Erweiterungen: Leveleditor, KI, Zufall

Erweiterungen

Wie angekündigt: Erweiterungen zu dem Projekt Boulder Dash

Heute:

- Leveleditor
  - in das Spiel eingebauter Editor, um Level zu erstellen
  - Unterstützung für die Karte an sich, aber auch für Levelregeln (pre/post)
- KI
  - Ein Computerspieler, der automatisch das Spiel spielt
  - Sollte zumindestens einige Level erfolgreich durchspielen; je mehr, desto besser
- Zufall: Levelregeln, die zufallsgesteuert nicht immer ausgeführt werden

Das meiste der auf diesen Folien vorgeschlagenen Umsetzungen soll als grobe Richtlinie dienen: Abänderungen und darüber hinausgehendes sind möglich

Leveleditor

Leveleditor ist dritter Modus neben Levelauswahl und Level spielen

Leveleditor besteht aus

- Level speichern: JSON-Datei muss geschrieben werden
- Karte bearbeiten: Felder ändern, Größe der Karte ändern, Zeit- und Edelsteinvoraussetzungen zum Gewinnen anpassen
- Regeln bearbeiten: Bequeme Eingabe von einzelnen Regeln, übersichtliche Anzeige der definierten Regeln
- Test- und Debugunterstützung: Level direkt aus dem Editor heraus ausprobierbar, danach Rückkehr in den Editor; Debugmöglichkeiten: Runden einzeln ausführen auf Tastendruck, Regeln debuggen

Regeln debuggen

Verschiedene Debugmöglichkeiten helfen beim Entwickeln von Levelregeln:

- Auch Levelregeln einzeln ausführen und Änderungen anzeigen
- Beim einzeln Ausführen für eine ausgewählte Stelle anzeigen, wieso die Regel dort nicht ausgeführt wird
- Levelzustand während der Ausführung ändern, ohne diese Änderung dauerhaft abzuspeichern
- Zusatzwerte anzeigen, zwecks Übersichtlichkeit besser nur ausgewählte Zusatzwerte

Karte bearbeiten

Prinzipiell reicht es, den Wert jedes einzelnen Feldes bearbeiten zu können

Besser sind mehr unterschiedliche Bearbeitungsmöglichkeiten:

- Undo/Redo: Letzte Änderungen rückgängig machen, Änderungen wiederherstellen
- Flächenfüllwerkzeug
- Bereiche an eine andere Stelle der Karte kopieren
- Linien ziehen
- Felder zufällig verteilen, beispielweise ein Füllwerkzeug, das nur mit einer einstellbaren Wahrscheinlichkeit das Feld ändert
- Viele weitere nützliche Kartenbearbeitungsvarianten möglich

KI

Es gibt verschiedene Ansätze, um eine KI umzusetzen; häufig verwendet werden:

- Betrachtung des Spielgraphens, Vorausberechnung von Zügen
- Regelbasierte Systeme, um auf konkrete Situationen reagieren zu können

Dabei sind folgende Aspekte zu beachten:

