Stephan Barth
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
WS 2017/18

Softwareentwicklungspraktikum Nebenfach
Hauptprojekt Boulder Dash

Version 1

1 Einführung

Das Hauptprojekt ist ein Computerspiel, das sich am Klassiker „Boulder Dash™“ von 1984 orientiert. Es gibt viele verschiedene Varianten dieses Klassikers, darunter auch frei verfügbare Varianten mit Open-Source-Lizenz, z.B. Epiphany oder eine dem Orginal nahekommende JavaScript-Version auf https://github.com/jakesgordon/javascript-boulderdash/.

In diesem Spiel steuert man einen Abenteurer, der Edelsteine einsammeln und mit diesen zu einem Ausgang kommen will. Dabei gräbt man sich zumeist durch das Erdreich, weicht Steinen und Gegnern aus und muss manchmal die Edelsteine den Gegnern abnehmen. Das Spiel läuft fortwährend ab, die Spielkarte ist immer kästchenbasiert. Viele verschiedene Spielkarten sind möglich (im folgenden auch „Level“ genannt).

Die folgende Beschreibung soll gleichermaßen die Spielregeln wiedergeben, so dass ein Spieler damit das Spiel spielen kann, wie auch eine ausreichend genaue technische Beschreibung liefern, damit man das Spiel anhand dieser Beschreibung implementieren kann.

Wenn in der Beschreibung steht, was der Spieler machen kann, dann ist damit natürlich auch gemeint, dass man diesen Aspekt implementieren muss.

2 Überblick

Das Spiel hat zwei mögliche Zustände:

- Levelübersicht: Alle momentan spielbaren Level werden gezeigt. Man kann sich eines dieser Level auswählen und spielen.
- Im Level: Dies ist das eigentliche Spiel. Man versucht das Level so gut wie möglich zu lösen.

Es wird gespeichert, welche Level bestanden wurden und falls ja, wie gut. Erfolgreiches Bestehen von Leveln kann weitere Level freischalten.

2.1 Levelübersicht

In der Levelübersicht wird angezeigt, welche Level es gibt und welche davon man als nächstes spielen kann. Nicht alle Level sind gleich am Anfang freigeschaltet. Um kompliziertere Level zu spielen, muss man in den einfachen Leveln erst eine bestimmte Mindestpunktzahl erreichen.

Punkte bekommt man durch erfolgreiches Lösen von bereits freigeschalteten Leveln. Für jeden erfolgreich gelösten Level gibt es ein bis drei Punkte. In die Punktvergabe fließt ein, wie schnell man das Level absolviert und wie viele Edelsteine man dabei sammelt. Wie sich daraus die Punkte berechnen, ist für jedes Level einzeln festgelegt, siehe dazu Abschnitt 3.

Die Zusammenstellung von Leveln und die Entscheidung, wann diese freigeschaltet werden, ist Ihnen überlassen.

2.2 Im Level

Der Spieler sieht einen rechteckigen Ausschnitt der ebenfalls rechteckigen Spielkarte des Levels. Die Spielkarte ist in ein Gitter von quadratischen Feldern unterteilt.

Jedes der quadratischen Felder ist mit genau einem Spielobjekt gefüllt. Das kann die Figur des Spielers sein, ein Felsblock, ein Gegner, ein Edelstein, ein Ausgang, Erdreich, welches der Spieler durchgraben kann, oder eines von vielen weiteren Objekten. Die folgende Abbildung zeigt ein stilisiertes Beispiel eines Levels der Breite 7 und der Höhe 5. Der Feldinhalt `ME` entspricht der Spielfigur, `PATH` ist ein begehbarer Weg, `MUD` ist Erdreich, durch das man sich graben kann, `WALL` eine nicht betretbare Wand, `STONE` ist ein Stein, der herunterfallen kann, `GEM` ist ein Edelstein und `EXIT` ein Ausgang.

WALL	WALL	WALL	WALL	WALL	WALL	WALL
MUD	MUD	MUD	MUD	STONE	MUD	EXIT
MUD	MUD	MUD	MUD	MUD	MUD	WALL
PATH	PATH	ME	MUD	MUD	MUD	WALL
MUD	MUD	MUD	MUD	GEM	MUD	WALL

Die Karte der meisten Level im Spiel wird natürlich größer sein, typische Level haben in jede Richtung mindestens 15 Felder, es kann aber auch deutlich größere Level geben.

Ziel des Spiels ist es, den Ausgang des Levels zu erreichen und dabei möglichst viele Edelsteine einzusammeln. Dabei muss man sich vor allem vor fallenden Felsblöcken vorsehen, die sich beim Graben durch das Erdreich lockern können. Auf der Karte kann es auch Gegner geben, welche selbstständig durch die Karte ziehen. Diese Gegner können die Spielfigur in Gefahr bringen, andererseits kann man ihnen manchmal auch ihre Schätze abnehmen. Der Ausgang öffnet sich erst, wenn man eine bestimmte Mindestanzahl an Edelsteinen gesammelt hat und obendrein ist auch noch die Zeit beschränkt.

Ein Level ist erfolgreich absolviert, wenn man es schafft, genügend Edelsteine zu sammeln und damit rechtzeitig zu einem Ausgang zu kommen. Dann erhält man ein bis drei Punkte, je nachdem, wie viele Edelsteine man gesammelt hat und wie schnell man war. Je mehr Punkte man in allen Leveln zusammengenommen gesammelt hat, desto mehr Level stehen dann im weiteren Spiel zur Auswahl.

Ist die zum Bestehen des Levels notwendige Zahl an Edelsteinen in der erforderlichen Zeit eingesammelt, und betritt die Spielfigur einen Ausgang, so ist dieses Level gewonnen. Danach wird wieder die Levelübersicht angezeigt.

3 Level und Karten

In diesem Abschnitt beschreiben wir, welche Daten die einzelnen Level enthalten und wie sie gegeben sind. Damit definieren wir das Datenmodell, welches im nächsten Abschnitt die Basis für die Definition der Spielmechanik sein wird.

