

ALADIN - Projektbericht

Projektseminar WS 2023/24

Inhaltsverzeichnis

1. Einleitung	1
1.1. Vorstellung ALADIN	1
1.2. Projektseminar	2
2. Aufgabentyp - Endlicher Automat	3
2.1. Motivation	3
2.2. Aufgabenbeschreibung	3
2.3. Anforderung	3
2.4. Planung	3
2.5. Technische Umsetzung	4
2.6. Ergebnis und Ausblick	13
3. Aufgabentyp - Clusteranalyse	14
3.1. Aufgabenbeschreibung	14
3.2. Anforderung	14
3.3. Planung	14
3.4. Technische Umsetzung	15
3.5. Ergebnisse	17
3.6. Ausblick	21
4. Aufgabentyp - Entscheidungsbaum	22
4.1. Aufgabenbeschreibung	22
4.2. Anforderungen	22
4.3. Planung	23
4.4. Technische Umsetzung in ALADIN	32
4.5. Ergebnisse	52
4.6. Ausblick	59
5. Aufgabentyp - Regex Puzzle	60
5.1. Aufgabenstellung	60
5.2. Aufgabenbeschreibung	60
5.3. Anforderung	60
5.4. Planung	60
5.5. Technische Umsetzung	61
5.6. Ergebnisse	67
5.7. Ausblick	68
6. Aufgabentyp - Chemie	70
6.1. Aufgabenstellung	70
6.2. Aufgabenbeschreibung	70
6.3. Anforderung	71
6.4. Planung	72
6.5. Technische Umsetzung	72

6.6. Ergebnisse	75
6.7. Ausblick	76
7. Reflexion	77
7.1. Jonas Hölzel	77
7.2. Norman Sebastian Arnold	77
7.3. Tanja Dietrich	77
7.4. Alessandra Ruff	78
7.5. Julius Wyrembek	79
7.6. Vincent Weise	79
8. Ausblick	81

1. Einleitung

Die Lehre an einer Universität oder Fachhochschule in Deutschland repräsentiert die höchste Ebene des Bildungssystems und ist darauf ausgerichtet, Studierenden umfassende theoretische Kenntnisse, praktische Fähigkeiten und wissenschaftliche Methodenkompetenz zu vermitteln. Um diese Fähigkeiten und Kompetenzen zu erlangen, benötigen die Studierenden vielseitiges Übungsmaterial, um das Wissen zu verinnerlichen.

Daher werden in der Lehre diverse Aufgabenformate benötigt, darunter beispielsweise Übungsaufgaben und Probeklausuren. Jedoch erfordert die Erstellung solcher Aufgaben einen erheblichen Zeitaufwand von Lehrkräften, die in der Regel dazu neigen, die Aufgaben manuell zu erstellen und zu korrigieren. Dies führt zu einer eingeschränkten Variation der Aufgabenstellungen, wodurch Studierende nur wenige oder gar identische Aufgabenformulierungen erhalten. Als Konsequenz entsteht ein Mangel an vielfältigen Übungsaufgaben für die effektive Vorbereitung auf Klausuren.

1.1. Vorstellung ALADIN

Die Bezeichnung ALADIN steht als Akronym für "Generator für Aufgaben und Lösungshilfen aus der Informatik sowie angrenzenden Disziplinen". Dieser Generator wurde entwickelt, um die zufallsbasierte Generierung von einer Vielzahl von Aufgaben zu ermöglichen. Studierende erhalten somit die Möglichkeit, diese Aufgaben online zu lösen.

Die aktuelle Ausrichtung von ALADIN liegt auf Aufgabenstellungen, die primär aus den Bereichen Produktionswirtschaft, Datenbanken und Projektmanagement stammen. Im Rahmen des Projektseminars soll gezeigt werden, dass ALADIN Aufgabenbereiche aus interdisziplinären Fachgebieten eingesetzt werden kann, um die Vielseitigkeit und Anwendungsbreite von ALADIN hervorzuheben. Dies soll zur Digitalisierung der Lehre beitragen.

Die Architektur der Software gliedert sich in zwei Hauptbereiche, wobei jeder Bereich sein eigenes Framework aufweist. Im Backend wird das ALADIN-Framework eingesetzt, das die Gliederung und Erstellung von Aufgaben und Lösungen steuert. Auf der Frontend-Seite kommt das zugehörige CARPET-Framework zum Einsatz, welches für das Design der Benutzeroberfläche verantwortlich ist und die Benutzerschnittstelle für die interaktive Bearbeitung und Präsentation der Aufgaben bereitstellt.

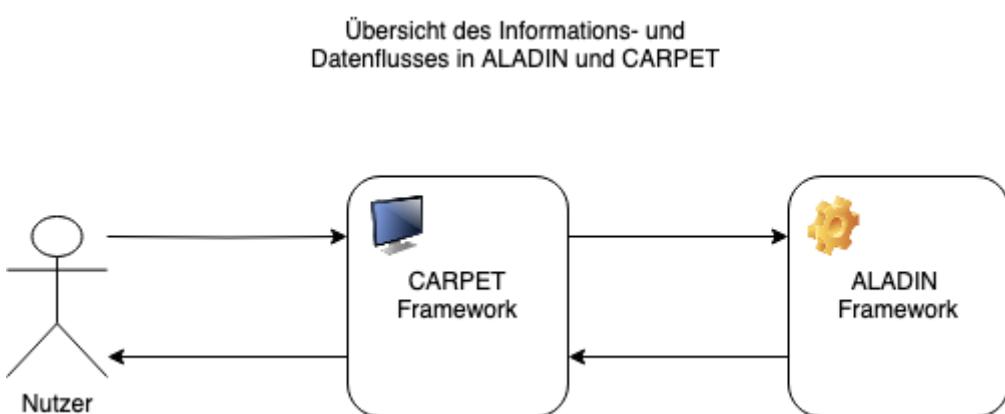


Abbildung 1.

1.2. Projektseminar

Im Verlauf des Projektseminars widmete sich die Projektgruppe, bestehend aus sechs Studierenden, der Konzeptionierung und Entwicklung von Aufgabengeneratoren in ALADIN. Dabei war die überwiegende Mehrheit der Teilnehmer eigenständig an der Entwicklung eines Generators beteiligt. Die Studierenden konnten aus verschiedenen Fachbereichen wie Musiktheorie, Chemie, Regular Expression, Spatial SQL, Endliche Automaten, Entscheidungsbäume, hierarchische Clusteranalyse und Phrase Structure Trees wählen, wobei auch eigene Ideen für die Umsetzung eines Aufgabentyps willkommen waren.

Ziel des Projektseminars

Das Projektseminar hat zum Ziel, neue Aufgabentypen in der ALADIN-Umgebung zu erforschen und umzusetzen.

Ablauf des Projektseminars

Auswahl eines Aufgabentyps

Die Studierenden wählen zu Beginn des Projektseminars den Aufgabentypen aus, den sie für die Entwicklung in ALADIN umsetzen möchten.

Einarbeitung in den Aufgabentyp

In den nächsten Wochen wurde ein generelles Verständnis für das Problem entwickelt und fachliches Wissen angeeignet, welches später helfen sollte, die Aufgabe in ein Programm umzusetzen.

Konzeption und Implementierung eines Prototyps

Folgend wurde zu jedem Konzept ein Prototyp in einer selbst ausgewählten Programmiersprache implementiert. Das Ziel bestand darin, ein grundlegendes Verständnis für die Komplexität zu entwickeln und eine Struktur für eine spätere Implementierung in ALADIN zu schaffen.

Umsetzung in ALADIN

Nachdem jeder Studierende die grundlegenden Funktionen seines Prototyps in einem externen Programm umgesetzt hat, begannen sie diesen in das vorhandene ALADIN Framework zu integrieren. Voraussetzung dafür war die Einarbeitung in ALADIN und CARPET sowie das fundamentale Verständnis über deren Zusammenarbeit.

Präsentation des entwickelten Aufgabentyps

Am Ende des Semesters wurden alle bearbeiteten Projekte den anderen Studierenden vorgestellt. Unter den Augen aller Professoren, die im Rahmen des Projektseminars ein Projekt betreut haben, konnten Studierende aus anderen Projekten Fragen zu den erbrachten Arbeiten stellen.

2. Aufgabentyp - Endlicher Automat

2.1. Motivation

Finite State Machines (FSMs) spielen eine zentrale Rolle in der Informatik und sind ein grundlegendes Konzept für das Verständnis und die Entwicklung von Software-Systemen. In der Praxis finden sie Anwendung bei der Steuerung von Verkehrsampeln, der Analyse und Verarbeitung von Text mit regulären Ausdrücken und bei der Implementierung von Programmiersprachen. FSMs ermöglichen es, das Verhalten von Systemen anhand von Zuständen und Übergängen präzise zu modellieren, was für das Design und die Analyse von Algorithmen sowie für das Erlernen grundlegender Konzepte der Informatik unerlässlich ist. Durch Abstraktion bieten FSMs einen Zugang zum Verständnis komplexer Systeme. Die Modellierung von komplexen Systemen und Prozessen durch FSMs ist ein wesentliches Werkzeug für Informatiker und trägt wesentlich zum Fortschritt in der Technologie bei.

2.2. Aufgabenbeschreibung

Die Aufgabe war es ALADIN um eine Aufgabe zu Endlichen Automaten zu erweitern. Dort soll es möglich sein, mehrere Aufgaben in ausgewählten Schwierigkeitsgraden zu dem Typ FSM zu generieren. Mögliche zu implementierende Aufgaben waren zum einen das Erzeugen des Zustandsdiagramms und lösen der Zustandstabelle bzw. umgekehrt. Oder das Erzeugen des Zustandsdiagramms und das angeben des zugehörigen input als regulären Ausdruck. Auch kann die Verarbeitbarkeit eines eingegebenen Wortes durch den Automaten überprüft werden.

2.3. Anforderung

Basierend auf der Aufgabenstellung ergeben sich folgende Anforderungen an das Programm:

1. Der Nutzer soll auswählen können, wie viele Aufgaben und in welchen Schwierigkeitsgraden generiert werden sollen.
2. Zu jeder Aufgabe soll ein zufälliger Endlicher Automaten generiert werden. Der dem vorher ausgewählten Schwierigkeitsgrad entspricht. Es soll also die Größe des Automaten variabel sein.
3. Der Automat soll als Graph dargestellt werden.
4. Es soll eine Zustandsübergangstabelle mit Lücken generiert werden, die vom Nutzer ausgefüllt werden muss. Diese Tabelle muss dann validiert werden. Auch das eingeben eines regulären Ausdrucks und eines Wortes, das der Automat akzeptiert, soll validiert werden.
5. Hat der Nutzer alle aufgaben bearbeitet, soll ihm eine Übersicht über die erreichten Punkte angezeigt werden.

2.4. Planung

Anhand von diesen Anforderungen, habe ich das Projekt in folgende Schritte unterteilt:

1. Generierung eines zufälligen FSMs mit einer variablen Größe. Das bedeutet die Anzahl der

Zustände und Übergänge soll variabel sein und je nach Schwierigkeitsgrad wachsen.

2. Erstellung einer Zustandsübergangstabelle, die zur Validierung der Nutzereingabe dienen soll.
3. Implementierung des User-Interfaces, das dem Nutzer die Möglichkeit gibt, die Aufgaben zu generieren, zu bearbeiten und die erreichten Punkte anzuzeigen.

2.5. Technische Umsetzung

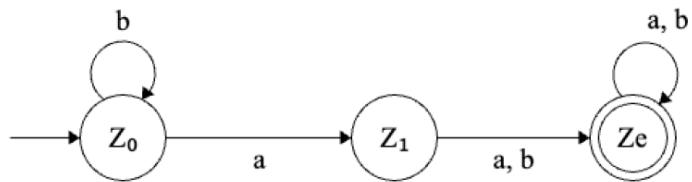
2.5.1. Mockups

Als erstes habe ich mir Gedanken gemacht, wie der Nutzer mit dem Programm interagiert und wie die Aufgaben auszusehen haben. Dafür habe ich in Figma ein paar Mockups erstellt, die mir als Vorlage für die Implementierung dienen sollten. Zum generieren der Aufgaben gibt es zwei Slider, einen um die Anzahl der zu generierenden Aufgaben auszuwählen und einen um den Schwierigkeitsgrad auszuwählen. Die Aufgaben bestehen dann aus dem Graphen des FSMs und einer Zustandsübergangstabelle, die vom Nutzer ausgefüllt werden muss bzw der Zustandsübergangstabelle und einem Editor, in dem der Nutzer den zugehörigen Graphen erstellen soll. Über die Buttons unten kann der Nutzer zwischen den Aufgaben navigieren.

Aufgaben Generieren



Erstellen Sie zu dieser FSM die passende Zustandstabelle.

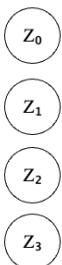


State	Input	
	a	b
Z_0		
Z_1		
Ze		



Erstellen Sie die zugehörige FSM zu dieser Zustandstabelle.

State	Input	
	0	1
Z_0	Z_0	Z_1
Z_1	Z_0	Z_2
Z_2	Z_0	Z_0
Z_3	Z_2	Z_1



2.5.2. Konzeption und Recherche

Angefangen habe ich damit, dass ich mir selbst eine Adjazenzmatrix erstellt habe und in einer Konsoleanwendung aus der Matrix dann einmal einen Graphen und eine Zustandsübergangstabelle generiert habe. Der Graph wurde dann in der Dot-Sprache beschrieben, ausgegeben und mit dem Online-Tool [Edotor](#) dargestellt.

Um den FSM aber zufällig zu generieren war mein erster Ansatz die Adjazenzmatrix zufällig zu generieren. Das hat aber nicht so gut funktioniert, da nicht garantiert werden konnte, dass der Automat auch wirklich zusammenhängend ist. Außerdem darf es keine unerreichbare Knoten und keine "Sackgassen" geben. Zudem darf von einem Knoten nicht zweimal die gleiche Kante zu verschiedenen anderen Knoten führen, da es sich sonst um einen nichtdeterministischen Automaten handeln würde. Da auch noch die Anzahl der Zustände und Übergänge variabel sein soll, gab es viele Edge-Cases, die beachtet werden mussten.

Ich habe viele verschiedene Ansätze ausprobiert unter anderem den Minimum-Spanning-Tree Algorithmus, aber keiner hat so funktioniert, dass 100% der Zeit ein brauchbarer Automat generiert wurde. Nach vielem Ausprobieren, kam mir die Idee nicht den Automaten zufällig zu generieren, sondern einen Regex zufällig zu generieren und aus diesem dann den Automaten zu generieren. Das hat dann auch funktioniert.

Bei meiner Recherche bin ich auf die "[Noam](#)"-Bibliothek gestoßen, die es ermöglicht Graphen, anhand von einem Regex, zu generieren. Ausgegeben wird der Graph dann in der Dot-Sprache und kann dann wieder mit Edotor oder beispielsweise mit [Graphviz](#) gerendert werden.

2.5.3. Implementierung in ALADIN

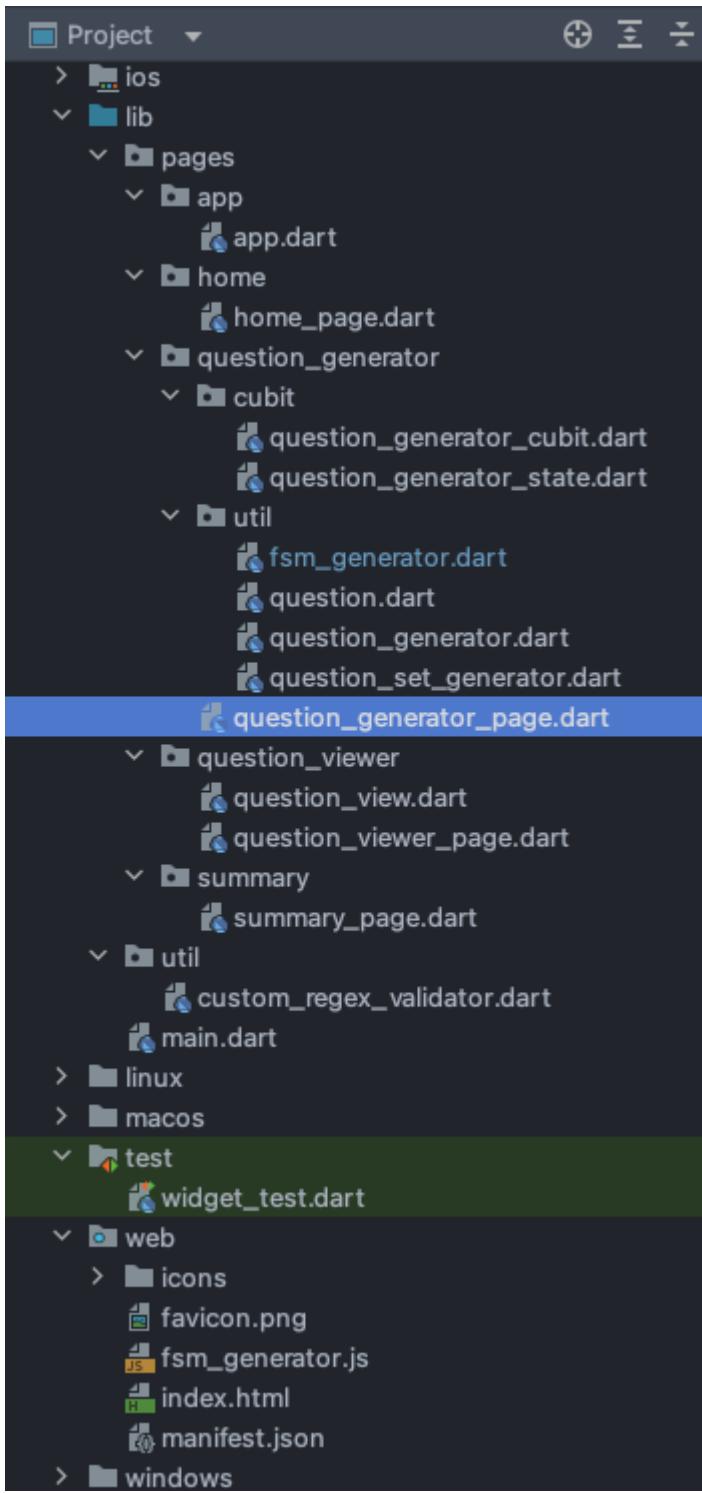
Die Einrichtung von ALADIN war auf meinem Mac leider sehr mühsam, es braucht eine spezielle Node-Version, die mit Macs mit M1-Chip kompatibel ist. Mit der Hilfe von Herrn Christ habe ich es dann zum Laufen bekommen. Allerdings hat sich nach einem Update wieder eine inkompatible Version installiert und ich habe sehr viel Zeit damit verbracht, ALADIN wieder zum Laufen zu bringen. Da die Zeit dann sehr knapp wurde und wir nur noch wenige Wochen bis zur Präsentation hatten, habe ich mich dazu entschieden, ein anderes Framework zu verwenden.

Meine Entscheidung fiel auf Flutter, da ich damit sehr viel Erfahrung habe und auch schon eine eigene App mit Flutter entwickelt und veröffentlicht habe.

2.5.4. Implementierung in Flutter

Flutter ist ein Framework von Google um plattformübergreifende Apps zu entwickeln. Im Fokus liegt dabei die Entwicklung von Android und iOS Apps. Aber auch Web-Apps und Desktop-Apps sind möglich. Flutter verwendet die Programmiersprache Dart, die von Google als Konkurrent zu TypeScript entwickelt wurde.

Ich habe also ein Flutter-Projekt erstellt:



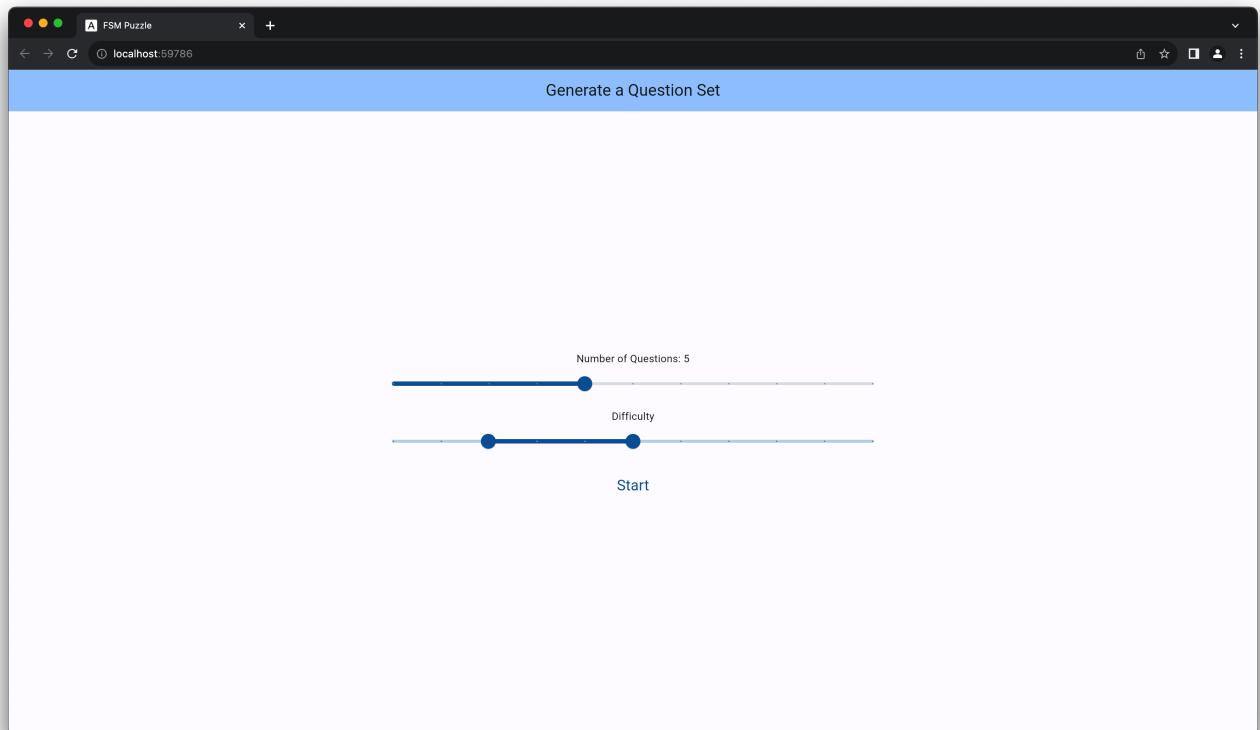
Durch die Verwendung von Flutter konnte ich leider nicht auf die in ALADIN schon integrierten Funktionen zurückgreifen, sondern musste vieles erst mal selbst implementieren. Gerade bei der UI musste ich von Null anfangen. Das hat mir aber auch die Möglichkeit gegeben, die UI so zu gestalten, wie ich es mir vorgestellt habe.

Die UI war auch schnell umgesetzt und es war möglich die Einstellungen für die Generierung der Aufgaben vorzunehmen und sich durch einen leeren Fragenkatalog zu klicken.

2.5.5. User-Interface

Das User-Interface besteht aus drei Seiten: Der Startseite, wo die Einstellungen zum Generieren der Aufgaben vorgenommen werden können. Es gib zwei Slider, einen um die Anzahl der zu generie-

renden Aufgaben auszuwählen und einen um den Schwierigkeitsgrad einzugrenzen. Außerdem gibt es einen Button, um die Aufgaben zu generieren. Beim Drücken dieser Buttons werden im Hintergrund die Aufgaben generiert und Es wird auf die nächste Seite weiter navigiert. Diese Seite stellt dann die erste Aufgabe da. Oben wird die Aufgabenstellung angezeigt, in der Mitte der Graph und darunter die Zustandsübergangstabelle. Rechts der Tabelle sind noch zwei Textfelder. Eins zum Angeben eines Wortes, welches vom Automaten akzeptiert wird und ein weiteres, wo der Regex dem der Automat entspricht angegeben werden soll. Am unteren Rand sind noch zwei Buttons um zwischen den Aufgaben zu navigieren. Ist man an der letzten Aufgabe angekommen, gelangt man über den Weiter-Button auf eine weitere Seite. Hier wird eine Übersicht über die erreichten Punkte angezeigt.



Fill the state table for the following FSM:
Regex hint: $b+(e+d)a$

State	Input	Next State
0	b	
0	e,d	
0	a	
1	b,e,d,a	
2	a	
2	b,e,d	
3	b,e,d,a	

Enter a word which gets accepted by the FSM:

Enter the Regex that describes the FSM:

3 / 5

Summary

- 1. Task: 10 / 10 ✓
- 2. Task: 7 / 9 ✗
- 3. Task: 7 / 7 ✓
- 4. Task: 5 / 7 ✗
- 5. Task: 8 / 13 ✗

Total Score: 37 / 46

[Back to Generator](#)

2.5.6. Generierung der Aufgaben

Nun muss das Aufgaben-Set erstellt werden. Dafür habe ich eine Klasse `QuestionSetGenerator` erstellt. Diese Klasse kann dann mit Hilfe der des `QuestionGenerator`'s eine Liste an `Questions` erstellen. Dafür wird der `QuestionGenerator` so oft aufgerufen, wie Aufgaben generiert werden sollen. Die generierten Aufgaben werden dann einer Liste angehängt. Dem Generator wird der Schwierigkeitsgrad übergeben, dieser wird zufällig innerhalb der Range, die vom Nutzer eingestellt wurde gewählt. Der `QuestionGenerator` generiert dann mit dem `FsmGenerator` den Dot-Code für den FSM.

Zur Generierung des FSMs musste ich dann die Noam-Library einbinden. Das war nochmal eine Herausforderung, da diese Library eine JavaScript Library ist und sie so nicht direkt in Dart verwendet werden kann. Also habe ich erstmal nach ähnlichen Libraries für Dart gesucht, bin da allerdings nicht fündig geworden. Auch mein Versuch die Library von ChatGPT umzuschreiben hat nicht funktioniert. Dies selbst zu machen hielt ich bei dem Umfang der Library auch für keine gute Idee. Also habe ich einen Wrapper um die, für mich notwendigen Funktionen in JavaScript geschrieben. Die Datei befindet sich unter [web/fsm_generator.js](#). Flutter-Web kann über ein JavaScript-Interop die JavaScript Funktionen der Datei im Browser ausführen. Dafür müssen die [fsm_generator.js](#)-Datei und die Noam-Library in die index.html als Script-Tag eingebunden werden.

```
41      <script src="https://ivanzuzak.info/noam/lib/browser/noam.js" defer></script>
42      <script src="fsm_generator.js" defer></script>
43  </head>
44  <body>
```

So gelang es mir dann die Noam-Library in Dart zu verwenden und den Dot-Code für den FSM zu generieren. Zunächst wird der zufällige Regex generiert, der dann verwendet wird, um einen Automaten zu generieren. Der generierte Automat ist ein eNFA, der dann in mehreren Schritten in einen DFA umgewandelt wird. Die Noam Library stellt die erforderlichen Funktionen bereit. Der DFA wird dann in der Dot-Sprache ausgegeben und zusammen mit dem Regex an den [QuestionGenerator](#) zurückgegeben.

2.5.7. Darstellung des FSMs

Da es in Flutter keinen nativen Dot-Renderer gibt, wird für die Darstellung die Seite quickchart.io verwendet. Sie bietet eine API an, die es ermöglicht, einen Graphen in der Dot-Sprache zu rendern und als SVG zurückzugeben. Das SVG kann dann in Flutter angezeigt werden. Dafür wird in der Klasse [QuickchartGraphviz](#) eine HTTP-Request URL generiert, beim Aufrufen gibt diese dann das SVG zurück. Dieses SVG wird dann in einem [SvgPicture](#)-Widget angezeigt.

2.5.8. Generierung und Validierung der Zustandsübergangstabelle

Die Zustandsübergangstabelle hat drei Spalten: 'State', 'Input' und 'Next State'. Die Spalte 'Next State' ist leer und muss vom Nutzer ausgefüllt werden. Dort muss eingetragen werden zu welchem Zustand man kommt, wenn man sich in dem Zustand aus der Spalte 'State' befindet und als Input eins der Zeichen aus der Spalte 'Input' bekommt. Da es für das erstellen der Tabelle notwendig war, durch den Graphen zu navigieren, habe ich den Dot-Code mit Hilfe der Library [graphviz2](#) in Objekte konvertiert. Durch diese Objekte konnte ich dann iterieren um die Tabelle generieren. Um das Durchnavigieren zu vereinfachen habe ich die Klasse [GraphvizHelper](#) erstellt. Die Tabelle wird durch zwei for-Schleifen generiert. Die erste Schleife iteriert über die Knoten und die zweite Schleife iteriert über die Kanten jedes Knotens. Daraus werden dann die Zeilen der Tabelle generiert.

```

127     List<Widget> _buildTableContent() {
128         final List<Widget> tableContent = [];
129         int textFieldsCount = 0;
130
131         for (var i = 0; i < _gvizHelper.nodes.length; i++) {
132             for (var j = 0; j < _gvizHelper.getInputsAt(_gvizHelper.nodes[i]).length; j++) {
133                 tableContent.add(_buildTableItemText(_gvizHelper.nodes[i].toString()));
134                 tableContent.add(_buildTableItemText(
135                     _gvizHelper.getInputsAt(_gvizHelper.nodes[i])[j]));
136                 tableContent.add(ValidatableTextField(
137                     id: textFieldsCount,
138                     origin: _gvizHelper.nodes[i],
139                     label: _gvizHelper.getInputsAt(_gvizHelper.nodes[i])[j],
140                     edgeExists: _gvizHelper.edgeExists,
141                     setValidation: (id, value) {
142                         _validationStates[id] = value;
143                     },
144                 ));
145                 textFieldsCount++;
146             }
147         }
148         return tableContent;
149     }

```

Das Validieren geschieht indem geprüft wird ob die Kante von dem Zustand aus 'State' mit dem Input aus 'Input' zu dem gleichen Zustand führt wie der der in 'Next State' angegeben wurde. Es gibt pro Lücke einen Punkt. Ist das Feld richtig, wird in der Liste `validationState` an der zugehörigen Stelle eine eins eingetragen.

