Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik

Entwicklung einer erweiterbaren Web-Benutzeroberfläche mit Spiel-Editor für ein SQL-Lernprogramm

Bachelorarbeit

Marcus Gagelmann
marcus.gagelmann@gmail.com

Erstprüfer: Prof. Dr. Stefan Brass

Zweitprüfer: PD Dr. Alexander Hinneburg

12. 03. 2021

Zusammenfassung

Im Zuge dieser Arbeit wurde eine Web-Applikation weiterentwickelt, welche Laien zum Editieren und Spielen von SQL-Textadventures befähigt. Ziel war es die Client-Server-Anwendung eines existierenden Lernprogramms zu überarbeiten und in diese einen relationalen Editor zu integrieren. An das vorhandene System wurde hierfür bereits vorher eine PostgreSQL-Datenbank angebunden. Die Anwendung, welche als Grundlage für die Arbeit dient, bot bisher nur eine rudimentäre Benutzerschnittstelle und musste sowohl grafisch als auch funktionstechnisch ausgebaut werden. Hierbei wurde unter anderem an der Umsetzung eines Plugin-Konzepts gearbeitet, welches eine einfach zu verwendende Schnittstelle zur Einbindung zusätzlicher Werkzeuge in den Client realisiert. Anforderung an den relationalen Editor ist es, dass dieser unabhängig von der aktuellen Version des Datenbankschemas funktioniert. Dafür ist das Einlesen des Datenbankschemas eines fertigen Spiels aus der Datenbank notwendig. Anschließend können zu dem eingelesenen Schema, mithilfe eines Graphical User Interface (GUI) und unterschiedlichen Hilfsfunktionen, neue relationale Spieldaten erstellt und editiert werden. Zu den Spielinhalten gehören in der Regel Räume, Gegenstände oder auch Non-Player-Characters (NPCs). Die Anwendung des Editors für Zwecke abseits des Lernprogramms, beispielsweise als Werkzeug zur Datenbank-Administrierung, wäre jedoch ebenfalls denkbar. Die erstellten Inhalte sind anschließend exportierbar und können zur Initialisierung eines neuen Spiels direkt aus dem Editor in die Datenbank geladen werden. Als Ergebnis dieser Arbeit ist es Autoren des Lernprogramms möglich, neue Spiele mit einer höheren Geschwindigkeit und einer geringeren Fehlerbelastung in einem internen Tool zu editieren. Weiterhin bewegen sich Nutzer des Lernprogramms nun in einer ausgebauten, modernen Anwendung, welche das Spielen und Lernen begünstigt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

Digitale Spiele werden in der modernen Welt von vielen Menschen als Freizeitmedium genutzt. Laut Untersuchungen des Verbandes „game“ [1] spielten 2019 etwa 34,1 Millionen Deutsche, zumindest gelegentlich, Computer- und Videospiele. Aufgrund des großen Interesses an digitalen Spielen konnte der Markt für Computer- und Videospiele im selben Jahr einen Umsatz von 6,231 Milliarden Euro verzeichnen. Diese Zahlen folgen dabei, mit einem Zuwachs von 6%, dem Trend der Jahre zuvor.

Unter Lernspielen versteht man die Verknüpfung einer spielerischen Handlung mit der gezielten Vermittlung von Wissen in einem Spiel. Diese Art der Spiele bietet heutzutage, vor allem in seiner digitalen Form, Potenzial an allen Lehreinrichtungen und in jeder Altersgruppe. Dieses Potenzial ist jedoch in den meisten Bereichen weiterhin ungenutzt. Der Verband „game“ schreibt hierzu in [1, S. 40]: „Unser Bildungssystem muss die Chancen von Games für die digitale Bildung in Schulen, Berufsschulen, Hochschulen, in der Weiterbildung und für das lebenslange Lernen nutzen.“ Ob die Verwendung von Lernspielen als Lehrmethode das aktive Lernverhalten unterstützen kann, wurde 2006 in einer Studie in den USA [2] untersucht. Hierfür wurden 41 Studierende aus Business-Kursen befragt, welche regelmäßig simple Lernspiele mit Kursinhalt in ihren Veranstaltungen verwendet haben. Dabei konnte ein positiver Effekt beim Lernengagement der Studenten, vor allem aufgrund des Wettbewerbgefühls gegenüber anderen Studenten, festgestellt werden. Eine weitere Studie [3], welche 2012 veröffentlicht wurde, untersuchte den Effekt eines Karten- und eines Computer-Lernspiels auf den Erfolg beim Lernen von Konzepten der Chemie im Vergleich zu herkömmlichen Lehrmethoden. Getestet wurden 105 Oberschüler aus Teheran mithilfe einer Prüfung im Fach Chemie. Hierbei schnitten sowohl das Kartenlernspiel als auch das Computer-Lernspiel signifikant besser ab, als die herkömmlichen Lehrmethoden. Weiterhin untersuchte eine Studie aus dem Jahre 2015 [4] das Element des Wettkampfes zwischen Studierenden in einer spielbasierten Lernumgebung. Die 142 Teilnehmer aus verschiedenen IT-Studienfächern wurden mithilfe eines Lernspiels im Themengebiet der ER-Diagramme unterrichtet. Hierbei erhielt eine der beiden Gruppen Zugriff auf eine spielinterne Anzeigetafel mit den Punktzahlen aller Gruppenmitglieder, um ein Wettbewerbsgefühl zu erzeugen. In dieser Studie konnte ermittelt werden, dass Mitglieder der Wettbewerbsgruppe in einem nachfolgenden Test signifikant besser abschnitten als Mitglieder der Kontrollgruppe, welche keine Informationen über den Punktestand anderer Spieler erhalten haben.

Adventure-Spiele bilden eines der großen Genres in der Geschichte der Computer- und Videospiele. Klassischerweise können Titel dieses Genres nur als Einzelspieler gespielt werden. Weiterhin lassen sie sich durch die Erkundung einer manipulierbaren Spielwelt sowie die Erzählung einer interaktiven Geschichte charakterisieren. Kerngedanke der Adventure-Spiele sind Rätsel, welche es zu lösen gilt und Hindernisse, mit welchen der Spieler regelmäßig innerhalb der Spielwelt konfrontiert wird. Diese Hürden werden vom Spieler bewältigt, um einen Fortschritt im Spielverlauf herbeizu-

führen. Das Erreichen von Fortschritt im Spielgeschehen wird in der Regel durch das Freischalten weiterer Teile der Spielgeschichte belohnt. Dadurch soll der Akteur zum Weiterspielen motiviert werden.

Textadventures bilden ein Subgenre der Adventure-Spiele. Textbasierte Adventure-Spiele zeichnen sich sowohl durch eine Spielwelt, als auch durch eine Geschichte aus, welche allein durch Texte beschrieben werden. Eine grafische Unterstützung des Spielers erfolgt nur in Ausnahmen. Auch die Kommunikation des Spielers mit dem Spiel wird in diesem Genre durch textuelle Befehle realisiert, anstatt beispielsweise einer Bewegungssteuerung mit Richtungstasten, wie man sie aus anderen Spielen kennt. Diese schlichte Form der Computerspiele entstand aus den begrenzten Möglichkeiten der 1970er Jahre heraus [5]. Vorreiter des Genres ist hierbei das Spiel „Adventure“ aus dem Jahre 1976 [6]. Seit Beginn der Entwicklung von grafischen Adventure-Spielen nehmen Textadventures jedoch lediglich eine untergeordnete Rolle im breiten Angebot der Computerspiele ein. Nichtsdestotrotz bieten Textadventures auch heute noch die Möglichkeit zur immersiven Geschichtenerzählung. Mithilfe dieser Geschichten kann heute, genauso wie damals, die Überwindung von Hindernissen im Spiel motiviert und belohnt werden. Eine Anwendung dieser Spielmechanik ist mitunter in Lernspielen denkbar. Hierfür werden die klassischen Spielrätsel durch pädagogisch wertvolle Aufgaben als Hindernisse ausgetauscht.

Der herkömmliche Weg, um Schüler und Studierende in SQL zu unterrichten, führte bisher über konventionelle Lehrmethoden in Form von Unterrichtsstunden und Vorlesungen. Diese Methoden haben bereits über viele Jahre ihre Funktionstüchtigkeit bewiesen. Das selbstständige Üben der erlernten Techniken außerhalb der üblichen Lehrumgebung kann sich jedoch für viele als Herausforderung aufgrund von mangelnder Motivation herausstellen. Im Rahmen mehrerer Arbeiten der Datenbank-Gruppe entsteht deshalb an der Martin-Luther-Universität Halle-Wittenberg (MLU) seit 2017 ein Programm zum Spielen von SQL-Textadventures. Bei dieser Anwendung wird der Fokus darauf gelegt, eine ähnliche Funktionsweise und ähnliche Möglichkeiten wie Klassiker des Genres, zum Beispiel Zork [7], zu bieten. Das Ziel des Projektes ist es, Studenten der Datenbankvorlesung spielerisch zum aktiven Lernen der „Structured Query Language“ (Abgekürzt: „SQL“) zu motivieren. Diese weitverbreitete, normierte Sprache für relationale Datenbanken vereint die Möglichkeiten der Datendefinition und Datenmanipulation. SQL gehört hierdurch zum Grundlagenwissen beim Umgang mit relationalen Datenbanksystemen. Zur Verknüpfung von SQL-Lehrinhalten und Textadventures wurde das Lernspiel der Datenbank-Gruppe so entworfen, dass sämtliche Daten eines aktuellen Spiels auf dem Zustand einer relationalen Spieler-Datenbank basieren. Hierdurch kann es dem Spieler zur Aufgabe gemacht werden, das Spiel durch SQL-Befehle zu steuern, anstatt textuelle Befehle (zum Beispiel: „rede mit Wirt“) zu verwenden. Mithilfe der Spielsteuerung wird die Verwendung von Update-Befehlen angereizt. Weiterhin motiviert der datenbankbasierte Ansatz den Spieler SQL-Anfragen zu formulieren. Hierbei soll der Spieler selbstständig die Datenbank nach Informationen durchsuchen, um keine Spielinhalte zu verpassen. Die Zielgruppe dieses

Projekts umfasst einen Personenkreis, welcher seine Kenntnisse über SQL erweitern möchte. In der ersten Version des Programms ist hierfür überwiegend mit Studenten der MLU zu rechnen, welche die Datenbankvorlesung besuchen.

Seit vielen Jahren bieten digitale Spiele auch Editoren zur Erstellung eigener Spielwelten an. Diese Editoren tragen oft maßgeblich zum langfristigen Erfolg eines Spiels bei. Ein Beispiel hierfür ist das Echtzeit-Strategiespiel „Warcraft III: Reign of Chaos“ aus dem Jahre 2002, welches 2019 neu veröffentlicht wurde [8]. Diesem Spiel hat vor allem der sogenannte „World Editor“, neben einer eigenen Kampagne und einem Mehrspielermodus, zu einer langfristig stabilen und treuen Community verholfen. Die immer neuen Konzepte aus der Community fesseln Spieler bis heute an das Strategiespiel. Der Erfolg des Editors von Warcraft wird in Kapitel 4.1 näher betrachtet. In dieser Arbeit werden die Features von Quest 5 [9] analysiert, einem der aktuell umfangreichsten Editoren für Textadventures. Dieses Autorensystem ist in seinen Mechaniken und Möglichkeiten konkret auf Textadventures spezialisiert. Vergleichbare Autorensysteme wie TADS [10] oder Twine [11] ermöglichen hingegen die Gestaltung von Interactive-Fiction. Interactive-Fiction ist ein Computerspielgenre, welches sowohl Parser-Based Games als auch Choice-Based Games umfasst. Textadventures bilden hierbei eine Teilmenge der Parser-Based Games [12]. Autorensysteme für die Erstellung von Interactive Fiction bieten aus diesem Grund auch Möglichkeiten zur Erstellung von Textadventures, sie werden jedoch nur selten für reine Spiele dieser Art verwendet. Grund hierfür ist der geringe Grad an Spezialisierung und der damit einhergehende bescheidene Funktionsumfang zur Textadventure-Erstellung. Auf die Diskussion von Autorensystemen zu Generierung von Interactive-Fiction wird in dieser Arbeit bewusst verzichtet, da Quest eine bessere Alternative zur Erstellung von reinen Textadventures ist.

Aufgrund der Entwicklung des Lernprogramms der Datenbankgruppe auf Basis einer relationalen Datenbank, werden zukünftige Textadventures des Lernprogramms grundsätzlich als eine Menge von relationalen Daten vorliegen. Solche Datenmengen sind in der Regel nur mit einem umfangreichen Wissen über SQL oder aber mithilfe unterstützender Anwendungsprogramme editierbar. Ziel des theoretischen Anteils dieser Arbeit ist deshalb, nach der Besprechung verwandter Arbeiten (siehe Kapitel 2.1), eine Einführung in den Textadventure-Spieleeditor Quest (siehe Kapitel 2.2) sowie in gängige relationale Editoren (siehe Kapitel 2.3). Für letztere werden sowohl die Verwendung existierender Software, als auch die Entwicklung einer neuen Datenbank-Schnittstelle diskutiert. Explizit werden „Adminer Editor“ in Kapitel 2.3.1 und „Oracle Forms“ in Kapitel 2.3.2 näher betrachtet. Dadurch soll eine Wissensgrundlage für die Implementierung eines Editors für das Projekt geschaffen werden. Weiterhin werden Grundlagen der funktionalen Programmiersprache Elm in Kapitel 2.4 besprochen. Diese Sprache zur Entwicklung grafischer Web-Oberflächen kompiliert zu JavaScript, weshalb Elm-Programme ohne Probleme in klassische HTML-Dateien eingebettet werden können. Zuletzt erfolgt eine ausführliche Beschreibung der ursprünglichen Client-Anwendung des Lernprogramms der Datenbank-Gruppe in Kapitel 2.5, um den Ausgangspunkt

dieser Arbeit zu verdeutlichen. Der Schwerpunkt von Kapitel 2. wird insgesamt auf die Vorbereitung der praktischen Anwendung der erklärten Konzepte gelegt.

Der praktische Anteil dieser Arbeit konzentriert sich auf die Überarbeitung von Design und Funktionalität der Client-Anwendung der Datenbankgruppe in der funktionalen Programmiersprache Elm, welche Nutzern einen spielerischen Anreiz zum selbstständigen Lernen, als Ergänzung zu konventionellen Lehrveranstaltungen, bieten soll. Die zum System zugehörigen Servlets und die Datenbank werden an einigen Stellen ebenfalls angepasst, dies geschieht jedoch in einem geringeren Ausmaß als bei der Client-Anwendung. In Kapitel 3.1 wird vorerst das CSS-Framework genauer betrachtet, welches in dieser Arbeit verwendet wurde. Im Anschließenden Kapitel 3.2 erfolgt anschließend eine Beschreibung und Auswertung der durchgeführten Änderungen am Client. Weiterhin findet eine Betrachtung der Farbgebung als designtechnischer Aspekt der Client-Anwendung in Kapitel 3.3 statt. Zuletzt werden im Zusammenhang mit dem Client der Prototyp eines Plugin-Konzepts in Kapitel 3.4 näher erläutert sowie weitere Forschungsmöglichkeiten für die Client-Anwendung in Kapitel 3.5 besprochen.

Als Grundlage für die Web-Applikation dienen die SQL-Textadventure-Datenbank, die Server-Anwendung und die Client-Anwendung, welche bereits von der Datenbank-Gruppe der Universität bereitgestellt werden. Alle diese Programme sind zu Beginn der Arbeit noch unfertig und werden im weiteren Verlauf überarbeitet und ergänzt. Das langfristige Ziel des Projekts ist es hierbei ein Anwendungssystem zu schaffen, welches Nutzern ein komfortables und umfangreiches Spiel- und Lernerlebnis ermöglicht. Im Rahmen dieser Arbeit wird deshalb angestrebt, sowohl den Ausbau der Client-Anwendung auf visueller und funktionaler Ebene in voranzutreiben, als auch den Prototypen eines eigenen Spiel-Editors grundlegend zu entwickeln. Dieser Editor soll das Erstellen und Verwalten neuer SQL-Textadventure-Spielwelten vereinfachen, welche in der folgenden Arbeit von nun an als „Spiel“ bezeichnet werden. Hierfür war es sowohl notwendig, die bestehende Schnittstelle zwischen der PostgreSQL-Datenbank und der Web-Applikation in Form von Java-Servlets zu erweitern, als auch ein eigenes Werkzeug für den genannten Zweck in die Web-Anwendung zu integrieren. In Kapitel 4.1 wird vorerst auf den Aufbau und die Bedienung des entwickelten Editors eingegangen. Im Anschluss findet in Kapitel 4.2 die Thematisierung von Fehlerkontrollen und des Sicherheitsaspekts statt. Genaue Beschreibungen zum verwendeten Datenformat sowie zum In- und Output des Editors sind in Kapitel 4.3, 4.4 und 4.5 zu finden. Zuletzt werden für den Editor in Kapitel 4.6 Möglichkeiten für die weitere Forschung besprochen. Die Umfunktionierung des Editors zur Bearbeitung realtionaler Daten, die keine Spieldaten sind, wäre eine solche Möglichkeit. Auch die Administration von Datenbanken wäre mithilfe des Tools nach einer Umfunktionierung denkbar.

Den Nutzern soll eine intuitive, grafische Benutzerschnittstelle zur Verwendung des Editors und zum Spielen der Textadventures zur Verfügung stehen. Zu diesem Zweck wird das UI jeder Seite der Client-Anwendung mithilfe von Design-Mustern auf visueller Ebene neu konzipiert, wodurch

unter anderem eine effektivere Viewport-Nutzung und ein ansprechenderes Aussehen angestrebt wird. Dafür wird auf die Thematik der Mensch-Maschine-Interaktion eingegangen. Diese ist sehr umfangreich und überschneidet sich mit dem Bereich der Psychologie. Eine Erläuterung von geläufigen Methoden und nötigem Hintergrundwissen sprengt den Rahmen dieser Arbeit. Stattdessen werden wichtige Punkte in den Praxisteil einbezogen, welche laut [13] beim Design von Grafical User Interfaces (GUI) Beachtung finden sollten.

2. Stand der Forschung

In diesem Kapitel werden neben fachlichen Grundlagen auch existierende Anwendungen besprochen, welche einen hilfreichen Ansatz umgesetzt haben. Daraus ableitend findet eine Betrachtung von für diese Arbeit potenziell interessanten Themen statt. Im Kontrast zu den hilfreichen Konzepten der vorhergehenden Projekte sollen in diesem Abschnitt auch Punkte ohne Mehrwert für diese Arbeit sowie mögliche Fehler erkannt und vermieden werden. Zuletzt erfolgt eine Diskussion von Anwendungsystemen, die eine potenzielle Fertiglösung für den zu entwickelnden Editor darstellen könnten.

2.1 SQL-Island

SQL-Island [14] ist aktuell das SQL-Lernspiel, welches den Textadventure-Ansatz am weitesten umsetzt. Das Spiel ist ein Projekt aus dem Jahre 2015 und stammt von der Universität Kaiserslautern. Man spielt mithilfe einer Web-Applikation, welche mit PHP, JavaScript und HTML entwickelt wurde. Im Back-End werden die SQL-Anfragen von einer SQLite-Datenbank beantwortet. SQL-Island verfolgt mit seiner Spielwelt einen sehr statischen Ansatz. Das Spiel ist hierbei lediglich ein vorgefertigtes SQL-Tutorial, welches einzelne Aufgaben durch eine Rahmenhandlung miteinander verknüpft. Der Nutzer kann dabei keine Entscheidungen fällen, welche den grundlegenden Spielablauf beeinflussen. Es werden lediglich Aufgaben in einer vorgegebenen Reihenfolge abgearbeitet. Dieses lineare Spielprinzip weicht hierbei grundsätzlich vom Ansatz des Lernprogramms der Datenbank-Gruppe ab. Hier soll nämlich ein Spiel mit einer offenen Spielwelt im Stil klassischer Textadventures entstehen.

Das Lernspiel kann über die dazugehörige Website [15] gespielt werden. Dem Spieler wird hierfür ein Input-Feld für SQL-Befehle, ein Output-Fenster mit der Anfragenhistorie sowie eine Übersicht über alle verwendbaren Tabellen zur Verfügung gestellt (siehe Abbildung 1). Zugriff hat der Nutzer auf die Tabellen aller Dörfer, Bewohner und Gegenstände der Spielwelt.

Auf Räume, Wege, eine Spielerposition und Bewegungsmöglichkeiten wird in SQL-Island verzichtet, da diese redundant für den ohnehin gesteuerten Spielablauf wären. Der Spieler bewegt sich nach Erfüllung einer Aufgabe automatisch innerhalb der Geschichte zur nächsten Aufgabe. Die Story selbst wird in SQL-Island getrennt von der Konsole mithilfe gezeichneter Charaktere erzählt. Auch dieses Konzept stellt eine große Abweichung vom klassischen Konsolen-Feedback bekannter Textadventures dar und wird innerhalb des Projekts der Datenbank-Gruppe anders umgesetzt. Fachlich vermittelt werden dem Spieler durch SQL-Island neben SELECT-,UPDATE- und DELETE-Anfragen auch der Umgang mit Gruppierungen, Aggregatsfunktionen und Joins. Eine Registrierung in der Anwendung ist vom Entwickler nicht vorgesehen.

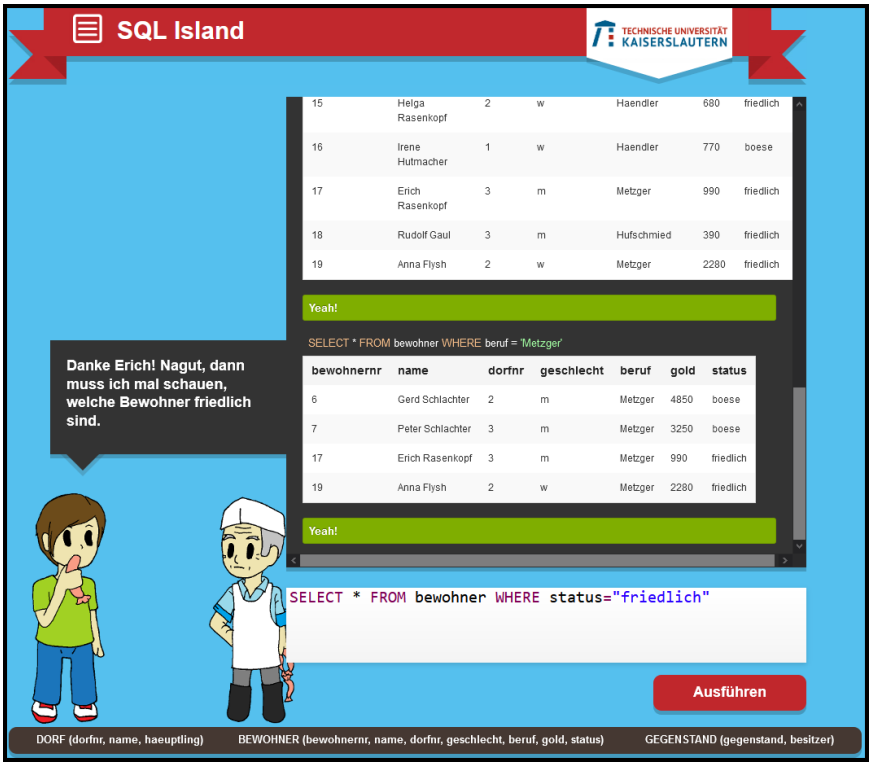

Abbildung 1: Spielausschnitt aus SQL-Island
Quelle: [15]

2.2 Autorensystem „Quest“

Quest [9] ist ein kostenloser Editor zur Entwicklung vollwertiger Spiele des Textadventure-Genres. Die Software ist sowohl als Programm zum Download, als auch als eigene Web-Applikation innerhalb der Website „textadventures.co.uk“ verfügbar. Die Website, welche auch Quest enthält, stellt Tools zum Erstellen, Teilen und Spielen von Interactive-Fiction bereit. Weiterhin bietet die Seite Funktionen zur Rezension und Kommentierung von Spielen an. Der Log-in mit einem Nutzer-Profil ist für die Verwendung der besagten Anwendungen erforderlich. Vorrangig wird in dieser Arbeit der Online-Editor betrachtet, welcher über „textadventures.co.uk“ [9] erreichbar ist. Anschließend erfolgt eine Untersuchung von Gemeinsamkeiten und Unterschiede zum Offline-Tool für Windows.

Im Kontext dieser Arbeit dient Quest als Musterbeispiel für einen Editor zur Erstellung von Textadventures. In diesem Zusammenhang existiert der offensichtliche Unterschied zum Projekt der Datenbank-Gruppe, dass Quest nicht für die Realisierung von SQL-basierten Textadventures konzipiert wurde. Dennoch bietet Quest seinen Nutzern ein vergleichbares Umfeld, wie es auch innerhalb des Projektes angestrebt wird. Eine Parallele bildet beispielsweise die Web-Anwendung von Quest, welche in gleicher Weise zum Spielen als auch zum Editieren von Textadventures verwendet werden kann. Mithilfe einer solchen Vorlage können wichtige Hilfsfunktionen für den zu entwickelnden Editor erkannt und übernommen werden. Zusätzlich können Probleme, welche bei der Modellierung der Grundstruktur von Quest nicht erkannt wurden, bei der Modellierung des

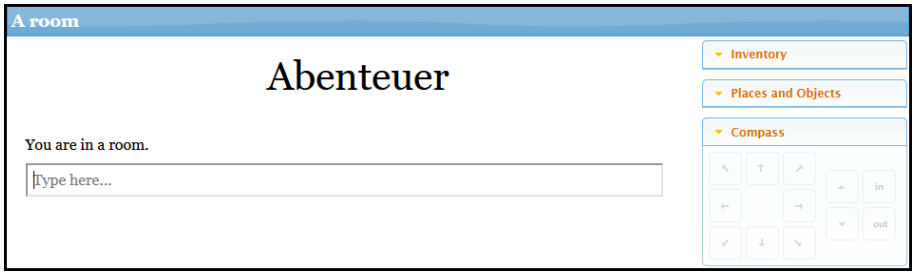

Abbildung 2: Quest Display „Standard“
Quelle: [9]

Abbildung 3: Quest Display „Retro“
Quelle: [9]

relationalen Editors im besten Fall direkt umgangen werden. Hierfür wird im Folgenden die Erstellung eines Spiels mithilfe des Online-Tools von Quest schrittweise beschrieben. Zusätzlich werden Mehrwerte für diese Arbeit diskutiert.

Bei der Erstellung eines neuen Spiels bietet der Quest-Online-Editor dem Nutzer initiale Einstellungen an, die das gesamte neue Spiel betreffen. Hierzu gehören mehrere Setup-Funktionen, welche die Festlegung von Metadaten ermöglichen. Unter anderem können ein Name, ein Autor, ein Entwicklungsjahr und ein Genre (zum Beispiel Comedy, Horror, Sci-Fi) für das Spiel gewählt werden. Weiterhin ist es möglich, vordefinierte Features zu aktivieren, die das grundlegende Spielverhalten verändern. Hierzu gehören spielerbezogene Eigenschaften wie Lebenspunkte, ein Score und ein Geldsystem, aber auch globale Optionen wie zum Beispiel Lichtverhältnisse sind möglich. Quest bietet dem User außerdem eine Anpassung des Spiel-Displays. Hierbei kann aus einer Reihe von Templates ausgewählt werden, welche anschließend unter anderem durch Farben, Schriftarten, Größeneinstellungen und Bilder-Uploads detailliert gestaltbar sind. Einige Beispiele für solche unterschiedlichen Templates sind in den Abbildungen 2 und 3 zu erkennen.