Level werden von Dateien im JSON-Format eingelesen. Eine Level-Datei enthält alle Daten, die man benötigt, um einen Level zu spielen. Einige Beispiellevel stehen auf der Praktikumsseite zum Download bereit.

Der prinzipielle Aufbau einer JSON-Leveldatei ist:

```
{ "name": "Gem Fever III",
  "width": 42,
  "height": 36,
  "gems": [1, 4, 16],
  "ticks": [1023, 511, 127],
  "pre":[
    ...
    ],
  "post":[
    ...
    ],
  "maxslime":15,
  "map":[
    ...
    ]
}
```

Die verschiedenen Attribute haben folgende Bedeutung:

- `name`: Welcher Name soll für dieses Level angezeigt werden, so lange man es spielt?
- `width`, `height`: Wie breit und wie hoch ist dieses Level?
- `gems`: Wie viele Edelsteine muss man einsammeln, um 1, 2 oder 3 Punkte in diesem Level zu bekommen?

  Man bekommt allerdings für das Level die Punkte nicht jedes Mal, sondern nur die Punkte, die maximal erreicht wurden. Besteht man beispielsweise mit 1, 2, 1 und dann mit 2 Punkten, so bekommt man insgesamt 2 Punkte.
- `ticks`: Wie viele Ticks hat man zur Verfügung, um 1, 2 oder 3 Punkte in diesem Level zu bekommen?

  Hier gilt für die Punkte das gleiche, wie bei den Edelsteinen.
- `pre`, `post`: In verschiedenen Leveln kann es viele verschiedene Gegenstände mit ganz unterschiedlichem Verhalten geben. Bestimmte Gegenstände sollen in den meisten Leveln das gleiche Verhalten haben. Steine, zum Beispiel, werden in den meisten Leveln einfach nach unten fallen. Für solche Gegenstände ist es sinnvoll das Verhalten ein einziges Mal für alle Level im Programm zu kodieren. Es gibt aber auch Gegenstände ohne feste Eigenschaften. Diese können sich je nach Level ganz unterschiedlich verhalten.

Zum Beispiel könnte sich Feuer in verschiedenen Leveln unterschiedlich ausbreiten und manche Gegenstände könnten in einem Level brennbar sein und in einem anderen nicht.

In den Attributen `pre` und `post` können Listen von Levelregeln angegeben werden, mit denen die Spielmechanik angepasst werden kann. Damit kann dann zum Beispiel das Verhalten von Gegenständen ohne feste Eigenschaften festgelegt werden. Es kann auch das Verhalten von eingebauten Gegenständen verändert werden. Die Details zu diesen Regeln werden im Abschnitt 4.5 erklärt. Beide Blöcke dürfen fehlen, in dem Fall wird der entsprechende Block als leere Liste behandelt.

- `maxslime`: Optionales Attribut: Maximale Anzahl von Schleimfeldern auf der Karte. Falls es nicht gegeben ist, so ist die Menge an Schleim unbegrenzt. Siehe Kapitel 4.4 für Details zum Schleim.

- `map`: Hier wird die eigentliche Karte des Levels gespeichert. Dies geschieht als Array von Arrays. Jeder innere Array steht für eine Zeile der Karte. Alle Zeilen haben die gleiche Länge.

  Jeder Eintrag in einem Zeilen-Array entspricht einem Feld auf der Karte. Der Eintrag ist entweder ein JSON-Objekt von der Form

  ```
  {
    "token": "swapling",
    "values": {"direction": 1}
  }
  ```

  oder ein String. Ein String `"foo"` ist dabei als Kurzschreibweise für ein Objekt mit leerem `"values"`-Eintrag zu verstehen: `{"token": "foo", "values": {}}`

  Um die Bedeutung dieser Einträge zu erklären, müssen wir zunächst das Datenmodell für Karten genauer erklären.

3.1 Karten

Wir haben bereits gesehen, dass die Spielkarte ein rechteckiges Gitter von Feldern ist. In diesem Abschnitt wird genau beschrieben, welche Daten die Felder enthalten können.

Zunächst enthält jedes Feld stets genau einen Gegenstand. Es gibt eine Reihe von möglichen Gegenständen, die im Folgenden aufgezählt sind. Die folgenden Grundgegenstände haben in allen Leveln das gleiche Verhalten, das in dieser Anleitung festgelegt wird:

- `ME`: Die Spielfigur, welche gesteuert wird.
- `MUD`: Erdreich, durch das man sich graben kann.
- `STONE`: Ein verschiebbarer Stein; kann herunterfallen.
- `GEM`: Einzusammelnder Edelstein; kann herunterfallen.
- `EXIT`: Der Ausgang, in den man gelangen soll, sobald man genügend Edelsteine gesammelt hat.
- `WALL`: Eine unzerstörbare, unverrückbare Wand.

- `BRICKS`: Eine Ziegelsteinwand.
- `PATH`: Freier Weg.
- `EXPLOSION`: Brennende Luft.
- `SLIME`: Schleim. Verbreitet sich, wird zu Steinen oder Edelsteinen.
- `SWAPLING`: Gegner, der sich nur hin- und herbewegt.
- `BLOCKLING`: Gegner, der sich gemäß linker-Hand-Regel an der Wand entlang bewegt.

  Linke-Hand-Regel bezeichnet dabei das allgemeine Orientierungsverfahren für ein Labyrinth, die besagt, man solle sich so bewegen, wie man sich bewegen würde, wenn man durchgehend die linke Hand an der Wand lässt. Der `BLOCKLING` folgt immer der Wand auf der linken Seite.
- `XLING`: Gegner, der sich gemäß rechter-Hand-Regel an der Wand entlang bewegt.

  Die rechte-Hand-Regel ist so ähnlich wie die linke-Hand-Regel, nur eben spiegelverkehrt.