2.5.9. Validierung des Regexes und des Wortes

Neben der Tabelle gibt es noch zwei weitere Text Felder. In dem einen muss der Nutzer den Regex eingeben, der den Automaten beschreibt und in dem anderen ein Wort, welches der Automat akzeptiert. Beide Felder werden validiert und bekommen zwei Punkte wenn sie richtig sind.

Bei dem Regex handelt sich hier nicht um den herkömmlichen Regex, sondern um eine vereinfachte Version, da dieser mit der Noam-Library kompatibel sein muss. Der Rexex akzeptiert Groß- und Kleinbuchstaben und die Ziffern 0 bis 9. Außerdem versteht er die Operatoren `()`, `$` als White-Space, `+` als 'ODER' und `*` als Kleene-Stern. Der eingegebene Regex wird dann mit dem Regex aus dem auch der Automat generiert wurde verglichen. Das ist nicht der optimale Weg, da es auch Regexes gibt, die den gleichen Automaten beschreiben, aber nicht gleich sind. Das wäre ein Punkt, den ich in Zukunft noch verbessern könnte.

Um das eingegebene Wort zu validieren, wird sich durch die Knoten des Automaten gehangelt bis der Knoten an dem wir uns befinden keine Kante mit dem nächsten Buchstaben des Wortes besitzt oder das Ende des Wortes erreicht wurde. Dann wird geprüft ob der Knoten in dem wir uns befinden ein Endzustand ist. Ist das der Fall, ist das Wort korrekt. Das passiert in der Klasse `RegexValidator`.

```

11     bool validate(String word) {
12         NodeId currentNode = _gvizHelper.nodes[0];
13         for (int i = 0; i < word.length; i++) {
14             final char = word[i];
15
16             final edgesFromCurrentNode = _gvizHelper.getEdgesFrom(currentNode);
17
18             bool found = false;
19             for (var edge in edgesFromCurrentNode) {
20                 final inputs = _gvizHelper.getInputsOfEdge(edge);
21                 if (inputs.contains(char)) {
22                     currentNode = NodeId(edge.rhs.target.toString());
23                     found = true;
24                     break;
25                 }
26             }
27
28             if (!found) {
29                 return false;
30             }
31         }
32
33         return _gvizHelper.isEndNode(currentNode);
34     }

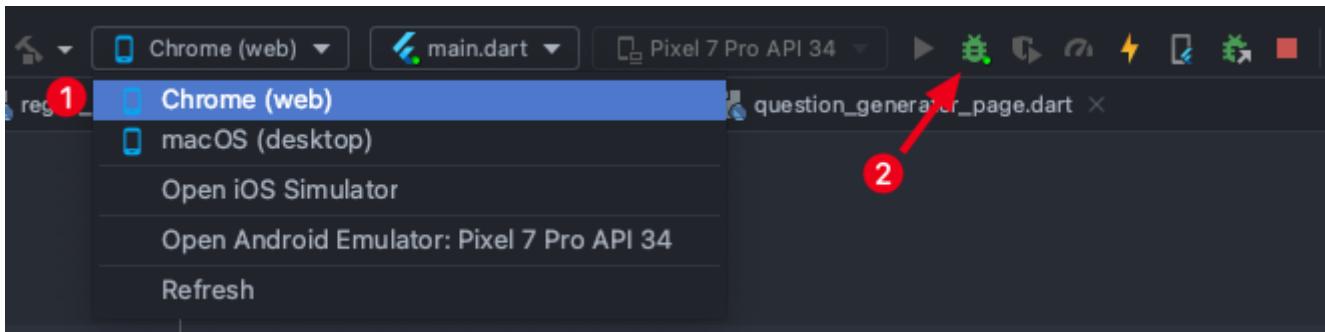
```

2.5.10. Punkteübersicht

Die erreichten Punkte werden dann auf der [SummaryPage](#) dargestellt. Dort werden die pro Aufgabe erreichten Punkte und die mögliche Gesamtpunktzahl angezeigt. Darunter wird dann die erreichte Gesamtpunktzahl über alle Aufgaben angezeigt. Über den Button 'Back to Generator' gelangt man wieder zurück zur Startseite.

2.5.11. Ausführen des Programms

Um das Programm zu starten, muss zuerst Flutter installiert werden. Eine Anleitung dazu gibt es auf der offiziellen Website von Flutter: <https://docs.flutter.dev/get-started/install>. Ist Flutter installiert, kann das Projekt mit dem Befehl `flutter run -d chrome` ausgeführt werden. Zum entwickeln empfiehlt sich Android Studio alternativ gibt es auch ein Flutter-Plugin für Visual Studio Code. In Android Studio kann das Projekt auch direkt über den Debugger gestartet werden:



2.6. Ergebnis und Ausblick

Das Programm hat eine schlichte und übersichtliche Oberfläche und ermöglicht dem Nutzer, ein Verständnis für Endliche Automaten zu entwickeln. Es generiert zufällige Aufgaben, ohne auf einen Datensatz an Beispielaufgaben zurückzugreifen. Das Programm legt den Grundstein für weitere Aufgaben, die sich mit Endlichen Automaten beschäftigen. Weitere Möglichkeiten für Aufgaben wären es, dem Nutzer einen Editor zur Verfügung zu stellen, mit dem er den Automaten anhand der Zustandsübergangstabelle oder des Regex selbst erstellen kann. Beim Generieren könnten noch weitere Einstellungen vorgenommen werden, wie zum Beispiel zwischen minimalen und nicht minimalen Automaten zu wählen und NFAs zu generieren.

Außerdem müsste die Validierung des Regex verbessert werden. Eine mögliche Lösung dafür wäre es den Automaten aus dem angegebenen Regex zu generieren, zu minimieren und mit dem minimalen ursprünglichen Automaten verglichen.

Flutter als Alternative zu wählen, hat es mir ermöglicht, mich auf die Generierung der Aufgaben zu konzentrieren und die UI so zu gestalten, wie ich es mir vorgestellt habe. Die Implementierung in Flutter war sehr angenehm und hat mir viel Spaß gemacht. Ich konnte mein Vorwissen gut anwenden und habe auch viel neues, wie zum Beispiel das einbinden einer JavaScript-Library, gelernt. Auch denke ich, dass die überführung des Projektes in ALADIN durch die Ähnlichkeiten von Dart und TypeScript nicht allzu schwer sein wird. Es würde auf jeden Fall das einbinden der Noam-Library erleichtern, da diese auch als npm-Package verfügbar ist. :doctype: book :lang: DE :hyphens:

3. Aufgabentyp - Clusteranalyse

3.1. Aufgabenbeschreibung

Die Aufgabe für das Projektseminar war es, ein Programm zur automatisierten Generierung von Aufgaben für die Hierarchische Clusteranalyse zu erstellen. Das Ziel bestand darin, dass die Studierenden die Durchführung einer hierarchischen Clusteranalyse üben können und Lehrende gleichzeitig ein Tool zu Verfügung gestellt bekommen, mit welchem Sie beliebig viele Aufgaben für die Studierenden erstellen können. Das Programm soll sowohl die Aufgabenstellung für die hierarchische Clusteranalyse, als auch ein Beispieldatenset generieren. Darüber hinaus soll die Ähnlichkeitsmetrik und das Linkkriterium wählbar sein. Aus Output soll ein Dendrogramm ausgeben werden.

Folgend ein Beispiel für eine mögliche Aufgabenstellung:

In dieser Aufgabe sollen Sie zeigen, dass Sie die hierarchische Clusteranalyse verstanden haben und diese anwenden können. Sie erhalten eine Sammlung von Datenpunkten, für die Sie die Analyse durchführen sollen.

Tabelle 1. Table1

Name Datenpunkt	Position X-Achse	Position Y-Achse
A	2	3
B	5	2
C	5	3
D	1	4
E	4	5

1. Zeichnen Sie im ersten Schritt die Datenpunkte in ein Koordinatensystem als visuelle Hilfe ein. 2. Berechnen Sie nun im nächsten Schritt die einzelnen Distanzmatrizen bis zu einer 2×2 Matrix und geben Sie Ihre Lösung an. Nutzen Sie für die Berechnung der Distanzen zwischen den einzelnen Datenpunkten die Euklidische Distanz als Berechnungsmethode sowie als Linkkriterium Single-Linkage

3.2. Anforderung

Aus diesen Gegebenheiten ergeben sich folgende Anforderungen an die Software:

- * Zufällige Generierung von Datensets
- * Umsetzung mindestens einer Methode zur Berechnung der Distanz
- * Umsetzung mindestens eines Linkkriteriums
- * Generierung der Aufgabe sowie der Lösung (Schrittweise inkl. aller Zwischenschritte)
- * Verwendung realistischer Wertebereiche
- * Ausgabe eines Dendrogramms

3.3. Planung

Um die zu Verfügung stehende Zeit effektiv zu nutzen, wurden vorher ein kurzer Projektplan mit mehreren Meilensteinen erstellt. Der erste Meilenstein sah die Einarbeitung in das Thema der Hierarchischen Clusteranalyse vor. Der zweite Meilenstein war die Erstellung eines Proof of Concept in

Form eines Python Scripts zur digitalen Lösung einer Hierarchischen Clusteranalyse. Der nächste Meilenstein war dann die Umsetzung des Frontends in CARPET sowie die Anbindung an ALADIN. Darauf folgend war geplant das Python Script aus dem PoC durch ALADIN aufrufen zu lassen und im Anschluss alle bestehenden Funktionalitäten zu verknüpfen. Anschließend sollte weitere Funktionalität in mehreren Iterationen hinzugefügt und getestet werden, bis das Projektseminar endet. Die abgeschlossenen Meilensteine wurden dabei in einem wöchentlichen Meeting präsentiert und dabei kurz bestehende Herausforderungen vorgestellt und sich Tipps zu deren Lösung eingeholt, um eine möglichst effiziente Entwicklung zu gewährleisten.

3.4. Technische Umsetzung

3.4.1. Entwicklung des Proof of Concept

Um sicherzustellen, dass die gestellte Aufgabe mittels der ALADIN- & CARPET Frameworks gelöst werden kann, wurde im ersten Schritt ein Proof of Concept (nachfolgend mit PoC abgekürzt) erstellt, um die Machbarkeit zu überprüfen. Der PoC bestand dabei aus einem Python Script, welche über ein CLI bedient wird. Die Anforderungen an den PoC waren vor allem die Abbildung des Algorithmus zum Schrittweisen lösen der hierarchischen Clusteranalyse. Im ersten PoC wurde sich, um bestehende Bibliotheken nicht neu zu schreiben und die zu Verfügung stehende Zeit effizient zu nutzen, die Python Bibliothek „scipy“ gewählt, welche bereits Methoden zur Durchführung der hierarchischen Clusteranalyse bietet. Nachdem ca. 75% des PoC fertig gestellt waren musste ich jedoch feststellen, dass die verbleibenden 25% nur zu realisieren wären, wenn ich die „scipy“ Bibliothek forken und stark modifizieren würde. Diese Option schloss ich jedoch aus, da absehbar war, dass in Zukunft keine Ressourcen zum maintainen des Forks zu Verfügung stehen würden. Daraufhin verwarf ich einen Großteil des PoC und entwickelt selbst ein Skript mit allen notwendigen Methoden, um die hierarchische Clusteranalyse durchzuführen und die dabei verwendeten Algorithmen zu verwenden zu können. Da bei der schriftlichen Lösung hierarchischen Clusteranalyse pro Iterationsschritt eine Distanzmatrix erstellt und dabei die Distanzen zwischen den einzelnen Datenpunkten berechnet werden, entschied ich mich dazu, diese Berechnung genauso in dem Python Script abzubilden. So war es möglich, Distanzmatrizen für alle Iterationsschritte zu erstellen. Die Python Bibliothek „scipy“ bietet im Vergleich dazu nur die Möglichkeit, ein Dendrogramm zu erstellen, aber nicht sich schrittweise diese Distanzmatrizen ausgeben zu lassen. Dies war der Hauptpunkt, weshalb ich mich gegen die Bibliothek entschied und die Funktionalität vollständig selbst entwickelte. Der zweite PoC war somit in der Lage, auf Basis eines vorher zu definierenden Datensets die hierarchische Clusteranalyse durchzuführen und die Lösung für jede Iteration und somit für jede Distanzmatrix zu berechnen. Darüber hinaus setze ich im PoC auch direkt die Generierung eines Dendrogramms auf Grundlage der zuvor durchgeföhrten Berechnungen um. Mittels des PoC war es mir somit möglich zu zeigen, dass sich die schriftliche Lösung der hierarchischen Clusteranalyse vollständig abbilden lässt. Gleichzeitig bildete der entwickelte PoC die Grundlage für die drauf folgende Entwicklung eines Programms zur Durchführung der Clusteranalyse mittels ALADIN.

3.4.2. Allgemeine Architektur

Die Software besteht einerseits aus dem ALADIN Framework, welches das Backend der Anwendung bildet, sowie dem CARPET Framework, welche ein Frontend in Form einer graphischen Weboberfläche zu Verfügung stellt.

Aufgrund des zu vor umgesetzten PoC mittels Python entschied ich mich dazu, dass das ALADIN Framework den bereits vorhanden Python Code mittels CLI aufruft, und die Daten, welche das Python Script zurückliefert, nimmt und anschließend verarbeitet. So war es mir möglich ein Großteil des Codes in Python zu schreiben und nur für den Datenaustausch zwischen dem Python Script und ALADIN das ALADIN Framework mittels Typescript zu modifizieren. Dazu entwickelte ich eine neue Komponente inkl. Aufruf und Verarbeitungsprozedere in Typescript, mit welcher es mir möglich war, dass ALADIN nun Python Scripte ausführen, Daten an das Script zu übergeben und die Daten, welche das Python Script zurück gibt, zu lesen. Die Umsetzung dieser Funktionalität sieht wie folgt aus:

```

import { exec } from "child_process";
import fs from "fs";
const path = require("path");

function runPythonScript(scriptPath: string, args: any): Promise<string> {
    try {
        return new Promise((resolve, reject) => {
            const command = `python3 ${scriptPath} ${args.join(" ")}`;
            console.log(command);
            exec(command, (error, stdout, stderr) => {
                if (error) {
                    reject(error.message);
                    return;
                }
                if (stderr) {
                    reject(stderr);
                    return;
                }
                resolve(stdout);
            });
        });
    } catch {
        console.log("Error");
    }
}

export async function clusterAnalysisMain(parameter: any) {
    let result = { foo: "bar" };
    console.log("PARAMETER:");
    console.log(parameter);
    try {
        let pythonScriptPath = "clusterAnalysis.py";
        pythonScriptPath = path.join(__dirname, pythonScriptPath);
        const argumentsToPythonScript = [...Object.values(parameter),
...parameter["nodeRange"]];
        (await runPythonScript(pythonScriptPath, argumentsToPythonScript)) as any;
        result = JSON.parse(fs.readFileSync(path.join(__dirname, "data.json"), "utf-
8"));
        console.log(result);
    } catch (error) {

```

```

        console.error("Error running Python script:", error);
    }
    return result;
}

```

Über diese Komponente wird das Python Script mit den entsprechenden Parametern, welches es aus dem Aufruf aus dem Frontend (durch CARPET) mitbekommt, aufgerufen. Das Script führt die Clusteranalyse Schritt für Schritt durch und gibt die Daten zur Abfrage der Nutzereingabe sowie die Ergebnisse zurück an das ALADIN Framework. Das ALADIN Framework wiederum sendet die Daten weiter an das Frontend, in welchem die Daten genutzt werden um die Weboberfläche zu generieren und es dem Nutzer zu ermöglichen seine Lösungen einzugeben und direkt kontrollieren zu lassen. Der Kontrollmechanismus funktioniert dabei mittels Javascript, d.h., dass die Daten zur Lösung bereits an das Frontend mitgegeben werden und während der Nutzer die Lösung eingibt keine weitere Kommunikation zwischen Frontend und Backend stattfinden muss.

3.5. Ergebnisse

Die Oberfläche, die der Nutzer vorfindet, bevor er die Aufgabe lösen kann sieht wie folgt aus:

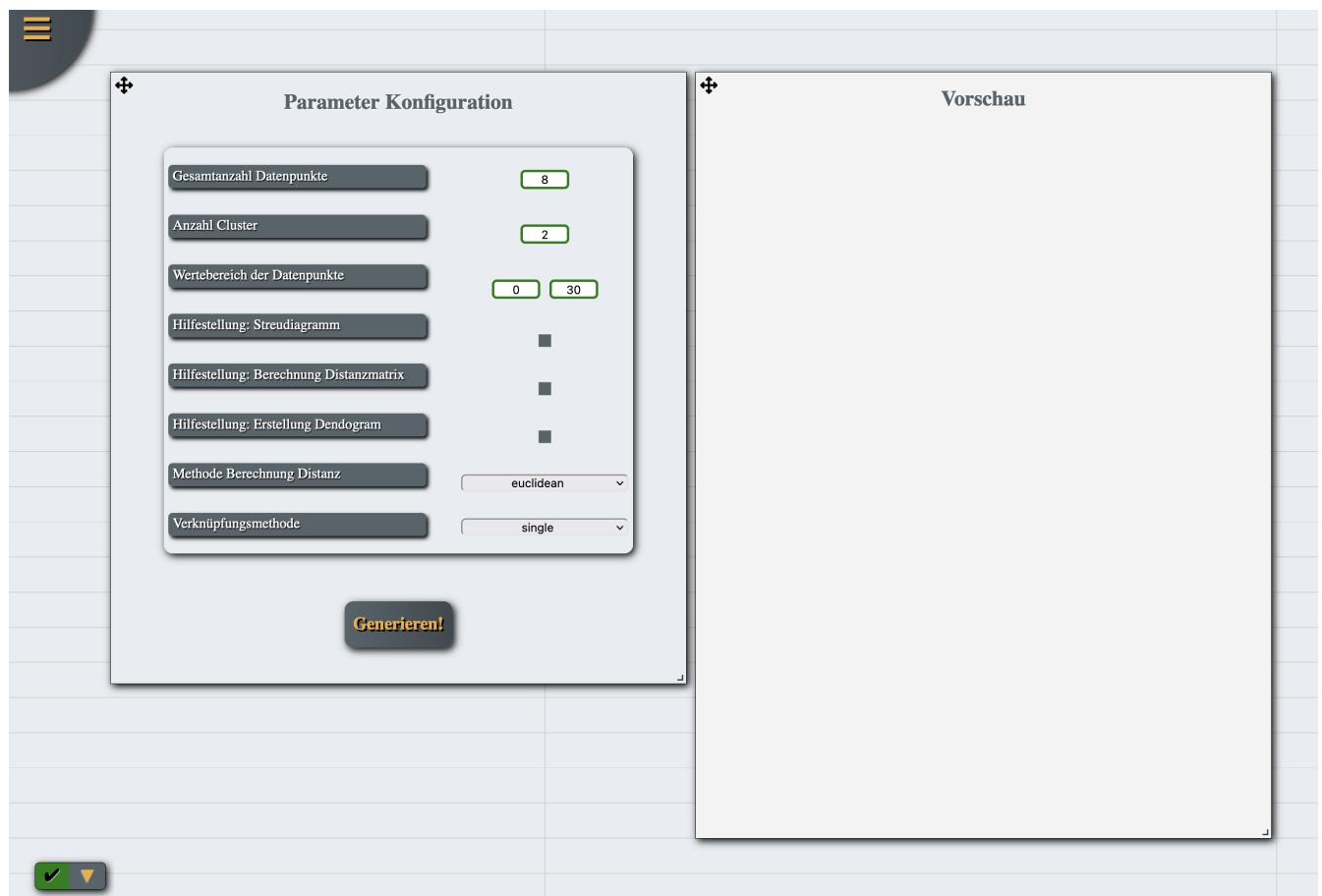


Abbildung 2. Ansicht Eingabemaske

Hier hat er die Wahl, ob er sich entsprechende Hilfen anzeigen lassen möchte oder nicht. Die Implementierung von Hilfen war nicht im ursprünglichen Scope enthalten, jedoch wurde in Abstimmung mit Herrn Paul Christ entschieden, dass es eine durchaus nützliche Funktionalität darstellt und deshalb mit implementiert. Nachdem der Nutzer die für ihn und seinen Anwendungsfall passenden Werte eingegeben bzw. ausgewählt hat, kann er auf den "Generieren" Button klicken, eine Aufgabe

generieren zu lassen. Dabei wird ihm direkt ein Streudiagramm angezeigt, so dass er sehen kann, ob die generierte Aufgabe seinen Anforderungen genügt.

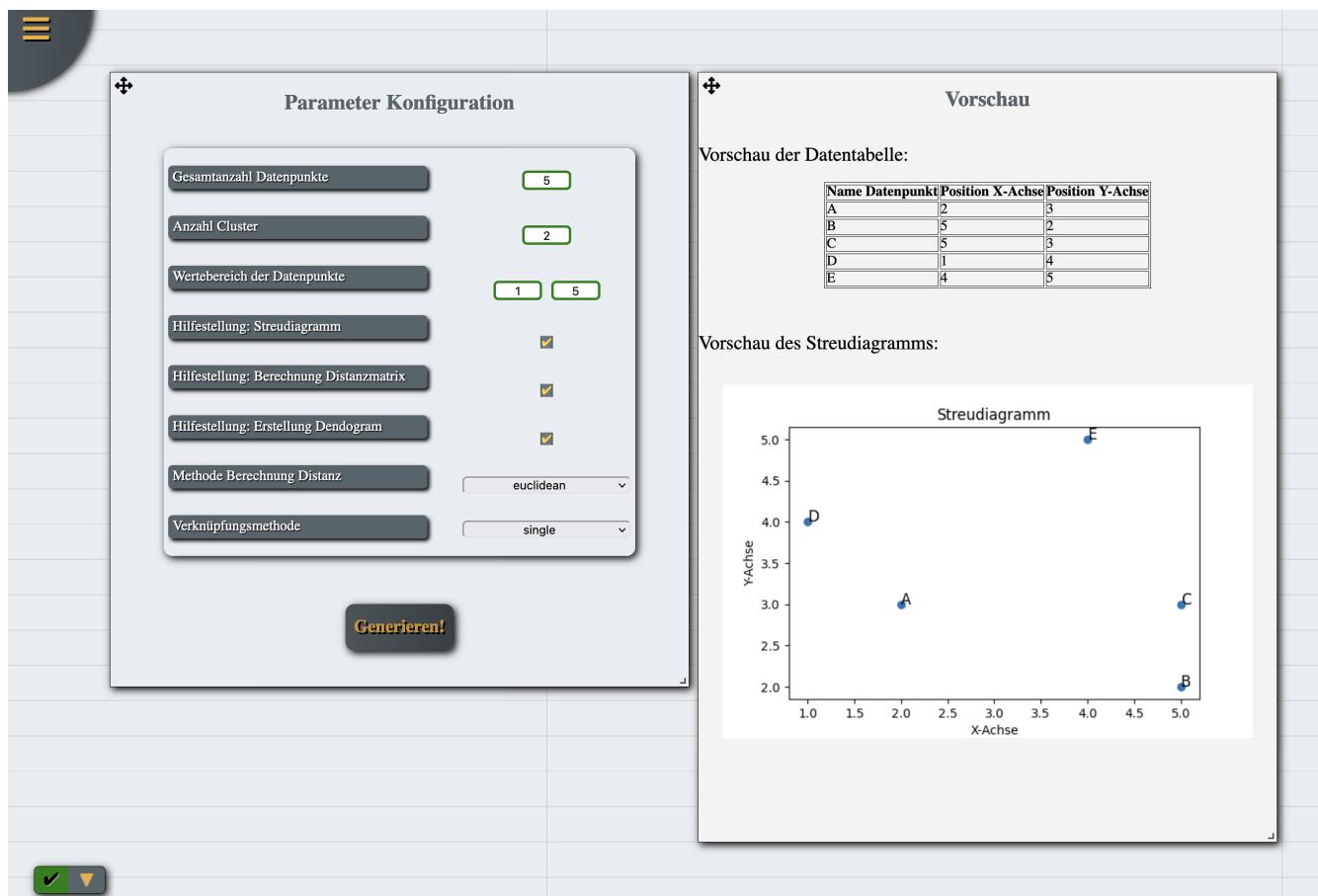


Abbildung 3. Ansicht Eingabemaske mit generierten Streudiagramm als Vorschau

Infofern dies zutrifft, kann der Nutzer die nächste Seite aufrufen oder eine neue Aufgabe generieren. Sofern der Nutzer sich entscheidet die Aufgabe zu lösen, sieht die Oberfläche wie folgt aus:

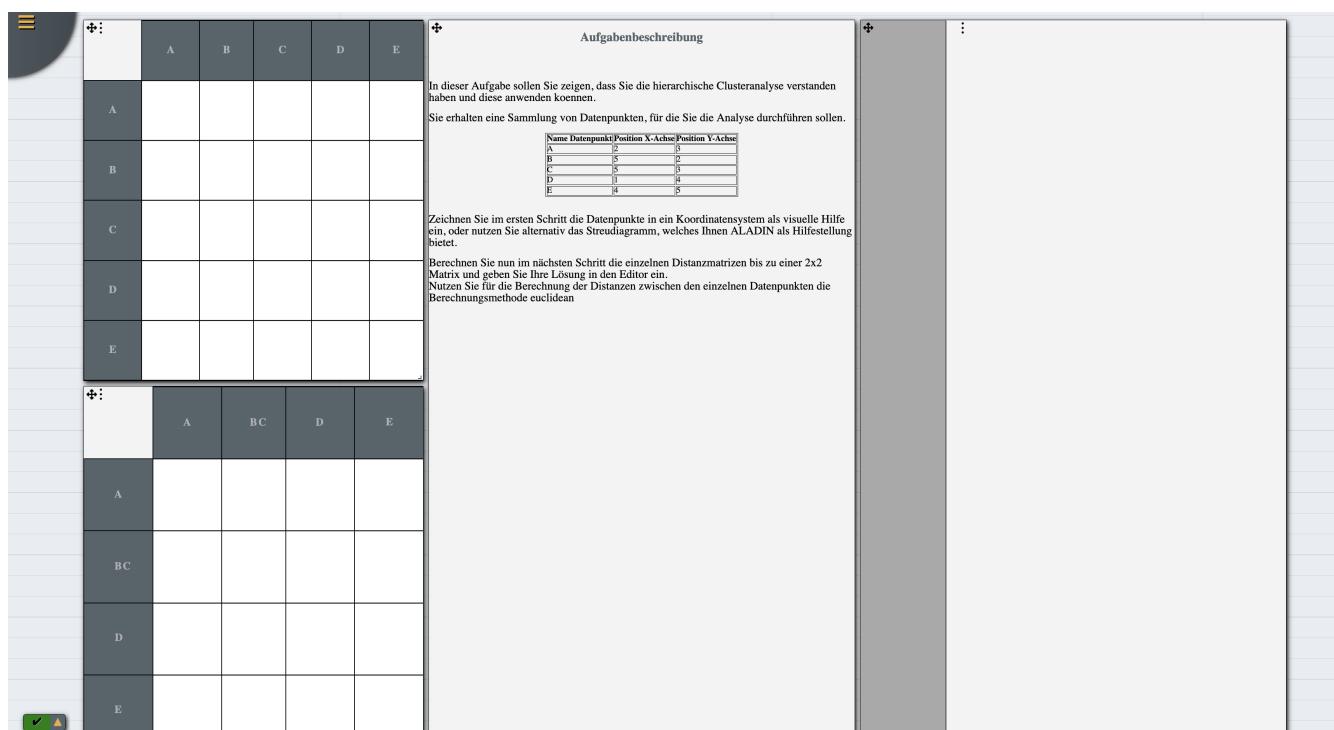


Abbildung 4. Ansicht Lösungsoberfläche

Dort kann er nun noch einmal die Aufgabenstellung lesen und die Aufgabe lösen, sowie seine Lösungen in die Eingabemaske eintragen und direkt sehen, ob seine Lösungen richtig waren oder er die Berechnung aufgrund eines Fehlers nochmal erneut durchführen muss.

	A	B	C	D	E
A					
B					
C					
D					
E					

	A	B C	D	E
A				
B C				
D				
E				

Aufgabenbeschreibung

In dieser Aufgabe sollen Sie zeigen, dass Sie die hierarchische Clusteranalyse verstanden haben und diese anwenden können.

Sie erhalten eine Sammlung von Datenpunkten, für die Sie die Analyse durchführen sollen.

Name Datenpunkt	Position X-Achse	Position Y-Achse
A	2	3
B	5	2
C	5	3
D	1	4
E	4	5

Sie haben bei der Aufgabenerstellung ausgewählt, dass Ihnen das Streudiagramm angezeigt werden soll. Das folgende Diagramm zeigt die verteilten Datenpunkte im Koordinatensystem.

