Entwicklung domänenspezifischer Sprachen mit ANTLR am Beispiel eines Pac-Man-Klons

Studiengang Medieninformatik Master
Beuth Hochschule für Technik Berlin

Marcel Brüning
s67176@beuth-hochschule.de

Simon Lischka
simon@lischka.co

Marcel Piater
s67357@beuth-hochschule.de

***Zusammenfassung*—Die nachfolgenden Seiten beschreiben die Entwicklung eines Pac Man Browserspiels mit JavaScript. Die Schwerpunkte des Projekts liegen in der automatischen Codegenerierung mit domänenspezifischen Sprachen (DSL) für die Levels und die künstlichen Intelligenz (KI). Die DSL der KI beschreibt verschiedene Strategien der vom Computer gesteuerten Spielfiguren.**

I. Über das Spiel

In diesem Absatz wird das Spielprinzip von Pac Man kurz vorgestellt und auf die Entwicklung eingegangen. Dabei werden auch die eingesetzten Technologien beschrieben.

A. Spielprinzip

Pac Man ist eine Spielfigur, die durch ein Labyrinth so schnell und effektiv wie möglich gesteuert werden soll um alle vorhanden Punkte einzusammeln. Auf dem Weg erschweren diverse Hindernisse den Siegeszug. Bewegliche Gegner (vier Geister) die von einer KI gesteuert werden sowie ein komplexes Labyrinth erschweren die Bewältigung des Spiels. Pro Spiel stehen verfügt Pac Man drei Leben. Wird er von einem Gegner gefasst, so geht ein Leben verloren. In dem Labyrinth gibt es durch das Sammeln von Münzen die Möglichkeit, die Gesamtpunktzahl zu erhöhen.

Über Früchte, welche Pac Man auf seinem Weg finden kann, gelangt dieser in den Angriffsmodus. In diesem Modus wird Pac Man für eine kurze Zeit vom Gejagten zum Jäger und kann seinerseits Geister fressen, was ihm wiederum zu zusätzlichen Punkten verhilft. Sollte es Pac Man innerhalb der drei Leben schaffen sämtliche Punkte auf dem Spielfeld zu konsumieren, so hat er das Level erfolgreich absolviert und startet ein neues Level.

B. Verwendete Technologien

Da das Spiel in JavaScript implementiert ist, muss der Codegenerator JavaScript-Code erzeugen, der von der Applikation angesteuert wird. Zur Interpretation von DSL-Dateien und Erzeugung von JavaScript-Code wird die Java-Implementierung von ANTLR4 verwendet.

Ein naheliegender Gedanke in der Planungsphase war es, die JavaScript-Implementierung von ANTLR zu verwenden. Auf diese Weise ließe sich eine einheitliche Programmiersprache in der gesamten Codebasis einsetzen.

Eine Rechtfertigung für dieses Vorgehen besteht darin, dass ein Team was sich nur auf eine Programmiersprache einarbeiten muss mit geringem Zeitaufwand einheitliche Coding-Standards etablieren kann, da diese nur für eine Sprache aufgestellt werden müssen. Ein programmiertechnischer Austausch zwischen allen Teammitgliedern, etwa durch Code Reviews oder die Klärung von sprachspezifischen Problemen würde vermutlich ebenfalls erleichtert werden. Das Ziel eines einheitlichen Einsetzens von JavaScript wäre also das Erlangen höherer Codequalität gewesen.

Da zur Java-Implementierung von ANTLR ausführlichere Dokumentation existiert als für die JavaScript-Variante und die Java-Implementierung etablierter erscheint, haben wir jedoch von der Verwendung von ANTLR für JavaScript abgesehen. Wir hatten die Vermutung, dass die zusätzliche Einarbeitungszeit und für uns unerwartetes Verhalten der JavaScript-Implementierung den zeitlichen Rahmen des Projektes übersteigen und in keinem Verhältnis zur höheren Codequalität durch den Einsatz einer einheitlichen Programmiersprache stehen würde.

Wir haben jedoch für die in JavaScript verfasste Codebasis gezielt Technologien zur Qualitätssteigerung ausgewählt. Hierzu gehört das Framework RequireJS, welches die Modularisierung und das Importieren von Klassen ähnlich wie in Java ermöglicht. Für das Einbetten von durch ANTLR erzeugte JavaScript-Klassen müssen diese als Module ladbar sein - RequireJS war für uns deshalb Voraussetzung um unser Vorhaben erfolgreich umsetzen zu können. Objektorientiertes Programmieren nach dem Paradigma Separation of Concerns (SoC) und der Aufbau einer übersichtlichen Projektstruktur werden durch Trennung von Klassen in einzelne Dateien durch RequireJS ebenfalls stark erleichtert.