Neben den Grundgegenständen gibt es noch weitere Gegenstände ohne feste Eigenschaften. Ihr Verhalten wird von jedem Level selbst in seiner JSON-Datei spezifiziert. Zu der Implementierung dieser Verhaltensweise, lesen Sie sich bitte die Levelregeln, Abschnitt 4.5 durch. Diese Gegenstände sollen kein Standardverhalten durch ihre Implementierung zugewiesen bekommen. Diese Gegenstände sind:

- `GHOSTLING`: Gegner, der sich auf levelabhängige Weise bewegt.
- `FIRE`: Feuer
- `NORTHTHING`, `EASTTHING`, `SOUTHTHING`, `WESTTHING`: Ein in die entsprechende Richtung zeigender, ansonsten nicht weiter spezifizierter Gegenstand.
- `POT`: Ein Kessel
- `SIEVE`: Ein Sieb
- `SAND`: Ein Sandhaufen

Jedes Feld enthält genau einen der oben aufgeführten Gegenstände. Zusätzlich können in jedem Feld noch sogenannte Zusatzwerte gesetzt sein. Zusatzwerte werden für die Implementierung der Spielmechanik verwendet. Es gibt zum Beispiel einen Zusatzwerte `FALLING`, welcher aussagt, ob sich der Gegenstand im Feld gerade im freien Fall befindet. Es gibt eine feste Anzahl von Zusatzwerte, die im Abschnitt 4.3 aufgelistet sind. In jedem Feld hat jeder Zusatzwert einen festen nicht-negativen Zahlenwert. Wenn davon die Rede ist, dass ein Zusatzwert gesetzt ist, so ist damit gemeint, dass er einen Wert > 0 hat. Ist der Zusatzwert $= 0$, so spricht man von nicht gesetzt.

Mit diesen Definitionen können wir nun die Bedeutung der Einträge in den `"map"`-Arrays der JSON-Dateien erklären. Ein JSON-Objekt der Form

```
{ "token": "swapling", "values": {"loose": 1, "falling": 0}}
```

beschreibt die Daten eines einzelnen Feldes. Der Gegenstand im Feld ist im Attribut `"token"` gegeben. Wenn ihre Werte im Attribut `"values"` nicht explizit aufgezählt sind, so werden Zusatzwerte nicht gesetzt. Im Beispiel kann man `"falling": 0` also auch weglassen.

4 Spielmechanik

Das Verhalten der verschiedenen Gegenstände in einer Karte wird durch die Spielmechanik spezifiziert. Die Spielmechanik ist rundenbasiert. Wird ein Level gespielt, so beginnt man anfangs mit der in der JSON-Datei spezifizierten Karte. Die Werte in der Karte werden dann fünf mal in der Sekunde in sogenannten Ticks aktualisiert. Ein Tick ist die kleinste Zeiteinheit des Spiels. Dieser Abschnitt beschreibt, was bei einem Tick passieren soll.

4.1 Ausgangssituation

Zu Beginn jedes Levels wird der Spielzustand initialisiert. Der Spielzustand umfasst die Werte in der Karte des Levels; dieser werden so initialisiert, wie in der JSON-Datei des Levels spezifiziert. Der Spielzustand enthält folgende weitere Werte:

- Anzahl der bereits eingesammelten Edelsteine, wird am Levelanfang auf 0 gesetzt.
- Vergangene Zeit in Sekunden seit Levelstart oder Startzeitpunkt: Dies wird benötigt, um zu sehen, ob die Zeit schon abgelaufen ist, aber auch, um anzuzeigen, wie viel Zeit noch übrig ist.

4.2 Berechnung eines Ticks

Der Spielzustand wird nun in fünf Ticks pro Sekunde aktualisiert.

Ein einziger Tick läuft folgendermaßen ab:

- Zurücksetzen der Zusatzwerte aller Felder entsprechend ihrer Bedeutung. Die Bedeutung der einzelnen Zusatzwerte ist im folgenden Abschnitt 4.3 erklärt.
- Die Levelregeln `pre` werden ausgeführt. Vergleiche dazu die Levelregeln in Abschnitt 4.5.
- Der Levelzustand wird entsprechend der Hauptregeln verändert, siehe Abschnitt 4.4.
- Die Levelregeln `post` werden ausgeführt. Vergleiche dazu die Levelregeln in Abschnitt 4.5.
- Die Berechnung ist damit zuende. Dies ist ein guter Zeitpunkt, um den aktuellen Spielzustand anzuzeigen.

4.3 Zusatzwerte und ihre Bedeutung

Jedes Feld enthält eine Reihe von Zusatzwerten. Dabei ist jeweils für jedes Feld einem Attribut eine natürliche Zahl zugeordnet.

In einigen Fällen ist es nur von Bedeutung, ob ein Zusatzwert $= 0$ oder $\neq 0$ ist. In den Fällen spricht man davon, dass ein Zusatzwert gesetzt ist ($\neq 0$) oder gelöscht ist ($= 0$). Soll ein Zusatzwert gesetzt werden ohne Angabe einer Zahl, so ist gemeint, dass der Wert auf 1 gesetzt wird.

Ferner gibt es noch eine Reihe von Zusatzwerten, die global sind, d.h. diese haben nur einen Wert auf der Karte und werden so behandelt, als wären sie auf jedem Feld gleich. Globale Zusatzwerte müssen natürlich nicht für die interne Darstellung auf jedem Feld gespeichert werden, es ist ausreichend sie für die ganze Karte einmal zu speichern.