Hilfe zur Distanzberechnung
Sie haben die euclidean Methode zur Berechnung der Distanz gewählt. Diese wird wie folgt berechnet:

$$d(a, b) = \sqrt{a^2 + b^2}$$

Um ein Dendrogramm aus einer Distanzmatrix zu erstellen, verwendet man zuerst einen hierarchischen Clustering-Algorithmus (wie z.B. Single-Linkage, Complete-Linkage oder Average-Linkage), um die Cluster-Hierarchie aus der Distanzmatrix zu erzeugen. Anschließend stellt man diese Hierarchie graphisch als Dendrogramm dar, wobei die Länge der Verbindungslien die Distanzen zwischen den Clustern repräsentiert.

Berechnen Sie nun im nächsten Schritt die einzelnen Distanzmatrizen bis zu einer 2×2 Matrix und geben Sie Ihre Lösung in den Editor ein.
Nutzen Sie für die Berechnung der Distanzen zwischen den einzelnen Datenpunkten die Berechnungsmethode euclidean

Abbildung 5. Ansicht Lösungsoberfläche ohne Eingabe der Distanzmatrix

Aufgabenbeschreibung

In dieser Aufgabe sollen Sie zeigen, dass Sie die hierarchische Clusteranalyse verstanden haben und diese anwenden können.

Sie erhalten eine Sammlung von Datenpunkten, für die Sie die Analyse durchführen sollen.

Name	Datenpunkt	Position X-Achse	Position Y-Achse
A		2	3
B		5	2
C		5	3
D		1	4
E		4	5

Sie haben bei der Aufgabenerstellung ausgewählt, dass Ihnen das Streudiagramm angezeigt werden soll. Das folgende Diagramm zeigt die verteilten Datenpunkte im Koordinatensystem.

Hilfe zur Distanzberechnung
Sie haben die euclidean Methode zur Berechnung der Distanz gewählt. Diese wird wie folgt berechnet:

$$d(a, b) = \sqrt{a^2 + b^2}$$

a ≡ Datenpunkt 1
b ≡ Datenpunkt 2

Um ein Dendrogramm aus einer Distanzmatrix zu erstellen, verwendet man zuerst einen hierarchischen Clustering-Algorithmus (wie z.B. Single-Linkage, Complete-Linkage oder Average-Linkage), um die Cluster-Hierarchie aus der Distanzmatrix zu erzeugen. Anschließend stellt man diese Hierarchie graphisch als Dendrogramm dar, wobei die Länge der Verbindungslien die Distanzen zwischen den Clustern repräsentiert.

Berechnen Sie nun im nächsten Schritt die einzelnen Distanzmatrizen bis zu einer 2×2 Matrix und geben Sie Ihre Lösung in den Editor ein.
Nutzen Sie für die Berechnung der Distanzen zwischen den einzelnen Datenpunkten die Berechnungsmethode euclidean

Abbildung 6. Ansicht Lösungsoberfläche mit Eingabe der Distanzmatrix

Insofern die eingegebenen Lösungen richtig sind, werden diese grün hinterlegt.

Die gesetzten Ziele wurden soweit vollständig erreicht. Es lässt sich somit sagen, dass das Programm zur Generierung von Aufgaben zur hierarchischen Clusteranalyse funktioniert und bereit für den Praxiseinsatz ist. Nachfolgend ein Vergleich der IST- und SOLL Stände:

Tabelle 2. Table2

Anforderung (SOLL)	Aktueller Stand (IST)
Zufällige Generierung von Datensets	gelöst (mittels in Python umgesetzter Generierungsfunktion)
Umsetzung mindestens einer Methode zur Berechnung der Distanz	gelöst (mittels in Python umgesetzter Berechnungsfunktion), es wurden in Summe sogar zwei Berechnungsmethoden umgesetzt
Umsetzung mindestens eines Linkkriteriums	gelöst (mittels in Python umgesetzter Berechnungsfunktion)

Anforderung (SOLL)	Aktueller Stand (IST)
Generierung der Aufgabe sowie der Lösung (Schrittweise inkl. aller Zwischenrésultate)	gelöst (mittels in Python umgesetzter Berechnungsfunktion)
Verwendung realistischer Wertebereiche	gelöst (mittels in Eingabevalidierung in CARPET sowie nochmals in Python)
Ausgabe eines Dendrogramms	gelöst (aktuell mittels scipy Bibliothek, Ausgabe eines Bildes mit einem Dendrogramm möglich, im nächsten Schritt muss das zeichnen und validieren des Dendrogramms in CARPET umgesetzt werden)

Die Umsetzung in ALADIN und CARPET stellte den umfangreichsten Teil und zeitintensivsten Teil des Projektseminars dar. Vor allem das Fehlen einer Dokumentation stellte eine große Herausforderung dar und resultierte darin, dass viel Zeit in das Testen, Ausprobieren und Verstehen des ALADIN & CARPET Frameworks investiert werden mussten. Oft stieß ich auf für mich unlösbare Herausforderungen, welche ich nur dank der Unterstützung von Herrn Paul Christ lösen konnte. Rückblickend betrachtet wäre es für mich deutlich effektiver gewesen, mithilfe einer Dokumentation schneller in die Frameworks einzutuchen und so die dadurch verbleibende Zeit in den Ausbau des Tool zu investieren.

3.6. Ausblick

Eine Möglichkeit das aktuelle Programm zu erweitern wäre, die bereits bestehende Komponente in ALADIN zum Erstellen von Diagrammen der DOT Notation zu modifizieren, so dass es Nutzern ermöglicht wird Dendrogramme per Drag&Drop zu erstellen und anschließend mit der Musterlösung zu vergleichen. So wäre es möglich den Prozess der schriftlichen Lösung einer Durchführung einer hierarchischen Clusteranalyse vollständig digital abzubilden und direkt kontrollieren zu lassen. Aufgrund der begrenzten Zeit war es mir nicht möglich, dieses Feature zu implementieren. Es ist jedoch meiner Meinung nach mit dem aktuellen Stand des Programms eine sehr gute Grundlage für die Erweiterung von genau solchen Features geschaffen wurden. Vor allem im Hinblick auf die eigentliche Aufgabenstellung, wurden die Anforderungen übererfüllt und bieten nun eine sehr gute Grundlage für die Implementierung weiterer Features. Ein weiteres dieser Features könnte beispielsweise auch die Umsetzung weiterer Ähnlichkeitsmetriken und Linkkriterien sein. Die in der Lehre auf häufigsten verwendeten wurde bereits umgesetzt, jedoch kann durch die Erweiterung um weitere Metriken es ambitionierten Studenten ermöglicht werden, auch diese zu üben und zu verinnerlichen. :doctype: book :lang: DE :hyphens:

4. Aufgabentyp - Entscheidungsbaum

4.1. Aufgabenbeschreibung

Im Rahmen des Projektseminars wurde ein Aufgabentyp entwickelt, der Studierenden die Berechnung und Erstellung eines Entscheidungsbaums ermöglicht. Die Grundidee besteht darin, dass Studierende anhand eines vorab generierten Datensatzes und eines festgelegten Algorithmus interaktiv einen Entscheidungsbaum für diesen spezifischen Datensatz aufbauen können. Dabei soll es möglich sein, Beschriftungen sowie berechnete Werte für die einzelnen Knoten und Kanten hinzuzufügen.

Beispiel 1. Aufgabenstellung für den Studierenden

"Erstellen Sie anhand des generierten Datensatzes einen Entscheidungsbaum. Nutzen Sie für die Berechnung den ID3-Algorithmus und für den Aufbau der Lösung den interaktiven Entscheidungsbaum. Tragen Sie die richtigen Beschriftungen und Werte in die Knoten und an den Kanten ein."

Die Aufgabe zielt darauf ab, den Studierenden die Möglichkeit zu bieten, die Berechnung eines Entscheidungsbaums mithilfe eines Algorithmus zu erlernen und zu vertiefen. Die Integration interaktiver Elemente ist darauf ausgerichtet, das Verständnis für den Algorithmus, als auch für den Aufbau eines Entscheidungsbaums zu erleichtern und einen praktischen Lernansatz zu fördern.

4.2. Anforderungen

Für den Aufgabentypen ergeben sich folgende Anforderungen:

1. Zufällige Datensatzgenerierung

Die Generierung eines Datensatzes erfolgt zufällig, wobei sich nur die Berechnungswerte ändern dürfen. Die Struktur des Baumes soll erhalten bleiben.

2. Datensatzgenerierung basierend auf vorhandenem Baum

Die Generierung des Datensatzes erfolgt anhand eines bereits vorgegebenen Entscheidungsbaums.

3. Interaktiver Aufbau des Entscheidungsbaums

Der Entscheidungsbaum sollte interaktiv aufgebaut werden können, wodurch beliebig viele Knoten und Kanten in unterschiedlichen Tiefen hinzugefügt oder entfernt werden können.

4. Überprüfung des eingegebenen Entscheidungsbaums

Der vom Studierenden aufgebauten Entscheidungsbaum wird auf Richtigkeit sowohl hinsichtlich der eingegebenen Werte als auch der Struktur überprüft.

5. Flexible Positionierung der Werte

Bei der Prüfung des Entscheidungsbaums wird berücksichtigt, dass die Werte der Knoten und Kanten auf der Baumebene an unterschiedlichen Stellen eingetragen werden können.

6. Berücksichtigung von Mehrfachlösungen

Es wird berücksichtigt, dass bei der Überprüfung des Entscheidungsbaums Mehrfachlösungen

möglich sind. Das bedeutet, je nach Berechnung für einen Knoten im Entscheidungsbaum können unterschiedliche Merkmale als korrekte Lösungsmöglichkeit gewertet werden.

7. Feedback nach erfolgreicher Lösung

Nach erfolgreicher Lösung der Aufgabe erhält der Studierende eine Benachrichtigung, dass die Aufgabe erfolgreich gelöst wurde.

8. Generierung neuer Aufgaben

Nach Abschluss der Aufgabe sollte die Möglichkeit bestehen, eine neue Aufgabe mit einem neuen Datensatz zu generieren.

4.3. Planung

4.3.1. Wöchentliche Fortschrittsbesprechung und Umsetzungsplan

Im Rahmen des Projektseminars wurde ein wöchentliches Meeting etabliert. Die Besprechungen fanden einmal in der Woche statt und wurden von Herrn Professor Munkelt und Herrn Christ geleitet. Das Meeting diente für die Evaluierung der gesetzten und erreichten Ziele, die Vorstellung und Diskussion von Ideen und Vorschlägen zur Umsetzung seines jeweiligen Aufgabentyps sowie für Diskussionen über auftretende Probleme und Unklarheiten.

Um die Umsetzung der Aufgaben und Ziele zu gewährleisten, wurden feste Zeiten für die Bearbeitung eingeplant. Zweimal pro Woche waren jeweils mindestens vier Stunden für die Arbeit an den Aufgaben und Zielen der Woche reserviert. Dies diente dazu, um sicherzustellen, dass das Projekt innerhalb der definierten Zeitvorgabe erfolgreich abgeschlossen werden konnte.

4.3.2. Eigenständige Berechnung einer ID3-Algorithmus-Aufgabe

Zu Beginn des Projekts stand die Auseinandersetzung mit dem Aufgabentypen zur Berechnung eines Entscheidungsbaums im Vordergrund. Ein entscheidender Schritt dabei war, die Aufgabe zunächst eigenständig zu lösen. Ziel war es, das persönliche Verständnis für den Aufbau und die wesentlichen Aspekte dieser Art von Aufgabe zu vertiefen. Dies beinhaltete die Analyse, welche Elemente bei der Konstruktion einer solchen Aufgabe von Bedeutung sind, welche Schritte in der Berechnung notwendig sind, welche Lösungswege existieren und wie das Endresultat, der Aufbau des Entscheidungsbaums, gestaltet sein sollte.

Im wöchentlichen Meeting wurde in Absprache festgehalten, dass der zu entwickelnde Aufgabentyp auf dem ID3-Algorithmus basieren sollte. Dies erforderte zunächst, das Verständnis für die Funktionsweise und Anwendung des ID3-Algorithmus zu erlangen. Weiter wurden Beispielaufgaben durchgerechnet, wobei verschiedene Problemstellungen berücksichtigt wurden, wie zum Beispiel die Berechnung von Aufgaben unter dem Gesichtspunkt von Mehrfachlösungen.

Das selbstständige Lösen der Aufgabe diente somit als Ausgangspunkt für die Ideensammlung für die Umsetzung und Entwicklung des interaktiven Aufgabentyps.

ANMERKUNG

Für die Recherche des ID3-Algorithmus und die Auswahl von Beispielaufgaben zur eigenständigen Bearbeitung wurden die folgenden Quellen herangezogen:

1. https://www.in.th-nuernberg.de/professors/Holl/Personal/DataMining_Basis.pdf

2. Literatur: Frochte, J. (2019). Maschinelles Lernen: Grundlagen und Algorithmen in Python. Carl Hanser Verlag.

3. Unterlagen aus dem 4. Semester: Toll, A. (2023). 7. Automatische Generierung von Entscheidungsbäumen. Modul Business Intelligence. Hochschule für Technik und Wirtschaft Dresden.

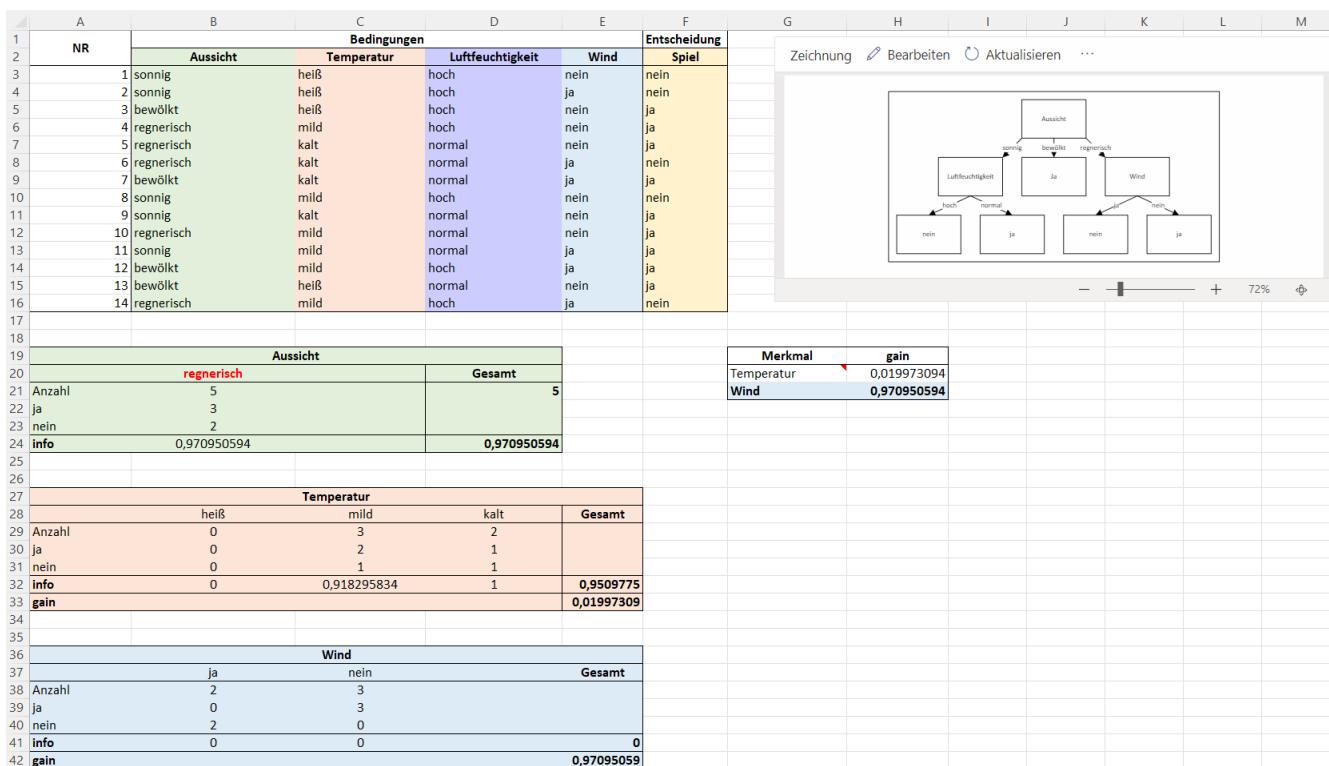


Abbildung 7. Beispiel Berechnungsaufgabe eines Entscheidungsbaums in Excel

4.3.3. Konzeption des interaktiven Aufgabentyps

Nachdem ein grundlegendes Verständnis für die Berechnung von Entscheidungsbäumen erworben war, war der nächste Schritt im Projekt die Entwicklung einer Aufgabenstellung für die Berechnung eines Entscheidungsbaums. Das Ziel bestand darin, wie eine Aufgabenstellung gestaltet werden könnte, um Studierende den interaktiven Aufbau von Entscheidungsbäumen zu ermöglichen. Dieser Schritt beinhaltete die Erstellung von Wireframes, um die Konzeptideen visuell zu erfassen.

Die erste konzeptionelle Idee sah vor, die Aufgabenstellung durch eine Art Menüführung zu strukturieren. Hierbei sollte der Studierende zu Beginn die relevanten Merkmale und deren Ausprägungen eingeben, woraus anschließend ein zufälliger Datensatz generiert werden würde. Basierend auf diesem Datensatz würde der Entscheidungsbaum schrittweise berechnet werden. Die Umsetzung erfolgte durch die schrittweise Anzeige eines Menüs für jede Knoten des zu konstruierenden Baums. In diesem Menü konnte der Studierende die Bezeichnungen der Merkmale und die berechneten Werte für den Informationsgewinn eingeben. Dabei wurde für jeden angezeigten Knoten des Baums das relevante Merkmal ausgewählt. Der iterative Prozess sollte es dem Studierenden ermöglichen, den Entscheidungsbaum schrittweise aufzubauen.

← → C X

ALADIN

Aufgabe: Entscheidungsbaum erstellen

Merkmale hinzufügen...

Merkmal1 Merkmalbezeichnung Ausprägung1 Ausprägung2 +

Merkmal2 Merkmalbezeichnung Ausprägung1 Ausprägung2 +

Zielmerkmal Merkmalbezeichnung Ausprägung1 Ausprägung2 +

Datensatz generieren i

Datensatz hochladen

Starten

Abbildung 8. Wireframe: Erster Entwurf des Startmenüs für die Aufgabenstellung

← → C X

ALADIN

Aufgabe: Entscheidungsbaum erstellen

Auswertung Wurzelknoten

Merkmal1	Aussicht	0.24874	✓
Merkmal2	Temperatur	0.02922	✓
Merkmal3	Luftfeuchtigkeit	0.15183	✓
Knoten	Aussicht	▼	

Aussicht	Temperatur	Luftfeucht.
<input checked="" type="checkbox"/> sonnig	heiß	hoch
<input checked="" type="checkbox"/> bewölkt	kalt	normal
<input checked="" type="checkbox"/> regnerisch	mild	hoch

i Alternative Lösung anzeigen

Weiter

Abbildung 9. Wireframe: Erster Entwurf für die Auswertung der eingetragenen Werte des Wurzelknotens

Die Weiterentwicklung des Aufgabentyps wurde durch eine Idee im Rahmen der wöchentlichen

Besprechung vorangetrieben, dass der Entscheidungsbaum interaktiv aufgebaut werden soll. Dies bedeutet, dass Studierende einen generierten Datensatz zur Berechnung angezeigt bekommen, um die berechnete Lösung in einer Umgebung aktiv als Entscheidungsbaum aufbauen zu können. Dabei soll zu Beginn kein vorgefertigter Baum vorhanden sein. Studierende sollen beliebig viele Knoten hinzufügen, entfernen und alle relevanten Werte wie die Bezeichnung des Merkmals bzw. die Klasse für Blattknoten und berechnete Informationsgewinne eigenständig eintragen können. Die Ausprägungen der Merkmale und die berechneten Entropien können entlang der Kanten des Baumes ergänzt werden. Nachdem der Studierende den Entscheidungsbaum vollständig aufgebaut und alle Werte eingetragen hat, kann er auf "Auswerten" klicken. Dabei werden die Stellen des Entscheidungsbaums angezeigt, die die korrekten Werte enthalten, sowie jene, die falsch sind. Dieser Prozess kann wiederholt werden, bis der Studierenden den Entscheidungsbaum vollständig und korrekt aufgebaut hat. Sobald die Aufgabe gelöst wurde, erscheint eine Meldung für den Studierenden und die Möglichkeit, eine neue Aufgabe zu berechnen. Eine weitere Idee bestand darin, den Studierenden unterstützende Hinweise anzubieten. Sollte beispielsweise Unklarheiten über die Formeln für die Berechnung des Informationsgewinns oder der Entropie bestehen, könnte der Studierende einen Hinweis-Button aktivieren. Nach einem Klick würden dann entsprechende Hinweise zur Formelberechnung angezeigt, um bei der Berechnung des Entscheidungsbaums zu unterstützen.

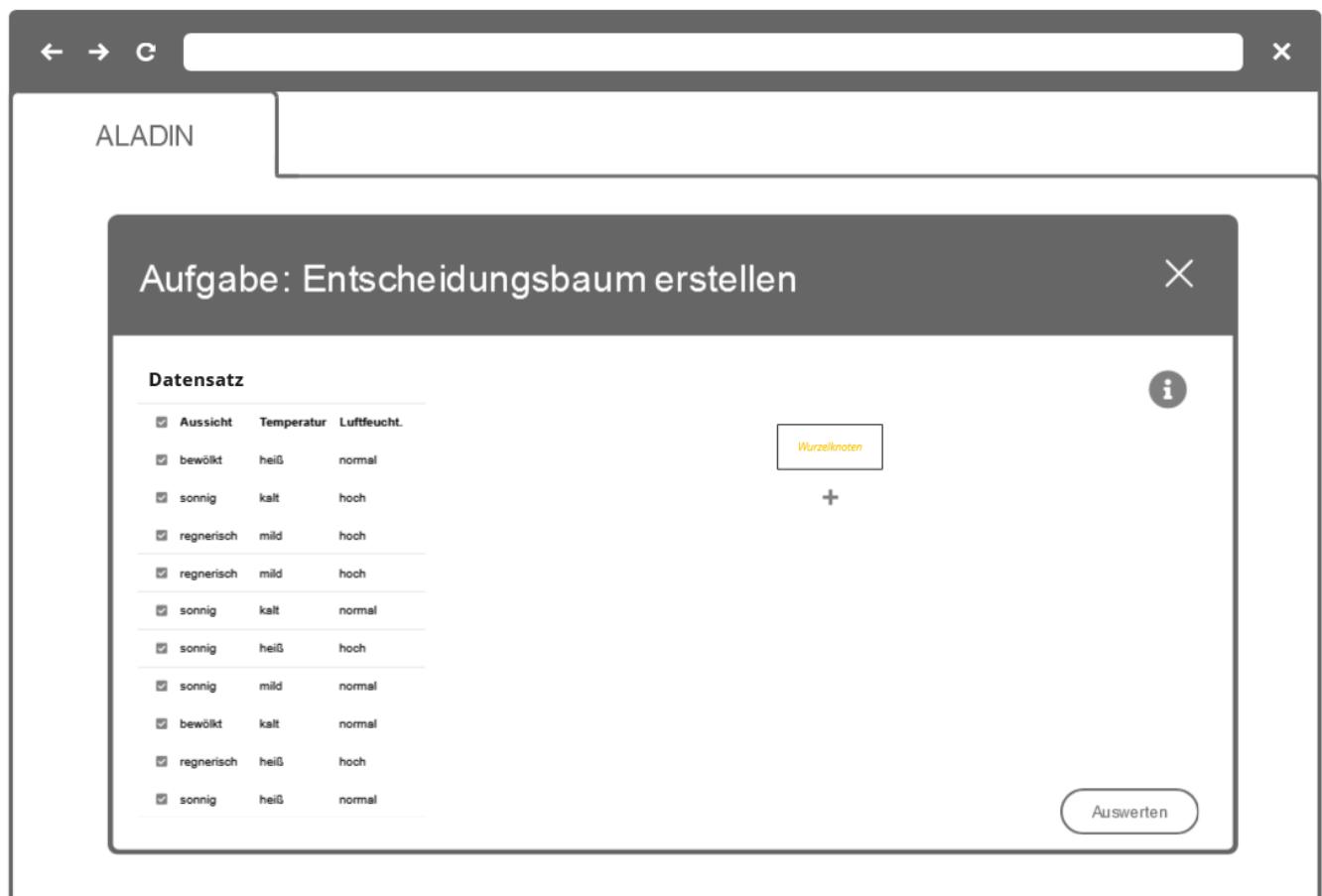


Abbildung 10. Wireframe: Finaler Entwurf der Startansicht mit dem Wurzelknoten des zu konstruierenden Entscheidungsbaums

← → C X

ALADIN

Aufgabe: Entscheidungsbaum erstellen

Datensatz

	Aussicht	Temperatur	Luftfeucht.
1	bewölkt	heiß	normal
2	sonnig	kalt	hoch
3	regnerisch	mild	hoch
4	regnerisch	mild	hoch
5	sonnig	kalt	normal
6	sonnig	heiß	hoch
7	sonnig	mild	normal
8	bewölkt	kalt	normal
9	regnerisch	heiß	hoch
10	sonnig	heiß	normal

```
graph TD; A[Aussicht] -- "Ausprägung eingeben  
Entropie eingeben" --> B[Luftfeuchtigkeit]; A -- "Ausprägung eingeben  
Entropie eingeben" --> C[Merkmale eingeben]; A -- "Ausprägung eingeben  
Entropie eingeben" --> D[Merkmale eingeben]; B -- "Ausprägung eingeben  
Entropie eingeben" --> E[Merkmale eingeben]; B -- "Ausprägung eingeben  
Entropie eingeben" --> F[ja]; C -- "Ausprägung eingeben  
Entropie eingeben" --> G[Merkmale eingeben]; C -- "Ausprägung eingeben  
Entropie eingeben" --> H[Merkmale eingeben];
```

The decision tree starts with 'Aussicht'. It branches into three paths: 'Ausprägung eingeben' and 'Entropie eingeben'. The first path leads to 'Luftfeuchtigkeit', which further branches into 'hoch' (high) and 'normal'. 'hoch' leads to 'Merkmale eingeben' and 'ja'. 'normal' leads to 'Merkmale eingeben'. The second path from 'Aussicht' leads to 'Merkmale eingeben'. The third path from 'Aussicht' leads to another 'Merkmale eingeben'. Each 'Merkmale eingeben' node has two sub-nodes: 'Ausprägung eingeben' and 'Entropie eingeben'. A large orange box highlights the path 'Aussicht -> Luftfeuchtigkeit -> hoch -> ja'.

