Vorstellung Hauptprojekt

Boulder Dash

Das Hauptprojekt des Praktikums 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.
- Die Boulder Dash Implementierung soll keine direkte Kopie einer existierenden Variante sein.

Es gibt eine ausführliche Beschreibung des Projekts auf der Vorlesungsseite. Dort wird auf Details eingegangen. Diese Folien sollen einen Überblick verschaffen und mit dem Projekt vertraut machen.

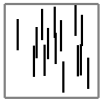
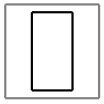
Spielprinzip

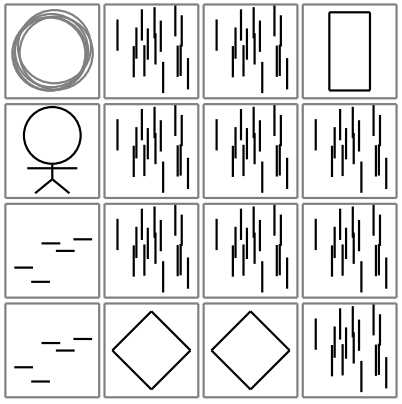
In Boulder Dash steuert man einen Abenteurer, der Edelsteine einsammeln und mit diesen zu einem Ausgang kommen will

- Das Spiel ist in Level unterteilt
- Es stehe unterschiedlich viele Level zur Auswahl, je nachdem
  - wie viele Level man schon bestanden hat,
  - wie gut man diese bestanden hat
- Jedes Level besteht hauptsächlich aus einer Karte, auf der man
  - Sich durch Erdreich graben kann
  - Edelsteine einsammeln kann
  - Gegnern und herunterfallenden Gegenständen ausweichen muss
  - Das Ziel finden muss

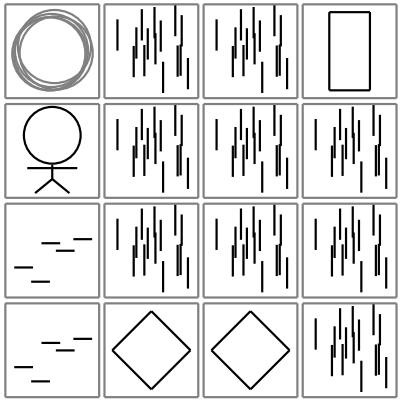
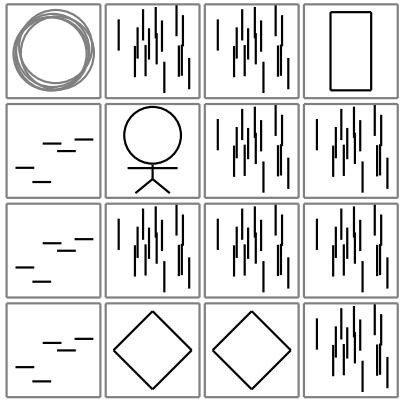
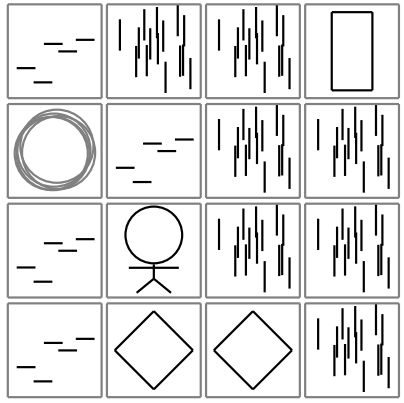
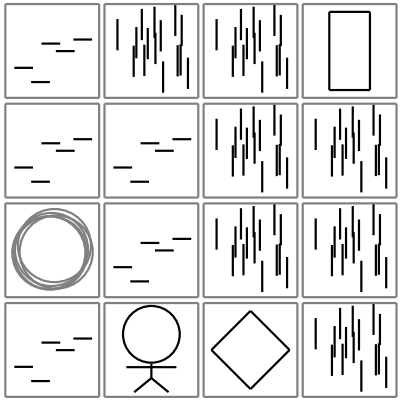
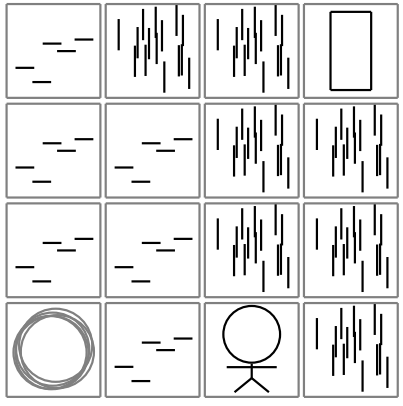
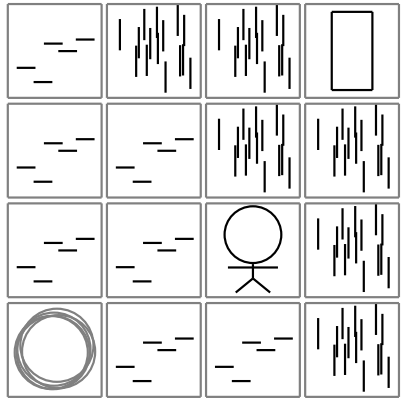
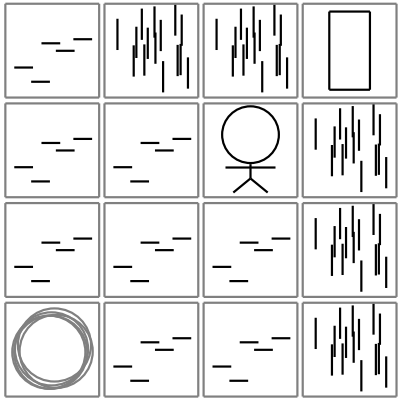
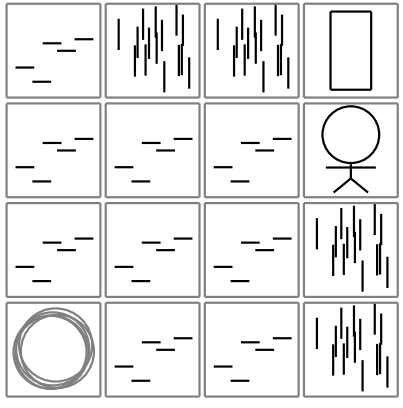
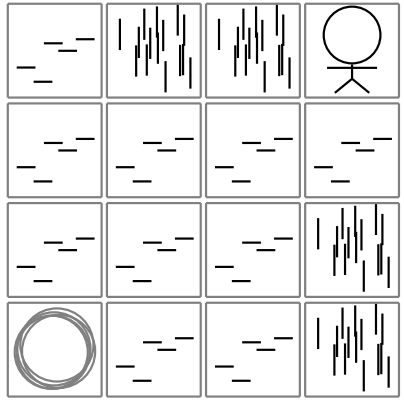
Beispiellevel