Des Weiteren sind einzelne Teile des Spieler-Interfaces aktivierbar und deaktivierbar. Zu den wichtigsten aktivierbaren Interface-Optionen in Quest gehören eine Karte, eine Location-Bar, die Kommandozeile sowie Dropdown-Sektionen mit einem Steuerkompass und verschiedenen Übersichten. Ist die Karte aktiviert, so zeigt sie eine Übersicht aller besuchten Räume und den Aufenthaltsort des Spielers im aktuellen Spiel an. Die Location-Bar zeigt den Namen des aktuellen Raumes. Eine Deaktivierung der Kommandozeile ist möglich. Diese Einstellung nimmt dem Text-

adventure jedoch die klassische Steuerung über textuelle Befehle. In diesem Fall ist es nötig, dass andere Interface-Optionen für die Spieler-Interaktion verwendet werden. Eine der Möglichkeiten zur Steuerung stellt der Kompass dar. Dieser umfasst Buttons zur Bewegung des Spielers in acht verschiedene Himmelsrichtungen sowie „hoch“, „runter“, „rein“ und „raus“. Zuletzt können dem Spieler verschiedene Übersichten zur Verfügung gestellt werden, welche einen Überblick über Gegenstände im Inventar und in der Umgebung gewähren. Diese Informationen müssen sonst mithilfe von Textkommandos über die Konsole abgerufen werden.

Unerfahrenen Autoren wird zur Einarbeitung empfohlen, unter dem Reiter „Settings“ den sogenannten „Simple Mode“ zu aktivieren. Dieser Modus stellt eine weniger komplexe Alternative zu der standardmäßigen Ansicht dar. Durch das Umschalten in diesen Modus werden viele von Quests fortgeschrittenen Funktionalitäten wie Skriptschnittstellen, Funktionen und Timer vor dem Nutzer verborgen. Diejenigen Aktionsmöglichkeiten, welche im „Simple Mode“ erhalten bleiben, werden jedoch nicht weiter beschränkt.

Obwohl man mithilfe von Quest auch funktionierende Spiele ohne die Verwendung von Skripten und Code erstellen kann, sind diese bei anspruchsvolleren Textadventures meistens notwendig. Insgesamt finden vier verschiedene Programmier- und Skriptsprachen in Quest Anwendung. Objekte und Räume können mit Skripten versehen werden. Solche Skripte können entweder direkt mithilfe der eigens für Quest entwickelten Skriptsprache ASL verfasst, oder aber mithilfe des GUI von Quest zusammengestellt werden. Über ASL wurden leider nur wenige Informationen von Seiten der Entwickler veröffentlicht. Sie soll Ähnlichkeiten zu C++ und Java aufweisen und viele voreingebaute Funktionen von Visual BASIC übernommen haben [16]. Ausgelöst werden können solche Skripte in der Regel durch Ereignisse im Spiel, wie zum Beispiel durch das Betreten eines Raumes oder auch durch das Betrachten eines Gegenstandes. Geschrieben ist Quest in der Programmiersprache C#. Der Nutzer muss laut Dokumentation [16] für die Erstellung eines Spiels jedoch keinerlei Kenntnisse in dieser Sprache besitzen. Weiterhin erstellt Quest im Hintergrund sowohl XML- als auch JavaScript-Code. In der Dokumentation wird behauptet, es sei lediglich in Ausnahmefällen notwendig, Code-Abschnitte dieser beiden Sprachen zu bearbeiten. Die Entwickler verwenden den XML-Code eines Spiels nur zur Einsicht, um Typfehler schneller finden zu können. JavaScript hingegen kann für außergewöhnliche grafische und funktionstechnische Anpassungen des Interfaces genutzt werden. In den allermeisten Fällen reicht jedoch die Benutzung der grafischen Benutzeroberfläche und das Schreiben von ASL-Code zur Erstellung und Anpassung von Spielen aus.

Zusätzlich werden von Quest noch weitere, weniger relevante Features unterstützt. Hierzu gehören unter anderem Kommandoeinstellungen, Einstellungen für Raumbeschreibungen und Timer. Fertige Textadventures, welche mit Quest erstellt wurden, können auf „textadventures.co.uk“ über die zugehörige Datenbank abgespeichert, veröffentlicht und gespielt werden.

Verallgemeinert bietet Quest ein großes Repertoire an vorbestimmten Einstellungen, welche

die Grundelemente (Objekte und Räume) einzeln oder in ihrer Gesamtheit beeinflussen können. Auf ähnliche Weise arbeiten die vom Spieler geschriebenen Skripte. Ein Skript stellt in sich eine sehr flexible Möglichkeit dar, Vorgänge und Abläufe innerhalb des Spiels zu beschreiben. Damit bilden Skripte eine mächtigere Alternative zu den vorgegebenen Funktionen des GUI. Jedoch sind auch die Skripte im Quest-Editor auf die Schnittstellen begrenzt, welche von den Entwicklern von Quest vorgegeben wurden. So kann ein Autor beispielsweise die Einstellung „Initialisierungsskript für dieses Objekt verwenden“ aktivieren, um mithilfe eines selbstbestimmten Skriptes in einem gesonderten Reiter den Startzustand des Objektes zu beeinflussen.

Weiterhin ist es möglich, Skripte beim Versuch der Aufnahme von Gegenständen auszuführen. Ein übergreifendes Skript zur Verwaltung des kompletten Objektes wäre jedoch aufgrund der klein angelegten Schnittstellen nicht möglich. Aufgrund der hohen Anzahl an Schnittstellen für Skripte in Quest, welche jeweils nur für einen kleinen Teil eines Spiels (beispielsweise einzelne Objekte) zuständig sind, besitzt ein fertiggestelltes Spiel oft eine größere Menge an unabhängigen Code-Abschnitten. Treten in dieser Menge Duplikate von Skripten auf, so wäre es schlechter Stil diese für jedes weitere Vorkommen zu kopieren und an anderer Stelle erneut einzufügen. Für dieses Problem stellt Quest die Möglichkeit zur Verfassung von Funktionen bereit. Funktionen sind ASL-Skripte, die auf globaler Ebene des Spiels wiederverwendet werden können. So könnte es an unterschiedlichen Punkten im Spiel nötig sein, den Charakter des Spielers wiederzubeleben. Hierfür wäre es möglich in jedem Fall eine einmalig formulierte Funktion „revive“ aufzurufen, anstatt das Skript zum Wiederbeleben des Spielers an jeder Stelle erneut einzufügen.

Zusammenfassend kann von Quest behauptet werden, dass der Editor eine große Vielfalt an unterschiedlichen Einstellungen und Unterstützungsfunktionen bietet. In Abbildung 4 ist zu erkennen, wie umfangreich allein die Liste an aktivierbaren Features für ein einziges Spiel ist. Ein Funktionsumfang solchen Ausmaßes konnte für dieses Programm realistisch umgesetzt werden, da der Rahmen des Möglichen eines jeden Spiels bereits durch das Programm vorgegeben wird. So kann beispielsweise die Spielmechanik „Geld“ im Feature-Tab eines Spiel (siehe Abbildung 4) aktiviert werden, weil das Konstrukt einer Währung während der Entwicklung von Quest in das Programm implementiert wurde. Ein solcher Funktionsumfang ist für den Editor des Lernprogramms der Datenbank-Gruppe im Rahmen dieser Arbeit nicht realisierbar, da hier jedes kompatible Spiel in Abhängigkeit zu einem grundlegenden Spielschema steht, das den Rahmen des Möglichen für diese Spiele festlegt. Ein solches Spielschema existiert jedoch in der aktuellen Version lediglich als Prototyp und lässt deshalb kaum Möglichkeiten zur Entwicklung ähnlicher Funktionen wie in Quest zu. Das Projekt der Datenbank-Gruppe ist schlichtweg auf einem zu frühen Entwicklungsstand. Genauere Ausführungen hierzu können in Kapitel 4.4 gefunden werden. Von der Möglichkeit zur Formulierung eigener Skripte ist der Editors des Lernprogramms ebenfalls noch weit entfernt. Der Ansatz ist jedoch nicht unrealistisch, da auch SQL-Anfrage-Sequenzen als Skript verfasst werden können.

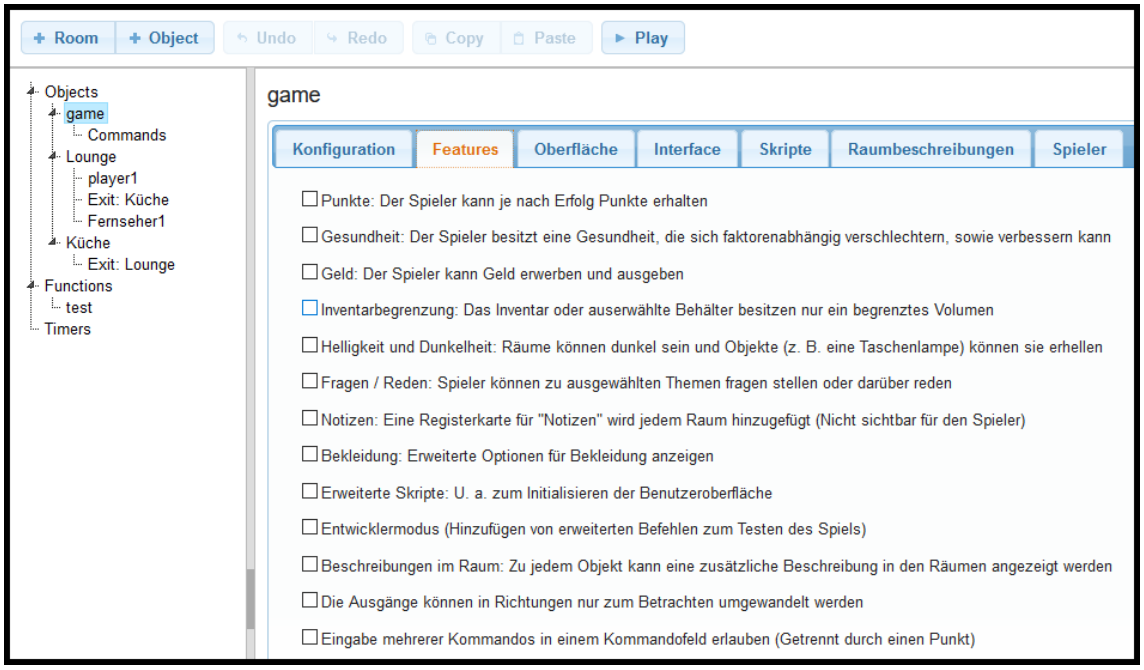

Abbildung 4: Optionsvielfalt in Quest
Quelle: [9]

Bei der Betrachtung von Fehlern in Quest wird deutlich, dass hier der Fokus auf einer generellen Vermeidungsstrategie liegt. Die Detektion und Anzeige von Fehlern besitzt insgesamt weniger Relevanz. Das Erstellen von Spielen in Quest bewegt sich in einem festgelegten Rahmen mit klaren Schnittstellen, wobei Einstellungsmöglichkeiten oft fest vorgegeben und nicht beliebig anpassbar sind. Aufgrund dieses kontrollierten Ansatzes ist die Bandbreite an Fehlerquellen im Editor von Quest relativ gering. Wenn ein Autor überwiegend nur aus Dropdown-Menüs mit vielen vorgegebenen Konfigurationsmöglichkeiten auswählen kann, oder aber Einstellungen lediglich über Checkboxen aktiviert und deaktiviert, so können kaum kaputte Spieldateien oder Laufzeitfehler durch menschliches Versagen entstehen. Die wenigen Fehler, die jedoch noch bei der Erstellung oder Bearbeitung eines Spiels möglich sind, werden vom Quest-Editor oft erkannt und dem Nutzer aufgezeigt. Solche Fehlermeldungen besitzen in Quest jedoch in vielen Fällen kaum Aussagekraft. Versucht ein Nutzer beispielsweise einen Weg zu erstellen, ohne aber einen zweiten Raum als Ziel auszuwählen, so stellt der Editor die in Abbildung 5 gezeigte Fehlermeldung dar. Diese Meldung vermittelt zwar, dass bei einer Aktion ein Fehler aufgetreten ist, der Nutzer erhält jedoch nur wenige Informationen über den Hergang und die Quelle des Fehlers. Ein Auffinden der Fehlerquelle könnte den Nutzer in komplexeren Szenarien dadurch viel Zeit kosten.

Ein weiteres Beispiel für die Fehlerbewältigung von Quest ist der Umgang mit fehlerhaftem Skript-Code. Enthält ein Skript einen oder mehrere Syntaxfehler innerhalb eines ansonsten syntaktisch korrekten Codes, so zeigt der Editor dem Nutzer einen „Fehler beim Laden des Skripts“ an und markiert den gesamten Quellcode rot, wie in Abbildung 6 zu erkennen ist. In diesem Fall wurde

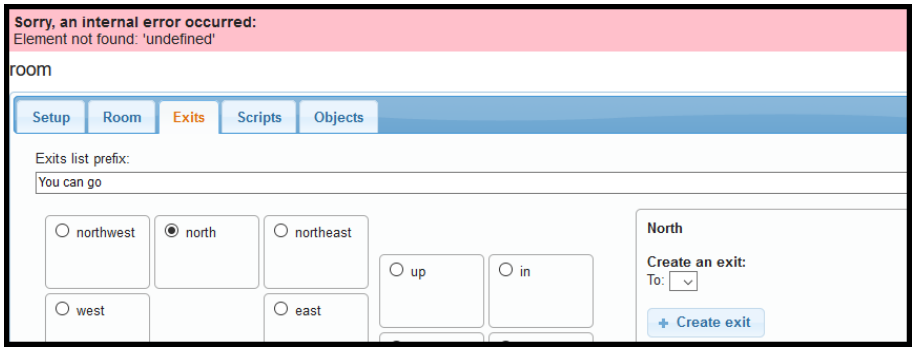

Abbildung 5: Fehlermeldung „Internal Error“ im Editor von Quest
Quelle: [9]

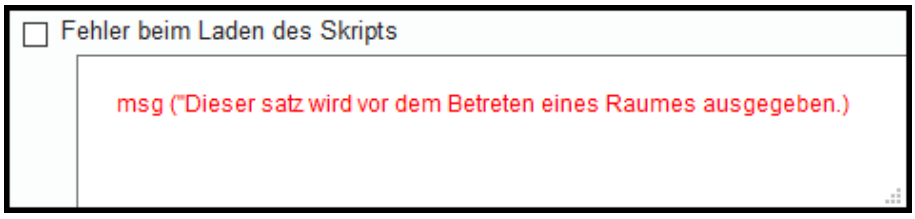

Abbildung 6: Ein Skript-Syntaxfehler im Editor von Quest
Quelle: [9]

ein Anführungszeichen für die Begrenzung eines String-Wertes vergessen. An diesem Punkt ist dem Nutzer bekannt, dass sein Skript einen Fehler enthält. Das Ausbleiben von Zeilenangaben und Fehlerkategorien macht es ihm jedoch schwer, ein effektives „Debugging“ in umfangreicheren Skripten durchzuführen. Dieser Nachteil wird auch in der Dokumentation von Quest [17] beschrieben. Es ist laut Dokumentation möglich Spiele fertigzustellen und zu starten, obwohl Syntaxfehler in ihren Skripten enthalten sind. Dies sollte jedoch vermieden werden, da Quest in solchen Fällen oftmals versucht die Probleme selbstständig zu korrigieren, um die Ausführbarkeit des Spiels zu gewährleisten. Dabei können Code-Abschnitte verloren gehen, wodurch ein noch größeres Durcheinander herbeigeführt wird [17]. Viel zu spät, nämlich erst, wenn die Ausführung eines erstellten Spiels aufgrund fehlerhafter Skripte durch Laufzeitfehler fehlschlägt, wird dem Nutzer über das Fenster des Spiels eine Liste von Fehlermeldungen angezeigt, welche genauere Vermutungen zur Fehlerursache zulassen. Ein Beispiel hierfür ist in Abbildung 7 zu sehen. Hier wurde versucht das Spiel, welches das fehlerhafte Skript aus Abbildung 6 implementiert hat, zu starten. Die Programmausführung wird hierbei trotz bestehender Fehler nicht aufgehalten. Das Programm selbst terminiert jedoch sofort.

Nicht jeder Autor bevorzugt es unter Online-Zwang zu arbeiten. Außerdem besteht unter vielen Autoren der Wunsch, die Spieldaten, welche sie unter großem Aufwand erstellt haben, zusätzlich auf der eigenen Maschine zu sichern. Die Abhängigkeit der Sicherheit ihrer Spiele von einer fremden Datenbank kann unter Umständen abschreckend wirken. Aus diesem Grund stellt Quest in der

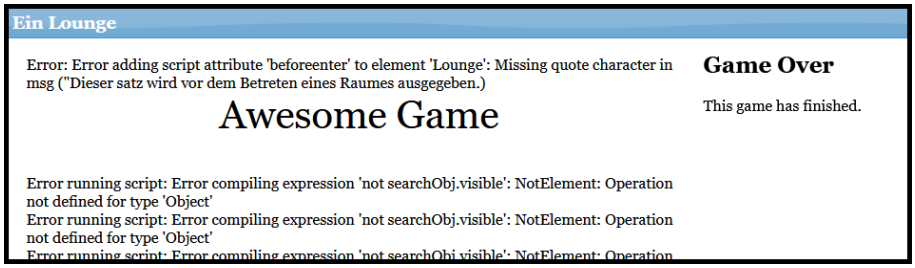

Abbildung 7: Ein Laufzeitfehler im Spiel-Fenster von Quest
Quelle: [9]

Version 5 ein Offline-Tool für Autoren bereit, welches das Erstellen, Speichern, Teilen und auch Spielen von Text-Adventures in einem eigenen Programm und nicht nur in der Client-Anwendung im Browser möglich macht. Das Programm kann über eine Setup-Datei auf Maschinen mit Windows-Betriebssystem installiert werden. Versionen des Editors für andere Betriebssysteme werden auf der offiziellen Website [9] nicht angeboten.

Das Offline-Tool von Quest stellt nicht nur eine Anwendung zur Verwaltung und Bearbeitung von Quest-Spielen dar, es ermöglicht bei einer bestehenden Internetverbindung auch das Browsen in der Spieledatenbank von Quest mithilfe einfacher Filterfunktionen. Eigene Spiele sowie Spiele der Datenbank können so offline mithilfe des Programms in ähnlicher Weise gespielt werden, wie es auch in der Web-Applikation der Fall ist. Hierfür muss sich das gewählte Spiel auf der Festplatte des Nutzers befinden. Nach dem Start des Quest-Programms befindet sich der Nutzer in einer Übersicht, welche den Wechsel zwischen den Reitern „Starten“ und „Erstellen“ anbietet. Hierüber kann entweder das Fenster zum Browsen in der Spieledatenbank, oder aber ein Untermenü zum Start des Editors aufgerufen werden. In diesem Untermenü kann der Nutzer ein neues Spiel erstellen, eine bestehende Spieldatei laden oder aber einen Link zur Website des Tutorials von Quest aufrufen. Die gleichen Funktionen zum Starten des Editors, zusammen mit Konfigurationsmöglichkeiten für Sprache und Aussehen des Programms, finden sich auch in dem Navigationsmenü in der oberen linken Ecke des GUI. Hier können Farben und Schriftarten angepasst sowie die Sounds des Editors ausgeschaltet werden.

Der Editor selbst besitzt einen eigenen Stil in Bezug auf sein Aussehen, das Layout und die Funktionen sind jedoch größtenteils identisch zur Online-Alternative. Unterschiede finden sich beim Editor im Umgang mit Spieldateien mit der für Quest einzigartigen Dateiendung „.aslx“. Der Online-Editor ermöglicht im Vergleich zum Offline-Tool keine lokale Verwaltung von Spieldateien. Die Daten in diesen Dateien liegen im XML-Format vor unter der Einhaltung gewisser Standards, welche jede Spieldatei erfüllen muss. Das äußerste Objekt jeder Spieldatei bildet ein ASL-Element, welches die verwendete Version der Skriptsprache angibt. Darin werden sämtliche Daten, von Bibliothek-Includes bis hin zu Räumen, Skripten und dem Spieler, abgespeichert [18]. In Abbildung 8 ist eine Beispielinstanz eines neuen unbearbeiteten Spiels als ASLX-Datei zu sehen. Es ist erkennbar, dass innerhalb des ASL-Elements zwei Bibliotheken mithilfe von Include-Elementen deklariert

```
<asl version="580">
    <include ref="Deutsch.aslx"/>
    <include ref="Core.aslx"/>
    <game name="test">
        <gameid>14568607-5dbe-46c0-8b90-df92e51eb227</gameid>
        <version>1.0</version>
        <firstpublished>2021</firstpublished>
    </game>
    <object name="room">
        <inherit name="editor_room" />
        <isroom />
        <object name="player">
            <inherit name="editor_object" />
            <inherit name="editor_player" />
        </object>
    </object>
</asl>
```

Abbildung 8: Beispielinstanz eines neu initialisierten Spiels als ASLX-Datei
Quelle: Eigene Darstellung

sind. Weiterhin beinhaltet das Spiel bereits zwei Objekte auf niedrigster Ebene. Zuerst ist ein Game-Objekt eingetragen, welches in jedem funktionierenden Spiel einmalig vorhanden sein muss. Zusätzlich existiert bereits ein Raum, in welchem sich das initiale Spieler-Objekt befindet.

2.3 Grafische Benutzerschnittstellen zur Datenbank-Interaktion

Für die Erstellung eines datenbankbasierten Editors ist die Realisierung einer intuitiven und leicht zu bedienenden Benutzerschnittstelle bedeutsam. In diesem Kapitel werden existierende Softwarelösungen für Probleme dieser Art vorgestellt und interpretiert.

2.3.1 Adminer und Adminer Editor

Vollständig entwickelte grafische Benutzeroberflächen zur Verwaltung von Datenbanken gibt es zum aktuellen Zeitpunkt in einer großen Menge. Einige Vertreter dieser Kategorie sind phpMyAdmin, TOAD (Tool for Oracle Application Developer) oder auch Adminer. Letzterer wird an dieser Stelle in Hinsicht auf den Nutzen als Editor für das Lernprogramm der Datenbank-Gruppe genauer untersucht.

Adminer wurde von dem tschechischen Programmierer Jakub Vrána im Jahre 2007 mit dem Ziel entwickelt, eine schlankere Alternative zu phpMyAdmin zu schaffen. Das gesamte Programm ist in einer einzigen PHP-Datei enthalten, welche lediglich auf dem Server abgelegt werden muss. Mithilfe von Adminer wird den Nutzern eine vollständige Datenbankverwaltung über die Verwendung eines GUI ermöglicht. Hierzu gehören sowohl Funktionen zur Manipulation von Metadaten, wie Datenbank-, Schema-, Tabellen- und Trigger-Definitionen, als auch Möglichkeiten zur direkten Datenänderung [19]. Ein solches für Adminer typisches Interface ist in Abbildung 9 erkennbar. Für die Verwendung des Tools ist ein Log-in mit Nutzername und Passwort erforderlich. Zu einem spä-

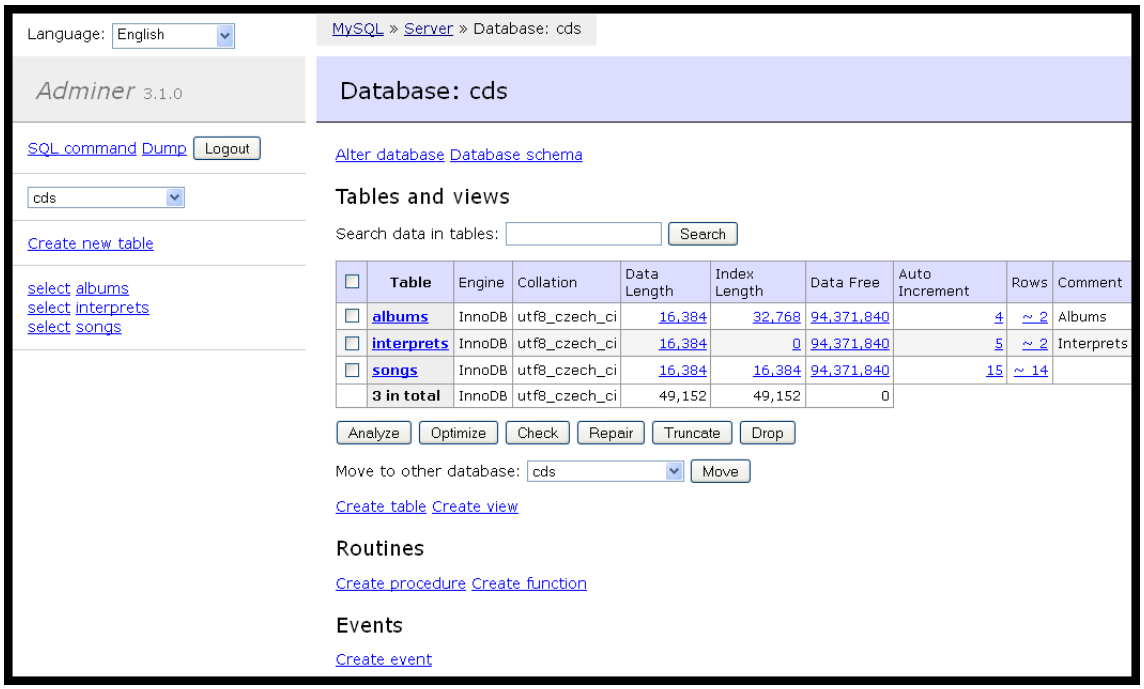

Abbildung 9: Ein typisches Adminer-GUI
Quelle: [20]

teren Zeitpunkt wurde zusätzlich das Programm „Adminer Editor“ veröffentlicht, welches eine vom Funktionsumfang reduzierte Variante des Adminer darstellt, diesem jedoch in der Grundstruktur gleicht. Hier wird der Fokus auf die Erstellung, Anpassung und Löschung von Daten in vorgefertigten Datenbankstrukturen gelegt. Die Änderung der Strukturen selbst ist jedoch mit diesem Tool, anders als mit Adminer, nicht vorgesehen. Adminer Editor wurde mit der Absicht entwickelt, die Anwendbarkeit und Benutzerfreundlichkeit im Vergleich zu Adminer weiter zu steigern und damit die Einarbeitungszeit in das Werkzeug für neue Benutzer zu reduzieren [20]. Zuletzt sollte Beachtung finden, dass sowohl Adminer als auch Adminer Editor bis zu einem gewissen Grad Erweiterungen unterstützen. Hierfür müssen lediglich die gewünschten Funktionen durch eigene Funktionen innerhalb des PHP-Codes ersetzt werden. Eine Übersicht aller überschreibbaren Methoden und eine API-Referenz mit den Bezeichnern der wichtigsten GUI-Elemente kann zur effektiveren Einarbeitung auf der offiziellen Website [21] gefunden werden.

