die Interaktion unabhängig von der Position des Empty-Elements, dadurch braucht man nur ein einziges, welches nach der Erstellung im Model gespeichert wird

  - Goal: Das Ziel (G), sobald man es erreicht, gilt das Level als geschafft
    Die Interaktion mit dem Goal ist immer möglich. Sobald man dieses Spielfeld betreten hat, gilt das Level als geschafft

Hinweis: genaue Beschreibungen zu ALLEN Methoden, finden Sie im Code

Durch diese Abstrakte Herangehensweise, ist es einfach neue Spiel Elemente hinzuzufügen, ohne was am bestehenden Code zu ändern.

Es ist wichtig zu verstehen, dass Methoden wie „move“ oder „interact“ immer erst prüfen ob dies Möglich ist und falls nicht, den Zustand des Spiels nicht verändern. Anhand der Namen der Methoden lässt sich das nicht unmittelbar erkennen. Es fehlt quasi ein „wantTo“ vor den Bezeichnern.

Des Weiteren nicht erkennbar: Das Model wird per Json über Konstruktoren generiert, die Koordinaten des Spielers werden dabei auf seine Startposition gesetzt.

Zuletzt muss noch erwähnt werden, inwieweit die playerPos zulässig ist.

[0, 1] -> Positive Bewegung in y-Richtung -> nach oben
[0, -1] -> Negative Bewegung in y-Richtung -> nach unten
[1, 0] -> Positive Bewegung in x-Richtung -> nach rechts
[-1, 0] -> Negative Bewegung in y-Richtung -> nach links