Auswerten

Abbildung 11. Wireframe: Finaler Entwurf für den Aufbau des Entscheidungsbaums und das Einfügen der Werte

ALADIN

Aufgabe: Entscheidungsbaum erstellen

Datensatz

<input checked="" type="checkbox"/> Attribut1	Attribut2	Zielattribut
<input checked="" type="checkbox"/> abc	123	ja
<input checked="" type="checkbox"/> def	123	nein
<input checked="" type="checkbox"/> abc	456	nein
<input checked="" type="checkbox"/> abc	789	ja
<input checked="" type="checkbox"/> abc	789	ja
<input checked="" type="checkbox"/> abc	789	nein
<input checked="" type="checkbox"/> def	789	nein
<input checked="" type="checkbox"/> ghi	123	ja
<input checked="" type="checkbox"/> ghi	456	nein
<input checked="" type="checkbox"/> def	123	ja

```
graph TD; A[Aussicht] -- "sonnig  
0.97095" --> B[Luftfeuchtigkeit]; A -- "Ausprägung eingeben  
Entropie eingeben" --> C[Merkmal eingeben]; A -- "Ausprägung eingeben  
Entropie eingeben" --> D[Merkmal eingeben]; B -- "hoch  
Entropie eingeben" --> E[Merkmal eingeben]; B -- "Ausprägung eingeben  
0.0" --> F[ja]; D -- "Ausprägung eingeben  
Entropie eingeben" --> G[Merkmal eingeben]; D -- "Ausprägung eingeben  
Entropie eingeben" --> H[Merkmal eingeben];
```

Auswerten

Abbildung 12. Wireframe: Finaler Entwurf für die Auswertung der Eingaben im Entscheidungsbaum

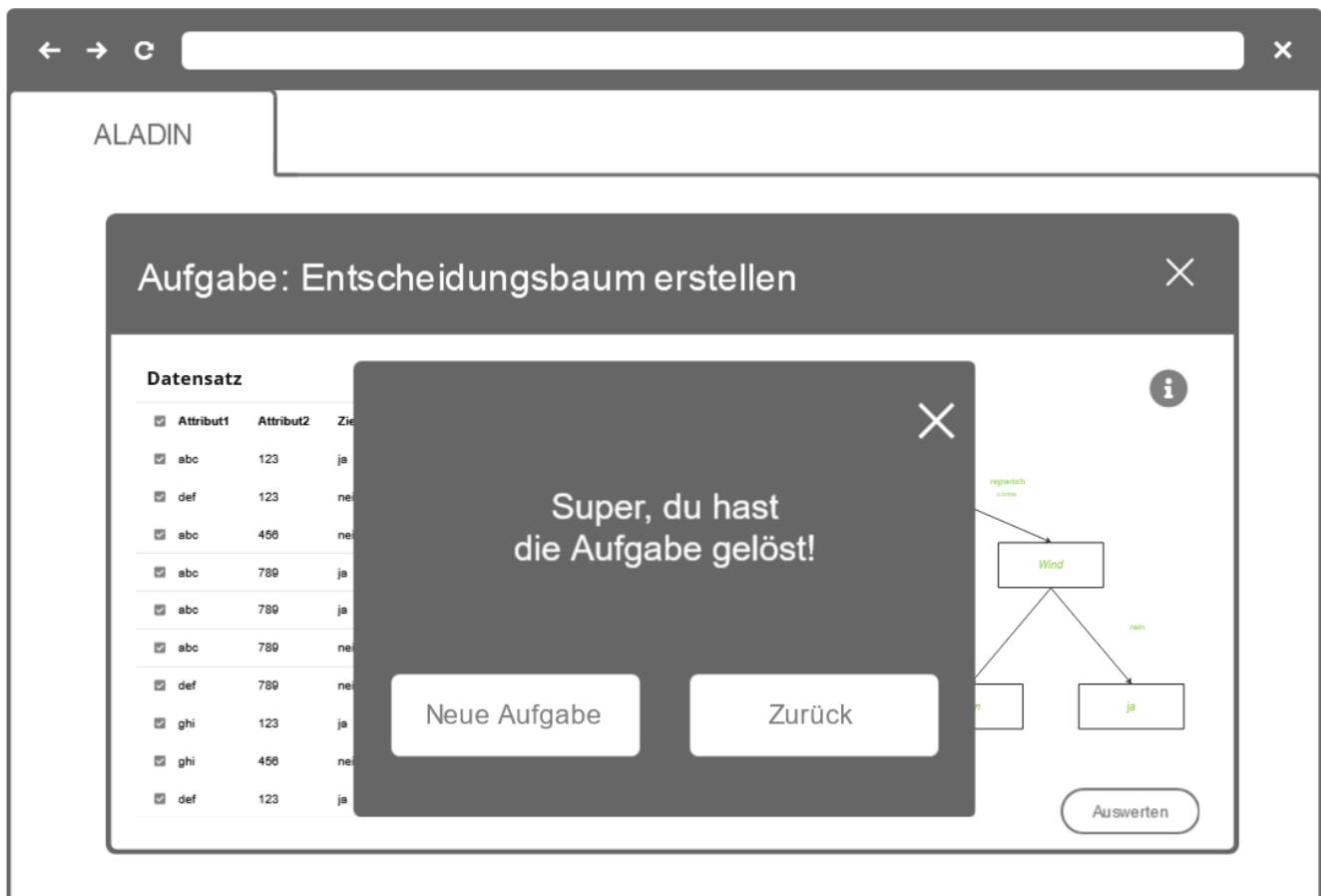


Abbildung 13. Wireframe: Finaler Entwurf für die Anzeige der Meldung über die erfolgreiche Lösung der Aufgabe

4.3.4. Entwicklung und Umsetzung des Prototyps

Nachdem der Entwurf für den Aufgabentypen festgelegt wurde, wurde mit der Umsetzung des Prototyps begonnen. Der Fokus lag zunächst auf der Implementierung des ID3-Algorithmus in der Programmiersprache Python mithilfe der Entwicklungsumgebung Jupyter Notebook. Da zu diesem Zeitpunkt noch keine Erfahrung mit Python und Jupyter Notebook vorhanden war, erfolgte zunächst eine Einarbeitung in diese Technologien. Die Entscheidung für Python fiel aufgrund seiner Eignung, komplexe Aufgaben wie die Berechnung von Entscheidungsbäumen einfach zu programmieren. Darüber hinaus wurde in den wöchentlichen Meetings hervorgehoben, dass Python-Programme problemlos in ALADIN integriert werden können, was die Wahl von Python weiter unterstützt.

Anfänglich wurden verschiedene Beispielcodes aus unterschiedlichen Quellen und Bibliotheken für den ID3-Algorithmus ausprobiert und getestet. Im Verlauf der Weiterentwicklung des Algorithmus wurde jedoch deutlich, dass es herausfordernd war, auf die berechneten Werte wie Informationsgewinn und Entropie zuzugreifen, da diese Aspekte in den Beispielcodes und Bibliotheken oft nicht ausreichend berücksichtigt wurden. Infolgedessen wurde der Algorithmus selbst entwickelt, um auf die berechneten Werte sowie die Struktur des Baums zugreifen zu können.

Bei der Entwicklung des eigenen Algorithmus wurde darauf geachtet, dass die berechneten Werte des Entscheidungsbäume gespeichert werden können. Gleichzeitig wurde versucht, die Struktur des Entscheidungsbäume zu speichern, um bei der Überprüfung des eingegebenen Entscheidungsbäume des Studierenden auf die korrekten Werte und Strukturen zugreifen zu können. Dazu wurden auch erste grafische Versuche mit der Bibliothek PyDotPlus unternommen, um Implementie-

rungsmöglichkeiten für die Darstellung eines Entscheidungsbaums zu skizzieren. Diese Versuche beinhalteten die Verwendung verschiedener Datensätze, um sicherzustellen und zu testen, dass der Algorithmus bei unterschiedlichen Datensätzen funktioniert, insbesondere bei solchen, die Mehrfachlösungen ermöglichen.

Tabelle 3. Übersicht über die verwendeten Technologien für die Entwicklung des Algorithmus

Technologie	Erläuterung
Jupyter Notebook	Wurde als Entwicklungsumgebung verwendet, für das Ausführen und Testen des geschriebenen Algorithmus
Python	Wurde als Programmiersprache verwendet, für das Schreiben des Algorithmus als Code
Pandas	Verwendete Bibliothek, um die Daten in tabellarischer Form (mit DataFrames) zu organisieren und die Verwendung von Funktionen für unterschiedliche Operationen der Datenbearbeitung
Numpy	Verwendete Bibliothek für die Verwendung von mathematischen Funktionen für die Berechnungen des Algorithmus
PyDotPlus	Verwendete Bibliothek für die grafische Darstellung des Entscheidungsbaums

Nach Abschluss der Algorithmusentwicklung wurde mit der Implementierung des Aufgabentyps als Prototyp begonnen. Die Backend-Entwicklung erfolgte weiterhin mit Python, wobei der zuvor implementierte Algorithmus verwendet wurde. Außerdem wurde Flask, ein Web-Framework für Python verwendet, um die Serveranwendung zu entwickeln. In der Frontend-Entwicklung kamen HTML, CSS und JavaScript für die grundlegende Entwicklung der Benutzeroberfläche zum Einsatz. Zudem wurde die JavaScript-Bibliothek D3.js verwendet, um den Entscheidungsbaum visuell umzusetzen. Basierend auf dem entwickelten Wireframe wurde die Aufgabe so implementiert, dass der Datensatz, aus dem der Entscheidungsbaum berechnet werden sollte, auf der linken Seite angezeigt wird, während rechts der interaktive Entscheidungsbaum dargestellt wird. Oberhalb wird die Aufgabenstellung für den Studierenden angezeigt.

Der interaktive Entscheidungsbaum kann folgendermaßen bedient werden: Durch einen Klick auf einen Knoten wird ein Prompt angezeigt, in dem der Studierende die Bezeichnung des Merkmals eingeben kann. Das gleiche Vorgehen gilt für einen Klick auf eine Kante, wobei hier die Bezeichnung sowie die zugehörige Entropie eingegeben werden können. Um weitere Knoten hinzuzufügen, kann der Studierende auf das "+"-Zeichen unterhalb des Knotens klicken und im Prompt die Anzahl der hinzuzufügenden Knoten auf der nächsten Ebene angeben. Ebenso kann er die Anzahl der Knoten reduzieren oder die Knoten auf der Ebene wieder löschen, indem er erneut auf das "+"-Zeichen unterhalb des Knotens klickt und die neue Anzahl eingibt oder die Anzahl auf 0 setzt, um die hinzugefügten Knoten zu entfernen. Dabei ist es möglich, auf jeder Ebene des Baumes die Anzahl der Knoten zu verändern. Allerdings werden die zuvor eingetragenen Werte für Knoten und Kanten mit den Standardwerten überschrieben, wenn die Anzahl auf der Ebene verändert wurde. Daher ist es erforderlich, nachträglich diese Werte erneut einzugeben.

Zur Überprüfung der Eingaben des Studierenden wird auf den Button "Speichern" geklickt. Dabei werden alle Knoten und Kanten rot eingefärbt, welche falsch sind. Hier werden folgende Prüfungen durchgeführt:

- **Schritt 1:** Generelle Überprüfung, ob die Eingabe der Bezeichnung korrekt ist. Zum Beispiel, ob die Bezeichnung des Merkmals vorhanden ist.
- **Schritt 2:** Um unterschiedliche Aufbaumöglichkeiten des Entscheidungsbaums zu ermöglichen, wird überprüft, ob der eingegebene Wert auf der richtigen Ebene eingetragen wurde.
- **Schritt 3:** Wenn die Eingabe korrekt ist und sich auf der richtigen Ebene befindet, wird überprüft, ob der übergeordnete Knoten bzw. die übergeordnete Kante korrekt ist.

Wenn alle Prüfungen erfolgreich verlaufen sind, wird die Eingabe des Studierenden als korrekt betrachtet, anderenfalls wird sie als falsch bzw. rot angezeigt.

Zur weiteren Kontrolle wurde im Prototyp eine Ausgabe implementiert, die den möglichen Fehler bei der Eingabe angibt. Um das Debugging zu erleichtern, wurde zwischen Knoten und Kanten unterschieden, um anzusehen, wo der Fehler aufgetreten ist. Es wurden die folgenden Prüfungen implementiert:

- **1. Prüfung:** "Label/Entropie nicht richtig"
- **2. Prüfung:** "Parent nicht richtig" - Hierbei handelt es sich um den übergeordneten Knoten bei einer Kante bzw. die übergeordnete Kante bei einem Knoten

Die Implementierung der Überprüfung auf Mehrfachlösungen mittels des ID3-Algorithmus wurde so realisiert, dass die eingegebenen Merkmale des Studierenden als Basis genutzt werden. Wenn während der Berechnung des Entscheidungsbaums an einem bestimmten Knoten mehrere Lösungen möglich sind, wird überprüft, welches Merkmal der Studierende an diesem Knoten eingetragen hat. Wenn die Eingabe mit einer der Lösungsmöglichkeiten übereinstimmt, wird dieses Merkmal für die fortlaufende Berechnung des Entscheidungsbaums verwendet. Andernfalls wird das zuerst berechnete Merkmal als Lösungsmöglichkeit ausgewählt. Durch dieses Vorgehen ist es nicht erforderlich, unterschiedliche Lösungsbäume zu speichern, was die Überprüfung des eingegebenen Entscheidungsbaums vereinfacht und beschleunigt.

Tabelle 4. Übersicht über die verwendeten Technologien für die Entwicklung des Prototyps

Technologie	Erläuterung
Backend	Python als verwendete Programmiersprache Flask als Webframework für Python für die Umsetzung einer Serveranwendung Pandas und Numpy als verwendete Bibliotheken
Frontend	HTML, CSS und JavaScript für den grundlegenden Aufbau und Umsetzung der Benutzeroberfläche des Aufgabentyps D3.js als verwendete JavaScript-Bibliothek zur grafischen Erstellung von Entscheidungsbäumen

Entscheidungsbaum

Aufgabenstellung

Berechnen Sie mithilfe des ID3-Algorithmus einen Entscheidungsbaum für das vorliegende Datenset.

Nutzen Sie den bereitgestellten interaktiven Entscheidungsbaum und gestalten Sie den Baum entsprechend. Kennzeichnen Sie die Attribute als Knoten, die Ausprägungen der Attribute sowie die Entropie an den Kanten.

Aussicht	Temperatur	Luftfeuchtigkeit	Wind	Spiel
sonnig	heiß	hoch	nein	nein
sonnig	heiß	hoch	ja	nein
bewölkt	heißt	hoch	nein	ja
regnerisch	mild	hoch	nein	ja
regnerisch	kalt	normal	nein	ja
regnerisch	kalt	normal	ja	nein
bewölkt	kalt	normal	ja	ja
sonnig	mild	hoch	nein	nein
sonnig	kalt	normal	nein	ja
regnerisch	mild	normal	nein	ja
sonnig	mild	normal	ja	ja
bewölkt	mild	hoch	ja	ja
bewölkt	heiß	normal	nein	ja
regnerisch	mild	hoch	ja	nein

Entscheidungsbaum

```

graph TD
    Root[Aussicht] -- "sonnig 0.97095" --> Node1[Decision 1]
    Root -- "bewölkt 0.0" --> Node2[ja]
    Root -- "regnerisch 0" --> Node3[Wind]
    Node1 -- "Label 1 0" --> Leaf1[ja]
    Node1 -- "Label 2 0" --> Leaf2[nein]
    Node3 -- "Label 1 0" --> Node4[Decision 1]
    Node3 -- "Label 2 0" --> Node5[Decision 2]
    
```

The decision tree starts at the root node 'Aussicht'. The first split is between 'sonnig' (entropy 0.97095) and 'bewölkt' (entropy 0.0). The 'bewölkt' branch leads to a leaf node 'ja'. The 'sonnig' branch leads to 'Decision 1'. 'Decision 1' splits into two labels: 'Label 1 0' leading to 'ja' and 'Label 2 0' leading to 'nein'. The second split is between 'regnerisch' (entropy 0) and 'Wind'. The 'Wind' branch leads to 'Decision 2'. 'Decision 2' splits into two labels: 'Label 1 0' leading to 'ja' and 'Label 2 0' leading to 'nein'.

Speichern
!
Auswerten

Falsche Knoten:

- Node ID: 2 (Decision 1) - Label nicht richtig
- Node ID: 4 (ja) - Parent nicht richtig
- Node ID: 6 (nein) - Parent nicht richtig
- Node ID: 12 (Decision 1) - Label nicht richtig
- Node ID: 14 (Decision 2) - Label nicht richtig

Falsche Kanten:

- Edge ID: 3 (Label 1, 0) - Label/Entropie nicht richtig
- Edge ID: 5 (Label 2, 0) - Label/Entropie nicht richtig
- Edge ID: 9 (regnerisch, 0) - Label/Entropie nicht richtig
- Edge ID: 11 (Label 1, 0) - Label/Entropie nicht richtig
- Edge ID: 13 (Label 2, 0) - Label/Entropie nicht richtig

Abbildung 14. Beispielansicht des Prototyps

4.3.5. Generierung neuer Datensätze

Bei der Generierung eines neuen Datensatzes für die Erstellung einer neuen Berechnungsaufgabe stand die Herausforderung im Vordergrund, die Struktur des Entscheidungsbaums aus der Berechnung des Originaldatensatzes unverändert zu lassen. Dabei sollten ausschließlich an den berechneten Werten wie Informationsgewinne und Entropien Veränderungen vorgenommen werden. Das Ziel bestand darin, neue Datenpunkte auf Grundlage des bereits vorhandenen Entscheidungsbaums zu generieren. Hierbei war es wichtig, die Verteilung der Datenpunkte beizubehalten, sodass auch Merkmale, die im Entscheidungsbaum nicht vorhanden sind, in der Berechnung des Entscheidungsbaums aus dem neuen Datensatz weiterhin nicht berücksichtigt werden. Dadurch sollte die Struktur des Baumes erhalten bleiben.

Bei der Umsetzung wurde die Bootstrap-Sampling-Methode gewählt. Die Bootstrap-Sampling-Methode ist eine statistische Methode, die es ermöglicht, aus einer vorhandenen Stichprobe wiederholt neue Stichproben zu ziehen. Dabei können Statistiken wie der Mittelwert oder andere Kennzahlen berechnet werden. Das Vorgehen beginnt mit einem Gesamtdatensatz, aus dem zufällig

Datenpunkte ausgewählt werden. Dabei können bereits ausgewählte Datenpunkte erneut gewählt werden. Dieser Vorgang wird mehrfach wiederholt, sodass eine Vielzahl von Stichproben erstellt wird.

ANMERKUNG

Für die Recherche der Bootstrap-Sampling-Methode wurden folgende Quellen verwendet:

<https://novustat.com/statistik-blog/resampling-methoden-bootstrapping-statistik-permutation-test.html>

<https://www.analyticsvidhya.com/blog/2020/02/what-is-bootstrap-sampling-in-statistics-and-machine-learning/>

Für die Generierung neuer Datensätze wurde daher die Bootstrap-Sampling-Methode gewählt, da sie eine unkomplizierte und effiziente Möglichkeit bot, basierend auf bereits vorhandenem Datensatz neue Datenpunkte zu generieren. Bei der Umsetzung der Methode war es daher möglich, den bereits bestehenden Entscheidungsbaum zu verwenden, um wiederholt neue Datensätze zu erstellen. Dabei war es möglich, nicht die Struktur des Entscheidungsbaums zu verändern. Die Umsetzung der Methode wurde mit Python realisiert.

4.4. Technische Umsetzung in ALADIN

Nachdem der Prototyp alle erforderlichen Funktionalitäten hatte, bestand der nächste Schritt darin, diese Funktionalitäten in ALADIN zu integrieren und den Aufgabentypen final zu entwickeln. Bei der Umsetzung des Aufgabentypen in ALADIN und in CARPET wurde sich an den bereits implementierten Gozintograph Aufgabentyp orientiert. Dabei wurde die Komponente für die Generierung eines Graphen übernommen bzw. weiterentwickelt. Dies ermöglichte eine erleichterte Einarbeitung und Umsetzung des Aufgabentypen.

Graphology

Als Erweiterung wurde die Bibliothek "Graphology" verwendet. Die Bibliothek bietet eine Sammlung an Funktionen, um die Erstellung und Verwaltung von Graphen zu erleichtern. Dadurch war es möglich, für den interaktiven Entscheidungsbaum eine flexible Struktur zu schaffen. Die Anwendungen der Funktionen und Event-Handler ermöglichen die einfache Durchführung von Manipulationen am Graphen, wie beispielsweise das Hinzufügen oder Entfernen von Knoten und Kanten. Graphology wurde im Backend (ALADIN) installiert und im Frontend (CARPET) eingebunden.

ANMERKUNG

Für die Installation von Graphology in ALADIN wurde `npm install graphology-types` verwendet. Die genutzten Funktionen wurden aus der Dokumentation entnommen: <https://graphology.github.io/>

4.4.1. Implementierung in ALADIN

Im Backend wurde der ID3-Algorithmus implementiert, um den Lösungsbaum zu berechnen, sowie die Überprüfung des vom Studierenden erstellten Entscheidungsbaums. Die Auswertung des eingegebenen Entscheidungsbaums erfolgt als JSON-Rückgabe an das Frontend, um weitere Bearbeitungen zu ermöglichen. Dabei wurden in ALADIN folgende Daten hinzugefügt oder geändert:

Im Verzeichnis: src\server\tempTaskGraphStorage\tasks\

- **DecisionTree.json**: Festlegung der Komponenten und Schnittstellen für den Aufgabentyp

Im Verzeichnis: src\Tasks\DecisionTrees\

- **DecisionTreeTask.py**: Python-Skript, welches von der Wrapperklasse in WrapperValidateDecisionTree.ts ausgerufen wird und zuständig für die Berechnung des Lösungsbaums sowie die Überprüfung des eingegebenen Entscheidungsbaums ist
- **DecisionTreeTaskFunction.py** : Beinhaltet Funktionen sowohl für den Aufbau des Lösungsbaums als auch für die Überprüfung des eingegebenen Baums
- **DataSampler.py**: Python-Skript, welches von der Wrapperklasse in WrapperDataSampler.ts aufgerufen wird und verantwortlich für die Erzeugung neuer Datensätze ist
- **WrapperDataSampler.ts**: Wrapperklasse für das Python-Skript zur Datengenerierung
- **WrapperValidateDecisionTree.ts**: Wrapperklasse für das Python-Skript zur Validierung des Entscheidungsbaums

Die Dateien **Task.ts** und **WrapperDecisionTree.ts** beinhalten die Funktionalitäten zur Erstellung eines Graphen aus dem Gozintograph Aufgabentyp. Die Funktionalitäten wurden vollständig übernommen.

Im Verzeichnis: src\Tasks\DecisionTrees\datasets

- **spiel.csv**: Ursprünglicher Datensatz, der als Basis für die Generierung eines neuen Datensatzes dient
- **generated_dataset.csv**: Neu generierter Datensatz im CSV-Format, der für die Berechnung des Lösungsbaums verwendet wird
- **generated_dataset.json**: Neu generierter Datensatz im JSON-Format, der für die Anzeige auf der Benutzeroberfläche im Tabellenformat dient

Erläuterung: DecisionTree.json

Festgelegte Schnittstellen

Es wurden die Schnittstellen 'generatedSampledData', 'generateGraph' und 'decisionTreeValidator' implementiert:

- **'generatedSampledData'**: Ermöglicht die Übergabe der neu generierten Daten
- **'generateGraph'**: Als Schnittstelle, um den Aufgabentyp zu erstellen
- **'decisionTreeValidator'**: Ermöglicht den Empfang des vom Studierenden eingegebenen Entscheidungsbaums für die weitere Verarbeitung

```
"API": [
  {
    "task": "decisiontree",
    "name": "generateSampledData",
    "httpMethod": "post",
    "params": { "parameters": "object" }
  },
  {
    "task": "decisiontree",
    "name": "generateGraph",
    "httpMethod": "post"
  },
  {
    "task": "decisiontree",
    "name": "decisionTreeValidator",
    "httpMethod": "post"
  }
]
```

```

        "task": "decisiontree",
        "name": "generateGraph",
        "httpMethod": "post",
        "params": { "parameters": "object" }
    },
{
    "task": "decisiontree",
    "name": "decisionTreeValidator",
    "httpMethod": "post",
    "params": { "parameters": "object" }
}
],
...

```

Hinzugefügte Komponenten für den Aufgabentyp

Für den Aufgabentyp wurden die Komponenten 'hints', 'TaskConfiguration', 'Output' und 'DecisionTree' integriert:

- **'hints'**: Die Verwendung von 'hints' ermöglicht die Implementierung von Hilfestellungen für den Studierenden, um während der Aufgabenbearbeitung Hilfestellungen anzuzeigen
- **'TaskConfiguration'**: Mithilfe von 'TaskConfiguration' wurde sowohl der "Generieren!"-Button implementiert, um neue Aufgaben zu generieren, als auch eine Aufgabenbeschreibung mit der Bedienungsanleitung des Entscheidungsbaums hinzugefügt
- **'Output'**: Dient dazu, die generierten Daten in Tabellenform auf der Benutzeroberfläche anzuzeigen
- **'DecisionTree'**: Die Komponente zeigt dem Studierenden den interaktiven Graphen an

Erläuterung: DecisionTreeTask.py und DecisionTreeTaskFunction.py

Da in der Datei DecisionTreeTask.py Funktionen aus DecisionTreeTaskFunction.py aufgerufen werden, wird hier ein Überblick darüber gegeben, wie diese Funktionen aufgerufen werden und welche Aktionen dabei durchgeführt werden. In dieser Datei werden spezifische Funktionen aufgerufen, die dazu dienen, den Lösungsentscheidungsbaum aufzubauen und ihn für die Überprüfung der Werte vorzubereiten. Die Validierungsfunktion für den eingegebenen Entscheidungsbaum des Studierenden wird hier aufgerufen und das Ergebnis der Überprüfung wird als JSON-Format umgewandelt und als Rückgabewert an die Wrapperklasse in der Datei WrapperValidateDecisionTree.py zurückgegeben.

Zu Beginn wird der generierte Datensatz aus der CSV-Datei geladen, um daraus das Zielmerkmal, dessen Klassen und die weiteren Merkmale zur Berechnung mit dem ID3-Algorithmus zu filtern. Sobald die Wrapperklasse aus der WrapperValidateDecisionTree.py Listen mit den enthaltenen Knoten und Kanten aus dem eingegebenen Entscheidungsbaum übergibt, werden diese Listen für die Überprüfung vorbereitet, sodass sie die benötigte Struktur aufweisen. Das bedeutet, dass jeder Knoten in der Liste der Knoten einem Tupel entspricht, was auch für alle Kanten in der Liste der Kanten gilt. Sobald dies umgesetzt wurde, wird die Funktion start_decisiontree_task() aufgerufen und die vorbereiteten Listen übergeben.

In der Funktion start_decisiontree_task() wird zunächst der Lösungsentscheidungsbaum mithilfe der Funktion build_decision_tree() berechnet. Diese Funktion führt die Berechnung des Entschei-

dungsbaums mit dem ID3-Algorithmus durch. Der Rückgabewert dieser Funktion entspricht einer Baumstruktur, die alle berechneten Werte enthält.

```
# Beispiel: Ausgabe des Lösungsbaums mit den berechneten Werten
{
  "Aussicht": {
    "entropies": {
      "Aussicht = sonnig": 0.87086,
      "Aussicht = regnerisch": 0.97987,
      "Aussicht = bewölkt": 0.0
    },
    "infogain": {
      "Aussicht": 0.32744
    },
    "sonnig": {
      "Luftfeuchtigkeit": {
        "entropies": {
          "Luftfeuchtigkeit = hoch": 0.0,
          "Luftfeuchtigkeit = normal": 0.0
        },
        "infogain": {
          "Luftfeuchtigkeit": 0.87086
        },
        "hoch": "nein",
        "normal": "ja"
      }
    },
    "regnerisch": {
      "Wind": {
        "entropies": {
          "Wind = nein": 0.0,
          "Wind = ja": 0.0
        },
        "infogain": {
          "Wind": 0.97987
        },
        "nein": "ja",
        "ja": "nein"
      }
    },
    "bewölkt": "ja"
  }
}
```

Aus diesem Rückgabewert werden die Baumstruktur (`decisiontree_structure()`) und die berechneten Werte für Informationsgewinn und Entropie abstrahiert (`extract_entropy_and_infogain()`).

```
# Beispiel: Ausgabe der Struktur des Lösungsbaums
{
```

```

"Aussicht": {
    "sonnig": {
        "Luftfeuchtigkeit": {
            "hoch": "nein",
            "normal": "ja"
        }
    },
    "regnerisch": {
        "Wind": {
            "nein": "ja",
            "ja": "nein"
        }
    },
    "bewölkt": "ja"
}
}

```

```

# Beispiel: Ausgabe des Dictionaries mit den Informationsgewinnen und Entropien
{'entropies': {'Aussicht = sonnig1': {'value': 0.87086, 'level': 1}, 'Aussicht = regnerisch1': {'value': 0.97987, 'level': 1}, 'Aussicht = bewölkt1': {'value': 0.0, 'level': 1}, 'Luftfeuchtigkeit = hoch1': {'value': 0.0, 'level': 3}, 'Luftfeuchtigkeit = normal1': {'value': 0.0, 'level': 3}, 'Wind = nein1': {'value': 0.0, 'level': 3}, 'Wind = ja1': {'value': 0.0, 'level': 3}}, 'infogain': {'Aussicht1': {'value': 0.32744, 'level': 0}, 'Luftfeuchtigkeit1': {'value': 0.87086, 'level': 2}, 'Wind1': {'value': 0.97987, 'level': 2}}}

```

Sobald dies abgeschlossen ist, wird eine Liste erstellt (build_decision_tree_list()), die den vollständigen Baum mit den dazugehörigen Werten darstellen soll. Dabei werden einzelne Tupel erzeugt, die entweder einen Knoten oder eine Kante repräsentieren sollen. Die Bezeichnungen der Kanten und Knoten werden hinzugefügt, ebenso wie die Ebene, auf der sich der Knoten bzw. die Kante befindet. Anschließend werden die übergeordneten Knoten und Kanten hinzugefügt (add_parent_label() und add_child_node()).

```

# Beispiel: Ausgabe der gesamten Liste des Lösungsbaums
[(1, 1, 'sonnig', 'edge', 0, 'Aussicht', 2, 'Luftfeuchtigkeit'),
(3, 3, 'hoch', 'edge', 2, 'Luftfeuchtigkeit', 4, 'nein'),
(5, 3, 'normal', 'edge', 2, 'Luftfeuchtigkeit', 6, 'ja'),
(7, 1, 'regnerisch', 'edge', 0, 'Aussicht', 8, 'Wind'),
(9, 3, 'nein', 'edge', 8, 'Wind', 10, 'ja'),
(11, 3, 'ja', 'edge', 8, 'Wind', 12, 'nein'),
(13, 1, 'bewölkt', 'edge', 0, 'Aussicht', 14, 'ja'),
(0, 0, 'Aussicht', 'node', None, 'None'),
(2, 2, 'Luftfeuchtigkeit', 'node', 1, 'sonnig'),
(4, 4, 'nein', 'node', 3, 'hoch'),
(6, 4, 'ja', 'node', 5, 'normal'),
(8, 2, 'Wind', 'node', 7, 'regnerisch'),
(10, 4, 'ja', 'node', 9, 'nein'),
(12, 4, 'nein', 'node', 11, 'ja'),
(14, 2, 'ja', 'node', 13, 'bewölkt')]

```

Danach wird die Liste in separate Listen für Knoten und Kanten aufgeteilt, um eine bessere und gezieltere Handhabung bei der Überprüfung zu ermöglichen. Zu jeder Liste werden entsprechend die Entropien und Informationsgewinne hinzugefügt (add_entropy() und add_infogain()).