Abschließend ist zu erkennen, dass Adminer Editor richtige Ansätze bietet, welche für den zu entwickelnden Editor des Lernprogramms benötigt werden. Mithilfe dieses Tools zur Datenbankverwaltung werden die meisten Funktionen einer Datenbank auch Nutzern zugänglich gemacht, die noch nicht ausreichend SQL beherrschen. Eine solche grafische Benutzerschnittstelle könnte in diesem Sinne also auch von Autoren des Lernspiels genutzt werden, um neue Spieldaten in bestehende Spielschemata einzufügen, sowie neue Spielschemata zu erstellen. Adminer Editor verwendet die Apache-Lizenz, welche eine freie kommerzielle als auch nicht-kommerzielle Nutzung der Software ermöglicht. Hierdurch könnten weitere Kosten für die Entwicklung des Lernprogramms gespart

werden. Probleme in Bezug auf die Zugriffsrechte einzelner Nutzer entstehen in der Regel bei der Verwendung von Adminer und Adminer Editor nicht. Diese bieten zwar keine direkten Funktionen zur Verwaltung einzelner Nutzer und Nutzerrechte, in ihrer zugrunde liegenden Datenbank können jedoch alle nötigen Vorkehrungen getroffen werden. Wichtig wäre es hierbei zu beachten, dass Spielautoren, welche größtenteils auf der gleichen Befugnisebene wie gewöhnliche Nutzer des Lernprogramms stehen, keinen allumfassenden Zugang zur Datenbank des Lernprogramms erhalten dürfen. Dies würde einen großen sicherheitstechnischen Fehler darstellen. Im schlimmsten Fall könnten Nutzerdaten anderer Spieler eingesehen, oder sogar die gesamte Datenbank gelöscht werden.

Auch die Erweiterung des Adminer Editors um Systeme zur gezielten Unterstützung von Autoren bei der Erstellung von Spieldaten könnte eingeschränkt möglich sein. Die Grenze des Möglichen bildet hierbei die API [21] des Programms. Genauere Untersuchungen der Schnittstelle und erste Erweiterungsversuche führen an dieser Stelle jedoch für die Thematik dieser Arbeit zu weit, da der praktische Einsatz der Software aufgrund des variierenden Technologie-Stacks unwahrscheinlich ist. Adminer und Adminer Editor wurden in PHP entwickelt. Eine Integration in das Elm-Programm des Projekts ist somit zwar möglich, wird jedoch von den Entwicklern noch nicht zufriedenstellend unterstützt. Eine gewisse Nähe des Programmcodes des Editors zur Client-Anwendung wäre für zukünftige Forschungen jedoch von Vorteil, wie in Kapitel 4.5 näher erläutert wird.

2.3.2 Oracle Forms

Im Vergleich zu Tools wie Adminer und Adminer Editor, welche eine nahezu vollendete Datenbankschnittstelle mit eingeschränkten Möglichkeiten zur Anpassung und Entwicklung darstellen, existieren auch Softwarelösungen zur selbstständigen Erstellung von individuellen Schnittstellen. Eines der bekanntesten Softwarepakete dieser Kategorie ist Oracle Forms, welches eine Komponente der Oracle Fusion Middleware [22] ist. Oracle Forms stellt seinen Nutzern Anwendungen zur Verfügung, deren Fokus auf der Entwicklung und Bereitstellung grafischer Benutzeroberflächen im Datenbankkontext liegt. Mithilfe dieser Anwendungen kann eine hocheffiziente und stark verknüpfte Datenbankadministration erreicht werden. Dabei ist eine Integration sowohl mit Java als auch mit Web-Services durchführbar [23].

In dieser Arbeit wird der Fokus der Diskussion auf das Anwendungspaket Oracle Forms Developer (OFD) gelegt, welches Bestandteil von Oracle Forms ist. OFD wird innerhalb der Anwendungsumgebung von Oracle Forms für die Entwicklung von reichhaltigen grafischen Benutzerschnittstellen (sogenannten „Forms“) eingesetzt, welche auf Oracle-Datenbanken zugreifen und die zugehörigen Daten präsentieren können. Dies wird sowohl in [23] als auch in [24] beschrieben. Hierbei entstehen Java-Client-Programme, ohne dass für den Nutzer die Notwendigkeit besteht Java-Code zu schreiben. Diese sind laut [24, S. 2] für die Arbeit mit dem Internet optimiert und ermöglichen die schnelle Berechnung großer Mengen an Daten, Analysen und Transaktionen. Für

die Arbeit innerhalb der Anwendung setzt OFD auf Konzepte des Rapid Application Developments (RAD), welche die benötigte Entwicklungszeit für Benutzerschnittstellen reduzieren sollen. Unter anderem werden dem Benutzer hierfür Assistenten (sogenannte „Wizards“) für die Erstellung und Modifikation von Datenblöcken und Layout angeboten. Weiterhin kann mit Property-Paletten gearbeitet werden. Diese dienen der Erstellung, der Bearbeitung und dem Vergleich von Properties unterschiedlicher GUI-Objekte, welche vom Nutzer in Modulen der erstellten Menüs und der Schnittstelle angelegt wurden [24, S. 3]. OFD bietet noch weitere solcher Unterstützungsfunktionen, eine ausführliche Beschreibung dieser würde den Rahmen dieser Arbeit jedoch überschreiten.

Für die zu behandelnde Thematik ist relevant, ob mithilfe der Möglichkeiten von Oracle Forms eine Benutzerschnittstelle erstellt werden kann, welche den Anforderungen eines Editors für das Lernprogramm der Datenbank-Gruppe genügt. Hierzu werden im offiziellen „Technical Overview“ [24] die Möglichkeiten, welche OFD dem Nutzer bietet, beschrieben. Dazu gehören unter anderem Baum-Steuerelemente, Tabs, Checkboxen, Pop-Up-Listen, Tooltips sowie berechenbare Ein- und Ausgabefelder. Weiterhin ermöglicht das Anwendungspaket die Integration eigener Java-Komponenten zur zusätzlichen Erweiterung der Benutzerschnittstelle. Denkbare Anwendungsbeispiele für diese Funktionalität sind benutzerdefinierte Rollover-Buttons, Hyperlinks und eine clientseitige Upload-Funktion. Ein Beispiel eines mit OFD erstellbaren User Interface (UI) ist in Abbildung 10 dargestellt. Die in OFD angelegten Module kommunizieren bei der praktischen Anwendung der Benutzerschnittstelle über einen einzigen HTTP-Listener, welcher einzelne Anfragen an ein Servlet weiterleitet, welches diese wiederum an das Forms-Laufzeitsystem verweist. Diese Laufzeitumgebung ist letztendlich verantwortlich für die Datenbankkommunikation. Sämtliche dieser Kommunikationsstrukturen sind in dem Anwendungspaket von Oracle Forms enthalten und jederzeit nutzbar [24, S. 7].

Das Produkt der Oracle Corporation, Oracle Forms, bietet Nutzern eine weit ausgebaute und längerfristig unterstützte Möglichkeit, professionelle Benutzerschnittstellen zu erzeugen, zu unterhalten und zu pflegen. Da die Software unter Verfolgung kommerzieller Ziele entwickelt wurde, ist sie größtenteils auf die alleinige Kompatibilität mit anderen Oracle-Produkten ausgelegt. Hieraus ergibt sich, dass Oracle-Forms auf dem standardmäßigen Anwendungsweg lediglich in Verbindung mit Oracle-Datenbanken verwendet werden kann. Die Migration von Oracle-Datenbanken hin zu PostgreSQL-Datenbanken unter der Weiterverwendung von Oracle-Forms stellt eine Aufgabe dar, welche selbst größere IT-Dienstleister bisher nicht vollständig bewältigen konnten [25].

Zusammenfassend kann behauptet werden, dass Oracle Forms durchaus ein mächtiges Werkzeug bei der Entwicklung eines Editors auf Datenbankbasis, oder auch des gesamten Lernprogramms sein könnte. Die Einschränkungen durch Inkompatibilität mit Konkurrenzprodukten, welche bei der Verwendung von Oracle-Software auftreten, würden jedoch eine vollständige Neustrukturierung des bestehenden Lernprogramms erfordern. Weiterhin wäre die Verwendung von Oracle Forms mit Lizenzkosten für das Projekt verbunden, welche bei der Anwendung von Alternativprodukten

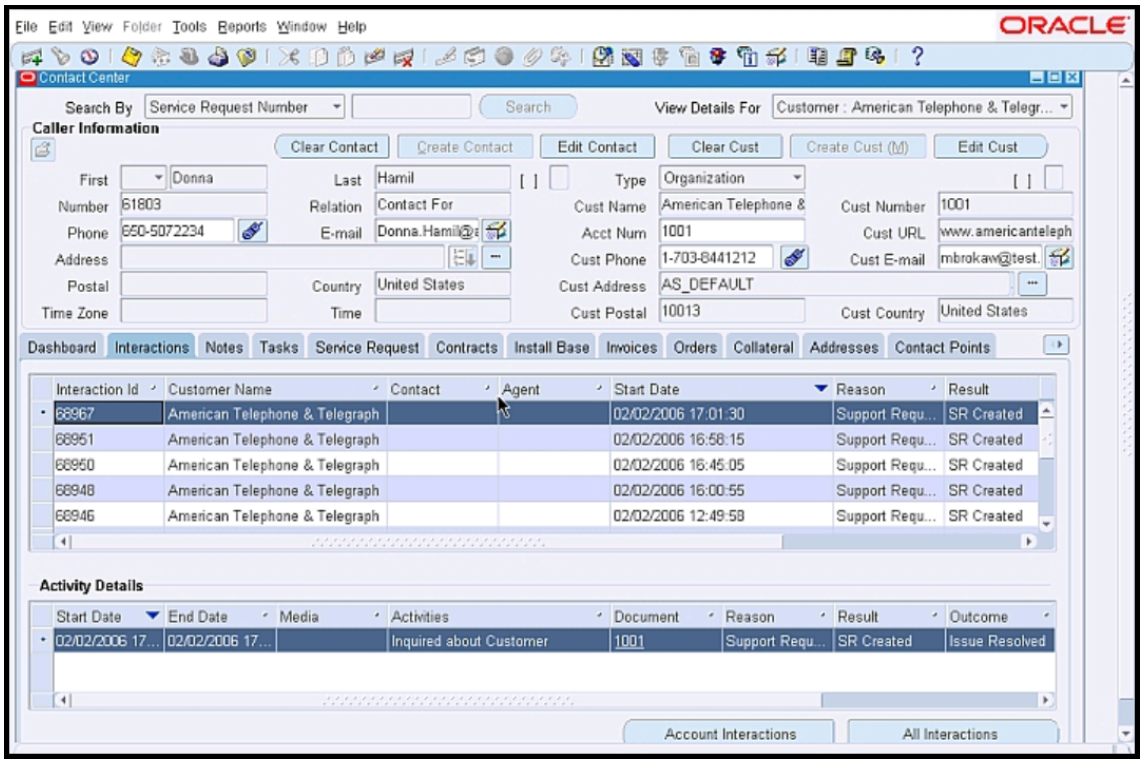

Abbildung 10: Ein Beispiel eines mithilfe von OFD erstellbaren UI
Quelle: [24]

vermieden werden könnten.

2.4 Grundlagen von Elm

In diesem Kapitel werden Grundlagen der weniger bekannten Programmiersprache Elm vermittelt, welche für den größten Teil der vorliegenden Arbeit verwendet wird. Hierbei wird auf den grundsätzlichen Aufbau und die Funktionsweise von Elm-Programmen eingegangen. Anschließend wird auf Elemente der Kernsprache verwiesen und diese werden peripher erläutert. In [26, S. 411] wird erklärt, dass Elm als eine deklarative Sprache zur Erstellung grafischer Web-Benutzeroberflächen geschaffen wurde, welche zu JavaScript kompiliert. Dieser kompilierte Code kann anschließend ohne weiteres in den HTML-Code klassischer Web-Applikationen eingebettet werden. Typische Aufgaben, denen sich Entwickler grafischer Oberflächen stellen müssen, sollen durch Elm vereinfacht werden. Unter anderem kann die Sprache durch seine funktionale Natur Laufzeitfehler garantiert ausschließen. Weitere Vorteile, welche Elm aufgrund des Konzepts der Funktionalität mitbringt, sind laut [27] benutzerfreundliche Fehlermeldungen, die zuverlässige Durchführbarkeit von „refactorings “ sowie die automatisch durchgesetzte semantische Versionierung aller existierender Elm-Packages [28].

Die Entscheidung zur Verwendung von Elm im Lernprogramm der Datenbank-Gruppe wurde mehrere Jahre vor Entstehung dieser Arbeit, zu Beginn des Projektes, getroffen. Ein Grund hier-

für könnte gewesen sein, dass Konzepte der Sprache theoretisch als auch praktisch an der MLU vermittelt werden. Dieser Fakt, kombiniert mit Vorteilen wie der Laufzeitfehlervermeidung, führte auch bei dieser Arbeit zu der Entscheidung, Elm für die weiteren Entwicklungen am Client zu verwenden.

Ein typisches Elm Programm der Art „Browser.application“ beschreibt ein Objekt, welches einen Initialisierungszustand, eine View-Funktion, eine Update-Funktion, eine Subscriptions-Funktion sowie Funktionen zur Aktualisierung der URL enthält. Es existieren auch andere, weniger umfangreiche Programmformate für Elm. Für das Projekt der Datenbank-Gruppe wurde jedoch „Browser.application“ gewählt, weshalb die Alternativen nur geringe Relevanz für diese Arbeit besitzen und daher bei den Erläuterungen vernachlässigt werden. Die drei erstgenannten Teile des Objektes bilden hierbei die Basis jedes Elm-Programms.

Im Initialisierungszustand wird vorerst der Zustand der Anwendung nach Programmstart im sogenannten „Model“ festgelegt. Zusätzlich werden erste auszuführende Anweisungen in Form von Kommandos an die Update-Funktion weitergegeben. Die View-Funktion stellt eine Möglichkeit dar, den aktuellen Zustand des Programms in darstellbaren HTML-Code umzuwandeln. Mithilfe der Update-Funktion kann der Zustand des Programms über Objekte eines Datentyps für Benachrichtigungen, den sogenannten „Messages“, angepasst werden. Messages werden unter anderem durch HTML-Events ausgelöst. Ein Beispiel hierfür ist das Klicken eines Buttons, welcher mit einer On-Click Funktion versehen wurde. Subscriptions erlauben es einem Elm Programm externe Events zu verfolgen. Hierzu zählen unter anderem die Uhrzeit, Mauszeiger-Events und auch geografische Änderungen. Im Programmcode des Projektes wurden bereits Vorbereitungen zur zukünftigen Verwendung von Subscriptions getroffen, vor der Entstehung dieser Arbeit haben sie jedoch noch keine Anwendung gefunden.

Ähnlich verhält es sich mit den Funktionen „onUrlRequest“ und „onUrlChange“. Diese dienen der Vor- und Nachbearbeitung des Programmzustandes bei der Verwendung von Links. Angestrebt wird für Web-Applikationen wie das Lernprogramm meist eine Verwaltung von URLs der Art, dass die Menge der URLs bijektiv auf die Menge der aufrufbaren Seiten abbildet. Das Laden einer konkreten URL soll also immer die gleiche Seite des Programms hervorrufen. Weiterhin sollte jede Seite ebenfalls nur durch die eine, ihr zugeordnete URL, aufrufbar sein. Für die URL-Funktionen des Elm Programms im Projekt sind bereits abseits dieser Arbeit unterstützende Funktionen implementiert worden. Die angestrebte Funktionalität wurde bis jetzt jedoch noch nicht vollständig umgesetzt.

Zuletzt ist das Konzept der Ports in Elm erwähnenswert. Diese dienen in einem Elm Programm als direkte Schnittstelle zu JavaScript-Programmteilen. Sie fungieren also nach dem Konzept einer API, haben jedoch im aktuellen Programm noch keine Verwendung gefunden.

Die Kernsprache von Elm besitzt Ähnlichkeit mit der populäreren funktionalen Programmiersprache Haskell. Wichtige Gemeinsamkeiten sind hierbei sowohl unveränderliche Variablen als auch

reine Funktionen. Die Eigenschaft der Unveränderlichkeit bedeutet für eine Variable, dass ihr Wert nach der Initialisierung konstant bleibt. Hierdurch werden gewöhnliche Ausdrücke wie „x=x+1“, welche in imperativen Sprachen regelmäßig Anwendung finden, in Elm unmöglich. Stattdessen könnte „x+1“ als Ausdruck definiert werden, welcher den Wert von x an der passenden Stelle in die Formel einsetzt. Konstanten wie x besitzen in Elm jedoch üblicherweise eine geringe Lebensdauer, wodurch ihr Bezeichner anschließend wiederverwendet werden kann. Das Konzept der reinen Funktionen steht im direkten Zusammenhang mit unveränderlichen Variablen. Diese Eigenschaft besagt, dass Funktionen in Elm sowohl bei gleichem Input den exakt gleichen Output liefern, als auch keine Effekte auf den Programmzustand neben der eigentlichen Funktionsdurchführung haben dürfen.

Vergleicht man Elm weiter mit Haskell so wird klar, dass auch hier ein Zusammenspiel von Funktionen zur Werttransformation und Ausdrücken zur Fallunterscheidung stattfindet. Solche Ausdrücke sind beispielsweise If- und Case-Statements. Komplexere Datenstrukturen sind in den meisten Fällen aus Listen, Tupeln und Records aufgebaut. Listen und Tupel sind hierbei bekannte Konzepte zur Organisation von mehreren atomaren oder auch komplexeren Daten. Mit ihrer Hilfe kann innerhalb von Elm das „mapping“ von Daten angewandt werden, das eine schlanke Option für die Transformation von Daten darstellt. Elm-Records hingegen beschreiben eine Art von Objekten, welche ohne Ordnung Werte unterschiedlichen Typs (sogenannte „Fields“) zusammen mit einem jeweils zugeordneten Namen verwalten. Der Zugriff auf einzelne Felder eines Records erfolgt durch das Ansprechen ihres zugehörigen Namens. Erstellt werden Records meist nach Vorbild eines vom Programmierer festgelegten „type alias“. Dieses Konzept ähnelt in seiner Funktionsweise einem Objekt in Java, welches aus einer Klasse mit Attributen erzeugt wird.

Weiterhin ermöglicht Elm die Arbeit mit benutzerdefinierten Typen. Diese nehmen immer einen Wert aus einer Menge von endlich vielen Werten an. In ihrer Verwendung besitzen benutzerdefinierte Typen die gleichen Freiheiten wie alle standardmäßigen Typen in Elm. Zusätzliche Informationen zu den verschiedenen Konzepten von Elm können auf der offiziellen Tutorial-Seite guide.elm-lang.org [27] nachgeschlagen werden. Weitere Ausführungen würden an dieser Stelle an der Thematik dieser Arbeit vorbeiführen.

2.5 Ausgangsposition

Diese Arbeit schließt an die vorhergehende Arbeit von Mitarbeitern der Datenbank-Gruppe an. Es wurde bereits ein spielbarer Prototyp erstellt, welcher nun weiter ausgebaut wird. Dieser Prototyp besteht im Kern aus Java-Servlets, welche auf einem Tomcat Webserver ausgeführt werden. Das Servlet leitet die Kommunikation zwischen der PostgreSQL-Datenbank und dem Spiele-Client. Die Datenbank verwaltet neben einer Spiel-Datenbank, welche alle Elemente des aktuell spielbaren Textadventures enthält und mehreren Spieler-Datenbanken, welche den Spielstand einzelner Nutzer abbilden, auch die registrierten Nutzer in einer gesonderten User-Datenbank. Der Spiel-

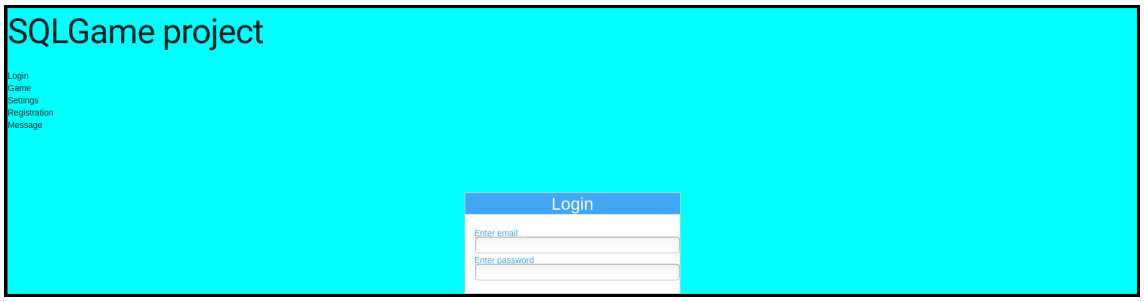

Abbildung 11: „LoginPage“ des alten Clients
Quelle: Eigene Darstellung

Client kommuniziert neue Registrierungen, Log-ins und Spielbefehle mithilfe von HTTP an die Servlets. Registrierungen, Log-ins und textuelle Spielbefehle (zum Beispiel: „go north“) werden im Servlet in SQL-Befehle übersetzt, welche die zuständigen Daten in der Datenbank korrekt ändern können. Das Servlet übermittelt der Datenbank über eine Java-Schnittstelle die ausstehenden SQL-Befehle. Innerhalb der Datenbank müssen anschließend lediglich die kommunizierten SQL-Befehle ausgeführt werden, um die Anweisungen des Clients in einen Datenbankzustand zu übertragen. Antworten der Datenbank auf SELECT-Befehle und Verwaltungsdaten, welche durch das Servlet ausgelesen werden (zum Beispiel: „Last-Log-in“), vermittelt das Servlet mithilfe von HTTP an die Client-Anwendung. Diese zeigt Antworten auf SELECT-Befehle in der Konsole an und verarbeitet übertragene Verwaltungsdaten.

Die ursprüngliche Client-Anwendung stellt eine Single-Page-Application dar, die ein User-Interface auf einem rudimentären Entwicklungsstand besitzt. Das Hauptmenü der ehemaligen Client-Anwendung wird auf jeder Seite, links zentriert, an gleicher Stelle angezeigt und umfasst den Titel des Projektes sowie fünf Menüpunkte. Der Platzverbrauch des Hauptmenüs auf jeder Seite der ehemaligen Anwendung ist enorm. Etwa 30% des Platzes jeder Seite wird durch das Menü eingenommen, welches sich horizontal über das Browser-Fenster ausbreitet. Jeder Menüpunkt ist hierbei auch weit rechts, außerhalb des optisch gekennzeichneten Bereiches, anwählbar und verweist auf eine eigene Client-Seite. In der folgenden Arbeit werden diese Seiten mit dem englischen Begriff „Page“ bezeichnet, um den Benennungen innerhalb der Implementierung folgen zu können. Insgesamt bieten die Seiten namens „LoginPage“, „GamePage“ und „RegistrationPage“ bereits Funktionalität, wohingegen „SettingsPage“ und „MessagePage“ zu leeren Seiten führen. Nach dem erstmaligen Öffnen der Client-Anwendung wird der Nutzer auf die „LoginPage“ verwiesen. Wie in Abbildung 11 zu erkennen ist, besteht diese Seite initial aus einer Überschrift und zwei Input-Feldern. Die Überschrift gibt dem Nutzer ein Feedback, auf welcher Seite des Clients er sich momentan befindet. Die Input-Felder sind mit „Enter email“ und „Enter password“ beschriftet. Nach Eingabe von formal korrekten Nutzerdaten erscheint ein grüner „Submit“-Button unterhalb des Passwort-Eingabefeldes. Formale Korrektheit bedeutet hierbei, dass beide Input-Felder nicht leer sind und die E-Mail-Adresse mindestens ein „@“-Zeichen und einen Punkt enthält. Wird eine formal inkorrekte

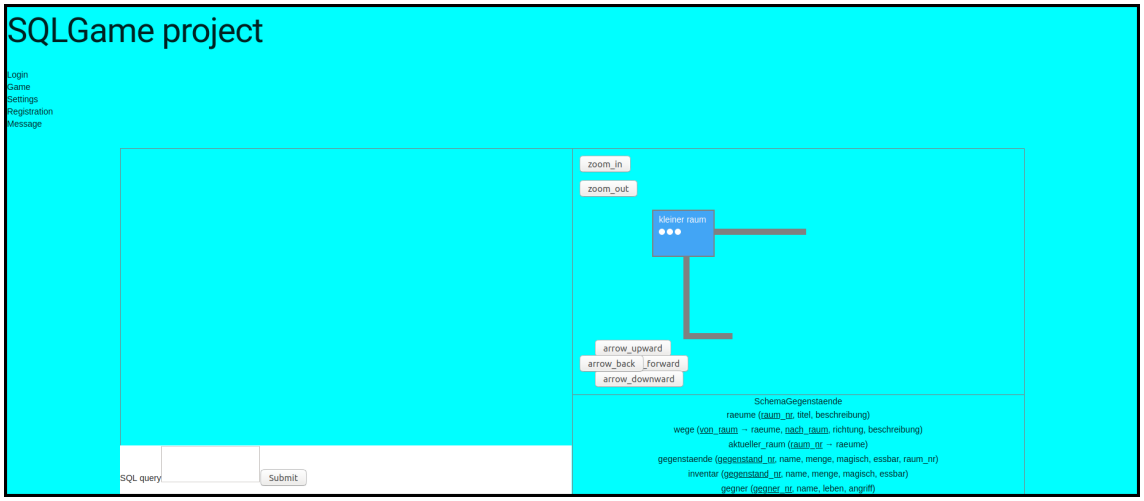

Abbildung 12: „GamePage“ des alten Clients
Quelle: Eigene Darstellung

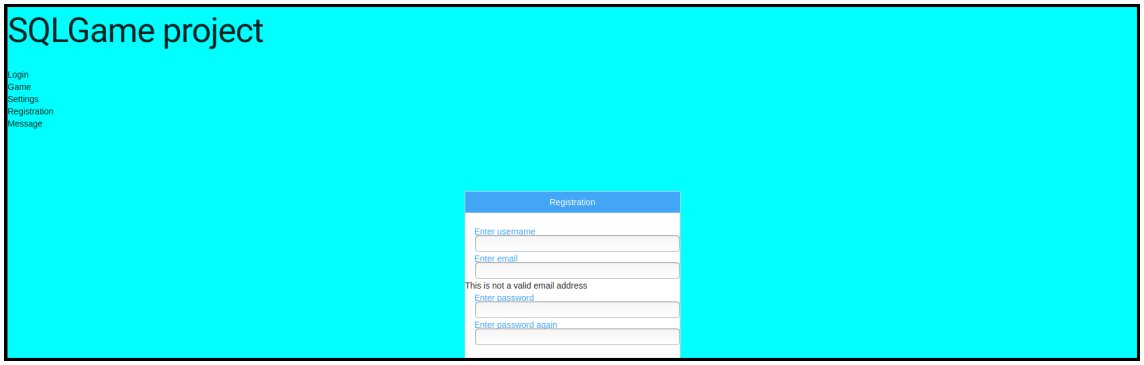

Abbildung 13: „RegistrationPage“ des alten Clients
Quelle: Eigene Darstellung

Zeichenkette in das E-Mail-Feld eingetragen, so erscheint die Fehlermeldung „This is not a valid email address“. Durch das Abschicken von Log-in-Daten, welche zu keinem registrierten Nutzer passen, wird die Fehlermeldung „Could Not Authenticate you“ angezeigt. Ein erfolgreicher Log-in leitet den Nutzer weiter auf die „GamePage“ der Anwendung, die in Abbildung 12 zu erkennen ist.

Die „GamePage“ besteht linksseitig aus einem Fenster für Konsolen-Output und einem darunterliegenden Fenster für Konsolen-Input, beschriftet mit „SQL query“ und einem dazugehörigen „Submit“-Button. Rechtsseitig ist ein Fenster mit einer grafischen Übersicht aller besuchten Räume und Wege zu erkennen. Dieser Kartenausschnitt kann, mithilfe der oberen beiden Buttons, rein- und rausgezoomt werden. Die unteren vier Buttons im Karten-Fenster besitzen eine ähnliche Funktionsweise wie der Steuerkompass im Autorensystem „Quest“. Sie ermöglichen die alternative Steuerung des Spielers durch das Klicken von Buttons, statt der eigentlich in Textadventures üblichen Steuerung durch textuelle Konsolenbefehle. Unterhalb der Kartenansicht befindet sich eine Übersicht über alle Tabellen der Spieler-Datenbank, in welchen der Spieler Zugriffsrechte besitzt.