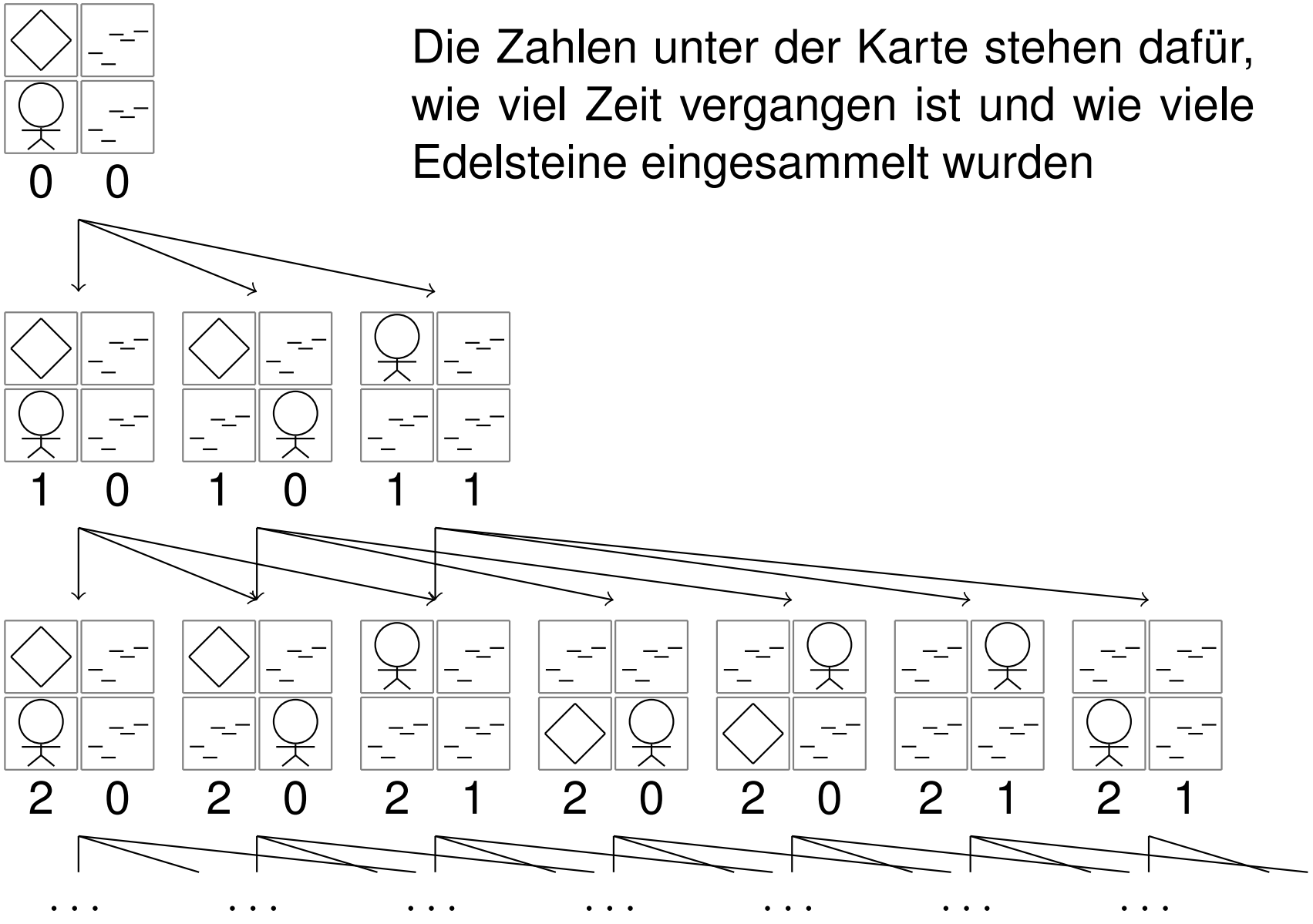
- Berechnung kann sehr zeitaufwendig werden
- Regelbasierte Systeme sind oft sehr schnell in der Berechnung, aber von mäßiger Qualität und nicht flexibel

Spielgraph

Ein wichtiges Konzept für eine KI ist der Spielgraph:

- Ein Graph mit den Spielsituationen als Knoten und möglichen Zügen als Kanten
- Wird normalerweise nicht vollständig berechnet, da zu groß
  - Tic-tac-toe: ca. 19 000 Knoten
  - Schach: ca. 10^{42} Knoten,
  - Boulder Dash (Level Text): ca. 10^{100} Knoten
- Es wird nur der Bereich berechnet, der nahe (=wenige Spielzüge entfernt) an der aktuellen Spielsituation liegt
- Auswahl eines guten Zuges ist damit ein Suchproblem auf einem Graphen, dafür sind gute Algorithmen bekannt, wir verwenden Breitensuche

Beispiel Spielgraph



Die Zahlen unter der Karte stehen dafür, wie viel Zeit vergangen ist und wie viele Edelsteine eingesammelt wurden

Spielstandbewertung

Idee: Alle Zugmöglichkeiten ausprobieren, den besten Zug nehmen

Festlegung, was gut ist: Berechnen einer Spielstandbewertung als eine Zahl, je größer, desto besser der Spielstand

- Gesammelte Edelsteine geben Pluspunkte
- Nähe zu Edelsteinen auf dem Spielfeld gibt Pluspunkte
- Spieler tot: Viele Minuspunkte
- viele weitere Möglichkeiten

Überlegen Sie sich selbst eine möglichst gute Spielfeldbewertung

Je besser die KI sich mit einer Spielfeldbewertung benimmt, desto besser die Bewertung (Ausprobieren!)

Breitensuche auf Spielegraph

Breitensuche aus dem Vorprojekt (Flughafen) angepasst:

- Es wird nicht mehr ein möglicher, sondern ein bester Weg gesucht
- Breitensuche wird abgebrochen, sobald Zeit zum Suchen vorbei ist (Zug muss rechtzeitig berechnet sein)

Dabei wird ein Knoten mit einer möglichst guten Spielstandbewertung gesucht, der mehrere Züge entfernt ist

Vom gefundenen Weg wird der erste Schritt genommen

Vorteil gegenüber direkten bestbewerteten Nachfolger nehmen: so können auch kleine „Pläne“ ausgeführt werden: z.B. Edelststein unter einem Stein befreien und dann nehmen

Regelbasierte Systeme

Regelbasierte Systeme reagieren direkt auf einzelne bekannte Situationen, beispielsweise

- Steht man neben einem Edelstein, ohne dass auf ihm etwas liegt, was einstürzen kann, so soll man ihn nehmen
- Einem Gegner muss man ausweichen
- Steht man nahe eines `xling`s, hat einen geeigneten Bereich mit Erdreich und einem Stein über sich, so kann man dem `xling` auf eine vorgegebene Weise einen Stein auf den Kopf fallen lassen