Zum Abschluss wird die Liste in die benötigte Struktur gebracht (prepare_lists()), um sie bei der Überprüfung verwenden zu können. Dies bedeutet, dass die Tupel in der Liste wie folgt strukturiert sind:

- **Tupel in der Lösungs-Knoten-Liste:** (KEY, LEVEL, LABEL, PARENTID, PARENTLABEL, INFO-GAIN)
- **Tupel in der Lösungs-Kanten-Liste:** (KEY, LEVEL, LABEL, PARENTID, PARENTLABEL, CHILDDID, CHILDLABEL, ENTROPY)

Die Listen solution_nodes und solution_edges dienen als Referenzwerte für die Überprüfung, anhand derer der eingegebene Entscheidungsbaum validiert wird.

```
# Beispiel: solution_nodes Liste
[(0, 0, 'Aussicht', None, 'None', 0.32744),
(2, 2, 'Luftfeuchtigkeit', 1, 'sonnig', 0.87086),
(4, 4, 'nein', 3, 'hoch', 0),
(6, 4, 'ja', 5, 'normal', 0),
(8, 2, 'Wind', 7, 'regnerisch', 0.97987),
(10, 4, 'ja', 9, 'nein', 0),
(12, 4, 'nein', 11, 'ja', 0),
(14, 2, 'ja', 13, 'bewölkt', 0)]
```



```
# Beispiel: solution_edges Liste
[(1, 1, 'sonnig', 0, 'Aussicht', 2, 'Luftfeuchtigkeit', 0.87086),
(3, 3, 'hoch', 2, 'Luftfeuchtigkeit', 4, 'nein', 0.0),
(5, 3, 'normal', 2, 'Luftfeuchtigkeit', 6, 'ja', 0.0),
(7, 1, 'regnerisch', 0, 'Aussicht', 8, 'Wind', 0.97987),
(9, 3, 'nein', 8, 'Wind', 10, 'ja', 0.0),
(11, 3, 'ja', 8, 'Wind', 12, 'nein', 0.0),
(13, 1, 'bewölkt', 0, 'Aussicht', 14, 'ja', 0.0)]
```

Die Funktion validateGraph() vergleicht den vom Studierenden eingegebenen Entscheidungsbaum mit dem berechneten Lösungsbaum. Hierbei werden die Listen der Lösungen (solution_nodes und solution_edges) sowie die erhaltenen Werte aus dem erstellten Entscheidungsbaum (received_nodes und received_edges) übergeben. Als Rückgabewert werden zwei Listen (result_nodes und result_edges) zurückgegeben, die die Auswertungen für jeden eingegebenen Knoten und jede eingegebene Kante enthalten. Hierbei werden lediglich die IDs gespeichert und der Wert 'true', wenn die Eingabe des Studierenden korrekt war, bzw. 'false', wenn die Eingabe falsch war.

```
# Beispiel: result_nodes Liste
[{'key': '1', 'result': False},
{'key': 'key_xlXFsd', 'result': False},
{'key': 'key_soGlGdt', 'result': False},
{'key': 'key_VEqhpdt', 'result': False},
```

```

{'key': 'key_B2MDrdt', 'result': False},
{'key': 'key_rEtVGdt', 'result': False},
{'key': 'key_NiSx0dt', 'result': False},
{'key': 'key_dDTWldt', 'result': True},
{'key': 'key_OmwAkdt', 'result': True},
{'key': 'key_Gyegzdt', 'result': False},
{'key': 'key_xl4ocdt', 'result': False}]

# Beispiel: result_edges Liste
[{'key': 'geid_243_0', 'result': False},
 {'key': 'geid_243_1', 'result': False},
 {'key': 'geid_243_2', 'result': False},
 {'key': 'geid_243_3', 'result': False},
 {'key': 'geid_243_4', 'result': False},
 {'key': 'geid_243_5', 'result': False},
 {'key': 'geid_243_6', 'result': False},
 {'key': 'geid_243_7', 'result': False},
 {'key': 'geid_243_8', 'result': False},
 {'key': 'geid_243_9', 'result': False}]

```

Abschließend wird überprüft, ob der vom Studierenden erstellte Entscheidungsbaum korrekt und vollständig ist. Im Fall, dass der Entscheidungsbaum vollständig richtig aufgebaut wurde, wird der Status von 'finished' auf 'true' gesetzt. Andernfalls bleibt der Status auf 'false'. Die Listen 'result_nodes' und 'result_edges', der Status von 'finished' sowie die berechneten Lösungswerte im Dictionary 'entropy_infogain_dict' werden als JSON-Format umgewandelt und als Rückgabewert an die Wrapperklasse gesendet.

```

import sys
import ast
import json
import pandas as pd
import numpy as np
from DecisionTreeTaskFunction import *

data_path_csv = 'src/Tasks/DecisionTrees/datasets/generated_dataset.csv'
dataframe = pd.read_csv(data_path_csv)

target_attribute = dataframe.columns[-1]
classes = dataframe[target_attribute].unique()
attributes = dataframe.columns[:-1]

def start_decisiontree_task(received_nodes, received_edges, classes):

    dt = build_decision_tree(dataframe, attributes, target_attribute, 0,
    received_nodes)
    decision_tree_structure_dict = decisiontree_structure(dt)
    entropy_infogain_dict = extract_entropy_and_infogain(dt)

    tree_list = []

```

```

build_decision_tree_list(decision_tree_structure_dict, tree_list=tree_list)

    tree_list = add_parent_label(tree_list)
    tree_list = add_child_node(tree_list)

    nodes_list = [item for item in tree_list if item[3] == 'node']
    edges_list = [item for item in tree_list if item[3] == 'edge']

    edges_list = add_entropy(edges_list, entropy_infogain_dict)
    nodes_list = add_infogain(nodes_list, entropy_infogain_dict)

    solutions_nodes, solutions_edges = prepare_lists(nodes_list, edges_list)
    results_nodes, results_edges = validateGraph(solutions_nodes, solutions_edges,
received_nodes, received_edges, classes)

    correctNodesCount = 0
    correctEdgesCount = 0

    for r in results_nodes:
        if r['result'] == True:
            correctNodesCount += 1

    for r in results_edges:
        if r['result'] == True:
            correctEdgesCount += 1

    if(len(solutions_nodes) == correctNodesCount and len(solutions_edges) ==
correctEdgesCount and len(solutions_nodes) == len(received_nodes) and
len(solutions_edges) == len(received_edges)):
        result = { 'nodes': results_nodes, 'edges': results_edges, 'finished': True,
'entropies': entropy_infogain_dict }
    else:
        result = { 'nodes': results_nodes, 'edges': results_edges, 'finished': False,
'entropies': entropy_infogain_dict }

    return (json.dumps(result))

# Erhalt der Nodes- und Edges-Liste aus WrapperValidateDecisionTree.ts
received_nodes_list, received_edges_list = sys.argv[1], sys.argv[2] if len(sys.argv) <= 3 else (None, None)

modified_nodes_list_1 = ast.literal_eval(received_nodes_list)
modified_nodes_list_2 = [tuple(inner) for inner in modified_nodes_list_1]
current_nodes_list = [(x[0], x[1], 0 if x[2] is None else x[2], x[3], x[4]) for x in modified_nodes_list_2]

modified_nodes_list = ast.literal_eval(received_edges_list)
current_edges_list = [tuple(inner) for inner in modified_nodes_list]

print(start_decisiontree_task(current_nodes_list, current_edges_list, classes))

```

Erläuterung: DataSampler.py

In der DataSampler.py wird die Datengenerierung implementiert. Zunächst wird der originale Datensatz geladen, um neue Stichproben zu generieren. Der Pfad, unter dem der neu generierte Datensatz gespeichert werden soll, wird mit 'src/Tasks/DecisionTrees/datasets/generated_dataset.csv' angegeben.

bootstrap_sample() und generate_data_point() - Funktion

Diese Funktionen dienen dazu, Bootstrap-Samples des Datensatzes mit bootstrap_sample() zu erstellen und Datenpunkte basierend auf dem Entscheidungsbaum mit generate_data_point() zu generieren.

check_generated_tree() - Funktion

Diese Funktion überprüft, ob die Struktur des Baumes aus dem generierten Datensatz der originalen Baumstruktur entspricht. Dies stellt sicher, dass bei unterschiedlichen Strukturen der Datensatz erneut generiert wird, bis er die gleiche Struktur wie der Originalbaum enthält.

sample_data() - Funktion

Diese Funktion ruft die Datengenerierung auf. Die while-Schleife sorgt dafür, dass der neue Datensatz generiert wird, solange er nicht der originalen Baumstruktur des Originalbaums entspricht. Mit 'num_samples' wird die Anzahl der Batches festgelegt, wie oft Daten generiert werden sollen. Die Größe des Batches entspricht der Größe des Ursprungsdatensatzes. Der Aufruf von pd.concat(...) erstellt ein neues DataFrame aus der Pandas-Bibliothek, welches die neu generierten Daten strukturiert. Die Funktion concat() wird verwendet, um mehrere Batches zusammenzuführen und einen vollständig neuen Datensatz zu erhalten.

Nach der Generierung des Datensatzes wird dieser sowohl im JSON-Format gespeichert, um ihn als Tabelle auf der Benutzeroberfläche anzuzeigen, als auch als CSV-Datei, um die Berechnung der Baumstruktur aus dem generierten Datensatz zu ermöglichen.

```
import pandas as pd
import numpy as np
from DecisionTreeTaskFunction import *

data_path_csv = 'src/Tasks/DecisionTrees/datasets/spiel.csv' # Originaler Datensatz
generated_data_path_csv = 'src/Tasks/DecisionTrees/datasets/generated_dataset.csv' # Generierter Datensatz

data = pd.read_csv(data_path_csv)
df = pd.DataFrame(data)

# Beispiel: Lösungsbaum-Struktur von spiel.csv
# decision_tree = {'Aussicht': {'sonnig': {'Luftfeuchtigkeit': {'hoch': 'nein',
# 'normal': 'ja'}},
#                   'bewölkt': 'ja',
#                   'regnerisch': {'Wind': {'nein': 'ja', 'ja': 'nein'}}}

decision_tree_original = get_decisiontree_structure(data_path_csv)

# Prüfung der Baumstruktur aus dem generierten Datensatz
```

```

def check_generated_tree(original_tree, generated_tree):
    original_json = json.loads(json.dumps(original_tree, sort_keys=True))
    generated_json = json.loads(json.dumps(generated_tree, sort_keys=True))
    return original_json == generated_json

# Bootstrap-Sampling-Funktion
def bootstrap_sample(data):
    # samplen der Datensätze in gleicher Größe wie die original Daten
    sample = data.sample(n=len(data), replace=True)
    return sample

# Funktion zum Generieren von Datenpunkten basierend auf dem Entscheidungsbaum
def generate_data_point(tree):
    current_node = tree
    while isinstance(current_node, dict):
        feature = list(current_node.keys())[0]
        values = list(current_node[feature].keys())
        selected_value = np.random.choice(values)
        current_node = current_node[feature][selected_value]
    return current_node

def sample_data():
    while True:
        num_samples = 5
        data = pd.concat([bootstrap_sample(df.apply(generate_data_point, axis=1)) for _ in range(num_samples)], ignore_index=True)

        # als JSON speichern für die Ausgabe als Tabelle

        data.to_json('src/Tasks/DecisionTrees/datasets/generated_dataset.json', orient='records',
                     lines=True)

        # als CSV speichern für die Berechnung des Entscheidungsbaums
        data.to_csv('src/Tasks/DecisionTrees/datasets/generated_dataset.csv',
                   index=False)

        # Entscheidungsbaum für die generierten Daten erstellen
        decision_tree_generated = get_decisiontree_structure(generated_data_path_csv)

        # Überprüfen, ob die generierten Daten den originalen Daten die gleiche
        Baumstruktur entsprechen
        if check_generated_tree(decision_tree_original, decision_tree_generated):
            print("Die generierter Datensatz entspricht der Struktur des originalen
Entscheidungsbaum.")
            break

sample_data()

```

Erläuterung: WrapperDataSampler.ts

Die Datei `WrapperDataSampler.ts` enthält die `Wrapper`-Klasse, die in TypeScript geschrieben ist und

dazu dient, die Ausführung des Python-Skripts DataSampler.py zu ermöglichen.

runPythonScript() - Funktion

Diese Funktion ermöglicht die Ausführung eines Python-Skripts mit übergebenen Argumenten. Hierbei wird die 'exec()' -Funktion aus dem 'child_process' Modul verwendet. Der Shell-Befehl für die Skriptausführung wird erstellt und bei erfolgreicher Ausführung wird die Standardausgabe zurückgegeben. Im Fehlerfall werden entsprechende Fehlermeldungen oder Standardfehlerausgaben behandelt und zurückgegeben.

readJsonFile() - Funktion

Diese Funktion liest eine JSON-Datei und gibt die geparsten Daten als Array von Objekten zurück. Sie filtert leere Zeilen und behandelt Lesefehler sowie Parsing-Ausnahmen. Die geparsten JSON-Daten werden als Promise zurückgegeben.

DataSamplingGenerator() - Funktion

Dies ist die Hauptfunktion, welche das Python-Skript DataSampler.py ausführt, die JSON-Datei einliest und eine Tabelle aus den Daten erstellt. Dabei werden mögliche Fehler bei der Ausführung des Skripts oder beim Lesen der JSON-Datei behandelt.

Erläuterung: WrapperValidateDecisionTree.ts

Die Datei WrapperValidateDecisionTree.ts enthält die Wrapperklasse, die in TypeScript geschrieben ist, um die Ausführung des Python-Skripts dtBuildDecisonTree.py zu erleichtern.

runPythonScript() - Funktion

Diese Funktion ermöglicht die Ausführung eines Python-Skripts über die Shell und nutzt die 'exec()' -Funktion aus dem 'child_process' Modul. Fehler, die während der Ausführung auftreten, werden über ein Promise behandelt, um eine asynchrone Ausführung zu ermöglichen.

createNodesList() und createEdgesList() - Funktion

Diese Funktion konvertiert eine Liste von Knoten und Kanten aus dem Graphen in ein Tupel-Format mit einer festgelegten Struktur, die für die spätere Überprüfung der Werte relevant ist.

- **Die Struktur eines Knoten-Tupels:** (KEY, LEVEL, LABEL, PARENTEDGE, INFOGAIN)
- **Die Struktur eines Kanten-Tupels:** (KEY, LEVEL, LABEL, PARENTKEY, PARENTNODELABEL, CHILDKEY, CHILDNODELABEL, ENTROPY)

DecisionTreeValidator() - Funktion

Dies ist die Hauptfunktion. Aus dem übergebenen Entscheidungsbaum an diese Funktion werden die Knoten- und Kantenlisten erzeugt, die für die Überprüfung übergeben werden sollen. Die Daten werden für die Übergabe so vorbereitet, dass sie als JSON-Daten in Zeichenketten umgewandelt werden. Hierbei werden die Zeichenketten mit Hochkommas gepräst, um eine korrekte Übertragung der Daten als Shell-Befehl zu ermöglichen.

Das Python-Skript DecisionTreeTask.py wird mit den übergebenen Listen als Argumente ausgeführt und erhält die Werte der Auswertung des Entscheidungsbaums zurück. Diese werden anschließend als JSON-Format gepräst und als Rückgabewert an das Frontend CARPET weitergegeben.

4.4.2. Implementierung in CARPET

Bei der Frontend-Umsetzung in CARPET wurde sich an dem Gozintographen Aufgabentyp orientiert. Da dieser Aufgabentyp bereits eine Graphenimplementierung enthielt, wurden die entsprechenden Komponenten modifiziert und erweitert, um eine visuelle Darstellung des Entscheidungsbaums zu ermöglichen.

In CARPET wurden folgende Dateien hinzugefügt bzw. geändert:

- Neue Komponente für die Erstellung eines interaktiven Entscheidungsbaums: **DecisionTree.vue** (im Verzeichnis: src\components\taskComponents\DecisionTree\DecisionTree)
- **Änderungen in Helperfunctions.ts**: Weitere Funktionen wurden hinzugefügt als Unterstützung für die Umsetzung des interaktiven Entscheidungsbaums
- **Änderungen in taskGraph.ts**: Eine Schnittstelle wurde implementiert, um den bearbeiteten Graphen des Studierenden an das Backend (ALADIN) zu übertragen

Erläuterung: DecisionTree.vue

Bei der Umsetzung der neuen Komponenten DecisionTree.vue wurde sich an der bereits existierenden DOTGraph-Komponente orientiert, die bereits im Gozintographen-Aufgabentyp verwendet wurde. Die DOTGraph-Komponente stellte eine Grundlage dar, um eine ähnliche Visualisierung eines Graphen für einen interaktiven Entscheidungsbaum zu implementieren. Dadurch konnte bereits bestehender Code wiederverwendet und erweitert werden.

Neuen Graphen anlegen und IDs speichern

Zu Beginn wird ein neuer Graph erstellt und ein Array zur Speicherung von IDs für jeden Knoten und für jede Kante. Dadurch wird die Überprüfung auf Vorhandensein einer ID ermöglicht, wenn eine neue ID generiert wird.

```
let graph = new Graph();
let ids: Array<String> = [];
```

Funktion: idGenerator()

Diese Funktion generiert automatisch eindeutige Zeichenketten als IDs für Knoten und Kanten im Graphen. Dabei wird überprüft, ob die generierte ID bereits vorhanden ist. Diese werden in das zuvor angelegte Array gespeichert.

```
const idGenerator = () => {
  let s = "";
  do {
    s = "key_" + createRandomString() + "dt";
  } while (ids.includes(s));
  ids.push(s);
  return s;
}
```

Wurzelknoten erstellen

Diese Funktion hat den Zweck, den ersten Knoten (Wurzelknoten) zu erstellen. Dabei werden der

erstellte Graph und ein Schlüsselwert für den Wurzelknoten übergeben. Es wurde festgelegt, dass der Wurzelknoten im Graphen immer den Schlüsselwert '1' hat, um eine eindeutige Identifikation zu gewährleisten. Schlüsselwerte (Keys) sind erforderlich für die Nutzung der Funktionen aus der Graphology-Bibliothek, um eindeutig Knoten und Kanten anzusteuern.

```
function createRootNode(graph, rootNodeKey) {  
    graph.addNode(rootNodeKey, {  
        id: idGenerator(),  
        label: "Wurzelknoten\\n(Info)",  
        name: "Wurzelknoten",  
        infogain: -1.0,  
        level: 0,  
        parentedge: "None"  
    });  
}
```

Event: Anklicken des "Generieren!"-Buttons

Nach dem Klick auf den 'Generieren!' -Button wird überprüft, ob bereits ein Wurzelknoten vorhanden ist. Falls nicht, wird ein neuer Wurzelknoten erzeugt. Wenn bereits ein Knoten existiert, wird der aktuelle Entscheidungsbaum gelöscht und ein neuer Wurzelknoten erstellt. Dadurch wird gewährleistet, dass beim Generieren eines neuen Datensatzes der Entscheidungsbaum entsprechend zurückgesetzt wird, um die neue Aufgabe von vorne beginnen zu können.

```
setTimeout(() => {  
    var generateButton = document.querySelector('.button');  
    generateButton.addEventListener('click', function() {  
        const rootNodeKey = '1';  
        let nodeExists = graph.hasNode(rootNodeKey);  
        if (!nodeExists) {  
            createRootNode(graph, rootNodeKey);  
        } else {  
            alert('Neuer Datensatz wurde generiert!');  
            graph.clear();  
            createRootNode(graph, rootNodeKey);  
        }  
    });  
}, 100);
```

Hinzufügen von Attributwerten: parentedge und childnode

Die Funktion setParentEdge() fügt die eingehenden Kanten zu einem Knoten hinzu. Dies ermöglicht spätere Überprüfungen, um sicherzustellen, dass der Knoten eindeutig zugeordnet werden kann. Gleichermaßen gilt für die Funktion setParentChildNode(). Diese Funktion fügt für jede Kante den Startknoten ('parentnode') und den Zielknoten ('childnode') hinzu.

```
function setParentEdge(key: any) {  
    graph.findNode((node, attributes) => {  
        if (key == node) {
```

```

graph.findInEdge((edge, attributes, source, target) => {
    if (node == target) {
        graph.setNodeAttribute(node, 'parentedge', attributes['name']);
    }
});
}
});

function setParentChildNode(key: any) {
    let nodeName = graph.getNodeAttribute(key, 'name')
    graph.findEdge((edge, attributes, source, target) => {
        if (key == source) {
            graph.setEdgeAttribute(edge, 'parentnode', nodeName)
        }
    });
    graph.findEdge((edge, attributes, source, target) => {
        if (key == target) {
            graph.setEdgeAttribute(edge, 'childnode', nodeName)
        }
    });
}
}

```

Farbige Hervorhebung der Knoten und Kanten

Die Funktionen colorNodes() und colorEdges() haben den Zweck, Knoten und Kanten farblich hervorzuheben. Nach der Überprüfung des Entscheidungsbaums im Backend und der Übermittlung der Auswertungen an das Frontend werden zu jeder Kante und zu jedem Knoten entsprechend ihren Werten (true oder false) in den Farben Grün oder Orange eingefärbt. Hierbei wird die Farbe als zusätzliches Attribut entsprechend hinterlegt.

```

function colorNodes(key: any, c: boolean) {
    if (c) { graph.setNodeAttribute(key, 'color', 'green'); }
    else { graph.setNodeAttribute(key, 'color', '#f1ad2d'); }
}

function colorEdges(key: any, c: boolean) {
    if (c) { graph.setEdgeAttribute(key, 'color', 'green'); }
    else { graph.setEdgeAttribute(key, 'color', '#f1ad2d'); }
}

```

Übermittelte Auswertungen der Knoten und Kanten aus dem Backend überprüfen

In der Funktion validateGraph() erfolgt die Überprüfung der Auswertungen, die vom Backend übergeben wurden. Hier werden die Funktionen zum Einfärben der Knoten und Kanten aufgerufen. Zudem wird überprüft, ob der Entscheidungsbaum bzw. der Graph bereits vollständig gelöst wurde. Diese Information wird aus den Lösungswerten der Auswertung ('parsendResult['status']) herausgelesen. Wenn dieser Wert den Wert true besitzt, gilt die Aufgabe als vollständig gelöst und es wird eine Meldung für den Studierenden angezeigt.

```

function validateGraph(parsedResult: any){
    const solutionNodesList = parsedResult['solution_nodes_list'];
    const solutionEdgesList = parsedResult['solution_edges_list'];
    const status = parsedResult['status'];
    for (let i = 0; i < solutionNodesList.length; i++) {
        colorNodes(solutionNodesList[i]['key'], solutionNodesList[i]['result']);
    }
    for (let i = 0; i < solutionEdgesList.length; i++) {
        colorEdges(solutionEdgesList[i]['key'], solutionEdgesList[i]['result']);
    }
    if (status) {
        setTimeout(() => {
            console.log("Finished: ", status);
            alert("Herzlichen Glückwunsch!\nSie haben die Aufgabe vollständig gelöst!");
        }, 200);
    }
    renderIfGraph();
}

```

Event-Handler bei Änderungen der Knoten und Kanten

Die Event-Handler in dieser Implementierung sind aus der Graphology-Bibliothek und ermöglichen die Reaktion auf Veränderungen im Graphen. Im folgenden Beispiel wird dies bei einer Attributsänderung eines Knotens verdeutlicht, wobei das Prinzip auf andere Event-Handler übertragbar ist. Insbesondere ist der Event-Handler von Bedeutung, wenn die Attributwerte 'label' (Knotenbeschriftung), 'parentedge' (eingehende Kante des Knotens) und 'infogain' (Informationsgewinn für den Knoten) verändert werden. Daher erfolgt zunächst eine Überprüfung, ob das geänderte Attribut eines dieser Werte ist. Falls nicht, wird die Funktion vorzeitig verlassen. Andernfalls wird der aktuelle Zustand des Graphen an das Backend gesendet, welches den vom Studierenden eingegebenen Graphen überprüft. Nachdem der Graph übermittelt wurde, wird die Auswertung als JSON unter 'parsedResult' gespeichert und für weitere Verarbeitung bereitgestellt.

Die folgenden Event-Handler wurden für diese Anwendung genutzt:

- **'nodeAttributesUpdated'**: bei Änderungen der Attributwerte 'label', 'parentedge' und 'infogain' eines Knotens
- **'edgeAdded'**: bei Hinzufügen einer Kante
- **'edgeAttributedUpdated'**: bei Änderungen der Attributwerte 'label', 'entropy', 'parentnode' und 'childnode' einer Kante
- **'nodeDropped'**: bei Entfernen eines Knotens

```

graph.on('nodeAttributesUpdated', function({key, type, attributes, name}) {
    if (name !== 'label' && name !== 'parentedge' && name !== 'infogain')
        return;

    unref(storeObject).store.dispatch('fetchSolution', graph.export())
        .then(result => {
            const parsedResult = JSON.parse(result);
            validateGraph(parsedResult);
            console.log("solution: ", parsedResult)
        })
})

```

```

    renderIfGraph();
  });
});

```

Überprüfung der Eingabewerte für den Informationsgewinn und die Entropie

Die Funktion isValidInputEntropy() und isValidInputInfo() wurden implementiert, um die Eingabewerte für die Entropie an den Kanten und den Informationsgewinn an den Knoten zu validieren. Dies soll sicherstellen, dass die übergebenen Werte vom Typ Float sind, um mögliche Komplikationen bei späteren Überprüfungen zu vermeiden. Bei der Validierung des Informationsgewinns wird darüber hinaus geprüft, ob die Eingabe entweder vom Typ Float oder vom Typ String mit dem spezifischen Stringwert 'decision'. Nur der Stringwert 'decision' ist zulässig, um Knoten zu identifizieren, die Klassen repräsentieren.

```

function isValidInputEntropy(input: string): boolean {
  const floatValue = parseFloat(input);
  if (!isNaN(floatValue)) {
    return true;
  }
  else {
    return false;
  }
}

function isValidInputInfo(input: string | number): string | number | null {
  if (typeof input === 'number') {
    return parseFloat(input.toFixed(5));
  } else if (typeof input === 'string') {
    if (input.toLowerCase() === 'decision') {
      return 'decision';
    }
    const parsedFloat = parseFloat(input);
    return isNaN(parsedFloat) ? null : parseFloat(parsedFloat.toFixed(5));
  }
  return null;
}

```

Interaktionen mit dem Graphen

Um den Studierenden eine interaktive Bedienung des Entscheidungsbaums bzw. des SVG-Graphen zu ermöglichen, wurden Event-Handler aus der D3.js-Bibliothek für die Knoten und Kanten implementiert. Dabei werden sowohl Links- als auch Rechtsklicks beim Anklicken eines Knotens abgefangen. Bei Kanten wird ausschließlich auf Rechtsklick reagiert.

```

setTimeout(() => {
  d3.selectAll('svg .node').on('click', null);
  d3.selectAll('svg .node').on('contextmenu', null);
  d3.selectAll('svg .edge').on('contextmenu', null);
  ...
}

```

Rechtsklick auf eine Kante: Kantenbeschriftung und Entropie ändern

Wenn eine Kante durch Rechtsklick ausgewählt wird, öffnet sich ein Prompt zur Eingabe der Kantenbeschriftung. Wenn die Eingabe für die Kantenbeschriftung leer ist, wird der Prompt geschlossen, ohne Änderungen an der Kante vorzunehmen. Falls jedoch eine Kantenbeschriftung eingegeben wurde, erscheint ein weiterer Prompt zur Eingabe der Entropie an der Kante. Die eingegebene Entropie wird auf seine Gültigkeit überprüft. Dies wird so lange wiederholt, bis ein gültiger Wert eingegeben wurde. Anschließend werden diese Werte als Attribute für die Kante gespeichert. Das Attribut 'label' fungiert als zusammengesetzter String, der die Kantenbeschriftung und die Entropie enthält und im Graphen angezeigt wird. Das Attribut 'name' enthält ausschließlich die Kantenbeschriftung, was für spätere Überprüfungen erforderlich ist.

```
// Kantenbeschriftung und Entropie ändern
d3.selectAll('svg .edge').on('contextmenu', (e) => {
    let edgeName = prompt("Kantenbeschriftung hinzufügen:", "");
    let edgeEntropyInput;
    edgeName = normalizeString(edgeName);

    if (edgeName == null || edgeName == "") {
        return;
    } else {
        do {
            edgeEntropyInput = prompt("Entropy hinzufügen:", "");
        } while (!isValidInputEntropy(edgeEntropyInput));
    }

    const edgeLabel = edgeName + "\n" + "(" + edgeEntropyInput + ")"
    const edgeId = e.target.parentNode.id;
    const filteredEdge = graph.filterEdges((n, a) => a.id == edgeId);

    if (filteredEdge.length <= 0 || edgeLabel == null || edgeLabel == "") {
        return;
    }

    const edgeEntropy = parseFloat(edgeEntropyInput);
    graph.setEdgeAttribute(filteredEdge[0], 'label', edgeLabel);
    graph.setEdgeAttribute(filteredEdge[0], 'name', edgeName);
    graph.setEdgeAttribute(filteredEdge[0], 'entropy', edgeEntropy);

    setTimeout(() => {
        let parentEdge;
        graph.findEdge((edge, attributes, source, target) => {
            if(filteredEdge[0] == edge){
                parentEdge = target;
            }
        });
        setParentEdge(parentEdge);
    }, 300);
});
```