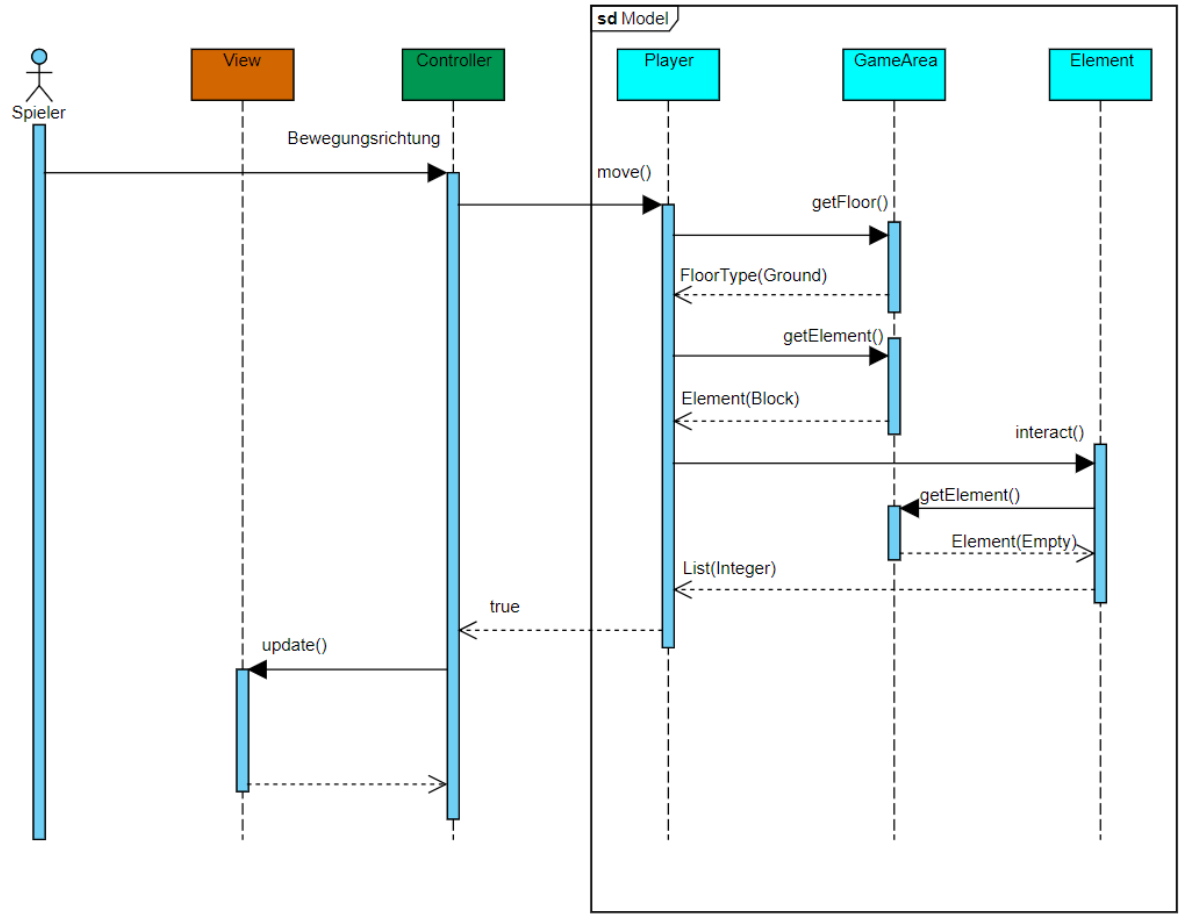


Abbildung 13 Sequenzdiagramm von move()

Zeigt einen Beispielablauf von `move()` wie beschrieben unter dem Eintrag: Player.
Das Diagramm wurde mithilfe von https://online.visual-paradigm.com/ erstellt.

3.3 Die View

```html
<!DOCTYPE html>

<html>

<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>Kwirk Escape</title>
  <link rel="stylesheet" href="styles.css">
  <link rel="icon" type="image/png" href="favicon.ico">
  <script defer src="main.dart.js"></script>
</head>

<body>
  <div class="container">
    <h1 id="title"> Kwirk Escape</h1>
    <div id="notification">press 'Esc' or 'double tap' to access the menu</div>
    <table id="selectLevel"></table>
    <table id="gameField"></table>
    <div id="menu">
      <div id="credits">
        <p><em>Credits</em></p>
        <p><b>Development Team:</b><br>
          Hummelsheim, Nicklas<br>
          Suer, Edward</p>
        <p><b>Graphic Design:</b><br>
          <a href="https://twitter.com/fred_was_geht" target="_blank" rel="noopener">Roehr, Frederik</a></p>
        <p><b>Pixel-Art:</b><br>
          <a href="http://opengameart.org/content/dungeon-crawl-32x32-tiles" target="_blank" rel="noopener">opengameart.org</a></p>
      </div>
      <p id="cleared"></p>
      <button type="button" id="close">Close</button>
      <button type="button" id="restart">Restart</button>
      <button type="button" id="levelSelection">Select Level</button>
      <button type="button" id="nextLevel">Next Level</button>
    </div>
  </div>
</body>

</html>
```

Abbildung 14 Das HTML-Dokument

Die View dient der Darstellung des Spiels und Menüs. Die View besteht aus einem HTML-Dokument und Methoden, die den DOM-Tree des Dokuments manipuliert.

Die View besitzt ein initiales HTML-Dokument, welches, basierend auf den Nutzereingaben, manipuliert wird. So kann die View sowohl ein Menü als auch das Spiel darstellen. Damit das funktioniert, muss die Klasse View als clientseite Logik von dem Dokument geladen werden.

Alles was für den Nutzbar zu sehen ist, befindet sich im `container`. Zunächst sieht er title, `notification` und `selectLevel`, sollte er ein Level ausgewählt haben, so werden die drei genannten Elemente versteckt und man sieht das `gameField`.

Von Natur aus nicht sichtbar, ist das `menu`, mit seinen (vom Zustand abhängigen) Buttons.

Die Folgenden Methoden werde nie von der View selbst, sondern immer vom Controller aufgerufen.

`update()`	Diese Methode aktualisiert das Spielfeld, basierend auf dessen Zustand. Dafür wird das Model übergeben. Die Methode kann nur durch Spieler Inputs über den Controller aufgerufen werden.
`generateGameField()`	Diese Methode dient dazu ein Spielfeld zu generieren, also die View so zu manipulieren, dass wir nicht mehr das Hauptmenü, sondern ein Spielfeld sehen. Sie wird aufgerufen, nachdem der Nutzer aus der `selectLevel` Tabelle ein Level ausgesucht hat. Außerdem wird die Methode auch genutzt, wenn der Spieler das Level neustarten möchte. Dazu wird das Level in das `gameField` geladen.