Funktionale Programmierung ist unter JavaScript prinzipiell möglich. Da wir mit dem funktionalen Programmierparadigma vertraut sind und es als modern und effizient empfinden, lag es in unserem Interesse, den Teammitgliedern dessen Einsatz zu erleichtern. Hierzu wurde Underscore.js eingesetzt. In kurzer Syntax und auf höherem Abstraktionsgrad sind so insbesondere häufig eingesetzte Idiome wie Listeniterationen abbildbar. Hierdurch ergibt sich weniger Raum für Fehler.

Kritische Funktionen wurden in JavaScript mit Unit Tests versehen. Hierzu setzen wir das Framework Jasmine ein. Eine

hohe Testabdeckung, wie sie mit Test Driven Development möglich ist, haben wir jedoch nicht priorisiert.

C. Umsetzung

Das zu spielende Level wird durch ein 2-Dimensionales Array abgebildet. Die Elementes des Spielfelds sind durch Zahlencodes im Array abgebildet. Eine Null steht dabei für ein freies Stück Weg. Die Eins und die Zwei stehen für ein begehbares Feld mit einem Punkt oder einer Frucht darauf. Wände werden durch eine Drei repräsentiert und sind von Pac Man und den Gegnern nicht begehbar. Auf diese Art und Weise können beliebig komplexe Levels erstellt und in das Spiel geladen werden.

Zur Darstellung werden für die genannten Elemente Bildobjekte in das Spiel geladen, die auf einem Canvas-Element des HTML Dokuments gezeichnet werden. Zum Zeichnen der Bildobjekte wird durch das Array iteriert und für jedes Array-Element das dem Zahlencode zugehörige Bildobjekt auf den 2D Kontext des Canvas gezeichnet.

Für die Spielfiguren wird ein zweites Canvas-Element verwendet, das transparent über den Canvas-Element des Levels liegt. Die Motivation hierfür ist, dass der Zustand der Spielfiguren sich in der Regel häufiger ändert als der des Spielfeldes. Ein Flackern, was durch zusätzliches Neuzeichnen des Levels bei jeder Figurveränderung entstehen kann, wird durch Trennung in zwei Canvas-Elemente reduziert.

Die Figuren werden genauso wie die Objekte des Levels als Bilder geladen. Sie operieren auf dem selben Koordinatensystem wie das Level.

Abbildung (REFERENZ EINFÜGEN) ist ein reduziertes Klassendiagramm und stellt den zuvor beschriebenen Sachverhalt dar. Die Objekte von Spieler und Gegner (Pac und Ghost) besitzen die Instanzvariablen gridX und gridY, die wie beschrieben den Wertebereich des Levels besitzen. Die Funktion checkMove() der Klasse Gameboard gleicht den nächsten Schritt einer Figur mit dem Array des Levels ab. Bei Zahlencode Null, Eins oder Zwei liefert sie true zurück und erlaubt den gewünschten Schritt der Figur. Entsprechend wird bei einer Drei, also einer Wand false zurückgeliefert und verhindert somit den nächsten Schritt der Spielfigur. Diese Funktion wird von Klassen Ghost und Pac bei der Umsetzung des nächsten Spielzugs verwendet.

Um die Bewegungen der Figuren sichtbar zu machen existiert die Funktion updateOnInterval(), die durch den Scheduler des Browsers alle 150 Millisekunden aufgerufen wird. Sie zeichnet das Level und die Spielfiguren mit den entsprechenden Informationen und Positionen neu. In der Methode werden Kollisionsabfragen der Klasse GameBoard aufgerufen, die prüfen ob Pac Man gerade einen Punkt bzw. eine Frucht frisst (Methode checkPacsEating()) oder mit einem Geist kollidiert ist (Methode checkKills()). Die im Spiel agierende KI berechnet in jedem Interval die Richtung der Geister neu. Ihre Berechnung bezieht sich also immer genau auf einen Spielzug und hat dementsprechend die nächste zu gehenden Richtung als Rückgabewert.

Abbildung (REFERENZ EINFÜGEN) stellt das FMC-TAM Diagramm des Spiels dar. Ghost und Pacman sind als Akteure dargestellt, die ihre Position in Abstimmung mit den Request-Antworten von checkMove aktualisieren. Akteur Strategy und Speicher Level werden durch ANTLR generiert. Im Zusammenhang mit Strategy ist Mittler GhostQuery relevant. GhostQuery stellt die Schnittstelle des generierten Codes zum Spiel dar und wird bei der Beschreibung der KI DSL in Abschnitt (REFERENZ EINFÜGEN) näher erläutert.

Interessant ist, dass die Akteure sich nicht automatisch auf die gleiche Modularisierungseinheit, also Beispielsweise nur Klassen abbilden. Checkmove etwa wird im aktuellen Entwicklungsstand durch eine einzelne Methode repräsentiert, Hitdetection durch mehrere Methoden der Klasse Gameboard. Akteure Pacman und Ghost beispielsweise existieren als eigene Klassen, ebenso wie Strategy und GhostQuery.