Für die Spielmechanik gibt es folgende Zusatzwerte mit folgenden Bedeutungen:

- `MOVED`: Ein Gegenstand wurde in das Feld hinein oder aus dem Feld heraus bewegt. Wird am Anfang jedes Ticks auf 0 zurückgesetzt.
- `FALLING`: Der Gegenstand im Feld ist momentan im Fall. Wird am Anfang jedes Ticks auf 0 zurückgesetzt.
- `LOOSE`: Der Gegenstand im Feld kann herunterfallen. Wird am Anfang jedes Ticks auf allen Feldern mit dem Gegenstand `STONE` oder `GEM` gesetzt; auf allen anderen wird es gelöscht.
- `SLIPPERY`: Ein Gegenstand, der über diesem Feld liegt, bleibt nicht liegen, sondern fällt wegen der Rutschigkeit seitlich nach unten. Wird am Anfang jedes Ticks auf allen Feldern mit den Gegenständen `STONE`, `GEM` und `BRICKS` gesetzt; auf allen anderen wird es gelöscht.
- `PUSHABLE`: Der Gegenstand in diesem Feld kann von der Spielfigur verschoben werden. Wird am Anfang jedes Ticks auf allen Feldern mit `STONE` gesetzt; auf allen anderen wird es gelöscht.
- `BAM`: Der Gegenstand in diesem Feld explodiert gleich. Wird am Anfang jedes Ticks auf allen Feldern auf 0 gesetzt.
- `BAMRICH`: Der Gegenstand in diesem Feld explodiert gleich, mit einer Explosion, die Edelsteine hinterlässt. Wird am Anfang jedes Ticks auf allen Feldern auf 0 gesetzt.
- `DIRECTION`: Strategieinformationen der Gegner. Dabei bedeutet 1 nach rechts, 2 nach oben, 3 nach links und 4 nach unten.

  Wird am Anfang eines Ticks nicht verändert.
- `A`, `B`, `C`, `D`: Dies sind Zusatzwerte, welche in den Levelregeln verwendet werden können. Die Hauptregeln sollen diese Zusatzwerte nicht ändern. Außer in den Levelregeln sollen diese Zusatzwerte nie geändert werden und durchgehend ihre Werte beibehalten.

Globale Zusatzwerte gibt es:

- `GEMS`: Anzahl der eingesammelten Edelsteine
- `TICKS`: Anzahl der seit Spielbeginn vergangenen Ticks
- `X`, `Y`, `Z`: Diese globalen Zusatzwerte entsprechen den `A`...`D`, bis auf dass sie globale Werte sind. Sie sollen also auch nie durch die Hauptregeln geändert werden, sondern nur durch die Levelregeln und durchgehend ihre Werte beibehalten.

4.4 Hauptregeln

Die Hauptregeln implementieren die Grundmechanik des Spiels. Diese werden in jedem Tick genau einmal ausgeführt. Jeder Gegenstand soll sich durch die Hauptregeln pro Tick höchstens einmal bewegen.

Im Folgenden ist zu beachten, dass vor den Hauptregeln bereits die `pre` ausgeführt werden. Durch Ausführung der `pre` könnten Zusatzwerte gesetzt werden. Zum Beispiel könnte es ein Level geben, in dem durch diese Regeln auf allen Feldern mit dem Gegenstand `POT` der

Zusatzwert `PUSHABLE` gesetzt werden. Allgemein gilt: Alle Regeln müssen die Bedeutung der Zusatzwerte respektieren. Die Hauptregeln können sich darauf verlassen, dass die `pre` die Zusatzwerte entsprechend ihrer Bedeutung setzen. Die Hauptregeln müssen aber ihrerseits für die `post` sicherstellen, dass sie die Zusatzwerte korrekt setzen.

Spielerbewegung. Zur Berechnung des Spielzustands ist es relevant, ob der Spieler zeitgleich Pfeil- und Shift-Tasten gedrückt hat oder nicht. Wurde keine Pfeiltaste gedrückt, so verbleibt die Spielfigur an Ort und Stelle.

Wurde eine Pfeiltaste gedrückt, aber ohne die Shift-Taste, so bewegt sich die Spielfigur (`ME`) in die entsprechende Richtung, falls es dort ein freies Feld (`PATH`) durchgrabbares Erdreich (`MUD`), oder einen Edelstein (`GEM`) gibt. Dort, wo die Spielfigur stand, verbleibt ein freies Feld (`PATH`). Falls auf dem Feld ein Edelstein war, so erhöht sich auch der Zähler der eingesammelten Edelsteine um 1.

Achten Sie dabei darauf, dass sich die Spielfigur gleichzeitig mit allen anderen Gegenständen am Ende des Ticks bewegt. Insbesondere darf sie sich nur um 1 Feld weit in jedem Tick bewegen, unabhängig davon, wie oft die entsprechede Pfeiltaste gedrückt wurde.

Beispiel, rechte Pfeiltaste gedrückt:

PATH	PATH	MUD	MUD
MUD	ME	MUD	MUD
MUD	PATH	MUD	MUD

wird zu

PATH	PATH	MUD	MUD
MUD	PATH*	ME*	MUD
MUD	PATH	MUD	MUD

Wurde gleichzeitig die Shift-Taste gedrückt, so wird nur das entsprechende Feld frei gegraben (falls möglich), aber nicht die Spielfigur bewegt. Ein Edelstein wird aber nur eingesammelt, wenn er nicht gleichzeitig am herunterfallen war; ist also das `FALLING`-Zusatzwert gesetzt, so wird er nicht eingesammelt. Das gleiche Beispiel, jetzt aber mit rechter Pfeiltaste und Shift-Taste gedrückt:

PATH	PATH	MUD	MUD
MUD	ME	MUD	MUD
MUD	PATH	MUD	MUD

wird zu

PATH	PATH	MUD	MUD
MUD	ME	PATH	MUD
MUD	PATH	MUD	MUD

Bei Druck der rechten oder linken Pfeiltaste kann ein vor der Spielfigur liegender Gegenstand verschoben werden, sofern der Zusatzwert `PUSHABLE` im Feld des Gegenstands gesetzt ist. Ohne Levelregeln kann nur der Gegenstand `STONE` verschoben werden, darum wird dieser in den Beispielen verwendet. Die Spielfigur kann immer nur einen einzigen Gegenstand verschieben und auch nur in ein Feld mit freiem Platz (`PATH`). Verschieben nach oben oder unten ist nicht möglich. Zudem ist diese Bewegung nicht mit der Shift-Taste kombinierbar, es ist also nicht möglich den Stein zu verschieben, ohne sich selbst zu bewegen.