Rechtsklick auf einen Knoten: Knotenbeschriftung und Informationsgewinn ändern

Bei einem Rechtsklick auf einen Knoten öffnet sich ein Prompt, welcher zur Eingabe der Knotenbeschriftung auffordert. Wenn die Eingabe leer ist, wird die Funktion vorzeitig beendet und es erfolgt keine Änderungen am Knoten. Andernfalls erscheint ein weiterer Prompt, der nach einer Eingabe des Informationsgewinns für den Knoten auffordert. Der eingegebene Wert wird überprüft. Bei einer ungültigen Eingabe wird erneut der Prompt angezeigt, bis ein gültiger Wert eingegeben wird. Anschließend werden die eingegebenen Werte als Attributwerte des Knotens gespeichert. Das Attribut 'label' fungiert als zusammengesetzter String, der die Beschriftung des Knotens und den Informationsgewinn enthält und für die Anzeige im Graphen verwendet wird. Das Attribut 'name' wird für die Knotenbeschriftung verwendet und ist für die späteren Überprüfungen relevant.

```
// Knotenbeschriftung und Informationsgewinn ändern
d3.selectAll('svg .node').on('contextmenu', (e) => {
    let nodeName = prompt("Label für Knoten ändern:", "");
    let nodeInfoGainInput: string | number;
    let nodeInfoGain;

    nodeName = normalizeString(nodeName);
    const nodeId = e.target.parentNode.id;
    const node = graph.filterNodes((n, a) => a.id == nodeId);

    if (node.length <= 0 || nodeName == null || nodeName == "") {
        return;
    } else {
        do {
            nodeInfoGainInput = prompt("Informationsgewinn hinzufügen:", "");
            nodeInfoGain = isValidInputInfo(nodeInfoGainInput);
        } while (nodeInfoGain === null);
    }

    graph.setNodeAttribute(node[0], 'name', nodeName);
    graph.setNodeAttribute(node[0], 'infogain', nodeInfoGain);
    graph.setNodeAttribute(node[0], 'label', nodeName + "\n" + "(" + nodeInfoGain +
")");

    setTimeout(() => {
        setParentChildNode(node);
    }, 200);
});
```

Linksklick auf einen Knoten: Neuer Knoten hinzufügen und Knoten entfernen

Bei einem Linksklick auf einen Knoten können zwei Funktionalitäten ausgelöst werden:

1. **Linksklick mit Eingabe einer Beschriftung für den Knoten:** Ein neuer Knoten wird unterhalb des angeklickten Knotens hinzugefügt
2. **Linksklick ohne Eingabe einer Beschriftung für den Knoten:** Der angeklickte Knoten wird entfernt

Bei einem Linksklick auf einen Knoten öffnet sich ein Prompt, um die Beschriftung des neu hinzufügenden Knotens einzugeben. Dabei wird überprüft, ob eine Eingabe vorhanden ist. Wenn die Eingabe leer ist, wird die Funktion zum Entfernen des angeklickten Knotens ausgeführt. Andernfalls öffnet sich ein weiterer Prompt, in dem der Informationsgewinn eingegeben werden kann. Es folgt die Erstellung eines neuen Knotens mit den eingegebenen Attributwerten sowie die Erzeugung einer neuen Kante, die vom angeklickten Knoten ausgeht und zum neuen hinzugefügten Knoten führt. Für die Kante werden entsprechende Attributwerte hinzugefügt, die zunächst Standardwerte beinhalten.

```
// Neuen Knoten hinzufügen/ Knoten entfernen
d3.selectAll('svg .node').on('click', (e) => {
    let nodeName = prompt("Neuer Knoten:", "");
    let nodeInfoGainInput: string | number;
    let nodeInfoGain;

    nodeName = normalizeString(nodeName);
    const nodeId = e.target.parentNode.id;
    const node = graph.filterNodes((n, a) => a.id == nodeId);

    if (node.length <= 0)
        return;

    if (nodeName == null || nodeName == "") {
        if (node[0] == '1')
            return;
        else if (graph.degree(node[0]) > 1) {
            alert('Es dürfen nur Blattknoten gelöscht werden!');
            return;
        }
        graph.dropNode(node[0]);
        return;
    } else {
        do {
            nodeInfoGainInput = prompt("Informationsgewinn hinzufügen:", "");
            nodeInfoGain = isValidInputInfo(nodeInfoGainInput);
        } while (nodeInfoGain === null)
    }

    // Neuer Knoten wird erstellt
    let newNode = graph.addNode(idGenerator(), {
        id: idGenerator(),
        label: nodeName + "\n" + "(" + nodeInfoGain + ")",
        name: nodeName,
        infogain: nodeInfoGain,
        level: graph.getNodeAttribute(node[0], 'level') + 2,
        parentedge: "None",
    });

    setTimeout(() => {

```

```

// Neue Kante wird erstellt
graph.addEdge(node[0], newNode, {
    id: idGenerator(),
    label: "Label\n(Entropie)",
    name: "Label",
    entropy: -1.0,
    parentnode: graph.getNodeAttribute(node[0], 'name'),
    childnode: graph.getNodeAttribute(newNode, 'name'),
    level: graph.getNodeAttribute(node[0], 'level') + 1,
});
setParentEdge(newNode);
}, 200);
});
}, 500);

```

Neue Funktionen bei HelperFunctions.ts

Um die Generierung von Schlüsselwerten und IDs für Knoten und Kanten zu unterstützen und auch die Normalisierung von Strings bei Eingabewerten zu ermöglichen, wurden entsprechende Hilfsfunktionen implementiert:

createRandomString()

Diese Funktion erzeugt eine zufällige Zeichenkette. Die Länge der Zeichenkette ist standardmäßig auf fünf festgelegt. In der while-Schleife wird ein zufälliges Zeichen aus dem vorherigen Zeichengenerator 'characters' ausgewählt und zu der Zeichenkette hinzugefügt. Sobald die Zeichenkette eine Länge von fünf erreicht, wird sie als Rückgabewert zurückgegeben.

normalizeString()

Diese Funktion normalisiert eine Zeichenketten mithilfe von regulären Ausdrücken. Standardmäßig ersetzt sie mehrere aufeinanderfolgende Leerzeichen durch ein einzelnes Leerzeichen. Der Parameter 's' ist der Eingabestring, der normalisiert werden soll. Der Parameter 'regex' gibt an, welche Zeichen ersetzt werden sollen.

```

const createRandomString = (length: number = 5) => {
  const characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
  let s = "";

  while (s.length < length)
    s += characters.charAt(Math.floor(Math.random() * characters.length));
  return s;
}

const normalizeString = (s: string | null, regex: string = "\s+") => {
  if (s == null)
    return null;
  return s.replace(new RegExp(regex, "gi"), "_").trim();
}

```

Neue Schnittstelle bei taskGraph.ts

Diese Funktion initiiert eine asynchrone POST-Anfrage an das Backend. Der Payload der Anfrage

setzt sich zusammen aus dem Graphen, welcher vom Studierenden auf der Benutzeroberfläche aufgebaut wurde, dem Typ 'decisiontree', sowie einer Anweisung 'decisionTreeValidator'. Letztere ruft die entsprechende Funktion im Backend auf und übergibt den Graphen zur Überprüfung der enthaltenen Werte. Die Funktion dient entsprechend als Schnittstelle für die Übertagung des eingegbenen Graphen an das Backend.

```
fetchSolution: async ({}, payload: Object) => {
  const result = await axios({
    method: 'POST',
    url: 'http://localhost:8000/api/decisiontree/decisionTreeValidator',
    headers: {
      "Content-Type": "application/json"
    },
    data: JSON.stringify({ graph: payload, type: 'decisiontree', instruction: 'decisionTreeValidator' })
  });
  return result.data;
}
```

4.5. Ergebnisse

4.5.1. Aktueller Stand

Die Umsetzung des Aufgabentyps konnte in ALADIN umgesetzt werden, wodurch es für die Studierende nun möglich ist, anhand von generierten Datensätzen einen Entscheidungsbaum zu berechnen und diesen auf interaktive Weise aufzubauen und zu lösen.

Tabelle 5. Übersicht über die erfüllten/nicht erfüllten Anforderungen

Anforderung	erfüllt/nicht erfüllt
Zufällige Datensatzgenerierung	erfüllt
Datensatzgenerierung basierend auf vorhandenem Baum	erfüllt
Interaktiver Aufbau des Entscheidungsbaums	erfüllt
Überprüfung des eingegebenen Entscheidungsbaums	erfüllt
Flexible Positionierung der Werte	erfüllt
Berücksichtigung von Mehrfachlösungen	erfüllt
Feedback nach erfolgreicher Lösung	erfüllt
Generierung neuer Aufgaben	erfüllt
Hinweis, um welchen Fehler es sich bei der Eingabe handelt	nicht erfüllt (bzw. nur beim Prototypen umgesetzt)

Sobald der Studierende auf den Aufgabentypen "DecisionTree" auf der Startseite von ALADIN

klickt, erscheinen drei Felder. Das linke Feld enthält die Aufgabenstellung sowie Hinweise zur Bearbeitung und Bedienungshinweise für die Benutzung des Entscheidungsbaums. Auf der rechten Seite erscheint der generierte Datensatz, der für die Berechnung verwendet wird. Unterhalb beider Felder befindet sich der Bereich, in dem der Studierende den Entscheidungsbaum erstellen kann.

Erstellung einer Aufgabe

Wenn der Studierende auf den "Generieren"-Button klickt, wird ein Datensatz und ein Wurzelknoten erzeugt. Bei erneutem Klicken auf den Button wird ein neuer Datensatz und ein neuer Wurzelknoten generiert. Dies bedeutet, dass der bisher aufgebaute Entscheidungsbaum entsprechend entfernt wird.

Index	Aussicht	Temperatur	Luftfeuchtigkeit	Wind	Spiel
0	bewölkt	kalt	normal	ja	ja
1	bewölkt	heiß	hoch	nein	ja
2	regnerisch	kalt	normal	ja	nein
3	sonnig	kalt	normal	nein	ja
4	regnerisch	mild	normal	nein	ja
5	bewölkt	mild	hoch	ja	ja
6	bewölkt	heiß	normal	nein	ja
7	regnerisch	kalt	normal	nein	ja
8	sonnig	kalt	normal	nein	ja
9	sonnig	kalt	normal	nein	ja
10	bewölkt	heiß	hoch	nein	ja
11	bewölkt	mild	hoch	ja	ja
12	sonnig	heiß	hoch	ja	nein
13	sonnig	mild	hoch	nein	nein

Abbildung 15. Ansicht des Aufgabentyps bei der Erstellung einer neuen Aufgabe

Knoten und Kanten bearbeiten

Durch Rechtsklick auf einen Knoten können sowohl die Beschriftung als auch der Informationsgewinn des Knotens geändert werden. Ein Prompt erscheint für die Eingabe der Beschriftung (Label)

des Knotens, gefolgt von einem weiteren Prompt für die Eingabe des Informationsgewinns. Das gleiche Vorgehen gilt auch für die Änderung der Beschriftung und Entropien an den Kannten. Wenn keine Eingabe für die Beschriftung des Knotens erfolgt, wird der Knoten aus dem Baum entfernt, sofern es sich um einen Blattknoten handelt. Falls der Knoten ein Blattknoten ist und eine Klasse eingetragen werden soll, wird beim Prompt für die Eingabe des Informationsgewinns die Bezeichnung "decision" eingetragen, um hervorzuheben, dass es sich um eine Klasse und kein Merkmal handelt.



Abbildung 16. Anzeige eines Prompts für die Bearbeitung eines Knotens

Neue Knoten hinzufügen

Durch Linksklick auf einen Knoten kann ein neuer Knoten direkt unterhalb des ausgewählten Knotens erstellt werden. Ein Prompt wird angezeigt, um die Beschriftung und den Informationsgewinn des neuen Knotens einzugeben. Dabei wird automatisch eine Kante zum neuen Knoten erstellt, der Standardwerte enthält, welche nachträglich geändert werden können.

The screenshot shows a user interface for generating a decision tree from a dataset. On the left, there's a sidebar with sections for 'Aufgabe' (Task), 'Bedienung' (Usage), and buttons for 'Generieren!' (Generate!) and 'Ändern' (Change). On the right, there's a table titled 'Datensatz' (Dataset) with columns: Index, Aussicht (Sight), Temperatur (Temperature), Luftfeuchtigkeit (Humidity), Wind (Wind), and Spiel (Game). The table contains 11 rows of data. A modal dialog box is centered over the interface, containing the text 'localhost:5173', 'Neuer Knoten:' (New Node:), and a input field with the value 'Luftfeuchtigkeit'. At the bottom of the dialog are 'OK' and 'Abbrechen' (Cancel) buttons.

Index	Aussicht	Temperatur	Luftfeuchtigkeit	Wind	Spiel
0	bewölkt	kalt	normal	ja	ja
1	bewölkt	heiß	hoch	nein	ja
2	regnerisch	kalt	normal	ja	nein
3	sonnig	kalt	normal	nein	ja
4	regnerisch	mild	normal	nein	ja
5	bewölkt	mild	hoch	ja	ja
6	bewölkt	heiß	normal	nein	ja
7	regnerisch	kalt	normal	nein	ja
8	sonnig	kalt	normal	nein	ja
9	sonnig	kalt	normal	nein	ja
10	bewölkt	heiß	hoch	nein	ja
			hoch	ja	ja
			hoch	ja	nein
			hoch	nein	nein

Abbildung 17. Anzeige eines Prompts für das hinzufügen eines neuen Knotens

Prüfung der Eingaben

Nachdem der Studierende eine Eingabe getätigt hat, erfolgt automatisch eine Überprüfung. Bei korrekter Eingabe am Knoten oder an der Kante wird das entsprechende Objekt grün hervorgehoben. Andernfalls bleibt der Knoten bzw. die Kante orange, um auf eine nicht korrekte Eingabe hinzuweisen.

Entscheidungsbaum

Aufgabe

Berechnen Sie anhand des generierten Datensatzes einen Entscheidungsbaum mit dem ID3-Algorithmus und erstellen Sie diesen.

- Fügen Sie die korrekten Beschriftungen für Knoten und Kanten hinzu.
- Geben Sie die Entropien und Informationsgewinne mit fünf Dezimalstellen an den entsprechenden Kanten und Knoten an.
- Setzen Sie 'decision' für die Knoten mit zugewiesener Klasse.
- Achten Sie auf die Groß- und Kleinschreibung!

Starten Sie, indem Sie auf 'Generieren!' klicken.

Bedienung

Knoten

Hinzufügen: Linke Maustaste (mit Eingabe)
Ändern: Rechte Maustaste
Löschen: Linke Maustaste (ohne Eingabe) - Nur Blattknoten!

Kanten

Ändern: Rechte Maustaste

Generieren!

Datensatz

Index	Aussicht	Temperatur	Luftfeuchtigkeit	Wind	Spiel
0	bewölkt	kalt	normal	ja	ja
1	bewölkt	heiß	hoch	nein	ja
2	regnerisch	kalt	normal	ja	nein
3	sonnig	kalt	normal	nein	ja
4	regnerisch	mild	normal	nein	ja
5	bewölkt	mild	hoch	ja	ja
6	bewölkt	heiß	normal	nein	ja
7	regnerisch	kalt	normal	nein	ja
8	sonnig	kalt	normal	nein	ja
9	sonnig	kalt	normal	nein	ja
10	bewölkt	heiß	hoch	nein	ja
11	bewölkt	mild	hoch	ja	ja
12	sonnig	heiß	hoch	ja	nein
13	sonnig	mild	hoch	nein	nein

```

graph TD
    Root((Aussicht  
0.1234)) -- "sonnig  
0.1234" --> Sonnig(( ))
    Root -- "bewölkt  
0.0" --> Bewölkt(( ))
    Root -- "regnerisch  
0.9635" --> Wind((Wind  
0.9635))
    Sonnig -- "normal  
0.0" --> Ja1((ja  
decision))
    Sonnig -- "hoch  
0.0" --> Nein1((nein  
decision))
    Wind -- "ja  
0.0" --> Ja2((ja  
decision))
    Wind -- "nein  
0.0" --> Nein2((nein  
decision))
  
```

Abbildung 18. Prüfung des Entscheidungsbaums durch farbliche Hervorhebungen

Meldung bei vollständiger Lösung der Aufgabe

Wenn der Studierende den Entscheidungsbaum korrekt und vollständig aufgebaut hat, wird eine Meldung angezeigt, die den erfolgreichen Abschluss der Aufgabe bestätigt.

Entscheidungsbaum

Aufgabe

Berechnen Sie anhand des generierten Datensatzes einen Entscheidungsbaum mit dem ID3-Algorithmus und erstellen Sie diesen.

- Fügen Sie die korrekten Beschriftungen für Knoten und Kanten hinzu.
- Geben Sie die Entropien und Informationsgewinne mit fünf Dezimalstellen an den entsprechenden Kanten und Knoten an.
- Setzen Sie 'decision' für die Knoten mit zugewiesener Klasse.
- Achten Sie auf die Groß- und Kleinschreibung!

Starten Sie, indem Sie auf 'Generieren!' klicken.

Knoten

Hinzufügen: Linke Maustaste (mit Eingabe)
Ändern: Rechte Maustaste
Löschen: Linke Maustaste (ohne Eingabe) - Nur Blattknoten!

Kanten

Ändern: Rechte Maustaste

Generieren!

Datensatz

Index	Aussicht	Temperatur	Luftfeuchtigkeit	Wind	Spiel
0	bewölkt	kalt	normal	ja	ja
1	bewölkt	heiß	hoch	nein	ja
2	regnerisch	kalt	normal	ja	nein
3	sonnig	kalt	normal	nein	ja
4	regnerisch	mild	normal	nein	ja
5	bewölkt	mild	hoch	ja	ja
6	bewölkt	heiß	normal	nein	ja
7	regnerisch	kalt	normal	nein	ja
8	sonnig	kalt	normal	nein	ja
9	sonnig	kalt	normal	nein	ja
10	bewölkt	heiß	hoch	nein	ja
11	bewölkt	mild	hoch	ja	ja
			hoch	ja	nein
			hoch	nein	nein

⊕ localhost:5173

Herzlichen Glückwunsch!
Sie haben die Aufgabe vollständig gelöst!

Weitere Aufforderungen von localhost:5173 verbieten

OK

```

graph TD
    Root["Luftfeuchtigkeit  
(0.9183)"] -- "normal  
(0.0)" --> ja1["ja  
(decision)"]
    Root -- "hoch  
(0.0)" --> nein1["nein  
(decision)"]
    bewolkte["bewölkt  
(0.0)"] --> ja2["ja  
(decision)"]
    bewolkte --> nein2["nein  
(decision)"]
    regnerisch["regnerisch  
(0.94566)"] --> ja3["ja  
(0.0)"]
    regnerisch --> nein3["nein  
(0.0)"]
  
```

Abbildung 19. Meldung an den Studierenden bei erfolgreiche Lösung der Aufgabe

Hilfestellungen für den Studierende

Wenn der Studierende Unterstützung bei der Lösung der Aufgabe benötigt, steht oben rechts ein Hinweis-Button als Glühbirne zur Verfügung. Durch einen Klick darauf erhält der Studierende Hinweise für die Bearbeitung der Aufgabe. Die Hinweise umfassen das Vorgehen des ID3-Algorithmus, Informationen zur Entropie sowie die Berechnungsformeln für den Informationsgewinn und die Entropie.

Entscheidungsbaum

Aufgabe

Berechnen Sie anhand des generierten Datensatzes einen Entscheidungsbaum mit dem ID3-Algorithmus und erstellen Sie diesen.

- Fügen Sie die korrekten Beschriftungen für Knoten und Kanten hinzu.
- Geben Sie die Entropien und Informationsgewinne mit fünf Dezimalstellen an den entsprechenden Knoten und Knoten an.
- Setzen Sie 'decision' für die Knoten mit zugewiesener Klasse.
- Achten Sie auf die Groß- und Kleinschreibung!

Starten Sie, indem Sie auf 'Generieren!' klicken.

Bedienung

Knoten

Hinzufügen: Linke Maustaste (mit Eingabe)
Ändern: Rechte Maustaste
Löschen: Linke Maustaste (ohne Eingabe) - Nur Blattknoten!

Kanten

Ändern: Rechte Maustaste

Generieren!

Datensatz

Index	Aussicht	Temperatur	Luftfeuchtigkeit		
0	bewölkt	kalt	normal		
1	bewölkt	heiß	hoch		
2	regnerisch	kalt	normal		
3	sonnig	kalt	normal		
4	regnerisch	mild	normal		
5	bewölkt	mild	hoch		
6	bewölkt	heiß	normal		
7	regnerisch	kalt	normal		
8	sonnig	kalt	normal		
9	sonnig	kalt	normal		
10	bewölkt	heiß	hoch	nein	ja
11	bewölkt	mild	hoch	ja	ja
12	sonnig	heiß	hoch	ja	nein
13	sonnig	mild	hoch	nein	nein

Hinweis 1: ID3-Algorithmus

- 1. Schritt: Berechnung Entropie
- 2. Schritt: Berechnung Informationsgewinn
- 3. Schritt: Auswahl des besten Merkmals
- 4. Schritt: Rekursive Anwendung (1. - 3. Schritt wiederholen)

```

graph TD
    Root((Aussicht  
0.39718)) --> Sonnig((sonnig  
0.9183))
    Root --> Bewölkt((bewölkt  
0.0))
    Root --> Regnerisch((regnerisch  
0.94566))
    Bewölkt --> JaDecision1[ja (decision)]
    Sonnig --> Luftfeuchtigkeit((Luftfeuchtigkeit  
0.9183))
    Luftfeuchtigkeit --> Normal((normal  
0.0))
    Luftfeuchtigkeit --> Hoch((hoch  
0.0))
    Normal --> JaDecision2[ja (decision)]
    Normal --> NeinDecision2[nein (decision)]
    Hoch --> JaDecision3[ja (decision)]
    Hoch --> NeinDecision3[nein (decision)]
    Regnerisch --> Wind((Wind  
0.94566))
    Wind --> Ja0[ja (0.0)]
    Wind --> Nein0[nein (0.0)]
    Ja0 --> NeinDecision4[nein (decision)]
    Nein0 --> JaDecision4[ja (decision)]
  
```

Abbildung 20. Anzeige einer Hilfestellung für das Vorgehen des ID3-Algorithmus

4.5.2. Aufgetretene Probleme

Die Umsetzung des neuen Aufgabentyps in ALADIN stellte eine Herausforderung dar, insbesondere bei der anfänglichen Schwierigkeit beim Verständnis des Frameworks. Das Fehlen einer vorhandenen Dokumentation machte den Einstieg schwierig und erschwerte somit die Einarbeitung.

Da das Verständnis nicht ausreichend vorhanden war, insbesondere bei der Handhabung der Datenübertragung zwischen Frontend (CARPET) und Backend (ALADIN), wurde um das Hindernis zu umgehen, eigenständig eine Schnittstelle entwickelt, um die notwendigen Daten, in diesem Fall die Übermittlung des aufgebauten Entscheidungsbaums vom Studierenden, übertragen zu können.

Aufgrund der zeitintensiven Einarbeitung konnten einige geplante Erweiterungen des Aufgabentyps nicht umgesetzt werden. Zu den nicht realisierten Erweiterungen gehören unter anderem die Klassifizierung neuer Datensätze sowie die Anwendung des Pruning-Verfahrens nach dem Aufbau des Entscheidungsbaums als erweiterte Aufgabe.

4.6. Ausblick

Durch die bisherige Entwicklung des Aufgabentypen wurde eine interaktive Plattform geschaffen, die es Studierenden ermöglicht, den berechneten Entscheidungsbaum mit dem ID3-Algorithmus zu üben.

Um den Lernumfang und die Anwendungsvielfalt zu erweitern, bietet das Projekt weitere Möglichkeiten zur Erweiterung:

1. Anzeigen des Lösungsbaums bzw. weitere mögliche Lösungsoptionen

Eine Erweiterung des Aufgabentyps könnte darin bestehen, dass nach Lösen einer Aufgabe der Lösungsbaum mit angezeigt wird. Insbesondere bei Aufgaben, bei denen Mehrfachlösungen vorhanden sind, könnte dies für die Studierenden interessant sein. Dadurch wird nicht nur die individuelle Lösung betrachtet, sondern auch alternative Lösungsoptionen. Dies könnte ein umfassenderes Verständnis für die Berechnung verschiedener Entscheidungsbaumstrukturen ermöglichen.

2. Hinweise zu den Fehlern im Baum dem Studierenden angeben

Eine weitere mögliche Erweiterung könnte die Implementierung von Hinweisen zu Fehlern bei den Eingaben und dem Aufbau des Entscheidungsbaums sein. Dadurch werden Studierende nicht nur darauf hingewiesen, an welcher Stelle ein Fehler aufgetreten ist, sondern erhalten auch eine Erklärung, warum der Fehler aufgetreten ist. Dies kann den Lerneffekt der Studierenden weiter fördern. Erste Ansätze zur Umsetzung von Hinweisen zu den Fehlern wurden bereits beim Prototypen entwickelt.

3. Pruning-Verfahren als Erweiterung des Aufgabentyps

Nachdem ein Entscheidungsbaum aufgebaut wurde, könnte eine Erweiterung des Aufgabentyps ermöglichen, den Baum mithilfe verschiedener Pruning-Verfahren zu "stutzen". Dies könnte Studierenden das Verständnis dafür vermitteln, wie Pruning-Techniken angewendet werden können, um übermäßig komplexe Bäume zu vereinfachen.

4. Klassifizierung neuer Datensätze als Erweiterung des Aufgabentyps

Eine mögliche Fortführung und Erweiterung des Aufgabentyps könnte darin bestehen, nachdem die Studierenden den Entscheidungsbaum durch ein Pruning-Verfahren gestutzt haben, neue Datensätze anhand des gegebenen Entscheidungsbaums zu klassifizieren. Dadurch könnte auch die praktische Anwendung des Entscheidungsbaums bei den Studierenden hervorgehoben werden.

5. Implementierung weiterer Algorithmen

Um den Lernumfang zu erweitern, könnten neben dem ID3-Algorithmus zusätzliche Algorithmen implementiert werden. Dies würde den Studierenden die Möglichkeit bieten, verschiedene Ansätze zur Berechnung von Entscheidungsbäumen zu üben. Beispiele für weitere Algorithmen könnten der C4.5-Algorithmus sein, eine Weiterentwicklung des ID3-Algorithmus oder der Random-Forest-Algorithmus.

6. Vielfältige Aufgabenthemen

Statt nur mit einem vorgegebenen Datensatz zu arbeiten, könnte aus verschiedenen "Themengebieten" Datensätzen ausgewählt werden. Dies würde die Generierung unterschiedlicher Aufgabenstellungen und den Schweregrad ermöglichen, wie z.B. die Kreditwürdigkeitsprüfung oder andere relevante Szenarien. Dies fördert sowohl die Vielseitigkeit des Lernmaterials als auch ermöglicht es, unterschiedliche Berechnungsfälle zu üben.

5. Aufgabentyp - Regex Puzzle

5.1. Aufgabenstellung

5.2. Aufgabenbeschreibung

Im Rahmen des Projektseminars war es meine Aufgabe, eine Möglichkeit zum Üben und Lernen von regulären Ausdrücken (RegEx) zu entwerfen und umzusetzen. Die genaue Beschaffenheit der Aufgabe war nicht festgelegt und lag in meinem Ermessen. Ich entschied mich für die Entwicklung eines RegEx-Kreuzworträtsels, bei dem die Nutzer zu gegebenen Zeilen- und Spalten-RegExen passende Zeichenketten finden müssen. Dieses erfolgt in Form eines Rasters, in welchem der Nutzer in jedes Feld den passenden Buchstaben eingeben muss. Zu jedem eingegebenen Buchstaben bekommt der Nutzer dann ein Feedback über die Richtigkeit der Eingabe.

5.3. Anforderung

1. **Zufällige Generierung von regulären Ausdrücken:** Die Anwendung sollte in der Lage sein, zufällige reguläre Ausdrücke zu generieren, die als Grundlage für das RegEx-Kreuzworträtsel dienen.
2. **Validieren der Nutzereingaben in Echtzeit:** Während die Nutzer die Buchstaben in das Kreuzworträtsel eingeben, sollte die Anwendung in Echtzeit die Eingaben validieren, um sicherzustellen, dass sie den RegEx-Mustern entsprechen.
3. **Abwechslungsreiche RegEx-Typen mit unterschiedlicher Komplexität:** Die generierten regulären Ausdrücke sollten eine Vielfalt an RegEx-Typen und unterschiedliche Schwierigkeitsgrade aufweisen, um eine Herausforderung für die Nutzer zu bieten.
4. **Übersichtliche und lesbare Darstellung der RegEx:** Die RegEx-Muster sollten auf eine übersichtliche und leicht lesbare Weise dargestellt werden, damit die Nutzer sie verstehen und interpretieren können.
5. **Lösbarkeit der Aufgabe sicherstellen:** Die generierten RegEx-Kreuzworträtsel sollten lösbar sein, wobei die RegEx-Muster angemessen herausfordernd, aber nicht unmöglich sein sollten.