Soll ein neuer Nutzer in der Datenbank registriert werden, so muss der Menüpunkt „Registration“ angewählt werden. In Abbildung 13 ist zu sehen, dass die „RegistrationPage“ des Web-Clients im Aufbau der „LoginPage“ ähnelt. Es existiert eine Überschrift, gefolgt von den vier Input-Feldern „Enter username“, „Enter email“, „Enter password“ und „Enter password again“. Ist in das E-Mail-Feld keine formal korrekte E-Mail-Adresse eingetragen, oder stimmen die Passwörter nicht überein, so wird eine Fehlermeldung angezeigt. Enthalten alle Eingabefelder mindestens ein Zeichen und ist die E-Mail-Adresse formal korrekt (siehe Seite 24), so erscheint unterhalb aller Eingabefelder ein grüner „Submit“-Button. Durch Betätigung dieses Buttons wird der neue Nutzer in die User-Datenbank aufgenommen.

3. Überarbeitung des Clients

Die Client-Anwendung des Lernprogramms befand sich vor der Durchführung dieser Arbeit in einem unzureichenden Entwicklungsstand. Der Funktionsumfang des Programms war nicht ausreichend, um viele grundlegende Aktionen als auch einen angenehmen Spielfluss zu gewährleisten. Unter anderem konnten den Nutzern weder optische Anpassungsmöglichkeiten geboten werden, noch wurden Spieler ausreichend durch intuitive Ein- und Ausgabe-Features unterstützt. Weitere Ausführungen zum ursprünglichen Stand des Clients können in Kapitel 2.5 eingesehen werden.

Die zielführende Idee bei der Überarbeitung des Clients war, eine Anwendung zu schaffen, welche die Funktionalität des ursprünglichen Programms erweitert und als benutzerfreundlichere Schnittstelle zwischen Mensch und Spiel fungiert. Hierfür mussten sowohl die grafische Oberfläche modernisiert, als auch die Menge an Funktionalitäten aufgestockt werden.

3.1 CSS-Framework

Im Rahmen dieser Arbeit soll die Front-End-Entwicklung mithilfe eines CSS-Frameworks durchgeführt werden. Diese Frameworks stellen eine Sammlung von Gestaltungselementen für eine einfache und standardisierte Webentwicklung dar. Eine solche Bibliothek wird in der Regel von erfahrenen Designern entwickelt und ist damit höchstwahrscheinlich besser durchdacht, als Eigenentwicklungen im Rahmen des Projektes sein würden. Die Entscheidung innerhalb dieser Arbeit fiel auf das CSS-Framework „Bootstrap“ in der Version 4.5.3. Diese Bibliothek stellt Nutzern Design-Elemente bereit, die in ihrer Funktionalität häufig durch JavaScript-Code erweitert sind. Besonderen Fokus legt das Framework hierbei auf die Schnelligkeit sowie die responsiven Eigenschaften seiner Elemente [29]. Es existieren durchaus gängige alternative CSS-Frameworks, welche statt Bootstrap hätten verwendet werden können. Einige Beispiele hierfür sind „Foundation“, „Tailwind“ oder auch „Bulma“. Von den untersuchten Frameworks erfüllt Bootstrap jedoch, subjektiv gesehen, die grafischen Ansprüche an das Projekt am besten.

Weiterhin existiert für die Sprache Elm, welche die Grundlage der Client-Anwendung bildet, eine importierbare Bibliothek zur direkten Nutzung von Bootstrap-Elementen mithilfe von Elm-Funktionen [30]. Die Verwendung dieser Bibliothek erspart dem Programmierer Arbeitszeit, da der zu schreibende Code für jede Art von Bootstrap-Elementen drastisch verringert wird. Elm-Bootstrap kann jedoch keine Vorteile in der Funktionalität aufweisen. Das Gegenteil ist eher der Fall. Die Mächtigkeit von Elm-Bootstrap ist abhängig von seinen unabhängigen Entwicklern. Änderungen und Updates des Bootstrap Frameworks bedeuten deshalb nicht, dass diese auch sofort in die Elm-Bootstrap Bibliothek übernommen werden, wodurch Elm-Bootstrap aktuell lediglich in einer veralteten Version bereitgestellt wird. So ist es beispielsweise nicht möglich mithilfe von Elm-Bootstrap das aktuelle Input-Feld der GamePage mit Prepend- und Append-Buttons so zusammenzuführen (Stand 16.11.2020), wie es in der bestehenden Anwendung der Fall sein soll (siehe Abbildung 18).

Da das Warten auf die Implementierung der nötigen Funktionalitäten durch die Entwickler keine Option darstellt, könnte eine hybride Lösung mit Elm-Bootstrap-Funktionen gewählt werden, welche durch klassisches Bootstrap-CSS erweitert wird. Diese hybride Lösung hätte jedoch einen schlechten Stil und eine geringere Übersichtlichkeit im Code zur Folge, weshalb die Entscheidung auf die alleinige Verwendung von Bootstrap-CSS gefallen ist und Elm-Bootstrap deshalb nicht in das Projekt importiert wird. Diese Entscheidung für besseren Code ist jedoch mit zusätzlichem Arbeitsaufwand verbunden, da zum jetzigen Zeitpunkt bereits Teile des Clients im Rahmen dieser Arbeit mit Elm-Bootstrap implementiert wurden. Sämtliche Icons, welche innerhalb dieser Arbeit verwendet werden, stammen aus der Font-Awesome-Bibliothek, die eine große Vielfalt an kostenlosen und hochwertigen Vektor-Icons bereitstellt. Die Entscheidung für diese Bibliothek wird hierbei durch vorhergehende positive Arbeitserfahrungen gestützt.

3.2 Überarbeitete Abschnitte und Funktionen

Für die Entwicklung einer Client-Anwendung, welche den grafischen Ansprüchen und der intuitiven Bedienbarkeit dieser Arbeit genügt, ist sowohl die Implementierung von neuen als auch die Überarbeitung bestehender Funktionalitäten unverzichtbar. Im folgenden Kapitel werden die wichtigsten Änderungen am Client beschrieben und die ausschlaggebenden Argumente und Hintergründe zu den Designentscheidungen erläutert.

3.2.1 Navigation

Der erste Schritt zur Verbesserung der Client-Anwendung stellt die Umgestaltung der Menüfunktion zur Navigation zwischen den einzelnen Seiten dar. Mit dem unüblich hohen Platzverbrauch des ursprünglichen Menüs ist eine Verschiebung aller Seiten des Clients auf der Y-Achse, um die Menühöhe nach unten, verbunden. Aufgrund dieser Verschiebung ist für die Verwendung des alten Clients auf jeder Seite ein, die gesamte Seite umfassender, Scrollbar auf der Y-Achse notwendig. Ein solcher Scrollbar stellt ein zusätzliches Navigationselement dar, welches Nutzer erkennen sowie im Zusammenhang mit der Gesamtanwendung verstehen müssen. Anschließend wird dem Nutzer bei jeder seiner Aktionen die Entscheidung abverlangt, das Navigationselement entweder zu verwenden oder zu ignorieren. Dieser Vorgang ist mit kognitiver Arbeit für den Nutzer verbunden. Für ein einzelnes Navigationselement ist diese aufzubringende Leistung nicht besonders hoch, aber im Rahmen der gesamten Anwendung betrachtet kann sich dieser Effekt aufsummieren [13, S. 56]. Folglich verringert sich mit jedem zusätzlichen Navigationselement die Übersichtlichkeit des Clients, was eine Reduzierung der Benutzerfreundlichkeit nach sich zieht. Ein sparsamer Umgang mit diesen Elementen ist also sowohl bei der Überarbeitung des Hauptmenüs, als auch bei der nachfolgenden Arbeit am Client zu beachten.

Als Ersatz für das ehemalige Hauptmenü des Clients wurde ein Bootstrap-Navbar-Element gewählt (siehe Abbildung 14) . Dieser nimmt auf horizontaler Ebene die gesamte Breite des Browser-

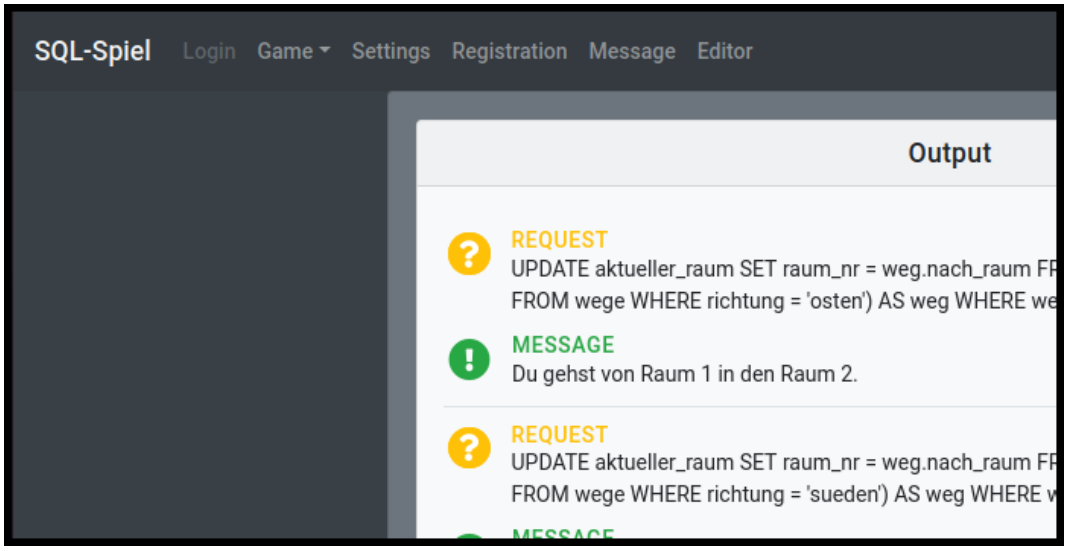

Abbildung 14: Ausschnitt vom neuen Hauptnavigationsmenü (oben)
Quelle: Eigene Darstellung

Fensters ein, wodurch auch in zukünftigen Versionen ausreichend Seiten-Tabs dargestellt werden können. Mit einer Höhe von 52 Pixeln ist der Navbar flach genug, um beliebige Inhalte der Seiten unter ihm, unter Vermeidung eines Scrollbars, darstellen zu können.

Tabs des Navbar, die aus dem aktuellen Zustand der Client-Anwendung nicht erreicht werden sollen, werden mithilfe der Bootstrap-Klasse „disabled“ deaktiviert. Deaktiviert bedeutet hierbei eine farbliche Kennzeichnung, ein Ausbleiben des üblichen Hover-Effekts sowie das Ausbleiben von Mauszeigeränderungen. Dies ist momentan für den Login-Tab der Fall, sobald ein Nutzer erfolgreich eingeloggt ist (siehe Abbildung 14). Ist noch kein Nutzer in den Client eingeloggt, so betrifft dieser Effekt sowohl die Tabs der GamePages als auch die der Settings und des Editors. Alle diese Seiten verwalten entweder aktuell Nutzerspezifische Daten, oder werden dies in zukünftigen Versionen tun, weshalb ein Zugriff vor dem Login keinen Sinn ergeben würde.

Zuletzt bietet das Navbar-Element des Bootstrap-Frameworks auch bereits implementierte Funktionen zur Realisierung einer responsiven Anwendung. Bei ausreichender Reduzierung der Breite des Viewports wird der Navbar in ein Dropdown-Menü transformiert, welches nach dem Öffnen die Seiten-Tabs vertikal angeordnet bereitstellt. Die Frage, ob der Client überhaupt eine responsive Anwendung werden soll, konnte aufgrund des frühen Entwicklungsstandes des Projektes noch nicht abschließend geklärt werden. Der Fokus der Entwicklung liegt in dieser Arbeit auf der Ausarbeitung einer Client-Anwendung im Vollbild-Format mit einer Bildschirmauflösung von 1920×1080 Pixeln. Es ist zu beachten, dass die Bildschirmauflösung nicht gleichzusetzen ist mit der Größe des Viewports im Browser. Sie kann zwischen den verschiedenen Arten von Browsern variieren. In dieser Arbeit wird mit Mozilla Firefox in der Version 85.0.1 innerhalb eines Ubuntu Betriebssystems gearbeitet. Aus den genannten Gründen ist die Relevanz der responsiven Eigenschaften des Navbar für diese Arbeit eher gering. Zukünftige Forschungen zur Bedeutung von

responsiven Eigenschaften für den Client könnten jedoch auf dieser Funktion aufbauen.

3.2.2 LoginPage und RegistrationPage

Die LoginPage und die RegistrationPage des Clients besitzen ein sich ähnelndes Layout mit ähnlichen Funktionen (siehe Abbildungen 11 und 13), weshalb bei ihrer Überarbeitung auf gleiche Weise vorgegangen wird. Das Layout beider Seiten ist bereits zufriedenstellend umgesetzt worden. In beiden Fällen werden die Inhalte der Seite mittig im Browser-Fenster innerhalb eines farblich hervorgehobenen Containers dargestellt. An dieser Stelle muss lediglich der ursprüngliche Seitenaufbau mithilfe des Bootstrap-Grid-Systems [29] imitiert werden. Das Bootstrap-Grid-System ist ein offiziell durch Bootstrap bereitgestelltes Layout-System.

Wie bereits in Kapitel 3.1 erwähnt, werden innerhalb dieser Arbeit die fertigen Bootstrap-Elemente aufgrund designtechnischer Überlegenheit den potenziellen eigenen CSS-Entwicklungen vorgezogen. Der Container innerhalb der LoginPage und auch innerhalb der RegistrationPage besteht ursprünglich aus Div-Elementen, welche durch die Verwendung des „style“-Attributs optisch für die Darstellung von Header- und Body-Inhalten angepasst wurden. An dieser Stelle bietet sich die Verwendung von Bootstrap-Cards an. Diese Elemente stellen Container dar, die neben ihrer designtechnischen Abgestimmtheit mit anderen Bootstrap-Elementen auch Header- und Footer-Funktionalitäten bieten. So können hierdurch auf der jeweiligen Seite der Name im Card-Header sowie die zugehörigen Bootstrap-Input-Elemente im Card-Body implementiert werden (siehe Abbildung 15).

Eine Eigenschaft, welche nicht äquivalent zur Vorlage umgesetzt wird, ist die Sichtbarkeit des „Submit“-Buttons in beiden Seiten. Im alten Client wurde dieser Button nur dann angezeigt, wenn auch formal korrekte (siehe Kapitel 2.5) Nutzerdaten in die Eingabefelder eingetragen worden sind. Für eine stabilere, konsistentere Wirkung der Seiten auf den Nutzer und damit einhergehender erhöhter Übersichtlichkeit werden die „Submit“-Buttons in der überarbeiteten Fassung des Clients jederzeit angezeigt. Solange jedoch keine formal korrekten Nutzerdaten in die entsprechenden Felder eingetragen wurden, bleibt der Button deaktiviert. Dieser Button-Zustand wird, anders als in Kapitel 3.2.1 beschrieben, mithilfe des HTML-Attributs „disabled“ implementiert und dem Nutzer auf ähnliche Weise wie in Kapitel 3.2.1 beschrieben grafisch vermittelt. Diese unterschiedlichen Herangehensweisen an die Implementierung von deaktivierten Elementen werden in der Bootstrap-Dokumentation [29] vorgegeben. Der Vorteil liegt hierbei in der Nutzung der ohnehin existierenden HTML-Funktion für solche Buttons, welche den Bootstrap-Entwicklern das Schreiben einer eigenen CSS-Klasse für diese Elemente erspart.

3.2.3 GamePage

Die wohl wichtigste Seite des Clients ist momentan die GamePage. Sie bietet dem Spieler konkrete Funktionalitäten zum Spielen von SQL-Textadventures sowie Hilfswerkzeuge für eine erleichter-

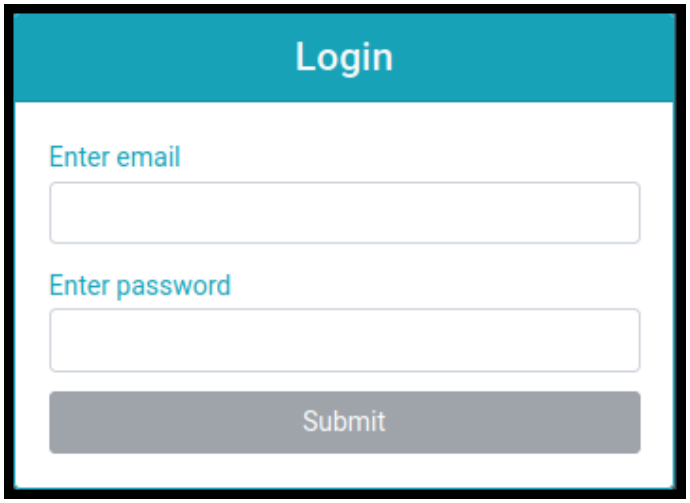

Abbildung 15: Bootstrap-Card innerhalb der neuen LoginPage
Quelle: Eigene Darstellung

te Übersicht und Bedienung (siehe Abbildung 12). Bei der Überarbeitung dieses Abschnitts war einer der ersten nötigen Schritte das Erstellen eines neuen Layouts. Hierbei musste der erhöhte Platzbedarf durch Plugin-Views (siehe Kapitel 3.4) mit den Anforderungen einer übersichtlichen und intuitiven Spielumgebung vereinbart werden. Zu diesem Zweck wurde eine grundsätzliche Orientierung des Seitenaufbaus an den Möglichkeiten, welche das Bootstrap-Grid-System [29] bietet, vorgenommen. Ein solches Grid kann entweder in einen Container mit einer festen maximalen Breite an Pixeln eingebaut werden, oder aber innerhalb eines responsiven Containers Anwendung finden, welcher 100% der Viewport-Breite umfasst.

Inspiriert von diesen Bootstrap-Vorgaben wurden zwei alternative GamePage-Views entwickelt, welche jeweils ihre eigenen Vor- und Nachteile mit sich bringen. Beide können über den Navbar des neuen Clients (siehe Kapitel 3.2.1) mithilfe eines Dropdown-Reiters aufgerufen werden.

Das Layout der ersten Ansicht teilt die GamePage in zwei grafisch getrennte Spalten auf. Hier wird nicht die volle Breite des Viewports genutzt, wodurch dem Nutzer insgesamt weniger zu erfassende Elemente gleichzeitig angezeigt werden. Die linke Spalte der ersten Ansicht enthält alle essenziellen Komponenten für das Spielen von SQL-Textadventures, weshalb ihre Bedeutung durch eine erhöhte Breite hervorgehoben ist. Essenzielle Komponenten sind sowohl das Input-Feld als auch das Output-Feld, welche die grundlegende Steuerung eines Spiels realisieren. Die rechte Spalte der ersten GamePage-View ist für die Anzeige von Plugins vorgesehen, die den Spieler zwar unterstützen, gleichzeitig aber keinesfalls essenziell für die Spielbarkeit des Lernprogramms sein sollen. Hier wird Nutzern die Option geboten, über ein Dialogfenster (sog. Modal) mithilfe von Checkboxen aus allen verfügbaren Plugins diejenigen zu aktivieren, die zur Bearbeitung des aktuellen Spielabschnitts am dringendsten benötigt werden. Diese werden dem Nutzer anschließend in der Plugin-Spalte vertikal angeordnet präsentiert. Hier kann bei Überfüllung zum benötigten Plugin gescrollt werden.

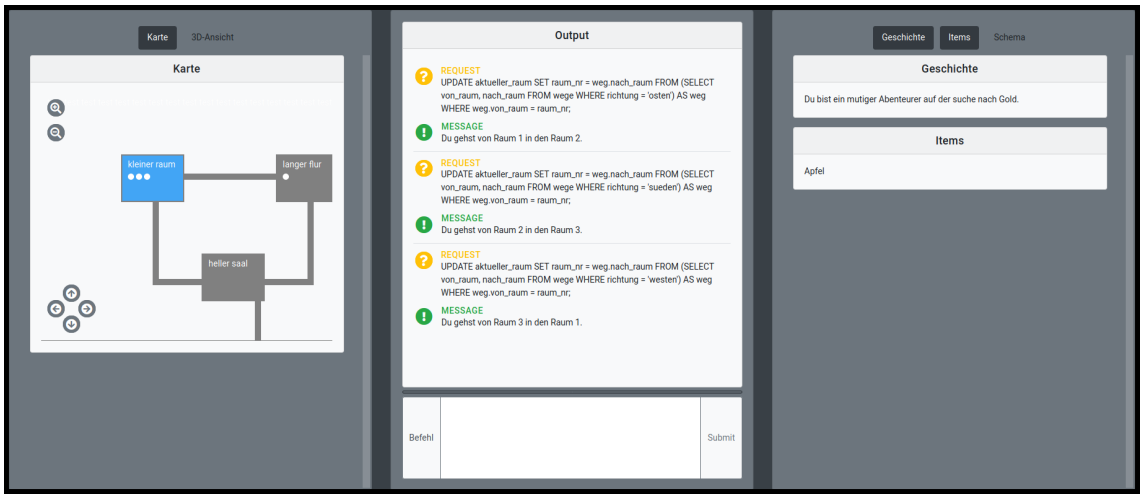

Abbildung 16: GamePage-View #2
Quelle: Eigene Darstellung

Einen etwas anderen Ansatz verfolgt die zweite GamePage-View, welche die gesamte Breite des Viewports einnimmt. Hier erfolgt eine Einteilung der sichtbaren Elemente in drei gleich große Spalten. Mithilfe dieses Konzepts können dem Spieler mehr relevante Objekte und Tools für die gleichzeitige Verwendung präsentiert werden. Die mittlere Spalte erfüllt hierbei den gleichen Zweck wie die linke Spalte der ersten Ansicht. Auch hier findet sich die Kombination von Input- und Output-Feld als Basiskomponente eines jeden Spiels. Linksseitig dieser Steuerelemente ist eine Spalte zur potenziellen Anzeige aller grafischen Plugins, wie beispielsweise der Karte aus Kapitel 2.5, positioniert. Auf der gegenüberliegenden Seite, rechtsseitig von den essenziellen Komponenten, werden wiederum alle textuellen Plugins präsentiert, wie in Abbildung 16 erkennbar ist. Hier soll der Spieler in Zukunft Schnellinformationen zu aktuellen Quests, seinem Inventar oder auch dem relationalen Schema seiner Spieler-Datenbank einsehen können. Eine Aktivierung sowie Deaktivierung von Plugins ist in dieser alternativen Form der GamePage über die Navigationsbalken am oberen Rand der Plugin-Spalten durchführbar. Hier sind in Abbildung 16 „Pill“-Buttons erkennbar, welche im ausgefüllten Zustand die aktuelle Verwendung eines Plugins anzeigen.

Durch die Entwicklung neuer, sowie den weiteren Ausbau bestehender Hilfsfunktionen innerhalb der GamePage kann die Benutzerfreundlichkeit des Lernprogramms weiter gesteigert werden. Ein Beispiel für dieses Potenzial ist bei den Einstellungsmöglichkeiten der Größenverhältnisse von Input und Output zu finden. Die ehemalige Version des Clients unterstützt keine Funktion zur Anpassung der Größe beider Abschnitte. Aufgrund dieser konstanten Größe ist es nicht möglich umfangreiche SQL-Anfragen im Input-Feld vollständig darzustellen. Der Spieler muss scrollen, um alle Abschnitte seiner Anfrage einsehen zu können, was die Übersichtlichkeit und damit eine effektive Arbeitsweise beeinträchtigen kann. Parallel kann ein Output-Fenster mit konstanter Größe die Darstellung zu weniger oder zu vieler Spielantworten bedeuten. Hierbei gilt es das Verhältnis zu finden, mit welchem genügend Antworten dargestellt werden und dennoch eine ausreichende

Übersichtlichkeit gewahrt wird. Da diese Faktoren einen sehr subjektiven Charakter besitzen und auch variieren, je nachdem ob der Spieler einen Eintrag in der Historie sucht oder nicht, ist die Entscheidung im Rahmen dieser Arbeit auf flexible Größeneinstellungen für Input und Output gefallen.

Diese Anpassbarkeit wird mithilfe eines „Resize-Bar“realisiert, welcher zwischen den beiden Abschnitten innerhalb der jeweiligen Spalte der GamePage platziert ist (siehe Abbildung 16 und 18). Ein Spieler kann diesen Balken mit dem Mauszeiger nach oben verschieben, um den Input-Abschnitt darunter in der Vertikale zu vergrößern und gleichzeitig den Output-Abschnitt darüber in der Vertikale um den gleichen Wert zu verkleinern. Umgekehrt verhält es sich in der entgegengesetzten Richtung, also nach unten. Hierbei ist erwähnenswert, dass die Höhe des Input- und des Output-Abschnitts jeweils minimal auf 10% und maximal auf 90% der Höhe des umschließenden Containers abgeändert werden kann. Der Balken zur Größeneinstellung wurde eigens für das Projekt mit HTML und CSS entwickelt. Eine Bootstrap-Lösung konnte nicht herangezogen werden, da Bootstrap keine Resize-Elemente oder -Elementeinstellungen unterstützt. Eine mögliche Alternative hätte die Verwendung des HTML-Property „resize“ dargestellt, welches eine vorimplementierte Option zur benutzerdefinierten Einstellung von Höhe oder auch Breite eines Textarea-Elementes bietet. Für die Zwecke dieser Arbeit wäre die Höhenverstellbarkeit relevant. Diese ist unterhalb des gewählten Textarea-Elements zwar unbeschränkt, oberhalb ist eine Höhenverstellung jedoch nicht vorgesehen. Da der Input-Abschnitt jedoch, wie die Konsole klassischer Textadventures, unterhalb des Outputs angeordnet wird und damit Größenanpassungen oberhalb des Inputs notwendig sind, kann die gewünschte Funktion nicht wie geplant mithilfe des HTML-Property „resize“ umgesetzt werden. Eine Eigenentwicklung stellt an dieser Stelle die naheliegendste Alternative dar.

Eine weitere Möglichkeit zur Umsetzung von Hilfsfunktionen und der Steigerung der Benutzerfreundlichkeit ist beim Ausgabeformat von Tabellen im Output-Bereich der GamePage zu finden. Durch die Ausführung von SELECT-Statements über die Konsole der GamePage ist es Spielern möglich sich Inhalte der Spieler-Datenbank im Tabellenformat ausgeben zu lassen. Diese Tabellen wurden bisher in ihrem vollen Zeilenumfang innerhalb des Output-Containers der GamePage angezeigt. Würde sich für den Spieler eine Ausgabetabelle mit einer besonders hohen Anzahl an Datenzeilen ergeben, sei es durch einen großen Datenbestand oder auch durch die exzessive Verwendung des „UNION ALL“-Operators, so könnte diese einen zu großen Teil des Output-Bereichs der GamePage in der Vertikale einnehmen. Zu groß bedeutet hierbei, dass der Spieler unverhältnismäßig lange im Output-Container nach oben scrollen muss, um zum oberen Ende der Tabelle zu gelangen. Dadurch wird beim Aufsuchen früherer Anfragen und Antworten der Spielfluss gestört und verlangsamt, was wiederum den Spielspaß und den Lerneffekt einschränken kann.