Beispiel, rechte Pfeiltaste gedrückt:

PATH	PATH	PATH	PATH	MUD
MUD	ME	STONE	PATH	MUD
MUD	MUD	MUD	MUD	MUD

wird zu

PATH	PATH	PATH	PATH	MUD
MUD	PATH*	ME*	STONE*	MUD
MUD	MUD	MUD	MUD	MUD

Dieses Beispiel verändert sich bei gedrückter rechter Pfeiltaste nicht mehr weiter.

Bei den mit `*` markierten Feldern ist zusätzlich der Zusatzwert `MOVED` gesetzt. Dieses verschwindet nach einem weiteren Tick aber wieder.

Gravitation. Gegenstände, bei denen der `LOOSE`-Zusatzwert gesetzt ist, können herunterfallen. Dies sind, sofern keine Levelregeln aktiv sind, nur die Gegenstände `STONE` und `GEM`. In den Beispielen wird immer der Gegenstand `STONE` verwendet.

Wenn Platz unter ihnen frei ist (=`PATH`), so fallen sie nach unten, in jedem Tick genau ein Feld weit. Ist der Platz unter ihnen zwar belegt, aber von einem Gegenstand, bei dem auch der Zusatzwert `SLIPPERY` gesetzt ist, so können sie nach links unten oder rechts unten fallen. Dies ist normalerweise bei einem Stein, Edelstein oder einer Ziegelsteinwand der Fall. Um nach rechts unten zu fallen, muss sowohl der Platz direkt rechts, wie auch rechts unten frei sein. Entsprechendes gilt, um nach links unten zu fallen. Nach rechts unten zu fallen hat dabei allerdings Vorrang vor nach links unten zu fallen.

Ferner darf im gleichen Tick kein Stein oder Edelstein ein erst in diesem Tick freigewordenen Platz belegen.

Beispiele (unabhängig von den gedrückten Tasten):

PATH	PATH	STONE	PATH	MUD		PATH	PATH	PATH*	PATH	MUD
MUD	PATH	PATH	PATH	MUD	wird zu	MUD	PATH	STONE+	PATH	MUD
MUD	MUD	MUD	MUD	MUD		MUD	MUD	MUD	MUD	MUD

PATH	PATH	STONE	PATH	MUD		PATH	PATH	PATH*	PATH	MUD
MUD	PATH	STONE	PATH	MUD	wird zu	MUD	PATH	STONE	STONE+	MUD
MUD	MUD	MUD	MUD	MUD		MUD	MUD	MUD	MUD	MUD

PATH	PATH	STONE	MUD	MUD		PATH	PATH	PATH*	MUD	MUD
MUD	PATH	STONE	PATH	MUD	wird zu	MUD	STONE+	STONE	PATH	MUD
MUD	MUD	MUD	MUD	MUD		MUD	MUD	MUD	MUD	MUD

MUD	MUD	STONE	MUD	MUD		MUD	MUD	STONE	MUD	MUD
MUD	MUD	STONE	MUD	MUD	wird zu	MUD	MUD	PATH*	MUD	MUD
MUD	MUD	PATH	MUD	MUD		MUD	MUD	STONE+	MUD	MUD

Bei den mit `*` markierten Feldern ist zusätzlich der Zusatzwert `MOVED` gesetzt. Dieser verschwindet nach einem weiteren Tick aber wieder.

Bei den mit `+` markierten Feldern sind zusätzlich die Zusatzwerte `MOVED` und `FALLING` gesetzt. Diese verschwinden nach einem weiteren Tick aber wieder.

Die Spielfigur und auch jeder Gegner kann von einem herabfallenden Gegenstand erschlagen werden. Das passiert, wenn ein Gegenstand in das Feld über der Spielfigur fällt. Unter einem Gegenstand zu stehen ist jedoch kein Problem, auch wenn dieser sonst fallen könnte. Ein Gegenstand über der Figur könnte sogar von einem Gegner seitlich verschoben worden sein. Um diese Situationen zu unterscheiden kann man der Zusatzwert `FALLING` verwenden.

Wird die Spielfigur (`ME`), ein `SWAPLING` oder ein `XLING` erschlagen, so wandelt er sich in ein 3×3-Feld aus Edelsteinen um. Wird ein `BLOCKLING` erschlagen, so wandelt er sich in ein 3×3-Feld aus `EXPLOSION` um.

Gegnerbewegungen. Es gibt drei Arten von Gegnern:

- `SWAPLING`: Ein einfacher Gegner, der sich wahlweise immer horizontal (von links nach rechts und zurück), oder vertikal (von oben nach unten und zurück) bewegt. Der `SWAPLING` ändert seine Richtung nur, wenn er nicht mehr weiter gehen kann.

- `XLING`: Ein Gegner, der sich immer gemäß Rechter-Hand-Regel an der Wand entlang bewegt. Man stellt sich vor, dass der Gegner mit der rechten Hand eine Wand berührt.

Dann bewegt sich der Gegner stets so an der Wand entlang, dass er die rechten Hand niemals von der Wand nehmen muss.

- `BLOCKLING`: Ein Gegner, der sich immer gemäß Linker-Hand-Regel an der Wand entlang bewegt.

Liegt auf dem Weg des Gegners die Spielfigur, so wird dieser ebenfalls in ein 3×3-Feld aus Edelsteinen oder Explosionen verwandelt, so wie wenn ihm ein Stein auf den Kopf gefallen wäre.

Das kann umgesetzt werden, indem hier der Zusatzwert `BAMRICH` gesetzt wird.