In diesem Level sind folgende Objekte zu sehen

-  die Spielfigur
-  Erdreich
-  freier Weg
-  ein Edelstein
-  ein Stein
-  der zu erreichende Ausgang



Beispiellevel: Möglicher Spielablauf

Gespeicherte Leveldaten

Das Spielprinzip lebt davon verschiedene Level zur Auswahl zu haben

- Level sind als Dateien gegeben: JSON-Format
- Jedes Level besteht aus
  - Karte (Höhe, Breite, Inhalt)
  - Gewinnbedingungen: Edelsteine zu sammeln, Zeit um das Level zu lösen
    3 unterschiedlich anfordernde Gewinnbedingungen
  - Extraregeln für das Level
    Können das Verhalten des Spiels zum Teil deutlich beeinflussen

Simulationsprinzip im Level – Zelluläre Automaten

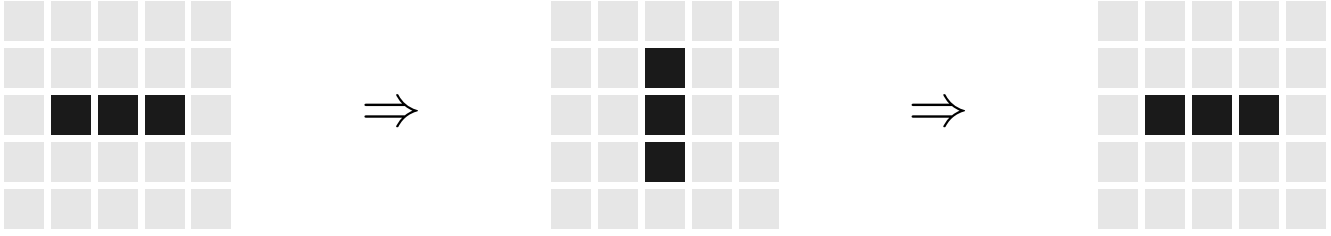
Die Simulation des Spiels basiert auf zellulären Automaten. Zelluläre Automaten dienen der Modellierung dynamischer Systeme.