Die Lösung für dieses Problem stellt das Konzept der Pagination (engl. Seitennummerierung) dar. Pagination wird in dieser Arbeit als Bezeichnung für ein GUI-Element gebraucht, das zur Nummerierung und Selektion unterschiedlicher Seiten eines digitalen Dokuments verwendet werden

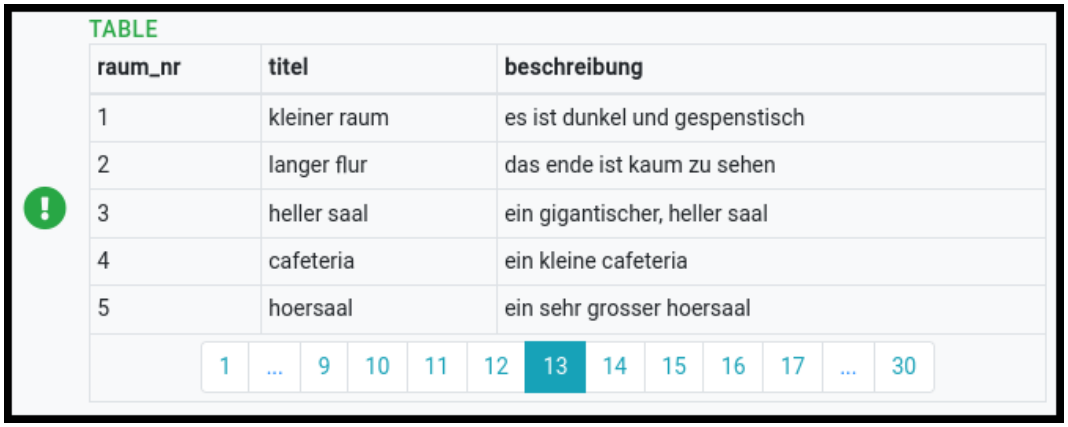

Abbildung 17: Große Beispieltabelle mit Bootstrap-Pagination
Quelle: Eigene Darstellung

kann. Im Falle der Tabellen innerhalb der GamePage werden mithilfe dieses Konzepts jeweils fünf Datenzeilen zu einer anwählbaren Seite zusammengefasst. Die Navigation zwischen diesen einzelnen Tabellenseiten erfolgt dann über das eigentliche Pagination-Element. Dieses stellt für jede Tabellenseite einen Button innerhalb einer Button-Group dar, welcher bei Betätigung die Ansicht der zugehörigen Tabellenseite hervorruft. Der Button der aktuell angewählten Seite wird hierbei immer mittig dargestellt. Vorgänger und Nachfolger werden entsprechend davor und danach bis zu einer maximalen Anzahl von neun aufeinander folgenden Buttons aufsteigend angeordnet. Existieren mehr als vier Vorgänger oder Nachfolger der aktuellen Seite, so wird dem Nutzer dies mithilfe von Auslassungspunkten und einem Abschließenden Button zum Öffnen der Ansicht der ersten oder auch letzten Seite vermittelt (siehe Abbildung 17).

Die grafischen Elemente der Pagination-Funktion konnte hierfür zwar von Bootstrap übernommen werden, der Algorithmus zum Wechseln der Tabellenseite und für den Umgang mit höheren Seitenanzahlen wurde jedoch im Verlauf dieser Arbeit eigenständig entwickelt. Grundsätzlich wird hierfür für jede Tabelle ein Integer-Wert innerhalb einer „tablePaginationList“ angelegt, welcher die Nummer der aktiven Seite der zugehörigen Tabelle angibt. Eine gesuchte Tabelle kann innerhalb dieser Tabellenliste gefunden werden, indem ein Abgleich der Elementreihenfolge mit der Output-Liste durchgeführt wird. Die Datenzeilen der aktuellen Tabelle werden anschließend aus dem entsprechenden „ConsoleEntry“ der gleichen Output-Liste gefiltert. Mithilfe der Nummer der aktuell darzustellenden Seite und allen Datenzeilen der Tabelle können nun Blöcke von jeweils fünf Datenzeilen gebildet und anschließend durch eine aufsteigend geordnete Button-Group mit den genannten Vorgaben dargestellt werden, wobei die aktive Seite als markierter Mittelpunkt der gesamten Pagination dienen kann. Wo vorher keine Maximalhöhe für Tabellen existierte, wird hiermit diese nun auf einen vertretbaren Wert eingegrenzt. Dieser entspricht der fünffachen Höhe einer Datenzeile der Tabelle, addiert mit der Höhe des Tabellenkopfes sowie mit der Höhe der Tabellenfußzeile, welche das Pagination-Element enthält.

Einer der eigentlichen Gründe für die Bereitstellung sämtlicher Outputs als listenförmige Timeline ist der potenzielle Wunsch von Spielern, bereits formulierte SQL-Anfragen für eine erneute Verwendung zu suchen, zu kopieren und in das Input-Textarea-Feld einzufügen. Hierbei könnte der Spieler entweder mit der Absicht handeln, die gleiche Anfrage erneut zu stellen oder aber die Anfrage auf neue Ziele anzupassen.

Dieser Prozess des Suchens, Kopierens und Einfügens stellt eine Aufwandsverkürzung für den Nutzer dar. Die Alternative, ohne die Verwendung der Timeline, ist die Neuformulierung jeder einzelnen Anfrage. Diese Methode zur Wiederverwendung von Anfragen durch den Nutzer wird im Rahmen dieser Arbeit weiter optimiert. Die Schritte des Kopierens und Einfügens von Anfragen aus der Timeline werden hierbei fokussiert betrachtet.

Hat der Spieler die gesuchte Anfrage innerhalb der Timeline gefunden, so muss diese mithilfe des Mauszeigers markiert werden. Anschließend kopiert der Nutzer den markierten String über das Rechtsklick-Menü des Mauszeigers oder über die Tastenkombination STRG-C. Zuletzt muss der kopierte String in das Input-Textarea-Feld der GamePage, wieder über das Rechtsklick-Menü des Mauszeigers oder über die Tastenkombination STRG-V, eingefügt werden. Alle diese Schritte zusammengefasst können den Nutzer je nach verwendeter Technik bis zu 10 Sekunden kosten. Dieser Wert wirkt auf den ersten Blick nicht sonderlich nachteilig für den Spieler. Betrachtet man dies jedoch über den gesamten Spielverlauf hinweg unter Berücksichtigung der vielfachen Durchführung solcher Aktionen, so kann sich die hierfür aufgebrachte Spielzeit des Nutzers erheblich summieren. Weiterhin ist zu beachten, dass kurze Störungen des Spielflusses den Nutzer in seinen Denkprozessen vom eigentlichen Ziel, der Formulierung einer passenden SQL-Anfrage, vorübergehend abbringen können. Um einen saubereren Spielfluss zu gewährleisten, wird der Prozess des Kopierens und Einfügens von Anfragen durch die Verwendung einer Hilfsfunktion innerhalb der GamePage für den Spieler verkürzt. Jedes Output-Objekt innerhalb der Output-Timeline wird zusammen mit einem zugehörigen kreisrunden Icon dargestellt. Diese Icons sollen dem Spieler anhand ihrer Farbe und ihres Aussehens einen schnellen Überblick über die Typen der unterschiedlichen Output-Objekte innerhalb der Timeline geben. Icons von Output-Objekten des Typs „request“ bieten zusätzlich im vorher erklärten Zusammenhang die Funktionalität von Buttons (siehe Abbildung 18).

Durch einen Klick auf ein solches Icon wird der vollständige Prozess des Kopierens und Einfügens der zugehörigen SQL-Anfrage vom Client übernommen. Anstatt innerhalb mehrerer Sekunden werden beide Schritte sofort durchgeführt. Somit wird die Zeit, welche ein Spieler für die Wiederverwendung einer Anfrage benötigt, drastisch reduziert. Dieser Ansatz ist jedoch wenig intuitiv, da die Icons, ohne Interaktion durch den Nutzer, grafisch nur geringfügig die Funktionalität eines Buttons präsentieren. Zusätzlich sind andere Icons in der Output-Timeline nicht anwählbar, wodurch der Nutzer vom Ausprobieren der Anwählbarkeit der „request“-Icons abgebracht werden könnte. Diese Funktion zur zeiteffizienteren Wiederverwendung von Anfragen muss also erst vom Nutzer

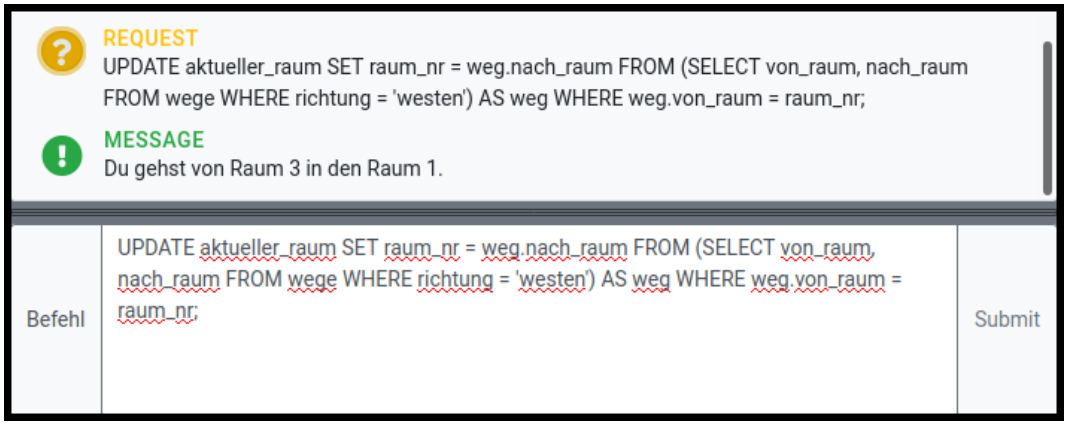

Abbildung 18: Betätigung eines „Anfrage Wiederverwenden“-Buttons
Quelle: Eigene Darstellung

entdeckt und verstanden werden. Im Verlauf eines Spiels ist dies mit hoher Wahrscheinlichkeit der Fall, da übliche Hover- und On-Click-Effekte von Bootstrap-Buttons für die „request“-Icons übernommen wurden. Sobald der Spieler seinen Mauszeiger über ein solches Icon bewegt sollte das optische Feedback ihn zur weiteren Entdeckung und somit zur Verwendung dieser Funktion bewegen.

Für eine bessere Einarbeitung von Nutzern in dieses und weitere Werkzeuge des Clients ist die Entwicklung eines „Walkthrough-Tutorials“ in zukünftigen Arbeiten des Projektes erwägenswert. Ein Beispiel für ein solches Tutorial bietet SQL-Island (siehe Kapitel 2.1). Dieses Konzept wird in dieser Arbeit jedoch nicht weiter vertieft. Zusätzlich zur Abkürzung des Kopierens und Einfügens von Anfragen könnte auch das Suchen von Anfragen in der Timeline vom System unterstützt werden. Hierfür wäre es möglich ein eigenes Suchfeld innerhalb des Clients für einen String-Vergleich, oder aber auch Filterfunktionen für die verschiedenen Objekttypen zu entwickeln. Die Verwendung des standardmäßigen Suchfeldes des Browsers funktioniert in der aktuellen Version jedoch bereits optimal. Der Fokus dieser Arbeit wird aus diesem Grund nicht weiter auf die Untersuchung dieses Ansatzes gelegt.

3.3 Farbgebung

Die Farbgebung ist im ehemaligen Client noch wenig durchdacht. Es wird schnell klar, dass diese Version des UI als erster funktionaler Prototyp und nicht als fertige Benutzerschnittstelle vorgesehen ist. Der gesamte Hintergrund des Body-Elements der Seite ist in einem hellen Cyan eingefärbt. Hierbei wurde ein einfacher RGB-Wert verwendet, welcher höchstwahrscheinlich ohne tiefere Überlegungen gewählt wurde. Im Vordergrund existiert neben dem weißen Input-Container keine weitere farbliche Anpassung. Alle Elemente werden auf dem cyanfarbenen Hintergrund präsentiert, manche kontrastreicher als andere (siehe Abbildung 12).

An dieser Stelle bietet eine Überarbeitung der Farbgebung ein hohes Potenzial zur Schaffung

eines UI, welches mithilfe von Tiefe und Kontrasten den Blick des Nutzers auf das Wesentliche lenkt. Generell ist es sinnvoll, das neue Farbschema auf einige wenige konkrete Farbtöne einzugrenzen. Tidwell [13, S. 294] beschreibt hierzu, dass Farben um die Aufmerksamkeit des Nutzers konkurrieren. Zu viele Farben könnten dem Client einen grellen Regenbogencharakter verleihen. Weniger sei an dieser Stelle deshalb häufig mehr. Andererseits sei eine gewisse Farbausprägung unentbehrlich für den Charakter einer modernen Web-Anwendung und kleinere Elemente innerhalb des UI, Beispielsweise Icons, dürfen laut [13, S. 295] aus diesem Muster auch in geringem Ausmaß ausbrechen. Hier sei es laut [13, S. 295] wichtig, das richtige Maß für die entsprechende Anwendung zu finden.

Für das Projekt wurden größtenteils die von Bootstrap vorgefertigten Farben verwendet. Auch hier wird wie am Anfang von Kapitel 3.1 davon ausgegangen, dass für die Auswahl dieser Farben wesentlich tiefere Überlegungen zum Design vorgenommen wurden, als sie im Rahmen des Projekts möglich gewesen wären. Die Wahl des Hauptfarbschemas fiel für den neuen Client auf Grautöne im Sinne eines modernen Dark-Mode-Designs. Hierfür werden die Bootstrap-Farben „secondary“ und „dark“ verwendet. Kombiniert werden die Grautöne sowohl mit dem Bootstrap-Farbschema „light“, welches nahezu weiß mit einem leichten Grauton ist, und mit dem Bootstrap-Farbschema „info“, welches das ursprüngliche Cyan in einer abgedunkelten Variante zurückbringt.

Neben der Farbtonauswahl zeigt sich als weiterer wichtiger Punkt, welchen es beim Farbdesign zu beachten gilt, die Anwendung der Farben in unterschiedlichen Helligkeitsstufen. Wie in [13, S. 291] beschrieben können dunklere Ausprägungen beim Betrachter einen pseudo-dreidimensionalen Hintergrundeffekt verursachen, wohingegen hellere Farben ein Element auf umgekehrte Weise hervortreten lassen. Dieser Effekt wird in den GamePages des neuen Clients genutzt, um sowohl die Spalten als auch die Card-Container der Plugins nach einem hierarchischen Konzept, ähnlich dem Document Object Model, hervorzuheben.

Als allgemeiner Hintergrund der Anwendung wird ein dunkles Grau („dark“) verwendet, um keine Aufmerksamkeit auf sich zu ziehen und die Augen des Nutzers möglichst wenig zu reizen. Vor diesem dunklen Hintergrund werden die Spalten der jeweiligen GamePage (siehe Kapitel 3.2.3) mithilfe eines etwas helleren Grautons („secondary“) hervorgehoben. Schlussendlich werden die wichtigsten Funktionalitäten innerhalb der Plugin-Cards und auch der Input- und Output-Container durch ein nahezu-Weiß („light“) für den Spieler kontrastreich gekennzeichnet (siehe Abbildung 16). Innerhalb dieses Grundaufbaus finden sich zusätzlich an unterschiedlichen Stellen einige wenige Cyan-Nuancen („info“), welche dem Client ein anspruchsvolleres Farbschema verleihen sollen.

Ein Beispiel hierfür ist der Header der Login-Card (siehe Abbildung 15). Die beschriebene Farbgebung wird auf allen Seiten des Clients im Rahmen dieser Arbeit auf äquivalente Weise umgesetzt.

3.4 Plugin-Konzept

Das SQL-Lernprogramm der Datenbank-Gruppe der MLU wird einer ständigen Entwicklung ausgesetzt sein. Zum aktuellen Zeitpunkt arbeiten bis zu fünf Personen an verschiedenen Teilen des Projektes. Hierbei entstehen unter anderem neue Module für den Client, wie zum Beispiel eine dreidimensionale grafische Ansicht der besuchten Räume. Auch nach Veröffentlichung der finalen Version durch die Datenbank-Gruppe sollen Entwickler angeregt werden, das Lernspiel mit eigenen Ideen auf ihre Bedürfnisse anzupassen. Um die zukünftige Arbeit anderer Entwickler zu unterstützen, ist das Entwicklungsziel der Modularität bei dem Web-Client weitestgehend verfolgt worden. Dabei soll ein übersichtlicher sowie struktureller Aufbau gewährleistet werden, welcher Raum zur Erweiterung bietet.

In diesem Zusammenhang wurde an einem Konzept gearbeitet, welches den Integrierungsprozess von zusätzlichen, nicht-spielnotwendigen Werkzeugen (sogenannte „Plugins“) in den Client vereinfacht und vereinheitlicht. Im Rahmen dieser Arbeit wurde hierbei das zugrundeliegende Konzept untersucht und implementiert. Konkrete Plugins werden hingegen abseits dieser Arbeit im weiteren Projektverlauf der Datenbank-Gruppe entwickelt.

Ein Beispiel für die Funktionalität eines solchen Plugins ist die interaktive Karte (siehe Kapitel 2.5), welche bereits vor Durchführung dieser Arbeit in die GamePage integriert wurde. Ein Fortschritt, welchen die Plugin-Struktur des neuen Clients jedoch im Vergleich zur ursprünglichen unstrukturierten Implementierung von Hilfswerkzeugen erzielt, ist die sowohl Server gesteuerte als auch Client gesteuerte Aktivierung und Deaktivierung sämtlicher Plugins. „Server gesteuerte Aktivierung“ bedeutet hierbei, dass die Menge an verfügbaren Hilfswerkzeugen für den Nutzer serverseitig bestimmt wird. Mithilfe dieser Einstellungsmöglichkeiten ist es Administratoren des Spiels möglich, den Schwierigkeitsgrad für alle Spieler indirekt durch Freigabe zusätzlicher Hilfen zu verringern oder aber durch weitere Einschränkungen zu erhöhen.

Zur Realisierung einer solchen Plugin-Konfiguration ist es notwendig, dass der zuständige Administrator eine eigene Elm-Main-Datei mit einer angepassten Liste von „pluginViews“ erstellt. Der kompilierte JavaScript-Code muss anschließend wie üblich in eine HTML-Datei eingebettet und über den Server aufgerufen werden. Die Client gesteuerte Aktivierung von Plugins bedeutet, dass der Nutzer aus der Menge der serverseitig vorgegebenen Erweiterungen diejenigen zur Verwendung auswählen kann, welche in der aktuellen Spielsituation benötigt werden. Ein einzelner Spieler kann also alle serverseitig aktivierten Plugins zur Lösung von Problemen im Spiel verwenden. Um selbst einen Überblick über die Werkzeuge zu bewahren, wird Spielern jederzeit ermöglicht, in ihrer lokalen Client-Anwendung die relevantesten Plugins anzeigen und unwichtige Plugins ausblenden zu lassen. Dies wird in der aktuellen Projektversion über Navigationsmenüs in der GamePage ermöglicht.

In zukünftigen Versionen ist eine Einbettung und Verwendung der Plugin-Struktur auf sämtlichen Seiten der Client-Anwendung denkbar. Beispielsweise könnte es ermöglicht werden, statisti-

sche Informationen über das aktuell bearbeitete Spiel im Editor aus- und einzublenden.

3.5 Weitere Forschungsmöglichkeiten

Bei der Überarbeitung der Client-Anwendung haben sich Ansätze für die weitere Forschung ergeben, welche jedoch den Rahmen dieser Arbeit sprengen würden. Im folgenden Kapitel werden diese Ideen vorgestellt, um eine Grundlage für zukünftige Arbeiten in dieser Richtung zu aufzuzeigen.

In Kapitel 3.2.3 wurde näher auf die Umsetzung der GamePage eingegangen. Unter anderem wurden dort zwei alternative Seiten-Layouts erklärt, welche beide Anwendung im neuen Client finden. Ein Nutzer kann also in der neuen Version selbst auswählen, welches der beiden Layouts ihm persönlich ein besseres Spielgefühl gibt. Eine Version des Clients, welche zur Veröffentlichung geeignet wäre, würde womöglich nur ein einziges GamePage-Layout implementieren, da mehrere Seiten mit gleicher Funktion einer übersichtlichen und intuitiven Anwendung entgegenwirken. Um beim Design der entgültigen GamePage ein bestmögliches Ergebnis zu erzielen, könnte mithilfe der aktuellen GamePage-Alternativen eine Studie durchgeführt werden. Gegenstand der Studie könnte die Frage sein, welche GamePage von der überwiegenden Anzahl an Probanden beim Spielen bevorzugt wird, welche GamePage einen schnelleren Einstieg in die Funktionsweise des Lernprogramms bietet oder aber auch mithilfe welcher GamePage ein konkretes Testspiel am schnellsten durchgespielt wird. Es sollte vor allem versucht werden abzuwägen, welches Layout (zwei- oder dreispaltig) und welcher Plugin-Zugriff (Modal oder Nav-Buttons) die Zwecke der GamePage am ehesten unterstützen. Als Probanden für die Studie könnten Teilnehmer der Datenbankvorlesung der Universität infrage kommen. Hierbei wäre es je nach Ziel der Studie sinnvoll, wenn die Testpersonen SQL bereits in ihren Grundzügen beherrschen würden, um einen realitätsnahen Spielfluss beobachten zu können.

Die grafische Karte, welche bereits in der alten Client-Version (siehe Kaptitel 12) vorzufinden ist, wurde im Rahmen dieser Arbeit mithilfe von Bootstrap-Buttons und Icons der Font-Awesome-Bibliothek optisch anspruchsvoller gestaltet. Sie bietet dem Nutzer dennoch lediglich Funktionalitäten für Zoom und Charakterbewegung. Für größere Spielkarten wären jedoch erweiterte Navigationsoptionen von Vorteil. Am meisten fehlt die Möglichkeit, den Viewport auf der Karte entlang der X- und der Y-Achse zu bewegen. Hierfür könnte sich eine Navigation mit der Maus eignen, welche durch ein „Ziehen“ der Karte durchgeführt wird. Diese Drag-Funktionalität wurde tatsächlich bereits vom ursprünglichen Entwickler der Web-Anwendung im Code bedacht, die Implementierung ist jedoch noch unausgereift.

Ähnlich wie die Karte ist auch eine Schemaanzeige bereits vor der Durchführung dieser Arbeit in den Client eingebaut worden. Diese Anzeige wird als relationales Modell der Spieler-Datenbank angezeigt und soll dem Spieler einen schnellen Überblick über alle ihm zur Verfügung stehenden Tabellen und Spalten geben. Bislang ist diese Anzeige jedoch statisch, weshalb sie zu jedem Zeitpunkt lediglich das relationale Modell der Spielerdatenbank des Testspiel „german“ anzeigen kann. Für

eine Client-Version, welche bereit für die Erstveröffentlichung ist, sollte hier definitiv die Anzeige des aktuellen Spielschemas ermöglicht werden, anstatt einen vorgefertigten String abzubilden.

Wie bereits in Kaptitel 2.4 angesprochen bietet der Elm-Code des alten Clients bereits erste Funktionen für den Umgang mit URLs, welche jedoch noch nicht den vollen Anforderungen einer standardmäßigen Web-Anwendung gerecht werden. Hier besteht weiterer Handlungsbedarf. Der gleiche URL-Aufruf sollte immer die gleiche Seite bei korrektem Login hervorrufen. Dies muss auch der Fall sein, wenn die URL manuell eingegeben wurde. Weiterhin sollte ein einheitlicher URL-Aufbau für die Web-Anwendung eingeführt werden. So ein einheitlicher URL-Aufbau könnte Nutzern das manuelle Formulieren einer Webadresse vereinfachen. Eine Möglichkeit hierfür wäre Beispielsweise die Form „$\backslash HostName \backslash PageName$“.

Mit Abschluss des praktischen Teils dieser Arbeit ist leider immer noch keine nutzbare Logout-Funktion in den Client implementiert worden. Für die Umsetzung dieser Funktion wird nicht nur eine clientseitige Implementierung innerhalb des Elm-Codes benötigt, es müssen auch tiefere Eingriffe in den bestehenden serverseitigen Java-Servlets vorgenommen werden. Die Durchführung dieser Eingriffe war innerhalb des zeitlichen Rahmens dieser Arbeit nicht möglich. Es ist jedoch klar, dass der Logout einen essenziellen Bestandteil des fertigen Clients bilden wird. Weitere Implementierungsversuche in dieser Richtung sind also unabdingbar. Ein konkreter Logout-Button könnte in der Client-Anwendung nach erfolgreichem Login den deaktivierten Login-Button im Navbar ersetzen. Hierfür müsste man lediglich eine weitere Ausnahme in der Funktion „tabTitles“ für den eingeloggten Zustand des Clients hinzugefügen. Mithilfe dieses Buttons sollte die Logout-Information an das entsprechende Servlet weitergeleitet und von dort in der User-Datenbank aktualisiert werden. Gleichzeitig müsste der Nutzer clientseitig auf die LoginPage weitergeleitet werden.

4. Entwicklung des Editors

Das Tool, welches im nachfolgenden auch als „Editor“ bezeichnet wird, soll Autoren als eine Entwicklungs-Pipeline zur Erstellung und Bearbeitung von Spielen für das Lernprogramm der Datenbank-Gruppe dienen. In diesem Kapitel werden die Modellierung und Implementierung von Front- und Back-End des Editors diskutiert. Einleitend soll ein Grundverständnis für die angestrebte Funktionalität des Editors geschaffen werden. Hierfür werden die wichtigsten Entwicklungsziele in 19 informell und allgemein aus der Sicht eines Endbenutzers verfasst. Diese Zielformulierungen, welche sich an dem Konzept von „User-Stories“ orientieren, sollen einen benutzerorientierten Rahmen zur Orientierung in den nachfolgenden Kapiteln schaffen.

Da das Lernprogramm der Datenbank-Gruppe auf einer relationalen Datenbank aufbaut, muss auch der Editor grundsätzlich auf die Bearbeitung relationaler Daten abgestimmt werden. Ein weiterer wichtiger Punkt bei der Entwicklung des Editors ist die Orientierung an den Features der Vergleichsprodukte (siehe Kapitel 2.2 und 2.3). Zum einen sollten hier vorteilhafte Funktionen wie die hybride Verwaltung fertiggestellter Spiele durch Quest (online und lokal) berücksichtigt werden, zum anderen ist es sinnvoll die Schwächen dieser Vergleichsprodukte möglichst zu vermeiden. Ein Beispiel für eine solche Schwäche ist die unzureichende Kontrolle der editierten Daten in Quest, die zu Laufzeitfehlern beim Spielen führen kann. Im Folgenden wird die Notwendigkeit eines Editors für das Projekt näher erläutert.

Die Erstellung und Bearbeitung von Spielen für das SQL-Lernprogramm wäre mithilfe anderer Anwendungen durchaus auch ohne einen Editor durchführbar. Solche Anwendungen können Standardwerkzeuge zur Administration von Datenbanken wie zum Beispiel Adminer [20], das pgAdmin-GUI Tool [31], oder auch das psql Command-Line Tool [32] sein. Jede dieser Anwendungen ist jedoch auf eine vollständige Datenbankverwaltung ausgelegt und bietet viele, für die Bedürfnisse des Nutzers redundante Funktionalitäten. Ein größerer Funktionsumfang und viele Freiheiten gehen immer auch mit einer höheren Einarbeitungszeit und einem breiteren Spektrum potenzieller Fehler einher. Ein Nutzer, welcher mit dem Tool seiner Wahl die gesamte Datenbank manipulieren kann, muss vor der Erstellung von Spielen erst einmal den Aufbau und die Funktionsweise des gesamten Lernprogramms verstehen können. Eine zugeschnittene Softwarelösung, welche dem Nutzer eine ausreichende Menge intuitiver Tools anbietet, nimmt ihm diesen Aufwand ab und lässt ihn sich auf seine Aufgabe als Autor konzentrieren. Weiterhin ist es mithilfe eines Editors nicht mehr nötig, die Datenbanksprache SQL zur Erstellung und Bearbeitung von Spielen einwandfrei zu beherrschen. Ein solches Tool ließe den Nutzer seine Aufgaben in einer intuitiven grafischen Benutzeroberfläche erledigen, wodurch CREATE-, INSERT- und UPDATE-Statements überflüssig für die Spielerstellung werden. Nichtsdestotrotz ist zu erwarten, dass Autoren des SQL-Lernprogramms ohnehin ausreichende Kenntnisse in SQL mitbringen werden.