Regelbasierte Systeme, Vor- und Nachteile

Regelbasierte Systeme bieten eine Reihe von Vorteilen gegenüber der Suche auf dem Spielgraphen:

- Die Implementierung ist schneller so weit, dass die KI ein erkennbares Verhalten zeigt und einfache Level lösen kann
- Es können wesentlich komplexere Verhaltensweise vorgegeben werden und die KI wirkt zeitweise intelligenter
- In vielen Fällen ist die Berechnung über Regelsysteme effizienter

Nachteile gegenüber einer Suche auf dem Spielgraphen sind

- Um alle Situationen abzudecken müssen sehr viele Regeln angegeben werden
- Fehler in regelbasierten Systemen sind oft schwer zu finden, gerade bei Regeln, die selten verwendet werden
- Regelbasierte Systeme kommen sehr schlecht mit unerwarteten Situationen zurecht, gerade die Levelregeln sind schwer zu erfassen

KI, Kombinierter Ansatz

Als Kompromiss kann man beide Systeme kombinieren:
Wenige Regeln für gewisse Situationen, bei denen die Suche schlechte Ergebnisse liefert können Vorteile beider Ansätze vereinen

KI, Möglicher Implementationsablauf

Als weitere Hilfestellung gibt es hier einen möglichen Ablauf eine KI zu implementieren; dabei implementiert man aufbauend eine immer bessere KI

1. Zufällige Auswahl der Aktion
2. Auswahl des nächsten Zugs mit bester Spielstandbewertung
3. Breitensuche nach Spielposition mit guter Spielstandbewertung
4. Regelbasiert komplexere Verhaltensweise hinzufügen, z.B. Gegner durch Steine erschlagen, sofern das in der Situation geht und hilft

Hinweis: Statische KI

Keine gute Idee ist es den Spielverlauf mit Spieleraktionen vorauszuberechnen und dann anzuzeigen:

- Kommt nicht mit Zufall in den Leveln klar (Schleim, Erweiterungen)
- Eine KI, die bei jedem Aufruf das gleiche macht, wirkt langweilig
- Kann nicht so erweitert werden, dass sie im Spiel auf Spielerwunsch jederzeit übernimmt

Hinweis: Selbstlernende KI

Erfahrungswert: viele Gruppen wollen eine selbstlernende levelabhängige KI implementieren, die funktioniert aber dann nicht richtig

Von einer selbstlernenden KI wird abgeraten, aber wenn man es trotzdem machen möchte, sollte man folgendes beachten

- Schwierigkeitsgrad von komplexeren Verfahren für selbstlernende KIs wird oft unterschätzt
- Vollständiges Verhalten zu erlernen ist das Hauptproblem
- Teilweises Erlernen ist einfacher: Erlernen der Parameter für die Spielstandbewertung funktioniert besser
  - Parameter bestimmen, wie viele Punkte welche Aspekte der Spielstandbewertung geben
  - Mehr Parameter: Besseres Endergebnis, aber viel längere Suche nach guten Werten. Erstmal maximal 4 Parameter!
  - Parameterwerte so optimieren, dass sie eine möglichst gute KI hervorbringen (evolutionäre Suche am einfachsten zu implementieren, viele Optimierungsverfahren funktionieren)

KI und Leveleditor

Eine gute KI kann den Leveleditor bereichern

- Level im Editor mit KI starten: Gibt ein Gefühl für das Level, ohne dass man es jedes Mal selbst spielen muss

- Ohne Anzeige im Hintergrund spielen: kann Daten für verschiedene hilfreiche Anzeigen liefern:
  - Mit welcher Chance schafft die KI dieses Level (falls Zufall in der KI oder im Spiel verwendet wird)?
  - Wie lange braucht die KI?
  - Wie viele Edelsteine sammelt sie?
  - Welche Edelsteine werden (nicht) gesammelt?

Zufall in Levelregeln

Die dritte Erweiterung führt Zufall in Levelregeln ein.

Dazu können die Objekte der Levelregeln einen zusätzlichen Eintrag `probability` haben:

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

Hat `probability` den Wert *p*, so gibt dies an, dass wenn eine Regel an einer Stelle passen würde, die Wahrscheinlichkeit, dass sie tatsächlich passt, nur noch *p* ist.

Gibt es keinen Eintrage `probability`, so wird, genau wie bei einem `probability`-Wert von 1, die Regel ganz normal ausgeführt, also immer, wenn sie passt.

Zufall und KI

Zufall (Zufallserweiterung, Schleim) in einem Level kann für ein anderes Ergebnis im Spielablauf, als in der Vorausberechnung sorgen

Einfache Möglichkeit: Besten Zug mehrmals vorausberechnen und der erste, der öfter als eine bestimmt Schranke kommt, nehmen

Komplexere Möglichkeit: Einen Schritt vorausberechnen und von jeder der möglichen Ergebnisse bestimmen, eine wie gute Position man erreicht, dann den Erwartungswert optimieren