- Ein zellulärer Automat agiert auf einem regelmäßigem Gitter von Objekten.
- Eine Belegung dieses Gitters wird Generation genannt.
- Der Automat wandelt Belegungen des Gitters in neue Belegungen um, berechnet neue Generationen.
- Dabei wird für jede Gitterposition nur die direkte Umgebung verwendet, um neue Werte zu berechnen.

Game of Life

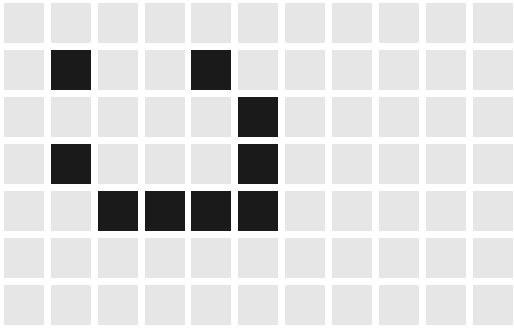
Ein berühmtes Beispiel für einen einfachen zellulären Automaten ist Conways Game of Life (Spiel des Lebens, 1970).

- Es gibt nur einen boolschen Wert auf jeder Gitterposition (Zelle): Tot/Lebendig.
- Beim Übergang von einer Generation zur nächsten
  - bleibt eine lebendige Zelle lebendig, wenn sie 2 oder 3 Nachbarn (in der 8-Nachbarschaft, also inklusive Diagonale) hat, sonst stirbt sie
  - wird eine tote Zelle lebendig, wenn sich 3 Nachbarn hat, sonst bleibt sie tot



Game of Life – Einfache Regeln, komplexes Verhalten

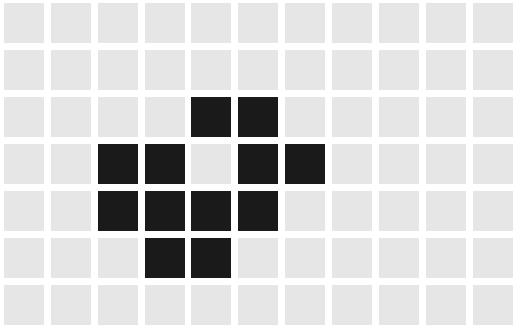
Obwohl die Regeln von Game of Life sehr einfach sind, gibt es interessante, komplexe Muster



$\Rightarrow$ Zelluläre Automaten sind geeignet, um ein komplexes Spiel mit einfachen Regeln zu modellieren.

Game of Life – Einfache Regeln, komplexes Verhalten

Obwohl die Regeln von Game of Life sehr einfach sind, gibt es interessante, komplexe Muster



$\Rightarrow$ Zelluläre Automaten sind geeignet, um ein komplexes Spiel mit einfachen Regeln zu modellieren.

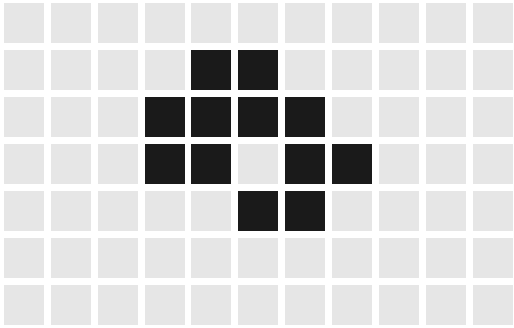
Game of Life – Einfache Regeln, komplexes Verhalten

Obwohl die Regeln von Game of Life sehr einfach sind, gibt es interessante, komplexe Muster

$\Rightarrow$ Zelluläre Automaten sind geeignet, um ein komplexes Spiel mit einfachen Regeln zu modellieren.