1. Als neuer Benutzer des Editors möchte ich ein Spielschema aus der Datenbank laden, um die Erstellung eines neuen Spiels zu initialisieren.
2. Als Autor eines SQL-Textadventures möchte ich meine lokale Spieldatei in den Editor laden, um diese zu überarbeiten.
3. Als Autor eines SQL-Textadventures möchte ich umfangreiche Hilfswerkzeuge verwenden, um mein Spiel effizienter zu editieren.
4. Als Autor eines SQL-Textadventures möchte ich auf eigene Fehler aufmerksam gemacht werden, um ein funktionierendes Spiel zu gewährleisten.
5. Als Autor eines SQL-Textadventures möchte ich mein Spiel lokal speichern, um es später weiter zu bearbeiten oder mit anderen zu teilen.
6. Als Autor eines fertiggestellten SQL-Textadventures möchte ich dieses in die Spiel-Datenbank laden, um das zukünftige Spielen in der GamePage vorzubereiten.

Abbildung 19: Angestrebte Funktionen formuliert als „User-Stories“
Quelle: Eigene Darstellung

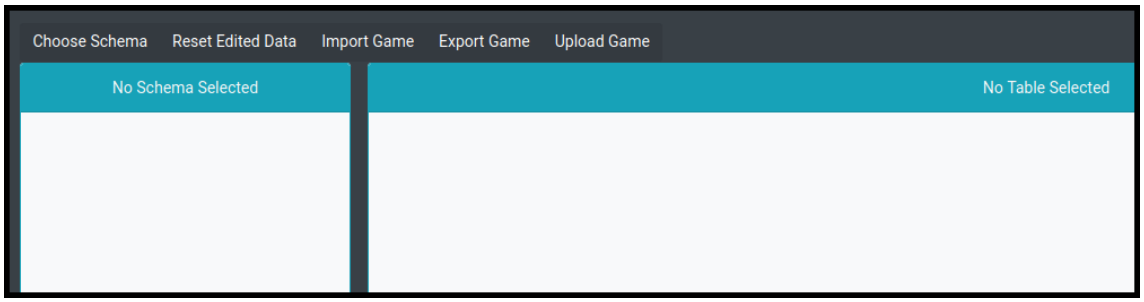

Abbildung 20: Ausschnitt vom Editor nach dem ersten Öffnen
Quelle: Eigene Darstellung

4.1 Aufbau und Bedienung

Der Editor bezeichnet ein Werkzeug innerhalb des Clients des Lernprogramms. Dieser integrierte Ansatz nimmt Autoren zusätzlichen Installationsaufwand ab und ermöglicht eine effiziente Verknüpfung von Nutzerprofilen und erstellten Spielen auf Ebene des Clients. Weiterhin könnte es durch die Nähe zum restlichen Client-Programm zukünftig möglich gemacht werden, editierte Spiele direkt nach Verwendung des Editors im Spiele-Reiter zu testen. Diese Funktion konnte in dieser Arbeit jedoch aufgrund des Entwicklungsstandes des Lernprogramms nicht weiter verfolgt werden. Die Möglichkeit, das Tool wie beim vergleichbaren Editor „Quest“ [9] zusätzlich als herunterladbares Programm anzubieten, stellt eine Chance zur Verbesserung der Benutzerfreundlichkeit dar und sollte in der Zukunft durchaus weiter verfolgt werden. Ein Offline-Werkzeug bietet Nutzern die Möglichkeit, auch an Orten ohne Internetverbindung Spiele zu editieren. Solche Verbesserungen der Benutzerfreundlichkeit können zur Popularität einer Anwendung beitragen und helfen, eine breitere Masse anzusprechen.

Das Editor-Tool innerhalb der Web-Applikation kann über den Hauptnavigationsbalken in der

oberen linken Ecke erreicht werden (siehe Abbildung 21). Auf der Seite des Editors wird dem Nutzer initial eine Auswahl von Funktionen in Form einer Button-Group sowie zwei leere Übersichten mit den Meldungen „No Schema Selected“ und „No Table Selected“ präsentiert (siehe Abbildung 20). Der erste Button ermöglicht es dem Nutzer, mithilfe eines Modals ein Schema für die Erstellung eines neuen Spiels aus der Datenbank in den Editor zu laden. Diese Schemata beinhalten in der Regel Tabellen mit Tabellennamen, Spaltennamen, Datentypen, Primärschlüsseln und Fremdschlüsseln.

Hat der Nutzer ein Schema ausgewählt, so werden die Tabellennamen in der linken Übersicht über ein Navigationsmenü angezeigt. Dem Nutzer ist es nun möglich, schnell zwischen den Tabellen des gewählten Schemas über die Navigation auf der linken Seite zu wechseln (siehe Abbildung 21). Wählt der Nutzer die nächste zu editierende Tabelle an, so werden ihr Name, ihre Spaltenköpfe sowie ihre Datenzeilen in der rechten Übersicht angezeigt (siehe Abbildung 21). Die eigentliche Funktionsweise des Ladens eines Schemas wird innerhalb des Systems über die Kommunikation von Elm-Client, Java-Servlet und Datenbank gelöst. Nach Betätigung des „Choose Schema“-Buttons wird zuallererst die Liste der Namen aller wählbaren Schemata vom Client angefragt. Diese wird an ihn weitergeleitet, nachdem das Servlet die entsprechenden Namen aus der Datenbank ausgelesen und mithilfe einer Blacklist gefiltert hat (siehe Kapitel 4.4).

Hat der Nutzer ein Schema zur Bearbeitung ausgewählt, so wird dieses ebenfalls vom Client angefragt, vom Servlet ausgelesen und anschließend im JSON-Format nach den in Kapitel 4.4 festgelegten Standards an den Client gesendet. Innerhalb des Clients lässt sich dieses Datenpaket im JSON-Format nun mithilfe der Elm-Bibliothek „Json-Decode“ in ein Format konvertieren, mit dem in Elm gearbeitet werden kann. Hierfür wird der innerhalb dieser Arbeit angelegte Datentyp „TableSchema“ verwendet, welcher sich nach den Vorgaben des JSON-Input-Schemas aus Kapitel 4.4 richtet. Die empfangenen Schemadaten werden an dieser Stelle verwendet, um den Editor für die kommende Arbeit zu initialisieren.

Zu Beginn besitzen Tabellen von neu geladenen Schemata keine Daten. Hier kann der Nutzer mithilfe des länglichen „+“-Buttons unterhalb der Spaltenköpfe neue, leere Datenzeilen zur Tabelle hinzufügen. Diese Funktion orientiert sich an dem Design-Pattern „new-item-row“ welches in [13, S. 193] näher beschrieben wird. Ein solches Element zum Hinzufügen von Daten veringere laut [13, S. 193] durch die lokal Positionierung an den neuen Datenzeilen den Navigationsaufwand für den Nutzer. Jede so hinzugefügte Datenzeile besteht aus einem Input-Feld pro Spaltenkopf sowie einem Button zum Löschen der Datenzeile und einem Button zum Wechseln in eine Zeilenansicht. Der Autor hat nun die Möglichkeit die Tabelle über die Input-Felder nach seinen Wünschen mit Spieldaten zu versehen. Für längere Einträge stellt die Kürze der Input-Felder jedoch ein Problem dar, weil jene Input-Felder kaum die Übersicht über einen gesamten Text gewähren können. Hierfür kann der Nutzer mithilfe der „Column“-Buttons (siehe Abbildung 22) in eine Property-Value-Ansicht einer Datenzeile wechseln (siehe Abbildung 23). In dieser Zeilenansicht werden die einzelnen Datenzellen der gewählten Datenzeile als Spaltenname-Wert-Paare dargestellt. Für jede Spalte

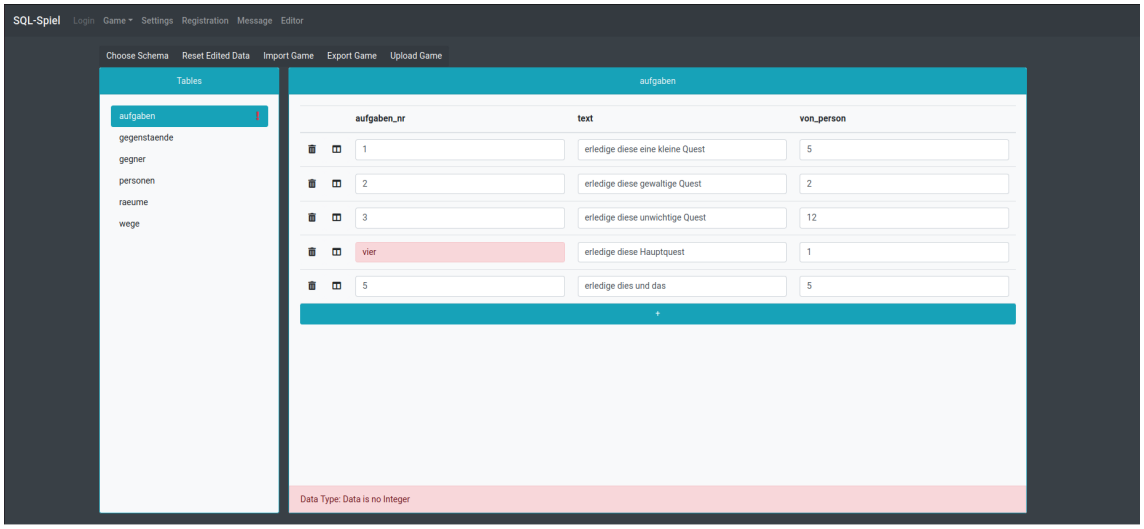

Abbildung 21: Beispielzustand des Editors mit Fehleranzeige
Quelle: Eigene Darstellung

Abbildung 22: Ausschnitt der Tabellenansicht des Editors
Quelle: Eigene Darstellung

kann ein vollständiges Textarea-Feld mit einer Höhe von vier Textzeilen und einer Scroll-Funktion bei längeren Texten ausgefüllt werden. Die Inhalte, welche in den Input-Feldern der Tabellenansicht erstellt wurden, werden hierbei in die zugehörigen Textarea-Felder der Zeilenansicht übernommen und umgekehrt. Ein Autor kann hier weitere Informationen zu den Spaltenschlüsseln über die „Key“-Buttons einsehen. In diesen wird angezeigt, ob die Spalte Teil des Primärschlüssels der Tabelle ist. Zusätzlich werden alle Fremdschlüssel der Spalte aufgelistet und die entsprechenden Tabellen der Fremdschlüssel verlinkt. Über diese Links gelangt der Nutzer zur Tabellenansicht der jeweiligen Fremdschlüsseltabelle. Zurück zu der Ansicht der aktuellen Tabelle gelangt der Nutzer über den Button mit dem linksgerichteten Pfeil-Icon in der oberen linken Ecke der Zeilenansicht (siehe Abbildung 23).

Eine weitere Funktion der Hauptnavigation besteht darin, sämtliche Datenzeilen des aktuell bearbeiteten Spiels zu löschen. Dieser Vorgang wird durch die Verwendung des Buttons „Reset

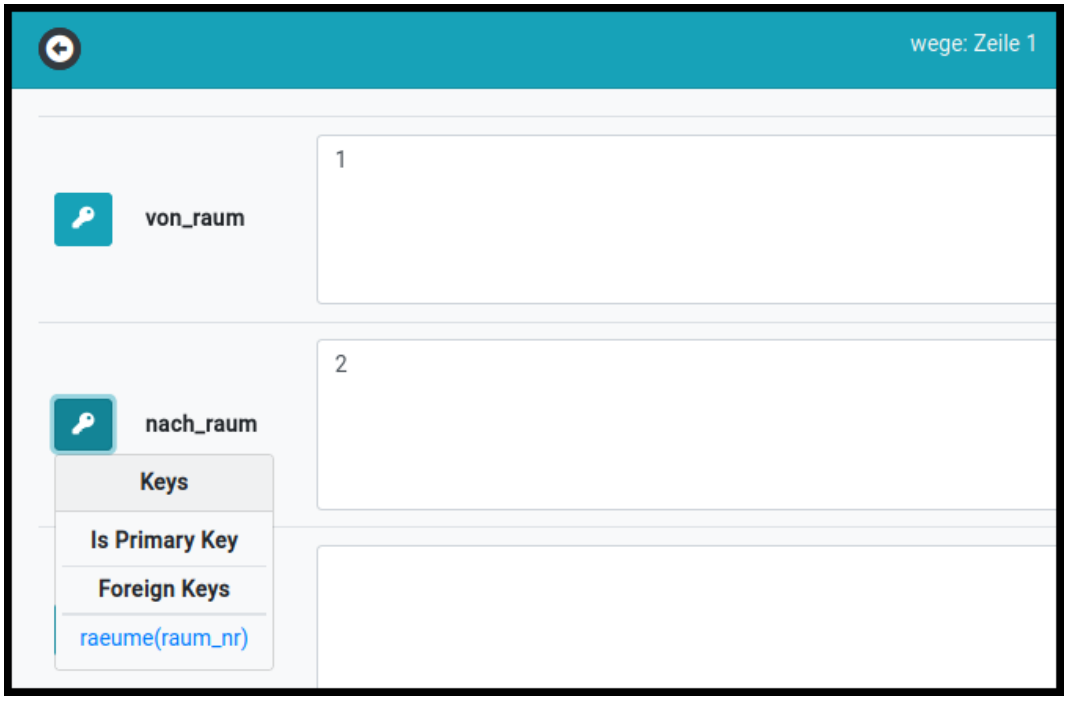

Abbildung 23: Ausschnitt einer Zeilenansicht des Editors
Quelle: Eigene Darstellung

Edited Data“ durchgeführt (siehe Abbildung 20). Hierbei wird dem Nutzer das Löschen jeder einzelnen Datenzeile erspart ohne das aktuelle Schema neu laden zu müssen.

Weiterhin können Autoren mithilfe des „Export Game“-Buttons das aktuelle Spiel vom Editor in ihren lokalen Speicher herunterladen. Hierfür wird mit Bestätigung des Downloads zuerst das aktuelle Datum, unter Berücksichtigung der Zeitzone des Nutzers, mithilfe von Tasks ermittelt. Dies geschieht Elm-intern in der Update-Funktion des Editors. Dieses Datum wird anschließend zusammen mit den gesammelten Daten des Editor-Models (Autoren, Schemaname, Tabelleninhalte, etc.) an eine Funktion weitergereicht, welche diese in dem geeigneten Datentyp „GameFile“ für den Download zusammenführt. Der genannte Datentyp bildet hierbei das Elm-Äquivalent zum Output-JSON-Schema, welches in Kapitel 4.5 näher erläutert wird. Zuletzt können diese Daten im GameFile-Format mithilfe der Elm-eigenen Bibliothek „Json.Encode“ in das JSON-Format überführt und unter Einhaltung des Dateistandards aus Kapitel 4.5 heruntergeladen werden.

Konzeptuell gesehen bildet das SQL-Spiel als Datei den Grundstein zur Verbreitung und Weiterentwicklung eigener Werke. Eine Plattform, welche das Hochladen und Teilen von Werken ermöglicht, wäre hierfür denkbar. Spieler auf der ganzen Welt könnten SQL-Spiele herunterladen, ausprobieren, anpassen und weiter teilen. Eine solche Website würde einen großen Schritt hin zum Aufbau einer Community und damit zur weiteren Verbreitung des Lernprogramms bedeuten.

Dass dieses Konzept bei Spielen des gleichen Genres bereits erfolgreich angewandt werden konnte, ist am Beispiel der Webseite textadventures.co.uk [9] unter dem Reiter „Play“ zu erkennen. Hier können Textadventure-Spiele, welche mithilfe der angebotenen Tools erstellt wurden, im Browser

gespielt und mit anderen Nutzern geteilt werden. Insgesamt wurden hier bereits 5435 verschiedene Spiele von Autoren auf der ganzen Welt veröffentlicht (Stand 09.02.2021). Ein Beispiel für den Erfolg, welchen ein solcher Editor in Verbindung mit einer Sharing-Plattform bringen kann, stellt das bereits im Kapitel 1. erwähnte Echtzeit-Strategiespiel „Warcraft 3“ dar. Das Spiel wurde im Jahre 2002 erstveröffentlicht und erhielt im Jahre 2004 eine Online-Plattform namens „Epic War“ [33] zur Archivierung und Verbreitung selbst erstellter Szenarien. Bis heute wurden dort insgesamt über 288000 Beiträge veröffentlicht (Stand 09.02.2021). Ein Beitrag besteht hierbei aus einer Karte, welche mithilfe des Hauptspiels geladen und anschließend gespielt werden kann. Eine Karte von Warcraft 3 ist also in der Funktionsweise vergleichbar mit einem Spiel des SQL-Lernprogramms. Diese Masse an Beiträgen umfasst nicht ausschließlich neue, einzigartige Karten, sondern auch neue Versionen von bereits existierenden Karten, welche durch andere Autoren überarbeitet und angepasst wurden. Hierdurch ergibt sich für das Spiel eine ganz eigene Dynamik, welche bis heute eine aktive Community an sich bindet. Allein am 08.02.2021 wurden wieder 18 neue Beiträge veröffentlicht. Dieser Trend von täglichen Uploads hält sich auf der Webseite bis heute konstant, was für ein 19 Jahre altes Spiel eine außergewöhnlich hohe Aktivität darstellt. Ein Editor in Verbindung mit einer Online-Datenbank zum Teilen von Spielen kann also durchaus ein Erfolgskonzept bei der Entwicklung eines Spiels darstellen.

Eine weitere Funktion des Editors ist es, die Daten von Spieldateien, welche auf der Festplatte eines Autors vorliegen, durch Betätigung des Buttons „Import Game“ in den Editor zu laden. Dies geschieht mithilfe des Standard-Dateiauswahlfensters des jeweiligen Betriebssystems. Ein Spielimport läuft hierbei in zwei Schritten ab. Zuerst wird der Name des Schemas des gewählten Spiels gelesen, das entsprechende Schema aus der Datenbank geladen und dieses im Editor initialisiert. Anschließend wird die importierte JSON Datei, in umgekehrter Weise im Vergleich zum Export, in eine GameFile umgewandelt, aus welcher alle nötigen Informationen zur Initialisierung des Editor-Models gewonnen werden können. Vor allem die Datenzeilen werden also sicher aus der Spieldatei in den Editor übertragen. Nach dem Import eines Spiels kann der Nutzer dieses im Editor mit vollem Funktionsumfang bearbeiten.

Zuletzt besitzt ein Autor die Möglichkeit, ein fertiges Spiel, welches sich im Editor befindet, mit einem Klick auf den Button „Upload Game“ als eigenes Schema in die Spiel-Datenbank zu laden. An diesem Punkt ist das erstellte Spiel beinahe spielbereit. Die Generierung eines Schemas in der Spieler-Datenbank zur Initialisierung des Spieler-Charakters steht jedoch noch aus. Die Generierung eines solchen Spieler-Schemas ließe sich beispielsweise durch ein Programm lösen. Diese Aufgabe steht jedoch immer in Abhängigkeit zum jeweiligen Spiel und kann aus diesem Grund nur schwer automatisiert werden. Ein weiterer Editor zur Erstellung von Spieler-Schemata aus Spieldateien durch einen menschlichen Nutzer wäre denkbar und könnte in den aktuellen Editor eingebaut werden. Diesen Ansatz zu untersuchen, würde jedoch den Rahmen der vorliegenden Arbeit überschreiten.

4.2 Fehlervermeidung und Sicherheit

Programme, welche auf Nutzerinteraktion ausgelegt sind, zeigen eine grundsätzliche Anfälligkeit für fehlerhafte oder auch missbräuchlichen Verwendung. Den Nutzer vor Fehlern durch das Programm zu bewahren, bedeutet, die Freiheit des Nutzers innerhalb des Programms auf alle technisch durchführbaren Aktionen einzuschränken. Hinsichtlich des hier zu besprechenden Editors muss dementsprechend Programmabbrüchen durch Java- und SQL-Fehler vorgebeugt, sowie Vor- und Nachbedingungen kritischer Funktionen des Elm-Programms eingehalten werden. Beispielsweise darf der Nutzer kein Spiel in die Datenbank hochladen können, solange im Editor kein Schema ausgewählt ist. Diese Aktion würde einen SQL-Fehler im nachfolgenden „Create Schema“ Statement und damit eine Exception im Java-Servlet auslösen.

Im Kontrast hierzu bedeutet der Schutz des Systems vor missbräuchlichen Handlungen durch den Nutzer, diesem lediglich Zugriff zu für ihn vorgesehenen Funktionen und Informationen zu gewähren. Einem regulären Nutzer des Editors sollte es beispielsweise nicht möglich sein, die Werke anderer Autoren aus der Datenbank zu löschen oder Passwörter anderer Profile einzusehen. Sowohl die Fehlerfreiheit als auch die Sicherheit können oft nur annäherungsweise vollständig gewährleistet werden und benötigen auch für diesen Zustand meist eine langwierige Phase der Entwicklung und des Testens. In dieser Arbeit wird das Thema der Fehlerfreiheit des Editors diskutiert. Sicherheitsaspekte hingegen werden in den Hintergrund gestellt.

Sobald der Autor eines Spiels versucht, sein Werk mithilfe der Hauptnavigation (siehe Abbildung 20) zu exportieren oder in die Datenbank zu laden, muss dieser zuerst einen Titel für sein Spiel wählen. Dieser Titel wird beim Export in den Dateinamen im Format „[Name].[Dateiendung]“ eingefügt und beim Upload als Name für das neue Spielschema verwendet. Nach Betätigung des jeweiligen Aktions-Buttons erscheint ein Modal, welches den Nutzer zur Namenswahl auffordert. Hier kann der Nutzer einen Namen eingeben und die Aktion entweder bestätigen oder zurückziehen. Namen, welche bereits in der Datenbank vorliegen, sind ungültig und können aus diesem Grund nicht gewählt werden. Versucht der Nutzer einen existierenden Namen zu wählen, wird eine Warnmeldung über ein Modal hervorgerufen und die Aktion abgebrochen. Ohne eine gültige Namenswahl könnte die Benennung einer Spieldatei gar nicht erfolgen oder sich mit einem bereits vorhandenen Namen überschneiden. Dies führt beim Upload des Spiels zu Problemen. Grund hierfür ist, dass die Durchführung von „CREATE SCHEMA“ Statements mit existierendem oder leerem Schema-Namen über das Java-Servlet zu Java-Exceptions führt, da die Datenbank diese Statements nicht ausführen kann.

Versucht der Nutzer wiederum eine Export- oder Upload-Aktion durchzuführen, ohne dass ein Schema in den Editor geladen ist, so wird die jeweilige Aktion abgebrochen und der Nutzer mithilfe einer Warnmeldung informiert. Auf gleiche Weise wird verfahren, falls der Nutzer versuchen sollte, die Daten des Editors zurückzusetzen, ohne überhaupt ein Schema geladen zu haben. Sowohl beim Wechsel des aktuellen Schemas, beim Zurücksetzen der aktuellen Daten im Editor als auch beim

Import von Spieldaten gehen sämtliche Fortschritte des aktuellen Editorzustands verloren. Um den Nutzer vor dem ungewollten Verlust von Daten zu bewahren, sind diese Funktionen durch einen Warnmechanismus abgesichert. Besteht die Gefahr des Datenverlusts, so wird der Nutzer, ebenfalls mithilfe eines Modals, auf die Gefahr hingewiesen und erhält die Möglichkeit, seine Aktion entweder zurückzuziehen oder zu bestätigen.

Bei der Verwendung des Editors müssen Nutzer nicht nur vor Fehlern bei der Bedienung von Steuer-Elementen geschützt werden. Die Erstellung und Bearbeitung von Inhalten stellt eine Fehlerquelle ähnlichen Ausmaßes dar. Grundlegende Überprüfungen zur Sicherstellung eines reibungslosen Arbeitsablaufs und zur Unterstützung des Nutzers werden in dieser Arbeit berücksichtigt und finden im praktischen Teil Anwendung. Im Editor werden die Informationen über Datentypen, Primärschlüssel und Fremdschlüssel des aktuell geladenen Schemas zur Kontrolle von editierten Daten verwendet. Daten, welche sich nicht an das vorgegebene Schema halten, könnten weitere Exceptions auslösen beim Versuch, diese in die Datenbank einzufügen.

Es wird für jeden Dateneintrag geprüft, ob der Datentyp der zugehörigen Spalte eingehalten wurde. Weiterhin wird kontrolliert, ob Primärschlüssel doppelt vorhanden oder leer sind. Auf gleiche Weise werden auch Composite-Keys überprüft, da für diese ähnliche Bedingungen gelten. Hierbei müssen jedoch, aufgrund der Funktionsweise von Composite-Keys, die Inhalte aller Primärschlüssel-Spalten jeder Datenzeile kombiniert und verglichen werden. Zuletzt überprüft der Editor, ob in den Spalten mit Fremdschlüsseln auch nur Einträge enthalten sind, die auch in ihren jeweils referenzierten Spalten vorkommen. Ergeben eine oder mehrere dieser Prüfungen Fehler, so werden diese dem Autor über verschiedene Feedback-Funktionen kommuniziert. Unter anderem erscheint in der Tabellennavigation auf der linken Seite (siehe Abbildung 21) ein rotes Ausrufezeichen hinter den Tabellen, die fehlerbehaftete Dateneinträge beinhalten. Weiterhin werden Fehler in der Ansicht der jeweiligen Tabelle durch rote Hervorhebung des betroffenen Input-Feldes gekennzeichnet (siehe Abbildung 21). Auf gleiche Weise werden Fehler in der Zeilenansicht über die Textarea-Felder aufgezeigt. In beiden Fällen ist es dem Nutzer möglich, die Maus über das betroffene Feld zu bewegen, um genauere Informationen zu erhalten. Hierbei werden am unteren Rand der Tabellen- oder Zeilenansicht Fehlermeldungen zu allen Fehlerquellen des aktuellen Feldes angezeigt, solange der Nutzer die Maus über dem fehlerbehafteten Feld hält (siehe Abbildung 21).

Bei Korrektur sämtlicher Fehler eines Feldes werden sowohl die rote Hervorhebung als auch die Fehlerinformation sofort aufgehoben. Die Markierung in der Tabellennavigation verbleibt so lange, bis alle Fehler innerhalb der Tabelle behoben wurden. In Abbildung 21 ist beispielsweise erkennbar, dass in der Spalte „aufgaben_nr“ in einer Zeile der String „vier“ statt dem korrekten Integer-Wert eingetragen wurde. Dem Nutzer wurde die fehlerhafte Tabelle markiert und er erhält durch eine Mausbewegung über das betroffene Feld die unten stehende Fehlermeldung „Data-Type: Data is no Integer“.