Schleim. Schleim verbreitet sich auf der Karte und interagiert mit einigen Objekten. Dabei wird auf den (horizontal und vertikal, nicht diagonal) benachbarten Feldern eines Schleimfeld in jedem Tick mit einer Wahrscheinlichkeit von 3% in Abhängigkeit des Feldes folgendes ausgeführt

- `PATH`, `MUD`: Aus diesem Feld wird ein Schleimfeld
- `SWAPLING`, `XLING`: Um dieses Feld entsteht ein 3x3-Feld aus Edelsteinen, so wie wenn dem Gegner ein Stein auf den Kopf gefallen wäre
- `BLOCKLING`: Um dieses Feld entsteht ein 3x3-Feld aus Explosionen, so wie wenn dem Gegner ein Stein auf den Kopf gefallen wäre
- Alle anderen Felder bleiben unverändert

Ist ein zusammenhängender Bereich (horizontal und vertikal) von Schleim komplett von Feldern, die unverändert bleiben, umgeben, so verwandelt sich dieser komplette Bereich in `GEM`.

Ist allerdings ein Wert für `maxslime` angegeben und es gibt auf der Karte mehr Schleim, als der Wert `maxslime` beträgt, so verwandeln sich alle Schleimfelder in `STONE`.

Explosionen. Alle Explosionen vom letzten Tick werden gelöscht, indem `EXPLOSION` überall durch `PATH` ersetzt wird. Das heißt, eine Explosion dauert maximal einen Tick.

Ist auf einem Feld der Zusatzwert `BAM` gesetzt (das kann zum Beispiel durch die `pre` passieren), so wird die 3×3-Umgebung um die Explosion auf `EXPLOSION` gesetzt. Ausgenommen sind dabei die Felder, die `WALL` oder `EXIT` enthalten. Diese bleiben wie sie sind. Ist dagegen `BAMRICH` gesetzt, so passiert da Gleiche wie bei `BAM`, nur mit `GEM` statt `EXPLOSION`.

4.5 Levelregeln

In einem Level kann die Spielmechanik durch Regeln verändert werden. Dies betrifft insbesondere diejenigen Gegenstände, die noch keine Verhaltensweisen in den Grundregeln zugeordnet bekommen haben, kann aber alle Arten von Gegenständen betreffen.

Dazu werden in jedem Tick sogenannte Regeln ausgeführt. Jede Regel beschreibt eine einfache Änderung des Spielzustands. Durch Hintereinanderausführung verschiedener Regeln kann eine beliebig komplizierte zusätzliche Spielmechanik entstehen.

Bevor wir zu den Regeln kommen, beschreiben wir erst einmal das Konzept von Regelbausteinen. Regelbausteine werden verwendet, um aus ihnen Regeln aufzubauen und sie sind einfacher als komplette Regeln.

Ein Regelbaustein ist als JSON-Objekt gegeben. Dieses ist von der Form `{"token": ..., "values": ... }`.

- Der Wert des Attributs `"token"` ist entweder ein String, ein JSON-Array von Strings, oder eine ganze Zahl.
- Der Wert des Attributs `"values"` ist ein JSON-Objekt, welches als Attribute Zusatzwerte enthält und als Werte jeweils eine ganze Zahl enthält. Beispiel: `{ "slippery": 1, "falling": 0}`.

Man spricht davon, dass ein Regelbaustein auf ein Feld passt, wenn die beiden folgenden Bedingungen erfüllt sind.

- Der Name des Gegenstands auf dem Feld ist in der Liste der Namen des Regelbausteins enthalten oder der Regelbaustein enthält den Namen „`*`“ (passt immer).

  Regelbausteine mit Zahlen (anstatt Namen) können nie auf ein Feld passen.
- Für jeden Zusatzwert, der in dem Regelbaustein vorkommt, gilt einer der folgenden Fälle
  - der Zusatzwert ist im Regelbaustein und auf dem Feld 0
  - der Zusatzwert ist auf dem Feld größer oder gleich dem Zusatzwert im Regelbaustein, ferner ist der Zusatzwert im Regelbaustein größer als 0

Ein Feld durch einen Regelbaustein zu ändern, bedeutet:

- Enthält der Regelbaustein zum Attribut `"token"` einen String, so wird das Token auf dem Feld durch den String des Regelbausteins ersetzt; ein Array vom Strings in einem Regelbaustein, welches ein Feld ändern soll, sind nicht zulässig.
- Enthält der Regelbaustein zum Attribut `"token"` eine Zahl k, so wird in dem Feld der Wert des Tokens durch den Wert des Tokens eines anderen Feldes ersetzt, und zwar durch das k-te gefundene Feld (siehe weiter unten)
- Bei allen Zusatzwerten mit dem Wert 0 wird der Wert auf dem Feld auf 0 gesetzt. Ist ihr Wert allerdings ungleich 0, so wird ihr Wert zum entsprechenden Zusatzwert auf dem Feld addiert. Ist das Ergebnis allerdings kleiner als 0, so wird dieser Zusatzwert auf dem Feld ebenfalls auf 0 gesetzt.

  Beispiele:
  - Feld alt: 5; Regelbaustein: 4; Feld neu: 9
  - Feld alt: 5; Regelbaustein: -4; Feld neu: 1
  - Feld alt: 4; Regelbaustein: -5; Feld neu: 0
  - Feld alt: 7; Regelbaustein: 0; Feld neu: 0

Eine Regel enthält zwei Listen von Regelbausteinen (`original` und `result`). Um die Regel anzuwenden, wird eine Abfolge von Feldern gesucht, auf welche die Regelbausteine in der ersten Liste passen. Die Regel spezifiziert dabei, ob in Reihen oder Spalten gesucht wird (`direction`). Wird eine passende Folge von Feldern gefunden, so werden die Felder entsprechend der Regelbausteine in der zweiten Liste geändert. Dieses Anwenden der Regel

wird so lange wiederholt, bis die ganze Karte abgearbeitet ist. Für jede Regel ist angegeben, ob dieses Abarbeiten der Karte in jedem Tick oder seltener passieren soll (genannt `situation`).