Game of Life – Einfache Regeln, komplexes Verhalten

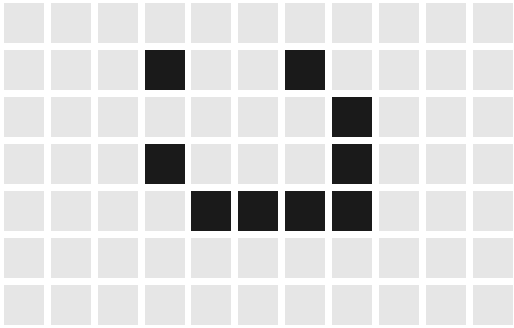
Obwohl die Regeln von Game of Life sehr einfach sind, gibt es interessante, komplexe Muster



$\Rightarrow$ Zelluläre Automaten sind geeignet, um ein komplexes Spiel mit einfachen Regeln zu modellieren.

Game of Life – Einfache Regeln, komplexes Verhalten

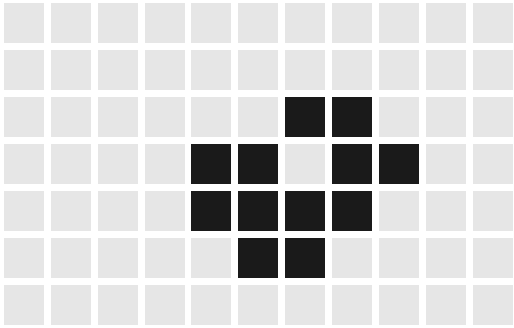
Obwohl die Regeln von Game of Life sehr einfach sind, gibt es interessante, komplexe Muster



$\Rightarrow$ Zelluläre Automaten sind geeignet, um ein komplexes Spiel mit einfachen Regeln zu modellieren.

Game of Life – Einfache Regeln, komplexes Verhalten

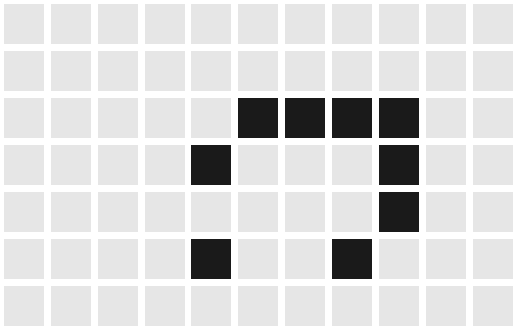
Obwohl die Regeln von Game of Life sehr einfach sind, gibt es interessante, komplexe Muster



$\Rightarrow$ Zelluläre Automaten sind geeignet, um ein komplexes Spiel mit einfachen Regeln zu modellieren.

Game of Life – Einfache Regeln, komplexes Verhalten

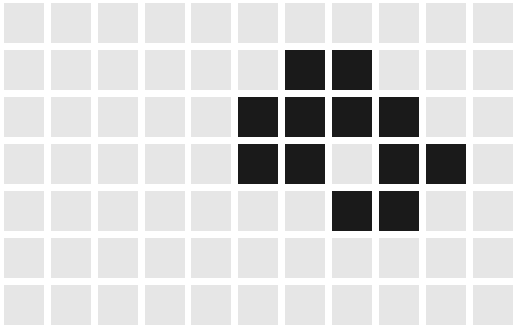
Obwohl die Regeln von Game of Life sehr einfach sind, gibt es interessante, komplexe Muster



$\Rightarrow$ Zelluläre Automaten sind geeignet, um ein komplexes Spiel mit einfachen Regeln zu modellieren.

Game of Life – Einfache Regeln, komplexes Verhalten

Obwohl die Regeln von Game of Life sehr einfach sind, gibt es interessante, komplexe Muster



$\Rightarrow$ Zelluläre Automaten sind geeignet, um ein komplexes Spiel mit einfachen Regeln zu modellieren.

Begrifflichkeiten

An verschiedenen Stellen in der Literatur haben die einzelnen Komponenten von zellulären Automaten verschiedene Namen. Eine Übersicht über die häufigsten

- Gleichbedeutend sind: Feld, Kästchen, Zelle, Tile
- Gleichbedeutend sind: Runde, Generation, Tick

Zudem gibt es zelluläre Automaten, die auf einer anderen Struktur als einem rechtwinkligen 2-D-Gitter basieren (damit beschäftigen wir uns aber nicht weiter)

Komplexere Zelluläre Automaten

Für viele Anwendungen eignen sich komplexere zelluläre Automaten, die auf jedem Feld mehr Daten haben, als nur einen boolschen Wert.