Es ist wahrscheinlich, dass eine Auslagerung von Hitdetection in eine eigenständige Klasse bei zunehmender Komplexität sinnvoll wäre und dadurch das Design durch die Einhaltung des SoC-Paradigmas verbessert werden würde. Durch die Aufstellung des FMC-TAM Diagramms lässt sich diese bevorstehende Designänderung antizipieren, auch wenn wir uns in der vorliegenden Implementierung dazu entschlossen haben, sie nicht umzusetzen.

II. Codegenerierung

ANTLR4 generiert auf Grundlage der DSL Lexer zur lexikalischen Analyse und Parser zur syntaktischen Analyse. Wir haben uns bei der Umsetzung der Level- und der AI DSL für das Einsetzen von Listener Klassen zum Traversieren des Abstrakten Syntaxbaums (AST) entschlossen.

Für beide Grammatiken erstellten wir Subklassen der generierten Listener. Diese Klassen enthalten jeweils eine Datenstruktur als Instanzvariable, die beim Traversieren befüllt wird. Im Fall der Level DSL handelt es sich um eine Liste, bei der AI DSL um eine Baumstruktur.

Erst nach Beenden der Traversierung werden die Daten in der Klasse CodeGenerator durch Interpretation der Datenstrukturen, die in den Listenern erzeugt wurden, geschrieben. Abbildung (REFERENZ EINFÜGEN) stellt die beteiligten Klassen dar. Bei totalValues und initialRoot handelt es sich um die erwähnten Datenstrukturen.

A. Level DSL

Der LevelListener ist für das Auslesen der Daten zuständig. In der Java Klasse LevelBaseListenerImplementation.java welche von der LevelListener.java erbt ist zum einen der Zugriff auf die Daten möglich zum anderen wurde an dieser Stelle die Überprüfung der Spielfeldgröße umgesetzt welches genau eine Arraygröße von 20x20 voraussetzt. Die parseLevel() Methode welche in der PacmanParseExecuter.java Klasse definiert ist erwartet eine .csv Datei als Argument. welche nach dem Schema wie in Tabelle 1 aufgebaut ist. Des Weiteren wird in dieser Methode die Initialisierung aller abhängigen Klassen, die für diesen Prozess nötig sind, erledigt.

Die CodeGenerator.java Klasse ist neben der Generierung für die KI auch für die Erstellung einer level.js Datei zuständig welche nach JavaScript Syntax erstellt wird. Obwohl die durch ANTLR4 generierten Lexer, Listener und Parser sowie der Codegenerator auf Java basiert muss der Codegenerator eine JavaScript-Datei erstellen. Diese ist insofern nötig da das Spiel auf auf dieser Technologie basiert und für die Weiterverarbeitung im Projekt unumgänglich ist. Die level.jswird direkt in dem Ordner `/dsl_pacman/pacman/levels/` generiert und von der Gameboardklasse verwendet um das Level zu erzeugen.

```
define([], function () {
  return {
    floor: 0,
    point: 1,
    fruit: 2,
    wall: 3,
    map: [
      [3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3],
      [3,1,1,1,1,1,1,1,1,1,3,1,1,1,1,1,3,1,1,2,3],
      [3,1,3,1,3,3,1,3,1,3,1,3,3,3,1,3,1,3,1,3,3],
      [3,1,3,1,1,1,1,1,1,1,1,1,1,3,1,3,1,3,1,3,3],
      [3,1,3,3,3,3,1,3,3,3,3,3,1,3,1,3,1,3,1,3,3],
      [3,1,1,1,1,1,1,1,1,1,3,1,1,1,3,1,1,1,1,1,3],
      [3,1,3,3,3,3,1,3,1,3,1,3,1,3,3,3,3,3,1,3,3],
      [3,1,3,3,1,1,1,1,1,2,1,1,1,1,1,1,1,3,1,3,3],
      [3,1,1,3,1,3,1,3,0,3,0,3,1,3,3,3,1,1,1,3,3],
      [3,3,1,3,2,3,1,3,0,0,0,3,1,3,2,1,1,3,1,3,3],
      [3,1,1,3,1,3,1,3,3,3,3,3,1,3,1,3,1,3,1,3,3],
      [3,1,3,3,1,1,1,1,1,1,1,1,1,3,1,3,1,3,1,3,3],
      [3,1,3,3,3,3,1,3,1,3,1,3,1,1,1,1,1,1,1,3,3],
      [3,1,1,1,1,1,1,3,1,3,1,3,1,3,3,3,3,3,1,3,3],
      [3,1,3,3,3,3,3,1,1,1,1,1,1,1,1,1,1,3,1,3,3],
      [3,1,3,1,1,1,1,3,3,3,3,3,1,3,3,3,1,1,1,3,3],
      [3,1,3,1,3,3,1,1,1,3,1,1,1,1,1,1,1,3,1,3,3],
      [3,1,3,1,3,3,1,3,1,1,1,3,3,3,3,3,3,3,1,3,3],
      [3,2,1,1,1,1,1,1,1,3,1,1,1,1,1,1,1,2,3],
      [3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3]
    ]
  }
});
```