5.4. Planung

Während des Projekts trafen wir uns wöchentlich, um den Fortschritt zu besprechen, mögliche Probleme zu identifizieren und die nächsten Schritte zu planen. Diese Treffen waren entscheidend für die Koordination und den reibungslosen Ablauf des Projekts. Im wesentlichen existierten drei Phasen im Projektverlauf:

1. Entscheidung und Konkretisierung eines Regex-Aufgabentyps

Zuerst habe ich mir Gedanken über den Aufgabentyp gemacht, den ich wählen möchte, um reguläre Ausdrücke zu üben. Dabei war es mir wichtig, einen Aufgabentyp zu wählen, der nicht nur dabei hilft, reguläre Ausdrücke zu üben, sondern auch Spaß macht. Da reguläre Ausdrücke oft als trocken und schwer zugänglich wahrgenommen werden, suchte ich nach einer Möglichkeit, sie spielerisch und ansprechend zu präsentieren. Dabei diente mir das hexagonförmige

Regex-Puzzle, das 2013 von Dan Gulotta für das MIT Mystery Hunt entwickelt wurde, als Inspiration. Da sich jedoch ein hexagonförmiges Puzzle, das zufällig generiert werden kann, als zu komplex erwies, entschied ich mich für einen zweidimensionalen Ansatz in klassischer Rasterform.

2. Realisierung der Aufgabe in einem externen Programm

Zunächst entwickelte ich einen Prototypen, der als eigenständiges Programm unabhängig von Aladin funktionierte. Zu dieser Phase gehörte das Festlegen auf eine Programmiersprache und das Konzipieren einer effizienten Programmstruktur. Nachdem diese Überlegungen angestellt wurden, begann ich die Funktionen zu implementieren. Dazu passte ich den Algorithmus zur Generierung von RegEx an, erstellte die Puzzle-Oberfläche und implementierte schließlich die Validierung der Nutzereingaben.

3. Integrieren in Aladin und Carpet

Nachdem diese Funktionen fertiggestellt waren, begann ich mit der Integration des Projekts in ALADIN. Dies benötigte die intensive Einarbeitung in das Framework, sowie das Verständnis für dessen Aufbau und Funktionsweise. Es mussten einige grundlegende Änderungen und Anpassen vorgenommen werden, um sicherzustellen, dass die Integration die Funktionsweise von ALADIN nicht behindert.

Nachdem diese drei Schritte durchlaufen wurden, waren nur noch einige Veränderungen und kleinere Fehlerbehebung vorzunehmen, bevor das Projekt final präsentiert wurde.

5.5. Technische Umsetzung

5.5.1. Alleinstehendes Programm

5.5.1.1. Python

Python war die naheliegende Wahl für die Umsetzung, da ich ein Raster erstellen und RegEx in den Zeilen- und Spaltenköpfen abbilden musste und ich am meisten Erfahrung in dieser Programmiersprache besitze. Dank der Vielzahl an vorhandenen Bibliotheken war Python besonders gut geeignet. Tkinter, eine Bibliothek für die Entwicklung grafischer Benutzeroberflächen in Python, bot eine einfache Möglichkeit, ein flexibles Raster zu erstellen und beliebig mit Werten zu füllen.

5.5.2. Algorithmus zur RegEx-Generierung

5.5.2.1. Allgemeiner Generierungsansatz

Um zufällige RegEx anzeigen zu können, war ein Algorithmus erforderlich, der diese generieren konnte. Statt von Grund auf einen eigenen Algorithmus zu entwickeln, entschied ich mich dafür, einen bereits vorhandenen Open-Source-Algorithmus zu verwenden und ihn anzupassen bzw. zu erweitern. Dafür habe ich den Algorithmus von [bcaluneo](#) auf GitHub verwendet und angepasst. Dieser Algorithmus nimmt eine Zeichenkette entgegen und generiert auf dieser Basis reguläre Ausdrücke. Da dieser jedoch nicht meinen Anforderungen entsprach und sehr statisch generierte, musste ich einige Änderungen vornehmen. So musste ich diesen zuerst von Typescript in Python umschreiben, sowie die Vielfalt und Häufigkeiten anpassen.

Der Algorithmus funktioniert folgendermaßen:

1. Ein Raster wird mit zufälligen Buchstaben und Zahlen gefüllt

W	T	R	K	P
L	S	V	A	C
J	O	E	Q	Y
G	Z	V	K	B
D	W	F	W	X

(hier beispielhaft nur Buchstaben)

1. Spalten und Zeilen werden als Zeichenketten kombiniert z.B hier 1. Zeile: WTRKP

W	T	R	K	P
---	---	---	---	---

Die Zeichenkette wird dem Algorithmus übergeben, der für jedes Zeichen einen zufälligen und passenden regulären Ausdruck generierte. Dabei war es wichtig sicherzustellen, dass die Überschneidung von Zeilen- und Spaltenausdrücken funktioniert. Dies wurde erreicht, indem die Zeichenketten nacheinander dem Algorithmus übergeben wurde, wodurch für jedes Zeichen im Raster ein entsprechender RegEx-Ausdruck generiert wurde. Die Generierten Ausdrücke werden dann an das Frontend gesendet, wo sie dann als Zeilen- und Spaltenköpfe angezeigt werden.

5.5.2.2. Umsetzung Algorithmus

Der Algorithmus setzt sich aus mehreren einzelnen Funktionen zusammen, die verschiedene Arten von RegEx generieren. Jede Funktion erhält ein Zeichen als Parameter und erzeugt einen regulären Ausdruck, der dieses Zeichen enthält.

```
def generate_character_set(char):

    num_chars_before = random.randint(0, 6)
    num_chars_after = random.randint(0, 6)

    start_char = chr(max(ord('A'), ord(char) - num_chars_before))
    end_char = chr(min(ord('Z'), ord(char) + num_chars_after + 1))

    return f'[{start_char}-{end_char}]'
```

In diesem Ausschnitt der `generate_character_set()`-Funktion wird ein Zeichenbereich generiert, beispielsweise [T-U]. Dieser Zeichenbereich umfasst alle Zeichen zwischen T und U in der ASCII-Zeichenfolge.

```
def generate_regex(word):
    result = "
```

```

skip = False
regexList = []

for i in range(len(word)):
    if skip:
        skip = False
        continue
    roll = random.random()
    if roll < 0.12:
        regexPart = generate_negated_character_set(word[i])
    else:
        roll = random.random()
        if roll < 0.25:
            if word[i].isnumeric():
                regexPart=generate_number_range(int(word[i]))
            else:      regexPart = generate_character_set(word[i])

```

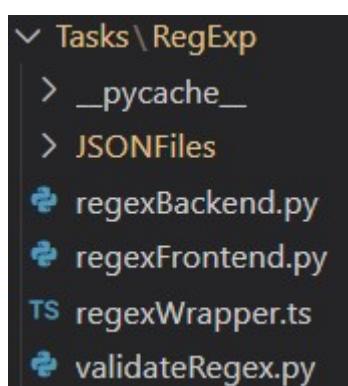
In diesem (unvollständigem) Ausschnitt der `generate_regex()`-Funktion wird eine Zahl zwischen 0 und 1 gewürfelt und basierend auf der gewürfelten Zahl die verschiedenen Funktionen zum Generieren von einzelnen RegEx-Teilen aufgerufen (siehe [\[generate_character_set\]](#)). Somit wird für jede Zeichenkette ein gemischter regulärer Ausdruck gebaut.

Die folgenden RegEx-Arten können nach aktuellem Stand von dem Algorithmus generiert werden:

- `.` → Ein beliebiges Zeichen
- `?` → Zeichen(gruppe) vor dem Fragezeichen ist optional und darf maximal einmal vorkommen
- `[A|B]` → "Oder": Nur eins der beiden Zeichen darf vorkommen
- `[ABC]` → Zeichenauswahl: Ein der eingeklammerten Zeichen darf vorkommen
- `[^ABC]` → Negation, alle Zeichen zulässig bis auf die in den Klammern stehenden
- `[A-D]` → Zeichenbereich, ein Zeichen im Bereich des in den Klammern stehendem Ausdrucks kommt vor
- `[0-9]` → Zahlenbereich, eine Zahl in dem Bereich kommt vor

5.5.3. Integrieren in Aladin und Carpet

Erklärung der Dateien



- **JSONFiles** → Dies ist der Ordner, in welchem die JSON Dateien zum Datenaustausch zwischen Frontend und Backend gespeichert werden.
- **regexBackend.py** → In dieser Datei befinden sich der Algorithmus zur Generierung der regulären Ausdrücke sowie die Funktion zum Validieren der Nutzereingaben
- **regexFrontend.py** → Diese Datei übernahm ursprünglich das Erstellen der Oberfläche, in der mein Puzzle in einem alleinstehendem Programm angezeigt werden kann. Die Elemente sind noch vorhanden und werden noch generiert, da diese zum Funktionieren benötigt werden. Angezeigt werden diese Elemente jedoch nicht mehr, sondern es wird nur sichergestellt, dass die richtigen Zeichenketten an den Algoirthmus zum Geneieren der RegEx geschickt werden.
- **regexWrapper.ts** → In dieser Datei befindet sich mein Typescript-Wrapper, welcher sicherstellt, dass mein Python Code in ALADIN aufgerufen wird. Diese Datei dient als Schnittstelle zwischen ALADIN und mein Python-Progamm.
- **validateRegex.py** → Hier wird die benötigte Validierungsmatrix erstellt, welche benötigt wird, um die Nutzereingaben vom Frontend farblich hervorzuheben, nachdem diese in meiner **regex-Backend.py** Datei geprüft wurden.

TypeScript:

Da Aladin und Carpet mittels TypeScript geschrieben wurden, musste ich Anpassungen treffen, sodass mein Python-Programm integriert werden konnte, ohne die Funktionsweise ALADINs zu beeinträchtigen. Dafür nutzte ich einen Wrapper, der in TypeScript geschrieben war, um mein Python-Programm mit den vom Nutzer eingegebenen Konfigurationsparametern über Konsolenbefehle aufzurufen. Diesen Ansatz benutze ich um mein Puzzle anfangs zu generieren und nach einer Nutzereingabe zu validieren.

Hier beispielhaft ein Ausschnitt der Wrapper-Funktion zum Generieren des Puzzles:

```
export function generateRegexPuzzle(regexData: RegexData) {
  try {

    const { columns, rows, seed } = regexData;
    const columnCount = columns;
    const rowCount = rows;
    const userseed = seed || Math.floor(Math.random() * 10000);

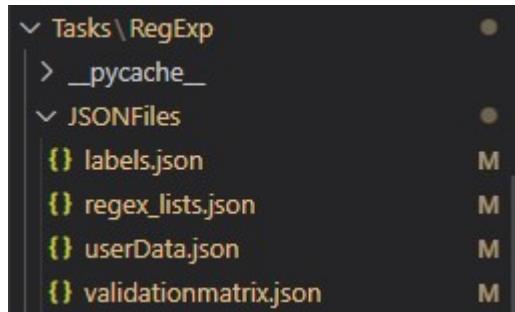
    const pythonScriptPath = 'src/Tasks/RegExp/regexFrontend.py';
    const argumentsToPythonScript = [columnCount.toString(), rowCount.toString(),
      userseed.toString()];
    runPythonScript(pythonScriptPath, argumentsToPythonScript);
    ...
  }
...
}
```

Das Python-Programm **regexFrontend.py** wird über einen Konsolenauftruf mit den vom Nutzer eingegebenen Parametern für Spalten (Columns), Zeilen (Rows) und den Seed gestartet.

JSON:

Da reguläre Ausdrücke viele Sonderzeichen enthalten und dies das Umschicken zwischen Frontend und Backend erschwerte, musste ich vorübergehend JSON-Dateien verwenden, damit Backend und Frontend kommunizieren können. So werden Nutzereingaben sowie Konfigurationsparameter in JSON-Dateien geschrieben und an anderer Stelle ausgelesen. Dies ist nicht der optimale Weg, jedoch eine vorübergehende Lösung, damit die Kommunikation zwischen Python und Typescript, sowie Backend und Frontend fehlerfrei funktionieren.

So existieren aktuell vier vorübergehende JSON Dateien:



- **labels.json** → Datei in welche Zeilen- und Spaltenregex geschrieben werden
- **regex_lists.json** → Datei, in welche Zeilen- und Spaltenregex als Liste geschrieben werden, die durch die Validierungsfunktion ausgelesen und zur Überprüfung verwendet werden kann
- **userData.json** → Datei, die jede Nutzereingabe in Matrixform abspeichert, sodass eine Validierungsmatrix basierend auf den Nutzereingaben erstellt werden kann
- **validationmatrix.json** → Datei, die die Validierungsmatrix erhält sodass durch das Frontend das färben der Felder vorgenommen werden kann.

5.5.3.1. Validierung

Zum Validieren nutzte ich eine Einfache Funktion: `check_input()`, die das vom Nutzer eingegebenes Zeichen, sowie den zugehörigen Zeilen- und Spaltenregex als Parameter übermittelt bekommt. Auf Basis dieser wird geprüft, ob das Nutzerzeichen beiden RegEx genügt und entsprechend ein *True* oder *False* zurückgegeben.

```
def check_input(user_input, regexRow, regexColumn):  
    flags = re.IGNORECASE  
    match_column = re.match(regexColumn, user_input, flags)  
    match_row = re.match(regexRow, user_input, flags)  
  
    if match_column is not None and match_row is not None:  
        if match_column.group() == user_input and match_row.group() == user_input:  
            return True  
    return False
```

Daraus wird im Folgenden in `validateRegex.py` eine Matrix erstellt, die zu jedem Feld im Raster entweder ein *True*, *False* oder *NULL* enthält. True und False, falls die Nutzereingabe jeweils richtig oder falsch ist und NULL, für noch offene Felder, ohne eingebenen Wert. Basierend auf der erstellten Matrix werden dann die Eingabefelder im Frontend rot, grün oder weiß gefärbt.

```

def createValidationMatrix(userData, regexListAllRows, regexListAllColumns):
    matrixColumns = len(userData[0]) if userData else 0
    matrixRows= len(userData)
    validationmatrix = [[None for _ in range(matrixColumns)] for _ in
range(matrixRows)]
    for row_index in range(matrixRows):
        for col_index in range(matrixColumns):
            user_input = userData[row_index][col_index]

            if user_input is None:
                validationmatrix[row_index][col_index] = None
            elif rb.check_input(user_input, regexListAllColumns[col_index][row_index],
regexListAllRows[row_index][col_index]):
                validationmatrix[row_index][col_index] = True
            else:
                validationmatrix[row_index][col_index] = False
    return validationmatrix

```

Bei einem 3 x 3 - Puzzle könnte die Validierungsmatrix dann folgendermaßen aussehen:

```

[[false, true, false],
[null, null, null],
[null, null, null]]

```

5.5.4. Schwierigkeiten und Probleme der Umsetzung

Das Generieren von regulären Ausdrücken ist eine sehr komplexe Aufgabe und obwohl das Projekt schlussendlich erfolgreich war und das Rätsel fast Fehlerfrei läuft, gibt es einige Funktionen, die nicht bzw. schwer umgesetzt werden konnten.

Generierung komplexer RegEx

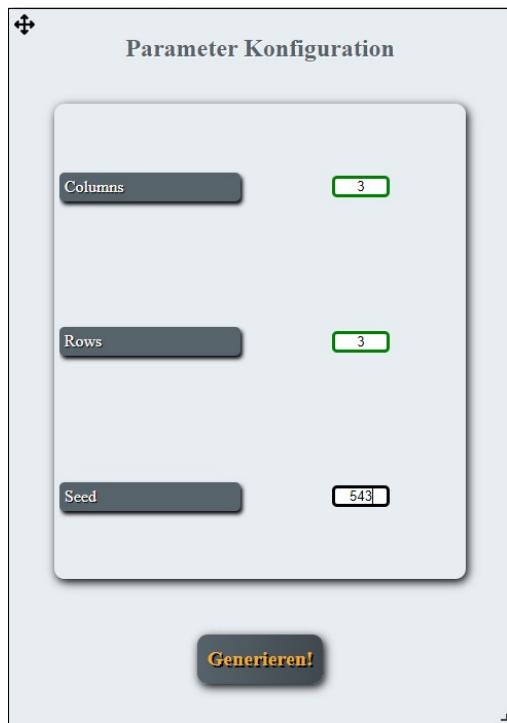
Ziel war es, eine breite Palette von RegEx-Arten in das Puzzle zu integrieren. Dazu erweiterte ich den Algorithmus während des Projekts kontinuierlich um weitere RegEx-Varianten. So konnte ich in meinem Prototypen viele weitere Arten generieren. Die neue Vielfalt an Ausdrücken führte jedoch aufgrund der Vielzahl an Sonderzeichen zu Kombabilitätsproblemen, wodurch das Programm nicht mehr wie erwartet funktionierte. Da ich bisher keine Lösung für dieses Problem gefunden habe, insbesondere unter Verwendung von JSON-Dateien als Kommunikationsweg zwischen Backend und Frontend, musste ich bei der Umsetzung auf einfachere RegEx-Arten zurückgreifen.

Bestehende Bugs

Bei der Integration von dem Puzzle in ALADIN kamen vereinzelt Fehler auf, deren Ursache unklar waren und somit nicht behoben werden konnten. Dazu gehörte der Bug welcher die erste Nutzereingabe in dem Rätsel überschreibt, da bei dieser, aus bisher unerklärlichem Grund, zwei Anfragen gesendet wurden.

5.6. Ergebnisse

5.6.1. Parameter Konfiguration



In diesem Fenster kann der Nutzer entscheiden, wie viele Zeilen und Spalten sein generiertes Rätsel haben soll. Zusätzlich gibt es die Möglichkeit einen bestimmten Seed einzugeben, durch welchen ein bestimmtes Puzzle reproduzierbar ist.

5.6.2. RegEx Puzzle

The screenshot shows a 4x4 grid for a regular expression puzzle. The first row is empty. The second row contains the expression $(P|H)P[^ZIC]$. The third row contains the expression $[G-Q].T^*$. The fourth row contains the expression $[^T-X][S-W][SGT]$. The columns are labeled with expressions: Column 1 is empty; Column 2 is $(H|A)[G-R]M$; Column 3 is $[O-Q].[UQJ]$; Column 4 is $[^S-V][T][^VMB]$.

$(P H)P[^ZIC]$			
$[G-Q].T^*$			
$[^T-X][S-W][SGT]$			

Die Konfigurationsparameter werden vom Backend verarbeitet, der Algorithmus generiert RegEx basierend auf dem Seed und das Rätsel wird schlussendlich angezeigt. Der Nutzer kann nun jedes

Feld ausfüllen

5.6.3. Validierung der Nutzereingaben

Die Validierung der Nutzereingaben erfolgt in Echtzeit, das heißt, sobald der Nutzer ein Zeichen in ein Feld einträgt, wird dieses vom Backend auf Korrektheit geprüft. Entsprechend wird das Eingabefeld grün markiert, wenn die Eingabe korrekt ist, und rot, wenn sie falsch ist. Hierfür nutzte ich die Python Regex-Bibliothek re, mit deren Hilfe ich durch alle Nutzereingaben iteriere und diese jeweils gegen die dazugehörigen Zeilen- und Spaltenregex prüfe. Dies ermöglicht es, dem Nutzer sofort Rückmeldung zu geben, ob das eingegebene Zeichen den Regeln entspricht.

♦:				
	(H A) 0-6 R*	[O-Q T QJ]		[^S-V D [^VNB]
(P H)P[^ZIC]	H	P	A	
[1-4](T K)D?	3	T	u	
[^B-M](X T)[^K-U]				

5.7. Ausblick

Hinweise und Hilfestellung zur Aufgabe:

Eine wesentliche Verbesserung besteht in der Bereitstellung von Hilfestellungen und einer Legende zu den verschiedenen RegEx-Typen, um auch Nutzern mit geringen Vorkenntnissen beim Lösen des Rätsels zu helfen. Außerdem ist es sinnvoll, eine klare Aufgabenstellung hinzuzufügen, damit die Erwartungen an den Nutzer eindeutig sind.

Erweiterung der Komplexität und Vielfalt der RegEx:

Derzeit deckt der Algorithmus nur eine begrenzte Anzahl von RegEx-Typen ab. Dies könnte in Zukunft erweitert werden, um die Vielfalt zu erhöhen. Obwohl ich einige weitere RegEx-Arten ausprobiert habe, die die Komplexität erhöhen würden, führte die Vielfalt an verwendeten Sonderzeichen zu Kompatibilitätsproblemen. Daher blieb ich bei der einfacheren Alternative.

Eindeutigkeit der Lösung:

Eine weitere Möglichkeit, die Schwierigkeit des Rätsels zu erhöhen, wäre, sicherzustellen, dass das Puzzle nur eine Lösung hat und nicht, wie derzeit, mehrere Lösungen zulässig sind. Dies ist jedoch

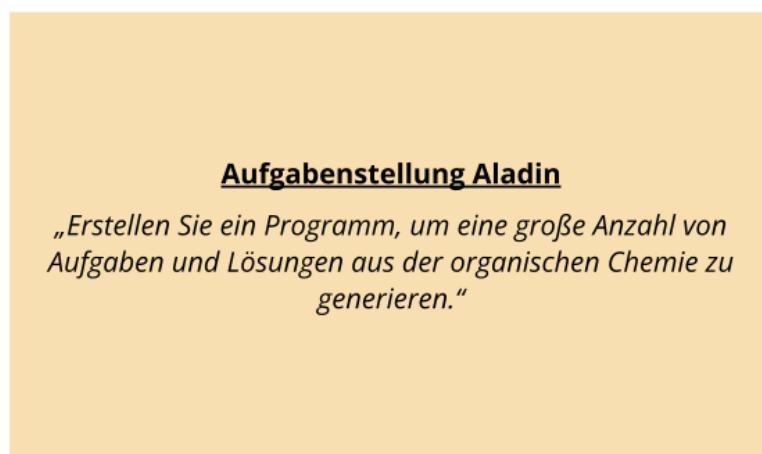
eine komplexe Aufgabe, die sorgfältig durchdacht werden muss, um den Lösungsraum weiter einzuschränken. Mögliche Ansätze könnten die Einführung eines Zeichenpools sein, aus dem die Nutzer festgelegte Zeichen zur Lösung verwenden müssen. Außerdem könnte der Algorithmus so angepasst werden, dass er nur ein bestimmtes Zeichen pro Feld zulässt, indem immer das Komplementär eines RegEx gebildet wird. Zum Beispiel könnte das Komplementär von [ABC] [^A-B] lauten. Dies sind einige Möglichkeiten, um die Schwierigkeit des Rätsels in Zukunft weiter zu erhöhen.

6. Aufgabentyp - Chemie

6.1. Aufgabenstellung

Die Aufgabe und das dazugehörige Programm wurden in Zusammenarbeit mit Prof. Dr. Teichert der Technischen Universität Chemnitz entwickelt. Maßstäbe und Begrenzungen der Aufgabenstellung richten sich inhaltlich nach seinen Anforderungen und Wünschen.

Chemie Generator



HTW D

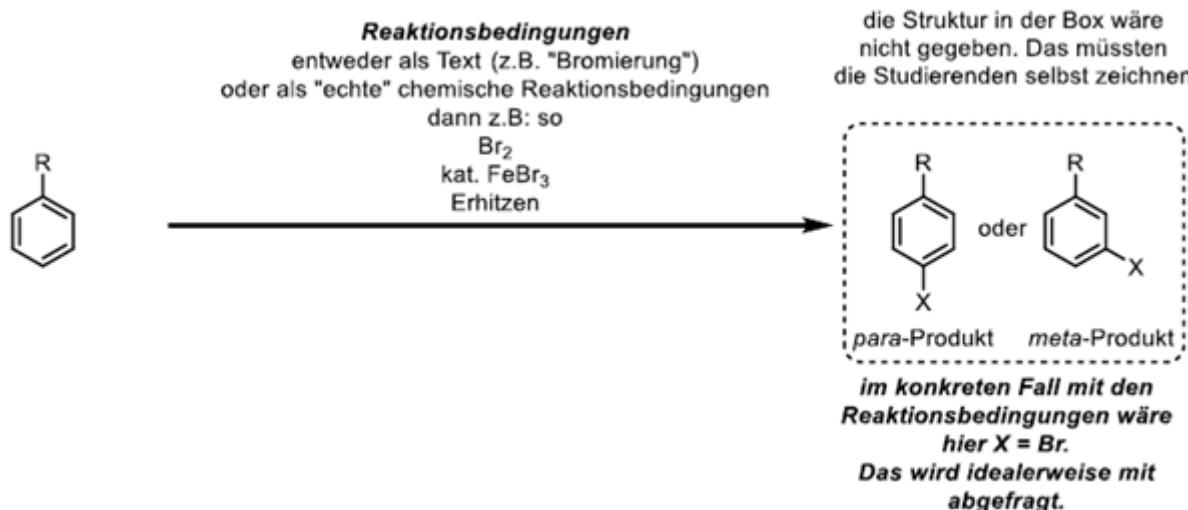
6.2. Aufgabenbeschreibung

Die Aufgabe erforderte es, ein Programm zu erstellen, welches aus einer endlichen, vorgegebenen Liste zufällig Stoffe auswählte und diese unter Einhaltung der chemischen Regeln in einer Reaktion darstellte. Diese Reaktion basiert auf dem Konzept der elektrophilen aromatischen Substitution und das Ziel besteht darin, dass der Student reaktionsbeschreibende Eigenschaften nennen und das fertige Reaktionsprodukt in einem implementierten Editor zeichnen muss. Diese zwei Aufgabenanteile sollen, anschaulich und intuitiv in der Verwendung, dargestellt werden.

6.2.1. Chemischer Hintergrund

Die elektrophile Substitution gehört zur Gruppe der Substitutionsreaktionen in der organischen Chemie. Ein Atom oder eine Atomgruppe wird durch ein elektrophiles Teilchen ersetzt. Ein Elektrophil hat eine positive Ladung oder Teilladung und ist daher bestrebt, Elektronen aufzunehmen (elektronenliebend). Das Programm beinhaltet ausschließlich die Reaktion mit Benzolringen, da diese eine der häufigsten Reaktionen dieses Typs sind. Dabei wird der Benzolring von elektrophilen Teilchen angegriffen und ein Wasserstoffatom verdrängt. Solch eine Reaktion beinhaltet abgesehen von Benzolring drei weitere Ausgangsstoffe. Die Reagenz (X) ist der Stoff der Reaktionsbedingung, in diesem Beispiel handelt es sich um eine Bromierung, folglich ist die Reagenz ein Brommolekül. Der Katalysator spaltet besagte Reagenz auf, um Reaktionen zu ermöglichen, in diesem Fall ist dies

der Katalysator FeBr₃. Der letzte Ausgangsstoff ist der Erstsubstituent ®, dieser ist sehr variabel und kann ein Ion oder ein ganzes Molekül sein. In der Reaktion reagiert sowohl der Erstsubstituent, als auch die Regenz an den Benzolring. Die Reagenz reagiert relativ zum Erstsubstituenten an einer von drei möglichen Positionen: Para, Meta, oder Ortho. Das ist abhängig von der Reagenz selbst, dem Erstsubstituenten, sowie jeglichen Reaktionsbedingungen.



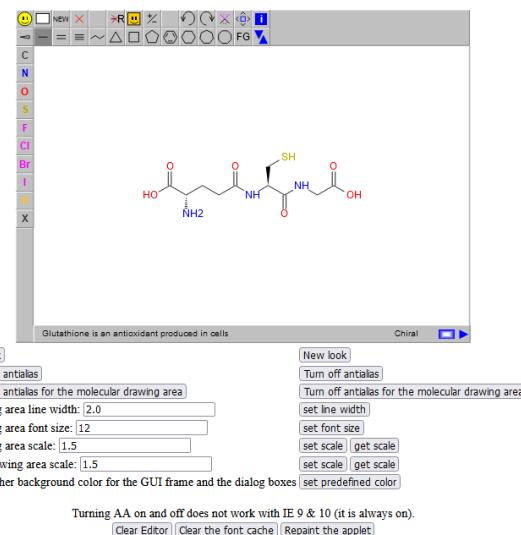
6.3. Anforderung

Die wichtigste Anforderung ist das intuitive Navigieren. Eine benutzerfreundliche Benutzeroberfläche sollte eine Navigation bieten, die für den Benutzer leicht verständlich ist. Dies umfasst klar gekennzeichnete Aufgabenfelder und Navigationselemente, die dem Benutzer eine nahtlose Interaktion mit dem System ermöglichen. Ein weiterer wichtiger Aspekt ist die wiederholbare Nutzung des Systems. Die Benutzeroberfläche sollte so gestaltet sein, dass Benutzer das System wiederholt nutzen können. Dies erfordert eine konsistente Strukturierung der Benutzeroberfläche und eine klare Darstellung von Funktionen und Optionen, sodass eine Nutzung ohne ständiges Neuladen der Seite erfolgen kann. Die Verständlichkeit der Aufgabenstellung ist ebenfalls entscheidend. Die Aufgabenstellung und Anweisungen innerhalb des Systems müssen klar und verständlich formuliert sein, um Missverständnisse zu vermeiden und den Benutzern eine klare Orientierung zu bieten. Des Weiteren muss das System einen Editor bereitstellen, der es den Benutzern ermöglicht, die chemische Reaktion visuell darzustellen. Dieser Editor muss benutzerfreundlich sein und eine einfache Bearbeitung der Reaktion ermöglichen. Benutzer müssen in der Lage sein, die Eigenschaften der Reaktion in das System einzugeben. Daher benötigt die Benutzeroberfläche eine Schnittstelle, die es den Benutzern ermöglicht, die relevanten Informationen präzise und korrekt einzugeben. Zusätzlich sollte das System eine Funktion zum Zeichnen des Reaktionsprodukts durch die Benutzer bereitstellen. Dies ermöglicht es den Benutzern, ihr Verständnis der Reaktion auf visuelle Weise zu demonstrieren. Schließlich muss das System die eingegebenen Informationen validieren, um sicherzustellen, dass sie mit den vorgegebenen Lösungen übereinstimmen. Dadurch wird sichergestellt, dass die Benutzer korrekte und akkurate Informationen eingeben, um ein direktes Feedback zu ihrem Wissensstand zu erhalten. Diese Anforderungen bilden die Grundlage für die Gestaltung und Entwicklung der Benutzeroberfläche, um eine optimale Benutzererfahrung zu gewährleisten.