Eine Regel ist durch ein JSON-Objekt folgender Form gegeben:

```
{ "situation": "...",
  "direction": "...",
  "original": [...],
  "result": [...]
}
```

Die Bestandteile einer Regel haben folgende Bedeutung:

- `situation`: In welcher Situation soll diese Regel ausgeführt werden? Mögliche Situationen sind
  - `any`: Diese Regel soll in jedem Tick ausgeführt werden.
  - `rare`: Im JSON-Objekt gibt es noch ein zusätzliches Attribut `sparsity`, welcher als Wert eine Zahl hat. Diese Zahl gibt an, alle wie viele Runden diese Regel ausgeführt werden soll.
  - `left`, `right`, `up`, `down`: Diese Regel soll nur dann ausgeführt werden, wenn durch das Eingabegerät angegeben wird, dass sich die Spielfigur in die entsprechende Richtung bewegen soll.

    Dies wird in der Implementierung typischerweise eine Pfeiltaste sein, ist aber nicht darauf festgelegt.
  - `metaleft`, `metaright`, `metaup`, `metadown`: Diese Regel soll nur dann ausgeführt werden, wenn durch das Eingabegerät angegeben wird, dass die Spielfigur das Feld in der entsprechenden Richtung freiräumen soll.

    Dies wird in der Implementierung typischerweise eine Pfeiltaste bei gedrückter Shift-Taste sein, ist aber ebenfalls nicht darauf festgelegt.
- `direction`: Der hier stehende Wert gibt an, in welche Richtung die Regel ausgeführt werden soll. Dabei bedeutet
  - `east`: Es soll eine Reihe von links nach rechts abgearbeitet werden.
  - `west`: Eine Reihe wird von rechts nach links abgearbeitet.
  - `north`, `south`: Eine Spalte wird von unten nach oben, oder von oben nach unten abgearbeitet.
- `original`: Eine Auflistung an Regelbausteinen, welche gefunden werden müssen, damit diese Regel aktiv wird.
- `result`: Eine Auflistung an Regelbausteinen, welche stattdessen in die Leveldaten geschrieben werden, wenn diese Regel ein passendes Teilstück gefunden hat.

Für die Ausführung der Regel wird das Spielfeld in Linien aufgeteilt. Eine Linie steht hierbei für eine Zeile oder Spalte. Jede Linie hat einen Anfangs- und Endpunkt. `direction` gibt die Richtung an, in die das Spielfeld aufgeteilt werden soll für diese Regel. Dabei wird jede Linie, die erzeugt wird, so lange wie möglich und das Spielfeld vollständig aufgeteilt.

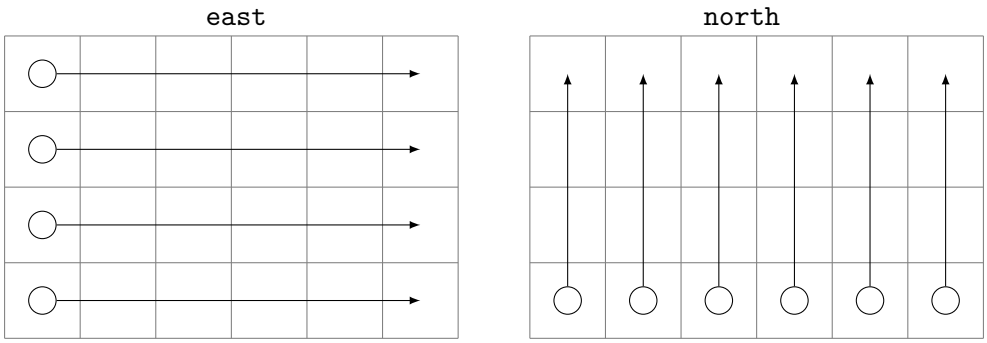

Abbildung 1: Unterteilung des Spielfelds in Linien, je nach Durchlaufrichtung der Regel

Abbildung 1 zeigt ein Spielfeld und seine Aufteilung in Linien unter verschiedenen Richtungen. Hierbei steht der Kreis immer für den Anfang der Linie und die Pfeilspitze für das Ende der Linie.

Für die Regel wird jede Linie für sich abgearbeitet. Die Reihenfolge, in der die Linien abgearbeitet werden, ist darum auch nicht von Bedeutung.

Erst wenn eine Regel fertig abgearbeitet ist, also auf alle Linien angewandt wurde, wird die nächste Regel abgearbeitet. Die Abarbeitung aller Regeln geschieht in jedem Tick, entsprechend der Reihenfolge, die in Abschnitt4.2 angegeben ist.

Für die Regel wird auf jeder Linie von Anfang an gesucht, ob es eine Abfolge von Feldern gibt, so dass die Regelbausteine aus `original` der Reihe nach auf die Felder passen. Gibt es so ein Vorkommen, so wird das erste Vorkommen genommen und die Felder durch die Regelbausteine aus `result` geändert. Danach wird ab dem Ende der gefundenen Stelle weitergemacht und erneut das nächste Vorkommen gesucht. Insbesondere wird selbst dann erst hinter dem Ende der gefundenen Stelle weitergemacht, wenn die Regel auf eine überlappende Stelle auch passen würde.

Ausführungsbeispiel: Eine Linie. In diesen Beispielen wird jeweils eine Regel auf einer einzelnen Linie ausgeführt. Dafür wird die Anwendung in Einzelteile zerlegt, damit man besser erkennen kann, wie die Ausführung der Regel zustande kommt.

Um den Inhalt im Text besser ansprechen zu können, sind die Felder in diesem Beispiel durchnummeriert.

In diesem Beispiel hat die Linie folgenden Inhalt.

0	1	2	3	4	5	6	7	8	9	10
PATH	STONE	STONE	PATH	PATH	STONE	STONE	PATH	STONE	PATH	STONE

Führt man nun folgende Regel aus (`east` hat hier keine Auswirkung, da es schon vorab verwendet wurde, um überhaupt auf diese Linie zu kommen)

```
{ "situation":"any",
  "direction":"east",
  "original":[
    {"token":"stone"},
    {"token":"stone"},
```

```
    {"token":"path"}
  ],
  "result":[
    {"token":"gem"},
    {"token":"path"},
    {"token":"stone"},
  }]
}
```

so passt die Regel nicht auf die Position ganz am Anfang (Position 0), da hier `PATH` steht und nicht `STONE`. Allerdings passt sie auf Position 1, darum wird sie hier ausgeführt. Dies führt zu folgendem Zustand.