- Zelluläre Automaten in der Wettervorhersage: Temperatur, Windgeschwindigkeit, Luftfeuchtigkeit, . . . auf jedem Feld
- Zelluläre Automaten in Computerspielen, Beispiel Stadtsimulation: Verkehr, Verschmutzung, Grundstückswert, . . . auf jedem Feld

Regeln für neue Feldinhalte in der nächsten Runde brauchen mehr, als nur Nachbarn zählen.

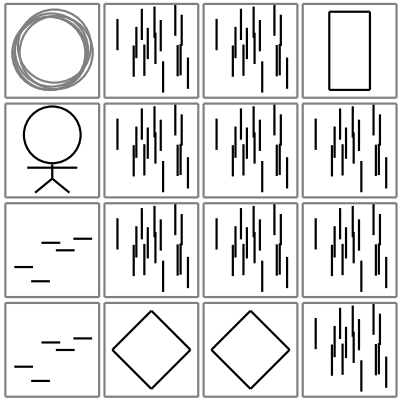
Vorteil von zellulären Automaten: Einfacher als andere Implementierungen, da nur Felder in der direkten Umgebung berücksichtigt werden müssen.

Datenmodell Boulder Dash

Boulder Dash kann auch als zellulärer Automat umgesetzt werden

- Jedes Feld beinhaltet einen String, sowie eine Reihe von benannten int-Werten, sogenannte Zusatzwerte
- Die string-Werte stellen den Inhalt der Karte dar
- Die Flags stehen für nicht darzustellende Zusatzinformationen, beispielsweise Verhaltensinformation der Gegner

Datenmodell, Beispiel



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

Regeln

Boulder Dash-Regeln sind komplexer, als die von Game of Life

Zur Implementierung von zellulären Automaten gibt es zwei prinzipielle Ansätze, verwendet werden meist Mischformen

- Alle Regeln werden direkt als Code implementiert
  - Hohe Ausführgeschwindigkeit
  - Wenig flexibel gegenüber nachträglichen Regeländerungen
  - Typischerweise in alten Spielen und sehr aufwendigen Simulationen verwendet
- Es wird ein System implementiert, in dem man die Regeln in einem anderen Format angeben kann
  - Etwas niedrigere Ausführgeschwindigkeit
  - Code hat weniger mit den konkret verwendeten Regeln zu tun

Boulder Dash lässt sich gut als Mischform implementieren

- Grundlegendes Spiel direkt in Code
- Extra-Regeln für einzelne Level im JSON-Format gegeben

Regeln, Grundablauf

Der Spielzustand ist in sogenannte Ticks unterteile: fünf Ticks pro Sekunde

Ein Tick läuft folgendermaßen ab:

- Zurücksetzen der Zusatzwerte aller Felder entsprechend ihrer Bedeutung
- Die Levelregeln `pre` werden ausgeführt
- Der Levelzustand wird entsprechend der Hauptregeln verändert
- Die Levelregeln `post` werden ausgeführt

Hauptregeln

Die Hauptregeln drücken das eigentliche Spiel aus:

- Die Spielfigur kann sich bewegen
- Gegner bewegen sich
- Verschiedene Gegenstände fallen nach unten
- Herunterfallende Gegenstände können Gegner sowie die Spielfigur erschlagen
- Schleim breitet sich aus
- Edelsteine können eingesammelt werden
- Hat man genügend Edelsteine gesammelt, so kann man ein Level durch einen Ausgang verlassen