6.4. Planung

Bei der Planung des IT-Projekts müssen verschiedene Aspekte berücksichtigt werden, die eng mit den chemischen Regeln und den Vorgaben von Prof. Teichert zusammenhängen. Diese Regeln dienen als Grundlage für die Entwicklung des Programms und müssen entsprechend umgesetzt werden. Dies kann durch die Implementierung von Algorithmen und Logik geschehen, die die chemischen Prinzipien abbilden und verarbeiten. Um auf die von Prof. Teichert bereitgestellten Daten zugreifen zu können, ist es notwendig, Schnittstellen zu implementieren, die es dem Programm ermöglichen, diese Daten abzurufen und zu verarbeiten. Dies kann beispielsweise durch JSON-Datei geschehen, hierbei werden die Daten in jenen Dateien gespeichert und später, wenn abgerufen von Aladin wieder ausgelesen. Unser Framework benötigt ein bestimmtes Datenformat, um effizient mit den Daten umgehen zu können. Ein strukturiertes Format wie JSON oder XML könnte verwendet werden, um die Informationen über chemische Reaktionen und Eigenschaften klar und einheitlich darzustellen. Die Benutzereingaben müssen durch eine benutzerfreundliche Oberfläche gestaltet werden, die klare Anweisungen und Eingabefelder bietet, um die erforderlichen Informationen von den Benutzern zu erhalten. Dies kann durch die Verwendung von Formularen, Drop-down-Menüs oder anderen interaktiven Elementen geschehen. Für die Darstellung und Zeichnung chemischer Reaktionen empfehlen wir den Einsatz des JSME Editors, da dieser von Prof. Teichert empfohlen wird und bereits von den Studenten vertraut ist. Dieser Editor erfüllt alle relevanten Anforderungen und bietet eine benutzerfreundliche Erfahrung. Die Implementierung eines externen Editors wie JSME erfordert möglicherweise die Integration entsprechender Schnittstellen oder Plugins in das Framework, um die Interaktion zwischen dem Programm und dem Editor zu ermöglichen. Dies kann durch die Nutzung von APIs oder anderen Integrationstechniken erfolgen, um eine reibungslose Zusammenarbeit zwischen beiden Komponenten sicherzustellen.

JSME test page



6.5. Technische Umsetzung

Zunächst folgte die Umsetzung eines Prototypen in Java, dieser diente zunächst dazu, die Aufgabe grundlegend zu modellieren, um die Komplexität zu erfassen. Die Interaktion folgte hier zunächst rein über Konsolen In- und Outputs, welche jedoch im Rahmen des Prototypen ausreichend waren, um erste Funktionen implementieren und testen zu können. Anschließend folgte die Analyse des vorhandenen Backends Aladin und dem dazugehörigen Frontend Carpet. Die zugrundeliegenden

Programmiersprachen sind Typescript, Vue, sowie Json, dies überschneidet sich jedoch nicht mit der Programmiersprache des erstellten Prototypen. Nach ersten Einschätzungen wäre eine direkte Implementierung der Aufgabenlogik in Front- und Backend eine komplexere Lösung als ein Aufruf des Java Programm, welches dann Inhalte entsprechend wiedergibt. Der Prototyp wurde anschließend entsprechend angepasst: Als Input erfolgt das Einlesen einer Json-Datei mit dem festen Namen "Substituenten.json", diese befindet sich im Root-Verzeichnis der Software, in der sie eingefügt wurde. Weiterhin gibt es eine Datei "Reagenz.json", welches sich ebenfalls im Root-Verzeichnis befinden muss. Der Prototyp liest den Inhalt dieser Datei aus, erstellt Parameter einer zufälligen Aufgabe, sowie deren Lösung und speichert alle benötigten Inhalte in einer Datei namens "Antwort.json". Diese Datei wird ebenfalls im Root-Verzeichnis erstellt. Das gesamte Programm liegt als Jar-Datei vor und kann, sollten alle Dateien entsprechend abgelegt sein, in jeder Ordnerstruktur aufgerufen werden. Diese Jar-Datei wurde dann entsprechend im Backend von Aladin im Ordner Tasks unter Chemistry eingefügt. Sie wird dann über das Backend aufgerufen, es werden neue Daten generiert und in Antwort.json gespeichert, aus welcher diese dann wieder ausgelesen und verwendet werden können.

```
// Pfad zur Java-Laufzeitumgebung
const javaPath = "java"; // Stellen Sie sicher, dass "java" im System-Pfad ist, oder geben Sie den vollständigen Pfad an

//Pfad zur JAR-Datei
const jarPath = path.join(
  __dirname,
  "/AladinJsonGenerator.jar"
);

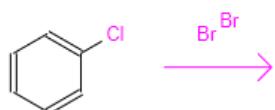
//Befehl zum Ausführen der JAR-Datei
const command = `${javaPath} -jar ${jarPath}`;
```

Wenn die Oberfläche von Aladin aufgerufen wird und der Nutzer die Kategorie Chemie auswählt, werden die benötigten Komponenten zusammengestellt und angezeigt. Dazu gehören das Aufrufen der Jar-Datei und das Verarbeiten ihrer Ausgabe, das Anzeigen der Aufgabenstellung und das Einbinden des JSME Editors. Dieser wird jedes mal dynamisch aufgerufen und in mehreren Ausführungen angezeigt.



Aufgabenstellung

Gegeben ist ein einfach substituiertes Benzolderivat. (siehe unten)
a) Geben Sie im nebenstehenden Dropdownfeld den Ort der Zweitsubstitution in einer elektrophilen aromatischen Substitution an!
b) Geben Sie bei Reaktionstyp im nebenstehenden Feld an ob Sie eine Überreaktion des Erstprodukts unter den gegebenen Reaktionsbedingungen erwarten?
c) (nächste Seite) Zeichnen Sie das zu erwartende Hauptprodukt!



Antwort

Position

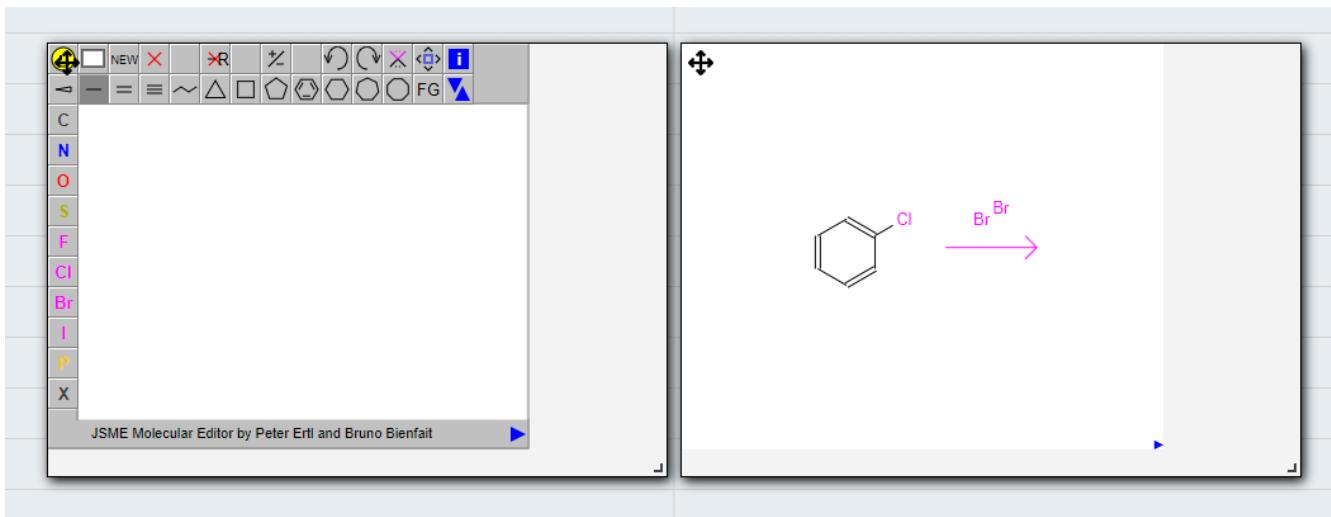
para

Reaktionstype

Ja

Um dem Nutzer die Aufgabenstellung zu verdeutlichen, wird auf der ersten Seite dem Nutzer lediglich ein Feld angezeigt, in welchem die Reaktion mit den dazugehörigen Reaktionsstoffen dargestellt wird. Dabei handelt es sich um einen JSME Editor, der mittels mehrer Parameter und einem String aufgerufen wird. Die Parameter dienen dazu, alle Schaltflächen für die Bearbeitung zu entfernen und den Editor somit auf Read-only einzustellen. Der String enthält einen komplexen "SMILES" Code, der vom Editor als Reaktionsgleichung erkannt und angezeigt wird. Dieser wurde ebenfalls von der Jar Datei erzeugt. Weiterhin gibt es eine Komponente, die zwei Dropdown Felder enthält. Der Nutzer muss zwei Teilaufgaben lösen, indem er die richtigen Elemente in den Feldern auswählt und anschließend seine Antworten überprüfen lässt. Erst wenn diese Benutzereingabe durch Carpet validiert wurde, erhält der Nutzer die Möglichkeit, auf die nächste Seite zu wechseln.

Auf der zweiten Seite wird der Nutzer aufgefordert, das chemische Hauptprodukt selbst darzustellen. Dazu wird hier erneut die Reaktionsgleichung mit einem Read-only Editor angezeigt. Weiterhin gibt es einen zweiten Editor, dieser ist jedoch leer und enthält alle benötigten Schaltflächen, damit der Nutzer verschiedene chemische Stoffe zeichnen kann. Jede Benutzereingabe sendet den Inhalt der JSME Editors als String an Carpet, wo dieser String mit der zuvor durch die Jar-Datei erstellten Lösung abgeglichen wird. Erst nach Eingabe der korrekten Lösung erhält der Nutzer eine visuelle Bestätigung und kann zurück zur Startseite von Aladin kehren.



6.6. Ergebnisse

Anforderung	Status
Geben Sie den Ort der Zweitsubstitution in einer elektrophilen aromatischen Substitution an!	gelöst (mittels eines Dropdown Feldes welches die Eingabe des Studenten erwartet)
Zeichnen Sie das zu erwartende Hauptprodukt.	gelöst (mittels einbinden des JSME-Editors)
Erwarten Sie eine Überreaktion des Erstprodukts unter den gegebenen Reaktionsbedingungen	gelöst (mittels eines Dropdown Feldes welches die Eingabe des Studenten erwartet)
Validierung der, von dem Studenten, eingegeben Daten	teilweise gelöst (mittels Validierungsfunktionen zum Abgleich der Eingaben mit der richtigen Lösung)

Die Umsetzung in Aladin und Carpet stellte den umfangreichsten Teil unserer Arbeit dar. Dabei war die Unterstützung von Herrn Christ von entscheidender Bedeutung, um ein fundiertes Verständnis für die beiden Frameworks zu entwickeln. Seine Kenntnisse halfen uns, die Herausforderungen zu bewältigen und die Funktionalitäten effektiv zu nutzen. Durch die Strukturierung der Probleme in zwei Bereiche konnten wir die Aufgaben klar unterteilen und effizient bearbeiten. Diese Herangehensweise ermöglichte eine gezielte Umsetzung und trug maßgeblich zum Erfolg des Projekts bei. Positiv betrachtet ermöglichten Aladin und Carpet eine umfassende Realisierung unserer Ziele und boten vielfältige Möglichkeiten zur Gestaltung und Implementierung.

6.6.1. Probleme

Während des Projekts traten verschiedene Probleme und Herausforderungen auf. Die Einarbeitung in das bestehende Framework war mit einer wesentlich höheren Komplexität verbunden, als zu Beginn des Projekts angenommen. Die Implementierung war sehr zeitaufwändig und erforderte intensive Unterstützung durch Herrn Christ. Darüber hinaus gab es während der Umsetzung einige technische Schwierigkeiten, die den Fortschritt zeitweise verlangsamt.

6.7. Ausblick

Das Programm stellt eine akzeptable Basis für eine Weiterentwicklung dar. Die Implementierung in Java besitzt geringe Komplexität und lässt sich schnell überarbeiten und entsprechend anpassen. Weiterhin sind alle notwendigen Komponenten in Aladin implementiert, sowie alle Schnittstellen, die dazu dienen, eine lauffähige Oberfläche zu erzeugen. Eine Erweiterung, die bereits teilweise implementiert ist, ist die Erweiterung um verschiedene mögliche Reagenzien. Das Programm arbeitet bereits mit einer Json-Datei, in welcher besagte Stoffe übergeben werden, jedoch sind diese aus chemischer Sicht nicht vollständig korrekt, sondern wurden lediglich aus informatischer Sicht für Testzwecke verwendet. Sollte eine Variation der Reagenz in Zukunft hinzugefügt werden, so müsste die Jar-Datei inhaltlich, chemisch korrekt, angepasst werden, um korrekte Ergebnisse liefern zu können. Weiterhin könnte der Nutzer auch weitere Reaktionsparameter, auch im Zusammenhang mit weiteren Reagenzien, abgefragt werden. Dazu müsste die Anzeige der Aufgabenstellung entsprechend verändert und eine Eingabe für den Nutzer geschaffen werden. Dazu bietet sich die bereits existierende Komponente, die die Dropdown Felder enthält an, es gibt jedoch auch eine Reihe weiterer Komponenten, die für solche Zwecke eine potentielle Lösung darstellen könnten.

7. Reflexion

7.1. Jonas Hölzel

Rückblickend betrachtet, muss ich sagen, dass ich anfangs das zufällige Generieren des Automaten deutlich unterschätzt haben und mich mit dem Ansatz dies über den Graphen zu machen etwas verrannt habe. Ich habe sehr viel Zeit aufgewandt, verschiedene Algorithmen und Ideen auszuprobieren, hatte aber keinen Erfolg. Das hat mich sehr frustriert und es hat meine Motivation für dieses Projekt etwas beeinflusst. Als ich dann aber einen neuen Ansatz gefunden habe, hat sich das geändert. Ich habe wieder Spaß an dem Projekt gefunden und konnte mich wieder besser auf die Umsetzung der Aufgaben konzentrieren.

Auch die Präsentation des Projektes vor den anderen Studierenden hat mir Spaß gemacht und ich finde, dass wir da als Team gut zusammen gearbeitet haben. Die Präsentation hat mich zwar auch etwas an Überwindung gekostet, dennoch fand ich es auch eine sehr gute Gelegenheit das Präsentieren vor Publikum zu üben. Dem Feedback der anderen Studierenden habe ich entnommen, dass sie das Projekt und die Idee dahinter interessant fanden.

Ich hätte mir generell in dem Projekt mehr Teamarbeit gewünscht. Es gab zwar das Chemie-Zweier-Team aber wir anderen haben doch eher jeder für sich gearbeitet. Wir konnten uns zwar untereinander austauschen, aber mit mehreren Leuten direkt in einem Team zu arbeiten ist doch nochmal was anderes.

Aus den anfänglichen Schwierigkeiten habe ich gelernt, dass es wichtig ist, sich nicht zu sehr in eine Sackgasse zu verrennen und auch mal einen anderen Ansatz zu probieren und das Problem von weiter weg zu betrachten. Auch nehme ich mit, dass es wichtig ist, sich Hilfe zu suchen. Ich bin jemand, der sehr ehrgeizig versucht Dinge ohne Hilfe zu lösen. Das versuche ich in Zukunft zu ändern.

Denn noch bin ich mit meinem Ergebnis zufrieden und denke, dass ich ein gutes Projekt abgeliefert habe, was die Möglichkeit für weitere Features bietet. Ich habe viel gelernt und konnte mein Wissen anwenden. Auch fand ich das Thema Automaten sehr interessant und bin mir sicher, dass ich von dem gesammelten Wissen in Zukunft nochmal Gebrauch machen werde.

7.2. Norman Sebastian Arnold

7.3. Tanja Dietrich

Im Rahmen des Projektseminars konnte ich wertvolle Erfahrungen sammeln und meine Kenntnisse vertiefen, im Hinblick auf die eigenständige Entwicklung einer generativen Aufgabe. Die Konzeptionsphase erwies sich für mich als entscheidender Schritt, in dem ich gelernt habe, wie die Konzeptionierung und Strukturierung eines Aufgabentyps für die Berechnung eines Entscheidungsbaums erfolgt und welche Komponenten dabei erforderlich sind. Im Rückblick erkenne ich, dass eine strukturierte Vorgehensweise für die Umsetzung in ALADIN ebenso hilfreich gewesen wäre, um die technische Umsetzung in ALADIN einfacher zu gestalten. Der implementierte Code hätte durch eine klare Strukturierung von Klassen und Funktionen deutlich übersichtlicher und verständlicher gestaltet werden können. Diese Erkenntnis nehme ich als wertvolle Lektion für kommende Projekte

aus dem Projektseminar mit.

Die Einarbeitung in ALADIN stellte für mich eine Herausforderung dar, besonders aufgrund meiner geringen Erfahrung in der Programmierung. Die anfängliche Einführung ermöglichte zwar einen groben Einblick in die Implementierung unserer Aufgabentypen in ALADIN, jedoch fehlte mir eine umfassende Dokumentation, die eine selbstständige und gezielte Einarbeitung in das Framework ermöglicht hätte. Dies führte zu hohem Zeitaufwand in der Einarbeitung, um das erforderliche Verständnis aufzubauen.

Ein weiterer Aspekt war die Einzelarbeit im Projektseminar, da Probleme häufig eigenständig bzw. allein bewältigt wurden. Ein verstärkter Austausch und gemeinsame Problemlösungen in der Gruppe hätten aus meiner Sicht zu einer einfacheren Bewältigung der Probleme führen können. Erst gegen Ende des Semesters wurde die Gruppendynamik besser genutzt, insbesondere bei der gemeinsamen Bearbeitung der Präsentation und des Projektberichts. Eine mögliche Überlegung für kommende Projektseminare wäre, dass mindestens zwei Studierende gemeinsam an der Entwicklung eines Aufgabentyps arbeiten. Dadurch kann die Bewältigung von Herausforderungen, speziell im Hinblick auf die Umsetzung in ALADIN, vereinfacht werden. Zudem würde dies dazu beitragen, die bisherige Einzelarbeitsweise zu überwinden und die Möglichkeit, den Austausch in der Gruppe zu verstärken.

Trotz der Herausforderungen war die Teilnahme am ALADIN Projektseminar eine wertvolle Erfahrung, die mir künftig bei weiteren Projekten von großem Nutzen sein wird. Die Entwicklung eines Entscheidungsbaum-Aufgabentyps hat nicht nur Freude bereitet, sondern mir auch ermöglicht, meine Programmierkenntnisse zu vertiefen. Darüber hinaus konnte ich mein im vierten Semester erworbenes Wissen über den ID3-Algorithmus erfolgreich in die Praxis umsetzen. Die wöchentlichen Meetings mit Herrn Professor Munkelt und Herrn Christ erwiesen sich sehr hilfreich, da sie wertvolle und unterstützende Hinweise gaben, um die Entwicklung des Programms in die richtige Richtung zu lenken. Zudem waren sie über die wöchentlichen Meetings hinaus stets erreichbar, um weitere Fragen oder Probleme zu klären. Die ALADIN-Lernplattform halte ich für eine sehr gute Idee, um praktische Übungen in verschiedenen Themengebieten durchzuführen und zu vertiefen. Ich bin der Überzeugung, dass sie Studierenden eine sehr gute Möglichkeit bietet, theoretisches Wissen aus Vorlesungen anzuwenden, zu üben und sich optimal auf Prüfungen vorzubereiten.

7.4. Alessandra Ruff

Als Studierender konnte ich mich leicht mit dem Problem des Mangels an Übungsaufgaben und der unzureichenden Vorbereitung auf Klausuren identifizieren. Daher hielt ich es für eine gute Idee, ein Tool zu entwickeln, das automatisch Übungsaufgaben generieren kann und sowohl Studierende als auch Professoren unterstützt. Insbesondere das Thema regulärer Ausdrücke bereitete mir schon während meines Studiums Schwierigkeiten. Ich hatte jedoch weder die Ausdauer noch die Ressourcen, um mich intensiv damit zu beschäftigen. Daher erschien mir das Projekt als ideale Lösung, um mein fachliches Verständnis von RegEx zu verbessern und gleichzeitig wertvolle Erfahrungen im Bereich von Softwareprojekten und ihrer Organisation zu sammeln.

Darüber hinaus konnte ich durch die Freiheit, welche Technologien verwendet werden sollen, meine Kenntnisse in Python weiter ausbauen. Ich habe gelernt, wie komplex die Aufgabe sein kann, viele verschiedene Programme in ein großes Framework zu integrieren, und wie wichtig eine umfassende Dokumentation beim Programmieren ist. Dabei hat mir leider teilweise die Dokumen-

tation zur ALADIn gefehlt, wodurch einige Aufgaben besonders bei der Integration als sehr schwierig bzw. teilweise nicht ohne Hilfe möglich waren. Leider fehlte mir teilweise die Dokumentation zu ALADIN, wodurch einige Aufgaben, insbesondere bei der Integration, als sehr schwierig oder teilweise nicht ohne Hilfe lösbar waren. Dennoch gelang es mir, einen neuen Aufgabentyp zu schaffen, der hoffentlich Studierenden helfen wird, ihre Kenntnisse in regulären Ausdrücken zu vertiefen.

Ich glaube, dass ALADIN in Zukunft großes Potenzial hat und auf verschiedene Teilbereiche erweitert werden kann. Deshalb bin ich sehr dankbar, Teil dieses Projekts gewesen zu sein und meinen eigenen Beitrag leisten zu dürfen.

7.5. Julius Wyrembek

Rückblickend betrachtet kann ich sagen, dass es ein sehr gutes Gefühl ist, positives Feedback von seinem Auftraggeber zu bekommen, nachdem man sich lange mit einer Aufgabe befasst hat. Während des gesamten Arbeitens an dem Projekt ist mir aufgefallen, dass es viel angenehmer ist, ein Programm mit realem Sachbezug zu entwickeln, bei dem man ständig Rückmeldung bekommt. Dies war möglich durch eine gute Strukturierung des Projektes, bei der wöchentliche Treffen durchgeführt wurden, um den aktuellen Stand der einzelnen Projekte vorzutragen. Eine Aufgabe selbst zu wählen, von der man fachlich anfangs keine Ahnung hatte, war für mich und Vincent Weise im ersten Moment reizvoll. Schon schnell wurde mir klar, dass es kein Vorteil war, Chemie in der Oberstufe abgewählt zu haben. Doch in Zusammenarbeit mit Herrn Prof. Teichert konnten wir innerhalb kürzester Zeit einen Einblick in den Bereich der Organischen Chemie bekommen, was uns helfen sollte, die Aufgabe zu erledigen. Eine Idee, wie wir es schaffen, eine zufällig ausgewählte Substitutionsreaktion in ein Framework einzubinden, war schnell entwickelt. Eine viel größere Hürde war für mich die Einarbeitung in die Frameworks Aladin und Carpet. Schlussendlich konnte ich mir dadurch neue technische Inhalte in den vergangenen Wochen aneignen. Zum einen habe ich erfolgreich Vue.js angewendet und meine Kenntnisse im Framework sowie in der Nutzung von JavaScript gefestigt. Da wir unser Backendprogramm mittels Java geschrieben haben, konnte ich auch dort mein Know-how erweitern. Schlussendlich kann ich von der Zeit wichtige Punkte in Sachen Projektorganisation im Team mitnehmen, sowohl das Koordinieren von Ressourcen als auch die Arbeitsverteilung nach individuellen Stärken.

7.6. Vincent Weise

Aladin war für mich ein sehr besonderes Projekt, da sich dieses stark in zwei Hälften unterteilte: zum einen die kontinuierliche Arbeit miteinander in einem Team an einem großen Projekt und zu anderen die inhaltliche Ausarbeitung einer Aufgabe und die eigentliche Implementierung.

Die Teamarbeit ist etwas das ich bereits in vielen verschiedenen Projekten in unterschiedlichem Ausmaß durchgeführt und miterlebt habe und Aladin dient dabei als eine weitere wertvolle Erfahrung auf die ich nun in zukünftigen Projekten zurückgreifen kann. Wir waren zwar eine große Gruppe jedoch arbeitete jeder mehr oder weniger allein, jedoch alle an einer Schnittstelle zu einem großen Projekt. So konnten wir uns inhaltlich zwar nicht über unsere direkten Aufgaben austauschen, aber im Bezug auf Methoden, Herangehensweisen und Inhalten zu Carpet und Aladin schon. Auch wenn das nicht die klassische Teamarbeit in einem Projekt ist, kann es doch eine sehr sinnvolle Herangehensweise sein, um zum Beispiel viele kleine Aufgaben zu erledigen.

Die inhaltliche Ausarbeitung entsprach hingegen nicht ganz meinen Erwartungen. Das Beschäfti-

gen mit dem Problem viele verschiedene Aufgaben zu generieren reizte mich sehr und auch die Ausarbeitung erste Prototypen empfand ich als eine spannende Aufgabe. Jedoch war das einbinden in das gestellte Framework eine sehr nervenzehrende Aufgabe. Dadurch dass das Framework bereits fertig war und wir nur etwas einbinden sollten, mussten wir uns sehr viel mit diesem beschäftigen und uns versuchen einzuarbeiten, leider war es aus meiner Sicht etwas zu komplex alles zu verstehen, weshalb die Ausarbeitung nur langsam voranschritt. Leider mussten wir oft um Hilfe bitten, da es oft Konzepte und Methoden gab deren Prinzip uns unbekannt waren. Grundsätzlich ist es sinnvoll zu lernen wie man sich in ein Framework einarbeitet, jedoch fand ich dieses etwas zu abstrakt für uns.

Trotzdem kann ich sagen das ich Aladin als eine sehr wertvolle Erfahrung festhalte und viel gelernt habe. Wir konnten alle Hürden bewältigen und ich werde mein Wissen in zukünftigen Projekten anwenden können.

8. Ausblick

Im Verlauf des Projektseminars haben die Teilnehmer erfolgreich Aufgabentypen aus verschiedenen Fachbereichen wie Chemie, Clusteranalyse, Entscheidungsbäume, Regular Expression und endliche Automaten konzipiert, entwickelt und erfolgreich in das ALADIN-Framework integriert. Durch diese Umsetzungen konnte das Framework um weitere Aufgabengeneratoren erweitert werden. Neben den bereits realisierten Aufgabentypen hat sich die Projektgruppe auch weitere Ideen und Entwicklungsmöglichkeiten für das ALADIN-Framework überlegt:

Integration von weiteren interdisziplinären Aufgaben

ALADIN kann um zusätzliche, vielfältige Aufgabentypen erweitert werden. Den Themenbereichen, die ALADIN umfasst, sind keine Grenzen gesetzt. So wären Aufgaben aus Fachbereichen der Modellierung, Geoinformatik, Musik, sowie vielen weiteren möglich. Diese könnten in Zukunft hinzugefügt werden, um die Vielfalt von dem, was mit ALADIN möglich ist, weiter zu erhöhen.

KI-Erweiterung

Durch den gezielten Einsatz von KI kann auch das ALADIN-Framework profitieren. In ALADIN könnte ein KI-Einsatz zum Beispiel bedeuten, dass die gelösten Aufgaben von der KI ausgewertet werden oder neue Aufgaben durch die KI generiert werden. Außerdem kann das Feedback der Nutzer, wenn bereits integriert, ausgewertet werden, um die Qualität der Aufgaben zu erhöhen, aber auch die Lösungshilfen könnten dynamisch mit KI-Einsatz generiert werden.

Tracking des Lernfortschritts

Damit der Lernprozess für jeden Studierenden individualisiert werden kann, wäre es erdenklich, für jede Aufgabe Punkte zu verteilen und den Lernfortschritt zu speichern. So könnten Statistiken erstellt werden, die den Studierenden helfen, ihre Problemfelder zu identifizieren und besser verstehen zu können. Dies könnte in Form von persönlichen Profilen für jeden Nutzer erreicht werden, auf welchem sie einen Überblick über das Erreichte sowie die noch offenen Themen erhalten.

ALADIN als Wissensvermittlungsplattform

Um ALADINs Vielfalt zu erweitern, wäre es sinnvoll, zusätzlich zu den Übungen auch Lernmaterialien zur Verfügung zu stellen. Dadurch könnten Studierende nicht nur ihr bereits vorhandenes Wissen festigen, sondern sich auch neues Wissen aneignen. Eine Kombination aus theoretischem Wissen über ein bestimmtes Thema und anschließendem praktischem Üben würde den Studierenden helfen, Lehrmaterialien aus dem Studium besser zu verstehen und zu verinnerlichen. Diese könnten kontinuierlich aktualisiert werden, sodass ALADIN den wandelnden Anforderungen gerecht wird.