0	1	2	3	4	5	6	7	8	9	10
PATH	GEM	PATH	STONE	PATH	STONE	STONE	PATH	STONE	PATH	STONE

Da die Ausführung an Position 1 insgesamt an drei Positionen potentielle Auswirkungen hatte (Position 1, 2, 3), so wird als nächstes an Position 4 getestet, ob die Regel passt. Da sie hier nicht passt, wird nun Position 5 getestet, wo die Regel erneut zur Anwendung kommt.

0	1	2	3	4	5	6	7	8	9	10
PATH	GEM	PATH	STONE	PATH	GEM	PATH	STONE	STONE	PATH	STONE

Die Regel hatte wieder auf drei Positionen potentielle Auswirkungen (5,6,7), darum wird als nächstes an Position 8 getestet, ob man die Regel hier anwenden kann. Dabei ist es unerheblich, dass die Regel jetzt an Position 7 passen würde, da Regeln ihre eigenen Auswirkungen aus dieser Ausführung nicht weiter beachten. Da die Regel auf Position 8 nicht passt, ist sie jetzt fertig, da sie an Position 9 und 10 nicht passen kann, da hier zu wenige Felder in der Linie übrig sind, als dass die Regel noch einmal passen könnte.

In einem weiteren Ausführungsbeispiel ist der Startinhalt der Linie nun der folgende:

0	1	2	3	4	5	6	7	8
STONE	STONE	STONE	STONE	STONE	GEM	STONE	STONE	STONE

Nun wird folgende Regel ausgeführt

```
{ "situation":"any",
  "direction":"northeast",
  "original":[
    {"token":"stone"},
    {"token":["stone","gem"]},
    {"token":"stone"}
  ],
  "result":[
    {"token":"gem"},
    {"token":"path"},
    {"token":"1"},
  }]
}
```

Diese Regel passt auf die Position 0, da hier dreimal `STONE` hintereinander vorkommt. Die durch die Regel verursachten Änderungen erzeugen `GEM`, `PATH`, `STONE`, da die Position 1 der Stelle an der die Regel passt auch Position 1 der Linie ist und hier `STONE` steht. Nun sieht die Linie also wie folgt aus:

0	1	2	3	4	5	6	7	8
GEM	PATH	STONE	STONE	STONE	GEM	STONE	STONE	STONE

Die Ausführung der Regel geht an Position 3 weiter, da es an Position 0, 1 und 2 eine potentielle Änderung gab. Das gilt auch dann, wenn sich der Inhalt an Position 2 gar nicht geändert hatte, was hier der Fall ist.

An Position 3 passt die Regel nicht, da `GEM` an Position 5 dem entgegensteht.

An Position 4 passt die Regel wieder, allerdings ist diesmal die Position 1 des Treffers die Position 5 der Linie. Also werden die Inhalte auf `GEM`, `PATH`, `GEM` geändert, die Linie hat danach folgenden Inhalt:

0	1	2	3	4	5	6	7	8
GEM	PATH	STONE	STONE	GEM	PATH	GEM	STONE	STONE

Danach würde es an Position 7 weitergehen, hier ist aber die Linie bereits zu kurz, als dass die Regel noch hineinpassen würde, die Ausführung der Regel ist also zuende.

Ausführungsbeispiel: Vollständige Karte. Dieser Abschnitt enthält ein Beispiel, wie die Felddaten aussehen können und wie sich die Levelregeln darauf auswirken.

Für dieses Beispiel steht in jedem Feld der Name des Gegenstands auf diesem Feld. Sind auf dem Feld Zusatzwerte wahr, so werden alle wahren Zusatzwerte hinter dem Gegenstandsnamen in eckigen Klammern aufgeführt.

Dieses Beispiel spielt auf einer Karte der Höhe 3 und Breite 4.

Zu Beginn dieses Beispiels sieht die Karte wie folgt aus:

PATH	STONE	PATH	PATH
PATH	ME	STONE	PATH
PATH	MUD	MUD	MUD

Wird nun die Regel

```
{ "situation":"down",
  "direction":"south",
  "original":[
    {"token":"me"},
    {"token":["mud", "path"]}
  ],
  "result":[
    {"token":"path", "values":{"moved":1}},
    {"token":"me", "values":{"moved":1}
  }]
}
```

ausgeführt, so ändern sich die Werte zu

PATH	STONE	PATH	PATH
PATH	PATH[MOVED:1]	STONE	PATH
PATH	ME[MOVED:1]	MUD	MUD

,

denn es gibt genau eine Stelle, bei der, von oben nach unten betrachtet, hinter ME direkt ein PATH oder MUD kommt. Diese Stelle wird dann entsprechend den Angaben in der Regel ersetzt.

Wird nun die Regel

```
{ "situation":"any",
  "direction":"east",
  "original":[{"token":"*"}],
  "result":[{"token":"0", "values":{"moved":0}}]
}
```

ausgeführt, so ändern sich die Werte zu

PATH	STONE	PATH	PATH
PATH	PATH	STONE	PATH
PATH	ME	MUD	MUD

,

denn * gilt an jedem Feld. Ferner ist diese Regel genau 1 Feld lang, darum würde unabhängig von der Richtung immer genau das gleiche herauskommen. Die Auswirkung von der Regel ist immer das Feld 0, also das Feld selbst wieder, hinzuschreiben, dabei aber der Zusatzwert MOVED zu entfernen.

Wird nun die Regel

```
{ "situation":"any",
  "direction":"west",
  "original":[{"token":"stone"}, {"token":"*"}],
  "result":[{"token":"1"}, {"token":"0"}]
}
```

ausgeführt, so ändern sich die Werte zu

STONE	PATH	PATH	PATH
PATH	STONE	PATH	PATH
PATH	ME	MUD	MUD

.