Umgesetzt wird diese Art der Fehlererkennung mithilfe der Funktion „dataCellErrorList“ inner-

halb des Elm-Codes. Diese berechnet für jede Datenzelle im Editor eine Liste von Strings, welche für jeden gefundenen Fehler eine ausformulierte Meldung enthält. Als Parameter bekommt die Funktion den Namen der aktuell ausgewählten Tabelle, die Zeilen- und Spaltennummer des betrachteten Dateneintrags innerhalb der Tabelle als Tupel sowie den aktuellen Inhalt der Datenzelle. Zusätzlich werden der Funktion auch die aktuellen Tabellen des Editors übergeben. Mithilfe dieser Daten können Überprüfungen zu den einzelnen potenziellen Fehlerquellen durchgeführt werden. Beispielsweise können der Name der aktuellen Tabelle sowie die Koordinaten der betrachteten Datenzelle verwendet werden, um den Spaltennamen der betrachteten Zelle innerhalb der Tabellenliste ausfindig zu machen. Durch diesen kann wiederum ermittelt werden, ob die entsprechende Spalte einen Primary-Key-Constraint besitzt. Trifft dies zu, so kann anschließend der Inhalt der Spalte auf Einträge untersucht werden, welche den gleichen Wert besitzen wie die betrachtete Datenzelle. Existieren solche Einträge, so wird die Flag zur Einbeziehung der Fehlermeldung „Primary-Key: Multiple Occurences“ im Rückgabewert der Funktion auf True gesetzt. Auf ähnliche Weise werden die Tabellen des Editors auch untersucht auf leere Primary-Keys, Composite-Key-Duplikate, falsche Datentypen, leere Foreign-Keys und auch Foreign-Keys, welche nicht in der referenzierten Spalte vorkommen.

Um den Nutzer vor regelmäßigem Ansichtswechsel zwischen der aktuell zu bearbeitenden Tabelle und Fremdschlüsseltabellen zu bewahren, besitzt der Editor eine Vervollständigungsfunktion. Diese bildet für einzelne Input- und Textarea-Felder sowohl mithilfe der zugehörigen Fremdschlüsselinformationen als auch unter Einbeziehung von Composite-Key-Informationen Vorschläge für den Nutzer, welche mithilfe des bisher eingegebenen Strings weiter eingegrenzt werden. Diese Vorschläge bekommt der Nutzer durch einen Klick in das gewünschte Input- oder Textarea-Feld mithilfe eines Dropdown-Menüs angezeigt. Hier kann der Nutzer einen Vorschlag seiner Wahl auswählen oder den Substring im aktuellen Feld mit den zusätzlichen Informationen korrekt zu Ende schreiben. Durch die Auswahl eines Vorschlags wird dieser in das aktuelle Feld eingefügt. Sämtliche Foreign-Key-Fehler werden dabei für dieses Feld behoben und damit auch entfernt.

Im Beispiel in Abbildung 24 existiert ein Fremdschlüssel von der Spalte „nach_raum“ auf die Spalte „raum_nr“ einer anderen Tabelle. Die referenzierte Tabelle enthält drei Räume, welche die Raumnummern 111, 112 und 120 besitzen. Auf der linken Seite der Abbildung ist erkennbar, dass der Nutzer bereits den String „1“ eingegeben hat. An diesem Punkt, oder auch schon vor der Eingabe von Zeichen, ist es dem Nutzer möglich, die Fremdschlüsselbedingung für das Feld durch Auswahl eines Vorschlags zu erfüllen. Er könnte jedoch auch weitere Zeichen hinzufügen, um die Auswahl an Vorschlägen einzugrenzen, wie man auf der rechten Seite von Abbildung 24 erkennen kann. Diese Vorgehensweise kann nützlich sein, falls dem Nutzer zu viele Räume für eine effiziente Auswahl vorgeschlagen werden. Würde in dem gezeigten Beispiel nun ein Composite-Key mit der „nach_raum“-Spalte und einer weiteren Spalte der gleichen Tabelle vorliegen, so könnte unter Umständen die zweite Einsatzmöglichkeit der Vervollständigungsfunktion beobachtet

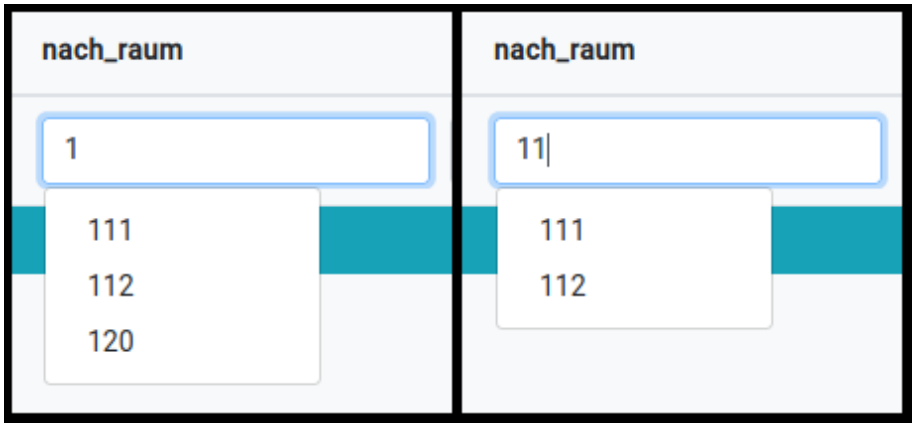

Abbildung 24: Anpassung von Vorschlägen aufgrund des Inputs
Quelle: Eigene Darstellung

werden. Hierzu müsste beispielsweise bereits ein vollständiger Composite-Key „111 - ABC“ in einer anderen Zeile und ein unvollständiger Composite-Key „___ - ABC“ in der aktuellen Datenzeile existieren. In diesem Fall würde der Vorschlag „111“ für die Spalte „nach_raum“ niemals erscheinen, da hierdurch ein Composite-Key-Duplikat vom Nutzer erstellt werden könnte.

Implementiert wurde die Vervollständigungsfunktion innerhalb der Funktion, welche auch für die Anzeige der einzelnen Input- und Textarea-Felder verantwortlich ist. Dies hat den Vorteil, dass hier bereits Informationen über die Koordinaten sowie den Inhalt der betrachteten Datenzelle vorhanden sind. Zusätzlich wird auch hier das Editor-Model mitgeliefert, um Einblicke in die aktuellen Tabellen zu gewährleisten. Mithilfe dieser vorliegenden Daten kann für jede einzelne Datenzelle vorerst die Liste aller ungefilterten Vorschläge berechnet werden. Zu diesem Zweck werden die Fremdschlüsselinformationen der aktuell betrachteten Spalte aus den Schemainformationen der aktuellen Tabelle herausgesucht. Im Anschluss kann hiermit ermittelt werden, welche anderen Spalten die aktuell betrachtete Spalte referenziert. Die Inhalte dieser referenzierten Spalten werden nun in einer Liste zusammengetragen, geordnet und Duplikate entfernt. Aus diesen ungefilterten Vorschläge werden danach alle Ergebnisse entfernt, welche nicht in ihrem vorderen Substring mit dem Wert der betrachteten Datenzelle übereinstimmen (siehe Abbildung 24). Falls die betrachtete Spalte einen Composite-Key bildet, so erfolgt abschließend eine Ausfilterung aller übrigen Vorschläge, welche zusammen mit den Einträgen der zugehörigen Composite-Key-Spalten in der aktuellen Zeile ein Composite-Key-Duplikat bilden würden. Die Vorschlagsliste der betrachteten Datenzelle ist an diesem Punkt vollständig bearbeitet und zeigt dem Nutzer nur Werte an, welche keine Meldungen bei der Fehlerprüfung des Editors ergeben.

4.3 Datenformat

Für den Austausch von Daten zwischen dem Elm-Client und dem Java-Servlet wird ein Format benötigt, nach dem die zu versendenden Daten strukturiert werden können und mit welchem sowohl

in Elm, als auch in Java unkompliziert kommuniziert werden kann. Für dieses Projekt bietet sich die „JavaScript Object Notation“ (kurz „JSON“) an. Diese wurde auch schon in dem vorgefertigten Prototypen des Projektes für die Kommunikation zwischen Client und Servlet verwendet.

Laut [34] ist dieses schlanke Datenaustauschformat so konzipiert, dass es sowohl für Menschen einfach zu lesen als auch für Maschinen einfach zu parsen ist. Außerdem folgt JSON vielen Konventionen, welche Programmierern aus der Familie C-basierter Sprachen, zum Beispiel Java, bereits bekannt sind. Dieser intuitive Aufbau erlaubt, neben dem maschinellen Datenaustausch, Menschen Einblicke in Spieldateien zu nehmen. Hierdurch erhöht sich die Transparenz gegenüber Entwicklern und auch Autoren, beispielsweise bei der Fehlersuche oder bei der manuellen Analyse von Spielen.

Weiterhin besitzen sowohl Elm als auch Java bereits Bibliotheken, welche den Umgang mit Daten im JSON-Format erleichtern. Ein Problem, welches jedoch mit der Wahl dieses Datenformats in Zusammenhang steht und betrachtet werden muss, ist die Möglichkeit des Betrugs beim Spielen durch das Einsehen von Spieldaten. Wäre es einem Spieler möglich, die Daten eines Spiels im JSON-Format einzusehen, so könnte dieser jene Daten als eine Art „Komplettlösung“ verwenden, um sämtliche Inhalte des Spiels vorauszusehen. Würde man sich an dieser Stelle für eine binäre Formatierung der Daten entscheiden, so könnten diese unmöglich von Menschen ohne größeren Dekodierungsaufwand und Fachwissen eingesehen werden. Die Gefahr des unkontrollierten Betrugs, aber auch die Übersichtlichkeit für Autoren und Administratoren würde minimiert werden.

An diesem Punkt ist jedoch zu beachten, dass Autoren erstellte Spiele aus dem Client-seitigen Editor entweder auf ihre lokale Maschine herunter-, oder in die Spiel-Datenbank hochladen. Innerhalb der Datenbank werden die Spieldaten lediglich zur Generierung und Aktualisierung der Spieler-Schemata genutzt. Während eines laufenden Spiels wird beim Login der Inhalt des Spieler-Schemas, nicht jedoch der Inhalt des Spiel-Schemas per HTTP frei lesbar an den Client des Spielers geschickt. Weiterhin muss betrachtet werden, dass das Spieler-Schema zu jedem Zeitpunkt lediglich vom Spieler bereits „freigespielte“ Informationen enthält, also nur diejenigen, zu dessen Einsicht er auch berechtigt ist. Ein Spieler kann in der momentanen Version also zu keinem Zeitpunkt unberechtigten Zugriff auf Daten des aktuell geladenen Spiel-Schemas über das Lernprogramm erhalten. Ein gewisses Risiko bleibt jedoch bestehen, da Autoren ihre heruntergeladenen Spiele in Dateiform auf anderen Wegen veröffentlichen können. Engagierte Cheater könnten auf zukünftig möglichen Plattformen zum Teilen von SQL-Textadventure-Spielen also auch ihr aktuell geladenes Spiel ausfindig machen.

Letztendlich überwiegen jedoch die Vorteile des JSON-Formats die Nachteile, weshalb eine Entscheidung für dieses Datenformat für den weiteren Verlauf dieser Arbeit getroffen wurde. Sicherheit kann in den seltensten Fällen vollständig garantiert werden. Der Kompromiss, welche hier eingegangen wird, genügt im Rahmen dieser Arbeit. Zudem wurde auch für Quest ein lesbares Dateiformat (siehe Abbildung 8) für lokal gespeicherte Spiele gewählt.

4.4 Input

Zu der Entwicklung eines Editors für das Textadventure-Lernprogramm gehört die Planung, welche Daten dem Tool zu Beginn jeder Verwendung übergeben werden müssen. Als potenzielle Input-Daten stehen sämtliche Daten zur Verfügung, welche auf der Datenbank, auf der lokalen Maschine des Nutzers oder zur Laufzeit im Client-Programm gespeichert sind. Benötigt werden für die Funktionen des Editors unter anderem Schema-Daten der Spiel-Datenbank. Es ist abzusehen, dass die Spiel-Datenbank im Verlauf der Entwicklung des Lernprogramms Anpassungen unterliegen wird. Die Wahrscheinlichkeit ist hoch, dass dem einzigen aktuell existierendem Testschema „german“ noch weitere Tabellen und Schlüssel hinzugefügt, manche aber auch wieder entfernt werden. Zusätzlich ist die Ergänzung weiterer „Default-Schemata“ neben dem einen bereits bestehenden abzusehen.

Es ist zu beachten, dass die Festlegungen und Metadaten eines Schemas den Rahmen des Möglichen für Spiele des Lernprogramms bestimmen. Im Vergleich hierzu bestimmt der Inhalt eines Schemas, also sämtliche Datenzeilen, welche ein Autor mithilfe des Editors erstellt hat, auf welche konkreten Spielinhalte ein Nutzer während des Spielens treffen kann. Das Schema gibt also die Menge aller potenziellen Spiele vor, welche mit seiner Hilfe entwickelt werden könnten. So kann ein Raum beispielsweise nur im Spiel existieren, solange das zugehörige Schema auch eine Tabelle mit Räumen verwaltet. Weiterhin kann nur sichergestellt werden, dass Gegenstände lediglich in existierenden Räumen liegen, wenn ein Fremdschlüssel von den Raumnummern der Gegenstände auf die Raumnummern der Raumtabelle verweist. Welche konkreten Räume ein Spieler besuchen kann, ist jedoch abhängig davon, welche Datenzeilen der Autor des Spiels zur Raumtabelle hinzugefügt hat. Hieraus erschließt sich, dass auch der genaue Ort eines Gegenstandes in unserem Beispiel davon abhängt, welche Raumnummer der Autor unter Einhaltung der Fremdschlüsselbedingung dem jeweiligen Gegenstand zugeordnet hat.

Es zeichnet sich also ab, dass vor Veröffentlichung des Projektes Mitarbeiter mit der Erstellung von mindestens einem funktionierenden Schema beauftragt werden müssen, um Spielautoren funktionierende Rahmenbedingungen zur Benutzung des Editors zu schaffen. Auf diesem Weg ist es den Administratoren des Lernprogramms möglich, Spielmechaniken und Spielbegrenzungen für Autoren festzulegen und anzupassen. Das aktuell existierende Schema „german“ genügt, um Mechaniken der Software auszuprobieren, es enthält jedoch noch einige Fehler und zu wenige Möglichkeiten für Autoren, um sinnvolle Spiele zu erstellen. Unter Berücksichtigung dieser bevorstehenden Aufgabe stellt sich die Frage, ob die Entwicklung eines eigenen Editors zur Erstellung von Schemata und nicht nur von Schemainhalten erstrebenswert ist. Hierbei sollte beachtet werden, dass bei der Erstellung eines Schemas in SQL keinerlei Vorgaben der Art existieren, wie es bei der Erstellung von Daten für ein Schema durch Tabellen, Spalten und Constraints der Fall ist. Aus diesem Grund könnte ein Editor für Schemata dem Nutzer kaum unterstützende Funktionen bieten, wie sie bei dem hier entwickelten Spieleditor beispielsweise durch Datentyp- und Schlüsselkontrollen Anwen-

dung finden. Ein solcher Editor könnte dem Nutzer also lediglich das Formulieren von „CREATE Schema“- und „ALTER Schema“-Statements vereinfachen, sowie die Kommunikation mit der Datenbank automatisieren.

Betrachtet man den aktuellen Zweck der Schemata im Zusammenhang mit dem Lernprogramm der Datenbank-Gruppe, so wird schnell klar, dass ihre Anzahl in einer fertiggestellten Version bei weitem nicht die Anzahl erstellter Spiele erreichen wird. Die Anzahl sinnvoller Rahmenbedingungen für Spiele hält sich offensichtlich in Grenzen, wohingegen die Anzahl an potenziellen Spielen für ein einziges Schema in der Theorie bereits unendlich groß ist. Realistisch gesehen wird es letztlich ein bis maximal zwei weit ausgebaute Schemata geben, welche von allen Spielautoren verwendet werden. Zusätzliche Möglichkeiten, welche ein Schema in Form weiterer Tabellen, Schlüssel, oder ähnlichem bietet, können vom Autor immer auch ignoriert werden. So kann ein Schema mit Raumtabelle auch zur Erstellung eines Spiels wie SQL-Island befähigen, in dem keine Räume benötigt werden.

Die Schemata der Spiel-Datenbank sollten also als ein Teil der zu entwickelnden Software angesehen werden, welcher immer auch verbessert und angepasst werden kann, jedoch zur Veröffentlichung des Lernprogramms einen annähernd fertigen und vollständigen Zustand erreicht haben sollte. Diese Schemata können dann innerhalb der Datenbank kopiert werden, um die erstellten Spieldaten des Editors in gesonderten äquivalenten Schemata unterzubringen. Diese Schemata sind also kein geeigneter Kandidat für einen eigenen Editor, welcher eher für die ständige dynamische Entwicklung von Spielinhalten nach der Veröffentlichung bestimmt ist, sowohl auf Seiten der Entwickler als auch der Nutzer.

Um den Editor auch nach den am Anfang von Kapitel 4.4 genannten Versionsänderungen ohne Quellcodeveränderungen verwenden zu können, müssen nach jedem Neustart aktuelle Metadaten des zu verwendenden Schemas eingelesen werden. Weiterhin sollen Autoren namentlich mit ihren erstellten Spielen verknüpft werden. Hierfür ist der Nutzername aus der User-Datenbank für den Editor notwendig. Zuletzt soll der Editor lokal gespeicherte Spieldateien zur erneuten Anpassung oder für den Upload in die Datenbank laden können. Gespeicherte Spiele liegen hierbei im JSON-Dateiformat vor. Aus diesem Grund muss der Editor JSON-Dateien einlesen können, welche sich auf dem Datenträger der lokalen Maschine des Nutzers befinden. Die Planung des Speicherns, Ladens und Teilens von editierten Spielen wird im Kapitel „Output“ genauer erklärt und diskutiert. Daten welche zur Laufzeit im Client vorliegen, werden für die Bearbeitungsfunktionen des Editors nicht benötigt.

Die Schemadaten der Spiel-Datenbank müssen vorerst vom Java-Servlet ausgelesen und anschließend im JSON-Format an den Elm-Client zum Editor gesendet werden. Für diesen Prozess ist ein einheitlich spezifiziertes JSON-Schema notwendig, nach welchem die zu sendenden Daten strukturiert werden. Das Schema dient außerdem bei der Entwicklung des Elm-Clients als Standard, um geeignete äquivalente Datenstrukturen anzulegen (siehe Kapitel 4.1). Diese Datenstrukturen helfen beiden Seiten der Software, die JSON-Daten zu speichern und auf diese zuzugreifen.

Die erste Überlegung bei der Erstellung des JSON-Schemas ist, wie das zu lesende Schema durch den Nutzer ausgewählt wird. Hierfür werden die Namen aller Schemata der Spiel-Datenbank vom Servlet ausgelesen. Alle Schemanamen, welche kein Spiel abbilden können, werden an dieser Stelle mithilfe einer Blacklist aus der Antwort entfernt. Dieser Filter funktioniert, da die Menge der Nicht-Spiel-Schemata in der Datenbank innerhalb einer Version des Lernprogramms unverändert bleibt. Zu diesen ausgeschlossenen Schemata gehört zum Beispiel „information_schema“, welches Metadaten zur Datenbank enthält. Anschließend werden die übrigen Schemanamen an den Editor gesendet. Hier wählt der Nutzer ein Schema aus, dessen Name dann im Body der folgenden HTTP-Request zurück an das Servlet übergeben wird. An diesem Punkt können die Metadaten des gewählten Schemas ausgelesen und als JSON strukturiert werden. Es wird eine Liste von Tabellen-Objekten benötigt, welche der Erstellung von zugehörigem Tabelleninhalt genügen. Jedes dieser Tabellen-Objekte sollte deshalb sowohl den Namen der Tabelle als auch eine Liste von Spaltenobjekten („columnSchemaList“) beinhalten. Die Spaltenobjekte müssen sowohl den Namen der Spalte („columnName“), ihren Datentyp („columnType“), die Information, ob die Spalte zum Primary-Key der Tabelle gehört („primaryKey“) als auch eine Liste der zugehörigen Foreign-Keys („foreignKeyList“) enthalten. Ein solches Objekt der Liste der Foreign-Keys beinhaltet hierbei den Namen der Fremdschlüsseltabelle („foreignTableName“) sowie den Namen der Fremdschlüsselspalte („foreignColumnName“). Abbildung 25 zeigt eine Beispielinstanz des modellierten Schemas, welche vom aktuellen Servlet versendet wurde. Sämtliche Listen wurden hierbei der Übersichtlichkeit wegen durch Auslassungspunkte verkürzt, da das Beispiel lediglich zum Verstehen des Schemas dienen soll. Das modellierte Schema ist in Abbildung 26 erkennbar. Dieses Schema wurde nach den Vorgaben des „JSON Schema Draft-07“ [35] angefertigt.

```
[
    {
        "tableName":"raeume",
        "columnSchemaList": [
            {
                "columnName":"raum_nr",
                "columnType":"int8",
                "primaryKey": true,
                "foreignKeyList":[]
            },
            {
                "columnName":"titel",
                "columnType":"varchar",
                "primaryKey": false,
                "foreignKeyList":[]
            },
            ...
        ]
    },
    {
        "tableName":"wege",
        "columnSchemaList": [
            {
                "columnName":"von_raum",
                "columnType":"int8",
                "primaryKey": true,
                "foreignKeyList": [
                    {
                        "foreignTableName":"raeume",
                        "foreignColumnName":"raum_nr"
                    }
                ]
            },
            {
                "columnName":"nach_raum",
                "columnType":"int8",
                "primaryKey": true,
                "foreignKeyList": [
                    {
                        "foreignTableName":"raeume",
                        "foreignColumnName":"raum_nr"
                    }
                ]
            },
            ...
        ]
    },
    ...
]
```

Abbildung 25: Input-JSON-Schema - Beispielinstanz
Quelle: Eigene Darstellung

```
{
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "tableName" : {"type": "string"},
            "columnSchemaList": {
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "columnName" : {"type": "string"},
                        "columnType": {"type": "string"},
                        "primaryKey": {"type": "boolean"},
                        "foreignKeyList": {
                            "type": "array",
                            "items": {
                                "type": "object",
                                "properties": {
                                    "foreignTableName": {"type": "string"},
                                    "foreignColumnName": {"type": "string"}
                                },
                                "required": [
                                    "foreignTableName",
                                    "foreignColumnName"
                                ]
                            }
                        }
                    },
                    "required": [
                        "columnName",
                        "columnType",
                        "primaryKey",
                        "foreignKeyList"
                    ]
                }
            }
        },
        "required": [
            "tableName",
            "columnSchemaList"
        ]
    }
}
```

Abbildung 26: Input-JSON-Schema
Quelle: Eigene Darstellung

Der Name des aktuellen Nutzers wird für die Zuordnung von Autoren zu Spielen benötigt. Hierfür wird dieser vom Servlet in einer eigenen SQL-Anfrage von der Datenbank direkt nach dem Login eines Nutzers in den Client abgefragt. Anschließend wird der Name als JSON-Objekt per HTTP an den Editor zur Weiterverarbeitung gesendet. Im Elm-Programm wird der Nutzername an das Elm-Model des Editors zur weiteren Verwendung übergeben. Da sich der Nutzername während einer Sitzung nicht verändern sollte, wurde die Abfrage an den ersten möglichen Zeitpunkt verschoben. Die Abfrage hätte auch bei einem Wechsel in den Editor-Tab oder auch beim Laden eines Schemas im Editor durchgeführt werden können. Hierbei wäre jedoch mindestens genauso viel Traffic entstanden, solange der Nutzer die Verwendung des Editors bezweckt. Mithilfe der gewählten Lösung befindet sich der Nutzername außerdem sofort nach dem Login im Client und kann auch für Funktionen außerhalb des Editors verwendet werden.

4.5 Output

Nach erfolgreicher Erstellung oder Bearbeitung eines Spiels im Editor müssen die durchgeführten Änderungen festgehalten werden. Hierfür ist zu planen, wie die editierten Daten zur Spielinitialisierung in die Datenbanken gelangen, wie Spiele längerfristig gespeichert werden können und auf welche Weise Werke mit Nutzern an anderen Maschinen geteilt werden können.

Zuerst ist abzuwägen, ob erstellte Spiele auf der Festplatte von Autoren als Datei abgespeichert werden sollten. Der vergleichbare Editor „Quest“ [9] lässt bei der Arbeit im Online-Tool nur die Speicherung von Spielen in der Datenbank der Website zu. Das lokale Speichern auf der Maschine des Nutzers als Datei ist lediglich mithilfe des herunterladbaren Offline-Programms möglich. Diese Spieldateien können anschließend auf der zugehörigen Webseite in die Datenbank von textadventures.co.uk geladen und dort verwaltet werden.

Für den Editor des SQL-Lernprogramms wird eine hybride Lösung beider Ansätze angewandt. Nutzern wird wie bei „Quest“ die Möglichkeit gegeben auf direktem Weg Spiele, welche sich aktuell intern im Editor befinden, in die Datenbank zu laden. Durch Betätigung eines Buttons werden die Daten im JSON-Format mithilfe des HTTP-Protokolls an das Servlet geschickt, um von dort aus über die bestehende Java-Datenbank-Schnittstelle die Datenbank zu aktualisieren. Hierbei wird zuerst das gewählte Schema aus der Datenbank vom Servlet kopiert, mithilfe des gewählten Namens in der Datenbank neu erstellt und anschließend mit den Daten des Spiels befüllt. Den Vorteil dieses Ansatzes stellt die Nähe der erstellten Daten zum Client dar, welche dem Nutzer in Zukunft das direkte Übernehmen eines neuen Spiels in die Gamepage ermöglichen könnte. Sofort nach der Fertigstellung im Editor könnte das Spiel zum Testen bereit sein. Bis diese Funktion aktiv genutzt werden kann, muss jedoch vorerst die Generierung von Spieler-Datenbanken aus fertigen Spielen gelöst und realisiert werden (siehe Kapitel 4.6). Ohne diese Funktion ist ein Spielen von Werken aus dem Editor noch nicht möglich. Mithilfe der oben genannten direkten Lösung muss der Nutzer keinen Aufwand investieren, um Speicher- und Lademechanismen des Clients für fertige

Spiele zu verstehen und zu verwenden.

Zusätzlich zu dieser Lösung wird Autoren die Möglichkeit gegeben, ihre Werke als Datei auf ihre Festplatte zu exportieren und auch zurück in den Editor zu importieren. Diese Entscheidung wurde mit der Voraussicht getroffen, dass zusätzlich zum Online-Editor, ähnlich wie bei Quest (siehe Kapitel 2.2), ein weiterer Offline-Editor entwickelt werden könnte. Auf diese Weise wird für die Entwicklung des Offline-Tools eine Schnittstelle angeboten, sodass für diese Aufgabe nicht mehr in den Programmcode des hier entwickelten Online-Editors eingegriffen werden muss. Weiterhin kann die Verbreitung selbst erstellter Inhalte durch den Umstand gefördert werden, dass Nutzer im Besitz von Spieldateien sein können. In Folge dessen besäße die Community die Möglichkeit, Spiele für das Lernprogramm auf eigenen inoffiziellen Seiten und auch privat auszutauschen. Durch dieses Konzept kann eine breitere Masse an Menschen auf das Lernprogramm aufmerksam gemacht werden, denn mehr aktive Sharing-Plattformen bedeuten auch mehr Möglichkeiten auf das Projekt zu stoßen.

Zuletzt bildet die Funktion des Spielexports den Grundstein für das Speichern und Laden von Spielständen, was über den Rahmen dieser Arbeit hinaus geht, aber durchaus in Zukunft in das Projekt eingebracht werden könnte. Für die Realisierung von Spielständen wäre das Speichern eines Zustands der aktuell verwendeten Spieler-Datenbank parallel in der Spieldatei, oder aber auch als eigene Spielstand-Datei nötig. Diese müsste dann zurück in die gleichnamige Datenbank geladen werden können, um den Zustand zum Zeitpunkt des Speicherns wieder herzustellen.