Listing 1. blabla

1) Spielfeldaufbau: Wie im Kapitel "Über das Spiel" beschreiben bildet das Spielfeld ein zweidimensionales Array ab, welches je nach Spielfeld-Design mit Zahlen von Null bis Drei befüllt wird. Für jeden Wert im Array werden dann die festgelegten Gegenstände auf der grafischen Oberfläche, dem Canvas Objekt dargestellt. Die Spezifikation sieht vor, dass ein Array der Größe von exakt 20x20 erstellt werden muss. Die Definition des Arrays wird in einer .csv Datei als Tabelle erstellt. Eine beispielhafte Darstellung die das Prinzip für den Aufbau des Spielfeldes veranschaulicht ist in Tabelle 1 dargestellt.

Zulässig sind als "Value" definiert, nur die Zahlen Null, Eins, Zwei und Drei in beliebiger Reihenfolge. Als Trennsymbol einzelner Values ist allein das Semikolon als "Separator" zulässig. Diese Anreihung von Value und Trennsymbol kann beliebig oft vorkommen solange bis das Ende einer Zeile, welche als "row" festgelegt ist, erreicht wird. Das Zeilenende wird durch ein "LineBreak" oder aber durch ein "EOF" signalisiert. Letzteres dient als terminales Symbol und signalisiert das Ende des einzulesenden Spielfeld-Arrays beziehungsweise beendet das Einlesen der Zeilen. Alle Zeilen zusammen werden als "field" spezifiziert.

Tabelle I
AN EXAMPLE OF A TABLE

3	3	3	3	3	0	freier Weg
3	2	1	1	3	1	Punkt
3	1	0	1	3	2	Frucht
1	1	1	2	3	3	Mauer/Hindernis
3	3	3	3	3		

```
grammar Level;

field       : row* EOF ;
row    : value (Separator value)* (LineBreak | EOF) ;

value       : Value ;
Separator   : ';' ;
LineBreak   : '\r'?'\n' | '\r';
Value : ('0'|'1'|'2'|'3')+ ;
```

Listing 2. Auszug aus der DSL spezifizierenden Grammatik level.g4

2) Grammatik: Um den korrekten Aufbau des Spielfelds zu gewährleisten werden zunächst domänenspezifische Gültigkeitsregeln festgelegt. In unserem Fall werden diese durch Zahlen, Sonderzeichen und reguläre Ausdrücke repräsentiert. Die Spezifikation der Arraygröße ist nicht Teil dieser DSL und die Überprüfung dafür findet an anderer Stelle statt welche im Kapitel "Codegenerierung mit ANTLR" beschrieben wird. Die in dem vorherigen Absatz, "Spielfeldeaufbau", festgelegten Regeln werden in einer level.g4 Datei(Abb.???) formal festgelegt um dann mit dem ANTLR4 Tool Java Code zu generieren welcher zur Überprüfung der Gültigkeit des aufzubauenden Spielfeldes (Vgl. Tab. 1) verwendet wird.

Zulässig sind als "Value" definiert, nur die Zahlen Null, Eins, Zwei und Drei in beliebiger Reihenfolge. Als Trennsymbol einzelner Values ist allein das Semikolon als "Separator" zulässig. Diese Anreihung von Value und Trennsymbol kann beliebig oft vorkommen solange bis das Ende einer Zeile, welche als "row" festgelegt ist, erreicht wird. Das Zeilenende wird durch ein "LineBreak" oder aber durch ein "EOF" signalisiert. Letzteres dient als terminales Symbol und signalisiert das Ende des einzulesenden Spielfeld-Arrays beziehungsweise beendet das Einlesen der Zeilen. Alle Zeilen zusammen werden als "field" spezifiziert

B. AI DSL

Eine mit Hilfe der AI DSL durch ANTLR generierte JavaScript-Klasse ermittelt auf Basis der aktuellen Richtung eines Geistes dessen Richtung für den folgenden Spielzug. Eingabe und Ausgabewert der AI-Klasse ist also die Richtung eines Geistes. Aus diesem Grund ist ein Verständnis der Tokens nötig, die für das Ausdrücken einer Richtung eingesetzt werden.

III. CONCLUSION

The conclusion goes here.

ACKNOWLEDGMENT

The authors would like to thank...

LITERATUR

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.