	Die Funktionalität als Restart kann nur über das Menü des Spiels, also nach dem Aufruf des Menü-Buttons (-> `menu()`) gecallt werden.
`homeScreen()`	Diese Methode zeigt den Startbildschirm von Kwirk Escape an. Hier kann man Level auswählen oder das Menü aufrufen
`showMenu()`	Zeigt das Menü an (es ist per doppelclick oder über die ESC-Taste erreichbar). Wenn das Menü während des Spielens gerufen wird, kann der Spieler wieder in Hauptmenü, das Level neu starten oder das Menü schließen. Sollte der Spieler im Ziel sein, kann er zusätzlich noch das nächste Level per Knopfdruck laden. Auf der Startseite zeigt das Menü die Credits an (z.T. mit Verlinkungen) und lässt sich auch schließen
`hideMenu()`	Lässt das Menü wieder verschwinden
`generateLevelList()`	Wird nur einmal gecallt (wenn man die Website lädt), zeigt eine Liste von verfügbaren Leveln an
`showLevel()`	Lässt das Level anzeigen
`screenShatter()`	Sollte der Spieler in eine Richtung gehen wollen, dies gelingt aber nicht (da die Bewegung nicht möglich ist) wird dies zusätzlich visualisiert, indem das gameField wackelt.

3.4 Der Controller

Der Controller ist, wie üblich im MVC, zur Ablaufsteuerung des Programms zuständig. In Unserem Spiel verarbeitet er dazu Nutzereingaben per Touchscreen oder Tastatur/Maus. Im Spiel selbst, kann man nur nach oben, unten, rechts und links gehen, sowie ein Menü aufrufen.

Kwirk Escape ist ein Spiel, welches zum Rätsel einlädt, dabei werden auf Scores, GameOvers oder TimeEvents (bisher) verzichtet. Dadurch ist es komplett von den Inputs des Spielers abhängig. Daher kann der Spieler nur auf eine Sache direkten Einfluss ausüben (die Spielfigur). So bildet sich ein schlanker Controller (was die Steuerung des Spiels angeht) ohne allzu komplexe Sachverhalte.

Der Controller wird als erstes instanziiert und steuert ab diesem Zeitpunkt alles was passiert. Zunächst werden die Level geladen und (als json String) in eine Liste gespeichert. Außerdem wird der „local Storage“ instanziiert, sofern noch nicht existent. Dach werden alle Listener erstellt. Dies beinhaltet die Steuerung des Spielers (Swipe oder Pfeiltasten/WASD), das Aufrufen des Menüs (DoubleTap oder ESC-Taste), die Buttons des Menüs und die Levelauswahlliste.

Die Listener übernehmen danach die Kontrolle über die View, und daher auch über den DOM-Tree, sowie kommunizieren sie mit dem Model.

Interessant ist, dass die Methode `movePlayer(dir)` einen boolschen Wert liefert, sollte der Rückgabewert false sein, so heißt das, dass der Spieler sich nicht bewegen konnte oder aber, dass er sich bereits im Ziel befindet – in dem Fall muss die update Methode nicht gecallt werden (Da sich nichts ohne Spielerinteraktion bewegt). Ansonsten wird die View nach dem Zustand des Models aktualisiert.

4 Levelkonzept

Hinweis: Kwirk Escape sieht keine Parameterisierungskonzepte vor

Level von Kwirk Escape werden als JSON eingelesen und interpretiert. Dies führt dazu, dass man Level ohne Probleme (also ohne was im Programm code zu ändern) hinzufügen, ändern oder löschen kann.

```
{
    "FloorType": [
      ["Wall", "Wall",    "Wall",   "Wall",   "Wall",   "Wall",   "Wall",   "Wall",   "Wall"],
      ["Wall", "Ground",  "Ground", "Ground", "Wall",   "Ground", "Ground", "Ground", "Ground", "Wall"],
      ["Wall", "Ground",  "Ground", "Ground", "Ground", "Ground", "Ground", "Ground", "Ground", "Wall"],
      ["Wall", "Ground",  "Ground", "Ground", "Wall",   "Ground", "Ground", "Ground", "Ground", "Wall"],
      ["Wall", "Wall",    "Wall",   "Wall",   "Wall",   "Wall",   "Wall",   "Wall",   "Wall",   "Wall"]],

    "Element": [
      {"id": 0, "type": "Empty"},
      {"id": 1, "type": "Goal"},
      {"id": 2, "type": "Block", "position": [[2, 5]]}],

    "Elements" : [
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 1, 0, 0, 2, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

    "Player": [2, 7]
}
```