Die Daten eines editierten Spiels benötigen ein einheitliches Format, in welchem sie auf der Festplatte des Nutzers abgespeichert werden. Hierfür eignet sich, wie im Kapitel 4.3 erläutert, erneut das JSON-Datenformat. Wie im Kapitel refinput ist auch hier die Modellierung eines JSON-Schemas zur Strukturierung und Validierung von Spieldaten nötig. Insbesondere wird auch hier wieder das JSON-Schema als Standard für die Festlegung äquivalenter Datenstrukturen im Elm-Client verwendet werden (siehe Kapitel 4.1).

Im äußersten JSON-Objekt jeder Instanz dieses JSON-Schemas sollte sowohl der Titel des Spiels, die Namen aller beteiligten Autoren, der Name des Spiel-Schemas aus der Datenbank, das Erstellungsdatum als auch das Datum der letzten Bearbeitung zu finden sein. Diese Metadaten eines Spiels sollen Lesern der Datei einen Überblick über das vorliegende Spiel geben und gehören aus Gründen der Lesbarkeit in das äußerste Objekt ganz oben. Zusätzlich wird auf der gleichen Ebene der Datei eine Liste von Objekten („gameTables“) angelegt, welche die eigentlichen Spieldaten enthält. Jedes Element der Liste beschreibt hierfür eine Tabelle. Ein solches Tabellenobjekt enthält wieder, wie bereits im Input-Schema (siehe Kapitel 4.4) beschrieben, den Namen der zugehörigen Tabelle („tableName“) und eine darauf folgende Liste mit Spieldaten der Tabelle namens „rowList“. Anders als im Kapitel Input werden die Daten in dieser Liste jedoch nicht in Spaltenform, sondern in Zeilenform strukturiert. Dies hat den Vorteil, dass die benötigten INSERT-Statements, welche zur Aktualisierung der Datenbank benötigt werden, im Servlet direkt und ohne Umformatierung

aus den Datenzeilen der Zeilenobjekte gebildet werden können. Innerhalb eines Zeilenobjektes sind leider keine festgelegten Properties möglich, da diese abhängig vom gewählten Schema sind. Zur Steigerung der Lesbarkeit der JSON-Dateien wurde hier bei der Implementierung mit dem „additionalProperties“-Attribut gearbeitet. Innerhalb von Objekten eines „rowList“-Arrays ist eine beliebige Anzahl beliebig benannter Properties zulässig. Lediglich der Datentyp konnte hier auf „string“ eingegrenzt werden. Durch dieses Konzept ist es möglich die Objekte mit Spaltenname-Wert-Paaren zu befüllen. Auf diese Weise können Leser der Datei schnell und eindeutig erkennen, was einzelne Werte einer Zeile bedeuten.

Eine Alternative zu diesem Ansatz wäre es jede Datenzeile ohne Tabellenkopf als gesonderte Liste zu speichern. Hierdurch könnte der Algorithmus im Servlet mit weniger Aufwand den gleichen Grad an Funktionalität erreichen bei einer geringeren Dateigröße und damit weniger Traffic. All diese Faktoren haben jedoch einen so minimalen Einfluss, dass kein spürbarer Unterschied für den Nutzer entstehen würde. Das Gegenteil ist eher der Fall. Die Rohdatei des Spiels könnte von keinem Nutzer mehr ohne größeren Zeitaufwand verstanden werden, da der nötige Kontext in Form der Spaltennamen entfernt wurde. Beide in Betracht gezogenen Ansätze bringen hierzu noch einen weiteren Nachteil mit sich. Es ist keine präzise Vollständigkeitsüberprüfung der Properties einer Datenzeile durchführbar wie es Beispielsweise für „tableName“ der Fall ist. Überschüssige, redundante, fehlende und falsch benannte Attribute könnten die Pipeline des Servlets ohne Probleme passieren. Diese Vollständigkeit muss somit an anderer Stelle, also im Editor, sichergestellt werden.

In den folgenden Abbildungen 27 und 28 wird das modellierte Output-JSON-Schema zusammen mit einem Beispiel für das Verständnis als Code dargestellt. Das Schema ist erneut, wie auch das Input-JSON-Schema aus dem Kapitel „Input“, nach den Vorgaben des „JSON Schema Draft-07“ [35] angefertigt worden.

```
{
    "title": "Awesome Game",
    "authors": [
        "Max Mustermann"
    ],
    "schemaName": "german",
    "creation_date": "11.11.2020",
    "last_modified": "12.02.2021",
    "gameTables": [
        {
            "tableName": "aufgaben",
            "rowList": [
                {
                    "aufgaben_nr": 1,
                    "text": "erste Aufgabenbeschreibung",
                    "von_person": "4"
                },
                {
                    "aufgaben_nr": 2,
                    "text": "zweite Aufgabenbeschreibung",
                    "von_person": "7"
                },
                ...
            ]
        },
        ...
        {
            "tableName": "raeume",
            "rowList": [
                {
                    "raum_nr": 1,
                    "titel": "Raum 1",
                    "beschreibung": "eine interessante Beschreibung",
                    "raumtext": "ein längerer Text",
                    "points": "100 200 300 400"
                },
                {
                    "raum_nr": 2,
                    "titel": "Raum 2",
                    "beschreibung": "eine weitere interessante Beschreibung",
                    "raumtext": "ein weiterer längerer Text",
                    "points": "400 300 200 100"
                },
                ...
            ]
        },
        ...
    ]
}
```

Abbildung 27: Output-JSON-Schema - Beispielinstanz
Quelle: Eigene Darstellung

```
{
    "type": "object",
    "properties": {
        "title": {"type": "string"},
        "schemaName": {"type": "string"},
        "authors": {
            "type": "array",
            "items": {"type": "string"}
        },
        "date_created" : {"type": "string"},
        "date_lastModified" : {"type": "string"},
        "gameTables": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "tableName" : {"type": "string"},
                    "rowList": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "additionalProperties": {"type": "string"}
                            }
                        }
                    }
                },
                "required": [
                    "tableName",
                    "rowList"
                ]
            }
        }
    }
}
```

Abbildung 28: Output-JSON-Schema
Quelle: Eigene Darstellung

4.6 Weitere Forschungsmöglichkeiten

Alle implementierten Strukturen und Features des Editors wurden bis zu diesem Punkt fertig besprochen. Es existieren jedoch noch weitere Ideen, welche aufgrund der Grenzen dieser Arbeit praktisch nicht weiter verfolgt werden konnten. Diese Ansätze sollten jedoch unbedingt in zukünftigen Versionen berücksichtigt und im besten Fall auch umgesetzt werden.

Als erstes ist anzumerken, dass die in Kapitel 4.4 und 4.5 formulierten JSON-Schemata noch nicht praktisch in der aktuellen Version des Projektes verwendet werden. Eine Anwendungsmöglichkeit, welche jedoch nicht vernachlässigt werden sollte, ist die Kontrolle von im Editor-Servlet ein- und ausgehenden JSON-Daten. Hierfür könnte beispielsweise die Java-Bibliothek „json-schema“ verwendet werden, die vorgefertigte Funktionen für genau diesen Zweck bietet.

Weiterhin muss erneut erwähnt werden, dass noch einige wichtige Schritte bis zum Spielen selbsterstellter SQL-Textadventures mithilfe des Clients gemacht werden müssen. Mit Fertigstellung dieser Arbeit existiert immer noch kein vollständig entwickeltes Spiel-Schema, das jedoch zur Erstellung eines sinnvollen Spiels benötigt wird (siehe Kapitel 4.4). Außerdem gibt es ebenfalls keine Möglichkeit ein fertiges Spiel, welches bereits vom Editor in die Datenbank geladen wurde, durch Generierung eines zugehörigen Spieler-Schemas zu initialisieren (siehe Kapitel 4.1). Diese Punkte sind essenziell für die Verwendung des Editors und sollten in zukünftigen Arbeiten dringend weiter untersucht werden.

Abschließend funktioniert die Überprüfung von Constraints, welche in Kapitel 4.2 besprochen wurde, wie vorgesehen, ist aber bei weitem nicht vollständig. SQL besitzt eine Vielzahl von Constraints in unterschiedlichen Varianten. Keine Beachtung haben in dieser Arbeit bisher Check-Constraints gefunden. Auf sie wurde noch nicht weiter eingegangen, da diese zum aktuellen Zeitpunkt nicht im Testschema „german“ vorkommen. Die Wahrscheinlichkeit ist jedoch hoch, dass sie in zukünftigen Versionen Anwendung finden werden. Ein Check-Constraint bezieht sich immer auf eine bestimmte Spalte einer Tabelle. Hier überprüft er, ob sämtliche Werte der Spalte einem vorgegebenen Boolean-Ausdruck genügen. Spaltenwerte, welche in Kombination mit diesem Ausdruck nicht den Wert „TRUE“ ergeben, dürfen nicht in der zugehörigen Spalte vorkommen. Zur Überprüfung von Check-Constraints müssen, bei der Arbeit von Autoren im Editor, zusätzliche Informationen beim Laden eines Schemas an den Editor gesendet werden. Es muss innerhalb des Editors ermittelt werden können, in welcher Tabelle, in welcher Spalte und mithilfe welchen Boolean-Ausdrucks ein Check-Constraint arbeitet. Diese Informationen können in PostgreSQL-Datenbanken in der Meta-Tabelle „pg_constraint“ im Schema „pg_catalog“ gefunden und mithilfe eines SELECT-Statements ausgelesen werden. Ein mögliches Statement ist in Abbildung 29 zu sehen. Hier wird „pg_constraint“ mithilfe von JOIN-Operatoren mit der Tabelle „constraint_column_usage“ zusammengeführt, aus welcher die Tabellen- und Spaltennamen der Check-Constraints entnommen werden.

Ein Statement dieser Art könnte im Servlet zum Auslesen der Check-Constraints aus der Daten-

```
select ccu.table_name,
ccu.column_name,
pgc.consrc as definition
from pg_constraint pgc
join pg_namespace nsp on nsp.oid = pgc.connamespace
join pg_class cls on pgc.conrelid = cls.oid
left join information_schema.constraint_column_usage ccu
on pgc.conname = ccu.constraint_name
and nsp.nspname = ccu.constraint_schema
where contype ='c'
order by pgc.conname;
```

Abbildung 29: Select-Statement zur Check-Constraint-Abfrage
Quelle: In Anlehnung an [36]

bank verwendet werden. Anschließend müssten die Daten der Constraints in den JSON-Objekten, welche im Kapitel 4.4 spezifiziert sind, für den Versand an den Editor untergebracht werden. Es bietet sich an, hierfür jedes Element in jeder „columnSchemaList“ mit einem weiteren Attribut zu versehen. Der Tabellenname und der Spaltenname können durch den Editor jedem Constraint bereits aufgrund der verwendeten Datenstruktur (siehe Abbildungen 25 und 26) zugeordnet werden. Das hinzugefügte Attribut sollte ein Array von Strings der Art „>4“, „<=ColumnName“ oder auch „<>0“ enthalten.

An dieser Stelle muss erläutert werden, dass für PostgreSQL noch viele weitere Operationen existieren, welche zum Datentyp Boolean auswerten und für Check-Constraints verwendet werden könnten. Darunter befinden sich unter anderem Predikatausdrücke als auch Musterausdrücke. Zu den bekanntesten Predikaten gehören die Ausdrücke „IS NULL“ oder auch „BETWEEN“. Das Pattern-Matching wird in PostgreSQL über die Operatoren „LIKE“ und „SIMILAR TO“ in Kombination mit regulären Ausdrücken gehandhabt. Bei Check-Constraints, welche Ausdrücke dieser Art verwenden, ist eine große Menge an Sonderfällen zu betrachten, weshalb diese auch gesondert bearbeitet werden sollten. Beispielsweise wäre für die korrekte Implementierung eines LIKE-Checks im Editor eine vollständige Erkennung regulärer Ausdrücke notwendig. Im Folgenden wird lediglich die Realisierung von Check-Constraints mithilfe von Vergleichsoperatoren betrachtet.

Das erweiterte JSON-Objekt wird anschließend weiter an den Editor gesendet. Hier können nun die Constraint-Daten in Elm-Records vom Typ „ColumnSchema“ gespeichert werden. Dieser Typ besitzt eine äquivalente Struktur wie Objekte der „columnSchemaList“ des Input-JSON-Schemas (siehe Abbildung 26) und dient der Überprüfung, ob ein Eingabefeld einen Constraint erfüllen muss. Ab diesem Punkt kann ähnlich wie im Kapitel 4.2 verfahren werden. Bei Aktualisierung eines Feldes, welches einen Check-Constraint erfüllen muss, wird der bisher eingegebene String zusammen mit allen Check-Constraint-Elementen der Liste der aktuellen Spalte zu einzelnen Elm-Boolean-Ausdrücken übersetzt. Ergibt einer dieser Ausdrücke den Wert FALSE, so werden an den üblichen Stellen (siehe Kapitel 4.2) Fehlermeldungen angezeigt. Die Daten über sämtliche Check-Constraints einer Spalte sollten ebenfalls über den Informations-Button in der Zeilenansicht (siehe

Abbildung 23) des Editors abrufbar sein. Zur Zeit wird in dem genannten Button ein Schlüssel-Icon angezeigt, da hier bislang nur Informationen zu Schlüssel-Constraints vermittelt werden. Bei Ergänzung von Check-Constraints zu diesem Reiter sollte ein für alle Constraints passenderes Icon eingesetzt werden.

Auf andere Constraint-Arten wird in dieser Arbeit nicht weiter eingegangen. Zur Vollständigkeit fehlen noch UNIQUE-, NOT-NULL- und EXCLUSION-Constraints. Zumindest die UNIQUE- und NOT-NULL-Constraints sollten auf ähnliche Weise wie die in diesem und in Kapitel 4.2 behandelten Constraints realisiert und implementiert werden können.

5. Fazit

Im Rahmen dieser Arbeit wurden Konzepte zur Aufwertung einer unausgereiften Client-Anwendung untersucht und realisiert. Diese haben das optische Erscheinungsbild, die intuitive Bedienbarkeit und auch den reinen Funktionsumfang für den Nutzer verbessert. Weiterhin konnte ein interner relationaler Spiel-Editor für den Client als zugeschnittene Softwarelösung entwickelt werden. Dieser Editor ermöglicht es Nutzern, eigene Textadventures auf Grundlage eines vorgefertigten Spiel-Schemas zu editieren und zu verwalten.

Im Client selbst konnte die ursprünglich platzverschwendende Hauptnavigation durch die gezielte Verwendung von Bootstrap-Elementen in ein schlankes Navbar-Objekt umgewandelt werden, welches genügend Raum für jeglichen Seiteninhalt lässt. Weiterhin wurde allen Seiten des Clients ein einheitliches und qualitativ hochwertigeres Aussehen verliehen, welches sich sowohl durch die durchgängige Implementierung stiltechnisch zusammenhängender Bootstrap-Elemente als auch eine designtechnisch durchdachte Farbgebung auszeichnet. Innerhalb der GamePage, welche den spielerischen Aspekt der Anwendung realisiert, konnten die meisten Fortschritte am Funktionsumfang erreicht werden. Hier wurde unter anderem eine Möglichkeit realisiert, welche den Nutzer zur effizienteren Wiederverwendung einer bereits gestellten Anfrage befähigt. Zusätzlich hierzu sind jetzt große Tabellenausgaben in der Konsole nur mithilfe eines Pagination-Systems untersuchbar, wodurch diese nicht mehr den Großteil des Output-Bereichs einnehmen können. Ein weiteres Feature für zukünftige Nutzer bildet der Resize-Bar der GamePage, welcher eine Anpassung der Größenverhältnisse von Input und Output in der Vertikale ermöglicht. Hierdurch kann sich der Nutzer situationsabhängig viel Platz für eine Suche im Output-Bereich nehmen, oder aber diesen für die Formulierung einer großen Anfrage im Input-Bereich nutzen. Zuletzt wurde die GamePage in zwei Alternativen aufgeteilt, welche beide über die Hauptnavigation zu erreichen sind und beide ein neues, vorteilhafteres, Layout bekommen haben. Hier werden weitere Untersuchungen in Aussicht gestellt, um das beste Layout für eine finale Version des Projektes zu ermitteln. Zuletzt konnte im Rahmen dieser Arbeit ein Plugin-Konzept ausgearbeitet werden, welches zukünftigen Entwicklern eine einheitliche Schnittstelle für zusätzliche optionale Funktionalitäten bietet. Dieses Konzept wurde vorerst lediglich in die GamePage implementiert. Eine Ausweitung auf den gesamten Client ist jedoch ohne weiteres möglich.

Der Spiel-Editor des Projektes wurde innerhalb dieser Arbeit von Grund auf geplant und implementiert. Hierfür wurden vorerst ein Seitenlayout konstruiert, die Darstellung von Schemadaten und Inhalten entworfen und anschließend grundlegende Verwaltungsfunktionen implementiert. Zu diesen Funktionen gehört das Laden eines Schemas aus der Datenbank, der Reset aller aktuellen Daten des Editors, der Export eines editierten Spiels auf die Festplatte des Nutzers, der Import von Spieldateien von der Festplatte des Nutzers sowie der saubere Upload fertiggestellter Spiele in die Datenbank. Die Arbeit innerhalb des Editors wurde dabei jedoch auch berücksichtigt. Hier werden dem Nutzer sowohl umfangreiche Fehlermeldungen als auch eine Vervollständigungsfunkti-

on zur effizienteren und übersichtlicheren Arbeit angeboten. Die Ansicht von editierten Daten kann außerdem zwischen einer Tabellenform und einer Zeilenform umgeschaltet werden, wodurch dem Nutzer entweder eine gröbere oder feinere Sicht auf die Daten gewährt wird. Programmtechnisch war für die Umsetzung des Editors eine Planung des Formats ein- und ausgehender Daten essenziell. Abschließend kann gesagt werden, dass diese Datenformate in jeder aufgezählten Funktion des Editors in irgendeiner Form zum Einsatz kommen und hiermit einen ungemeinen Wert für den Gesamtzusammenhang der Arbeit darstellen.

Es ist zu erkennen, dass am Entwicklungsstand des Lernprogramms durchaus Fortschritte mit dem Verlauf der Arbeit zu verzeichnen sind. Es existiert jedoch noch eine umfangreiche Menge weiterer Ansätze für das Projekt, welche im Rahmen dieser Arbeit aus zeittechnischen Gründen nicht weiter betrachtet werden konnte. Grund hierfür sind die Umstände, unter welchen die Arbeit an dem Projekt begonnen wurde. Die ursprüngliche Version des Projektes wurde zuletzt im Jahre 2017 weiterentwickelt. Die letzte Person, welche umfangreiche Änderungen am Programmcode vorgenommen hat, war leider nicht mehr am Projekt beteiligt. Hierdurch und aufgrund fehlender Dokumentation wurde die Einarbeitung wesentlich erschwert. Weiterhin existierten viele angefangene Baustellen innerhalb des Clients, die noch zu Ende geführt werden mussten. Einige davon konnten in dieser Arbeit berücksichtigt, auf andere konnte hingegen nur verwiesen werden.

Inwieweit das SQL-Textadventure-Lernprogramm der Datenbank-Gruppe zukünftigen Studenten, Schülern oder auch anderen Wissbegierigen ein effektives Lern- und Spielumfeld bieten kann, bleibt abzuwarten. Hierbei wird sich in zukünftigen Arbeiten zeigen, welche weiteren Verbesserungen in den Bereichen Design, Funktionalität, Fehlervermeidung und Sicherheit möglich sind. Der Weg zu einer funktionstüchtigen Anwendung zum Spielen und Lernen ist jedenfalls nicht mehr weit.

Literaturverzeichnis

[1] game – Verband der deutschen Games-Branche e.V., *Jahresreport der deutschen Games-Branche*, Website, 2020, Adresse: https://www.game.de/wp-content/uploads/2020/08/game-Jahresreport-2020.pdf, abgerufen am 22. Oktober 2020.

[2] J. M. Ritzko und S. Robinson, „Using Games To Increase Active Learning“, *Journal of College Teaching & Learning (TLC)*, Jg. 3(6), 2006, Adresse: https://doi.org/10.19030/tlc.v3i6.1709, abgerufen am 22. Oktober 2021.

[3] H. Rastegarpour und P. Marashi, „The effect of card games and computer games on learning of chemistry concepts“, *Procedia - Social and Behavioral Sciences*, Jg. 31, S. 597–601, 2012, World Conference on Learning, Teaching & Administration - 2011, ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2011.12.111. Adresse: http://www.sciencedirect.com/science/article/pii/S1877042811030400, abgerufen am 04. November 2021.

[4] N. E. Cagiltay, E. Ozcelik und N. S. Ozcelik, „The effect of competition on learning in games“, *Computers & Education*, Jg. 87, S. 35–41, 2015, ISSN: 0360-1315. DOI: https://doi.org/10.1016/j.compedu.2015.04.001. Adresse: http://www.sciencedirect.com/science/article/pii/S0360131515001001, abgerufen am 04. November 2021.

[5] interactive-fiction.de, *Was bedeutet Interactive Fiction?*, Website, 2009, Adresse: http://interactive-fiction.de/, abgerufen am 22. Oktober 2020.

[6] C. Muckenhoupt, *Adventure*, Website, 2000, Adresse: http://web.archive.org/web/20120510201955/http://www.wurb.com/if/game/1, abgerufen am 07. Januar 2021.

[7] Infocom, *Zork*, Website, 1980, Adresse: http://textadventures.co.uk/games/view/5zyoqrsugeopel3ffhz_vq/zork, abgerufen am 23. Oktober 2020.

[8] Blizzard-Entertainment, *Warcraft 3: Reforged*, Website, 2002, Adresse: https://eu.shop.battle.net/de-de/family/warcraft-iii, abgerufen am 10. März 2021.

[9] textadventures.co.uk, *Quest*, Website, 2011, Adresse: http://textadventures.co.uk/quest, abgerufen am 23. Oktober 2020.

[10] M. J. Roberts, *TADS - the Text Adventure Development System*, Website, 2001, Adresse: http://www.tads.org/, abgerufen am 24. Oktober 2020.

[11] C. Klimas, *Twine / An open-source tool for telling interactive, nonlinear stories*, Website, 2009, Adresse: https://twinery.org/, abgerufen am 24. Oktober 2020.

[12] ifwiki.org, *What is „interactive fiction“?*, Website, o. D., Adresse: http://www.ifwiki.org/index.php/FAQ, abgerufen am 29. Oktober 2020.

[13] J. Tidwell, *Designing Interfaces: Patterns for Effective Interaction Design*. Sebastopol: O'Reilly Media, 2005, First Edition.

[14] J. Schildgen und S. Deßloch, „SQL-Grundlagen spielend lernen mit dem Text-Adventure SQL Island“, in *Datenbanksysteme für Business, Technologie und Web (BTW 2015)*, T. Seidl et al., Hrsg., Bonn: Gesellschaft für Informatik e.V., 2015, S. 687–690, Adresse: https://dl.gi.de/handle/20.500.12116/2448, abgerufen am 10. März 2021.

[15] J. Schildgen, *SQL-Island*, Website, 2014, Adresse: https://sql-island.informatik.uni-kl.de/, abgerufen am 29. Oktober 2020.

[16] textadventures.co.uk, *Quest 5 - Dokumentation*, Website, o. D., Adresse: http://docs.textadventures.co.uk/quest/, abgerufen am 24. Oktober 2020.

[17] ——, *Introduction to coding with Quest*, Website, o. D., Adresse: https://docs.textadventures.co.uk/quest/quest_code.html, abgerufen am 15. Februar 2021.

[18] ——, *ASLX File Format*, Website, o. D., Adresse: https://docs.textadventures.co.uk/quest/aslx.html, abgerufen am 15. Februar 2021.

[19] J. Vrána, *Architecture of Adminer*, Website, 2009, Adresse: https://php.vrana.cz/architecture-of-adminer.php, abgerufen am 17. Februar 2021.

[20] ——, *Adminer*, Website, 2021, Adresse: https://www.adminer.org/de/, abgerufen am 10. März 2021.

[21] ——, *Adminer - Erweiterungen*, Website, 2021, Adresse: https://www.adminer.org/de/extension/, abgerufen am 17. Februar 2021.

[22] Oracle, *Oracle Fusion Middleware*, Website, 2021, Adresse: https://www.oracle.com/de/middleware/technologies/, abgerufen am 16. Februar 2021.

[23] ——, *Working With Oracle Forms*, Website, 2021, Adresse: https://docs.oracle.com/en/middleware/developer-tools/forms/12.2.1.4/working-forms/index.html, abgerufen am 16. Februar 2021.

[24] ——, *Oracle Forms Services & Oracle Forms Developer 12c Technical Overview*, Website, 2016, Adresse: https://www.oracle.com/technetwork/developer-tools/forms/documentation/forms-overview-2858191.pdf, abgerufen am 16. Februar 2021.

[25] R. Sijmons, *The Possibilities of Postgres: Nibble IT's Database Migration Story*, Website, 2020, Adresse: https://www.enterprisedb.com/blog/complete-migration-from-oracle-to-postgresql-database-including-oracle-forms-nibble-it-story, abgerufen am 16. Februar 2021.

[26] E. Czaplicki und S. Chong, „Asynchronous Functional Reactive Programming for GUIs“, in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Ser. PLDI '13, Seattle, Washington, USA: Association for Computing Machinery, 2013, S. 411–422. DOI: 10.1145/2491956.2462161. Adresse: https://doi.org/10.1145/2491956.2462161, abgerufen am 10. März 2021.

[27] E. Czaplicki, *An Introduction to Elm*, Website, 2012, Adresse: https://guide.elm-lang.org/, abgerufen am 14. Februar 2021.

[28] ——, *Elm Packages*, Website, 2012, Adresse: https://package.elm-lang.org/, abgerufen am 14. Februar 2021.

[29] getbootstrap.com, *Getting started*, Website, o. D., Adresse: https://getbootstrap.com/docs/4.6/getting-started/introduction/, abgerufen am 19. Februar 2021.

[30] M. Rundberget, *Build responsive Elm applications using Bootstrap 4.* Website, o. D., Adresse: http://elm-bootstrap.info/, abgerufen am 19. Februar 2021.

[31] pgAdmin Development Team, *pgAdmin v4.30*, Website, 2021, Adresse: https://www.pgadmin.org/, abgerufen am 08. Februar 2021.

[32] The PostgreSQL Global Development Group, *PostgreSQL Client Applications*, Website, 2020, Adresse: https://www.postgresql.org/docs/current/app-psql.html, abgerufen am 08. Februar 2021.

[33] Epic War, *WC3MapDB 2.2.5*, Website, 2004, Adresse: https://www.epicwar.com/maps/, abgerufen am 09. Februar 2021.

[34] json.org, *Einführung in JSON*, Website, 2020, Adresse: https://www.json.org/json-de.html, abgerufen am 24. November 2020.

[35] json-schema.org, *Understanding JSON Schema*, Website, o. D., Adresse: http://json-schema.org/understanding-json-schema/, abgerufen am 23. November 2020.

[36] dataedo.com, *List all check constraints in PostgreSQL database*, Website, 2019, Adresse: https://dataedo.com/kb/query/postgresql/list-check-constraints-in-database, abgerufen am 13. Februar 2021.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und die allgemeinen Grundsätze guter wissenschaftlicher Praxis beachtet habe.

Marcus Gagelmann