Levelregeln, Format

Eine Levelregel ist ein JSON-Objekt der folgenden Form

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

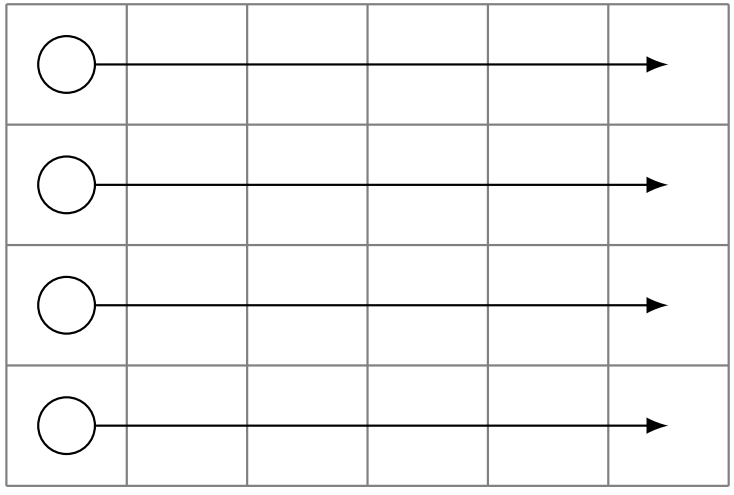
wobei

- `situation`: Wann soll diese Regel ausgeführt werden (jede Runde, wenn die Spielfigur angewiesen wird nach rechts zu gehen, ...)
- `direction`: In welche Richtung wird diese Regel ausgewertet
- `original`: Nach was wird gesucht
- `result`: Was wird stattdessen eingefügt

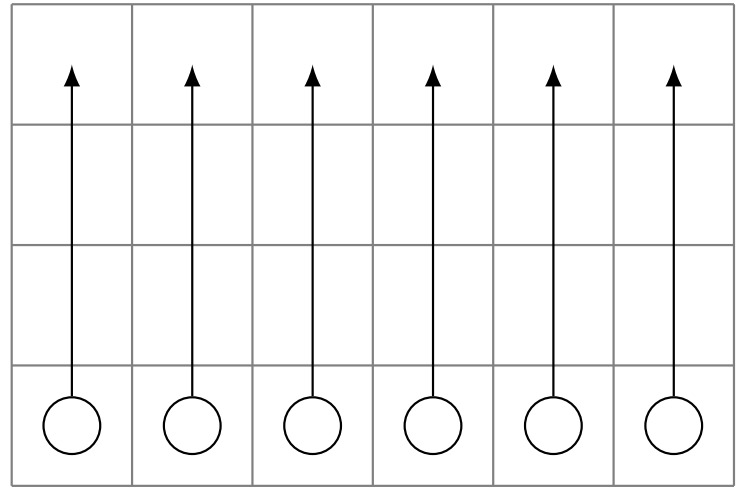
Levelregeln, Regelrichtung

Zur Ausführung einer Regel wird die Karte in Linien unterteilt; die Richtung gibt an, wie diese Unterteilung aussieht:

`east`



`north`



Jede Linie wird getrennt von Anfang bis Ende abgearbeitet

- Innerhalb der Linien ist die Reihenfolge wichtig
- In welcher Reihenfolge die Linien abgearbeitet werden, ist unwichtig

Levelregeln, Ersetzung

Zur Bearbeitung einer Linie sind die Arrays `original` und `result` relevant:

- Zwei gleichlange Arrays
- `original` gibt an, nach welchen Gegenständen in dieser Linie gesucht werden soll
- `result` gibt an, womit gefundene Stellen in dieser Linie ersetzt werden sollen

Beispiel:
`original`: [`GEM`, `STONE`]
`result`: [`PATH`, `PATH`]

Levelregeln: Objekte in original und result

`original` und `result` können mehr als reine Ersetzung von festen Gegenständen durch feste Gegenstände
Einträge sind Objekte der Form

```
{"token":..., "values":{"value1":47, ...}},
```

- `values` kann festlegen, dass bestimmte Zusatzwerte bestimmte Werte haben müssen; im `result`-Teil können diese geändert werden
- in `original`: `token` mit einem einfachen String (der nicht `*` ist), legt fest, dass genau dieses Objekt gefunden werden soll
- in `original`: `token` mit einem Array von Strings legt fest, dass einer dieser Strings gefunden werden soll
- `*` in `original`: passt auf jedes Objekt
- in `result`: Zahlen geben an, dass der Gegenstand aus dem, was gefunden wurde, hierhin geschrieben werden soll

Beispiel: Levelregel

Ein Beispiel einer Levelregel ist

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

Zur kompakten Darstellung auf den Folien kürzen wir sie ab zu

`stone stone|gem stone -> gem path 1 (north)`

Beispiel: Levelregel, Anwendung auf Linie

```
stone stone|gem stone -> gem path 1 (north)
```

Angewandt auf die Linie

0	1	2	3	4	5	6	7	8
stone	stone	stone	stone	stone	gem	stone	stone	stone

Ersetzung an Position 0:

0	1	2	3	4	5	6	7	8
gem	path	stone	stone	stone	gem	stone	stone	stone

Ersetzung an Position 4:

0	1	2	3	4	5	6	7	8
gem	path	stone	stone	gem	path	gem	stone	stone

Beispiel: Levelregel, Anwendung auf Karte

```
stone stone|gem stone -> gem path 1 (north)
```

stone	stone	stone	stone
mud	stone	gem	mud
stone	stone	mud	mud
mud	stone	mud	stone
mud	stone	stone	stone
stone	stone	gem	stone
stone	stone	stone	stone
stone	stone	stone	stone

$\rightsquigarrow$

stone	stone	stone	stone
mud	stone	gem	mud
stone	stone	mud	mud
mud	path	mud	stone
mud	gem	gem	stone
stone	stone	path	stone
path	path	gem	path
gem	gem	stone	gem

Levelregeln und Zusatzwerte

Wie vorhin schon angedeutet gibt es auf den Feldern auch einige ganzzahlige Werte, sogenannte Zusatzwerte.

Ihre Hauptbedeutung haben sie im Zusammenspiel Levelregeln, Hauptregeln.

Beispiele:

- `FALLING`-Zusatzwert kann verwendet werden, um herauszufinden, welche Gegenstände gerade fallen
- `BAM`-Zusatzwert kann in den prerules gesetzt werden, um die Hauptregeln zu veranlassen eine 3-Kästchen Explosion zu verursachen
- `LOOSE`-Zusatzwert kann verwendet werden, um zu steuern, welche Gegenstände herunterfallen können

Globale Zusatzwerte

Einige Zusatzwerte sind global, also auf jedem Feld gleich

- Beispiele: gems (gesammelte Edelsteine), ticks (vergangene Zeit)
- Wird ein globaler Zusatzwert gesetzt, so ändert er sich auf allen Feldern
- Auf jedem Feld müssen die globalen Zusatzwerte gleich sein
- Wird durch eine Regel ein globaler Zusatzwert mehrmals an verschiedenen Stellen geändert, so ändert sich dieser globale Zusatzwert auch stärker
  Beispiel: `{ ..., "original":[{"token":"slime"}], "result":[{"token":0,"values":{"gems":1}}]}` erhöht den globalen Zusatzwert gems um so viel, wie es Schleim auf der Karte gibt
- Es empfiehlt sich die globalen Zusatzwerte nur einmal zu speichern

Darstellung

- Die Levelauswahl soll in einer Weise dargestellt werden, dass man übersichtlich die zur Verfügung stehenden Level erkennen kann

- In einem Level soll die Anzeige die Spielfigur gut erkennbar darstellen, aber auch das Level selbst übersichtlich darstellen
  Hilfreich sein kann: Zoomen, zentrieren

Wie genau die Levelauswahl und Karte angezeigt werden, ist nicht vorgegeben.

Material

Materialien zum Hauptprojekt auf der Vorlesungsseite

- Ausführliche Projektebeschreibung mit vielen Implementierungsdetails.
- Ein Levelpack mit einer kleinen Auswahl von Leveln:
  - Text: Durch Spielgegenstände ist das Wort „Text“ geschrieben; hilft zu testen, ob man die Orientierung der Level richtig implementiert hat
  - Wand: Demonstriert von links und rechts wachsende Wände durch einfache Levelregeln
  - Bewegung: Dieses Level enthält verschiedene Arten von Gegnern
  - Labyrinth: Globale Levelwerte werden verwendet, um die Wände im Level beim Sammeln von Edelsteinen zu verändern
  - Schleimer: Ein Testlevel mit Schleim und Gegnern
  - Spiegelgeist: Enthält Levelregeln, die sich auf die Steuerung der Spielfigur beziehen

Ablauf

- Neue git-Verzeichnisse für das Hauptprojekt anlegen
  Gruppenname muss wieder im Verzeichnisnamen vorkommen.
- Durch die Projektbeschreibung ist der Pflichtteil der Aufgabe vollständig gegeben.
  Es werden noch kleine Ergänzungen vorgeschlagen werden; man kann sich auch selbst Ergänzungen überlegen
- Endabnahme findet zu einem noch zu bestimmenden Termin statt.
  Dazu erheben wir im Plenum ein Stimmungsbild (keine verbindliche Abstimmung).
  - Ende Vorlesungszeit
  - Ende Vorlesungsfreie Zeit
  - Sonstige Termine