Abbildung 15 Ein Level in Kwirk Escape

In Abb. 15 sehen wir ein Level von Kwirk Escape als .json. Es wird folgends gespeichert:

FloorType: speichert die Bodentypen des Spiels in einem 2-dimensionalen Array. Dabei ist der Inhalt eines Eintrags im Array gleich der String-Repräsentation des FloorTypes

Element: speichert alle Elemente, die im Level vorkommen auf eine ID. Das Empty Element wird nur einmal gespeichert, da es sich um ein statisches Element in jedem Spielfeld handelt. Die ID wird genutzt, um die Position der Elemente einfacher zu halten. Dazu wird zusätzlich der Eintrag „Elements“ genutzt.

Elements: ist ein 2-dimensionales Array welche die explizite Position der Elemente beinhaltet. Dazu werden als Referenz die (nur im .json befindlichen) IDs genutzt. type bestimmt den Typen des Elements (z.B. Empty, Block, usw.) Bestimmte Elemente haben noch weitere Felder. So hat Block noch ein Feld position und GateWay position und joint. Dies ist vom Programm so vorgegeben!

Player: speichert lediglich die Startposition des Spielers in einem Array

All diese Einträge (der .json) müssen existieren, damit ein Level erstellt werden kann.

Im Controller wird, bei der Instanziierung, alle Level eingelesen und als json String gespeichert. Dies dient dem Zweck, dass, sobald man die Seite geladen hat, auch offline spielen kann (solange man die Seite nicht neu lädt). Im Nachhinein, wird dann das spezifische Level an die Konstruktoren des Models übergeben.

Eine RuleSheet und weiter Informationen zu dem Levelsystem in Kwirk Escape, finden Sie hier

5 Anforderungen und Beiträge

5.1 Nachweis der Anforderungen nach „Corona Chronicles“

Hinweis: Die folgenden Einträge basieren und unserer Bewertung

ID	Kurztitel	✓	O	×	Erläuterung
AF-1	Single-Page-App	X			Kein Multiplayer, wird entsprechend deployt
AF-2	Komplexität vs Konzept	X			v.a. schnell und intuitiv fassbar
AF-3	Dom-Tree	X			Kwirk Escape folgt dem MVC und DOM-Tree
AF-4	Target-Device	X			Mobile und Desktop aber Mobile First
AF-5	Mobile-First	X			Swipes und DoubleTap wird genutzt, mehr nicht
AF-6	Intuitive-Spielfreude	X			v.a. schnell und intuitiv erfassbar, spaß durch Rätselnatur
AF-7	Levelkonzept	X			Level werden als .json interpretiert
AF-8	Speicherkonzepte	X			Es wird local storage genutzt, um den Levelfortschritt zu speichern
AF-9	Basic Libraries	X			Es wurden Standard Libraries verwendet
AF-10	Keine Spielereien	X			Es wurden keine Spielereien implementiert
AF-11	Dokumentation	X			Alles wurde entsprechend dokumentiert

Table 1 Nachweis der Anforderungen

5.2 Verantwortlichkeiten im Projekt

Hinweis: Es wurde zumeist im Pair-Programming gearbeitet

Komponente	Detail	Hummelsheim	Suer
Model	Model (Klasse)	V	U
	- FloorType	V	U
	- Element	V	U
	- Empty	V	U
	- Block	V	U
	- Gateway	V	U
	- Goal	V	U
View	HTML-Dokument	U	V
	Gestaltung	U	V
	View-logik	U	V
Controller	Eventhandeling	U	V
	Parametrisierung	entfällt	
	Level	V	U
Dokumentation		V	U

INFO:
V = Verantwortlicher
U = Unterstützer

Table 2 Verantwortlichkeiten im Projekt

5.3 Tatsächliche Verteilung der Komplexitätspunkte

Komponente	Gewähltes Konzept	Punkte	Hummelsheim	Suer
Target Device	Desktop und Browser	3	1	2
Steuerung	Gestensteuerung	2	1	1
Darstellung	rasterbasiert	1	0	1
Levelsystem	komplex	2	2	0
Insgesamt:		8	4	4

Table 3 Tatsächliche Verteilung der Komplexitätspunkte