BOSCH
Technik fürs Leben

DHBW
Duale Hochschule
Baden-Württemberg

Sudoku Helper

Studienarbeit

Bachelor of Science

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Ruben Hartenstein, Annika Harter

10.06.2022

Bearbeitungszeitraum	22.10.2022 - 10.06.2022
Matrikelnummer, Kurs	2746235, 4810277, TINF19ITA
Ausbildungsfirma	Robert Bosch GmbH, Stuttgart
Betreuer	Sebastian Trost

Selbstständigkeitserklärung

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema: *Sudoku Helper* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 10.06.2022

Ruben Hartenstein, Annika Harter

Abstract

Sudokus sind Logikrätseln und werden in verschiedenen Medien publiziert. Es gibt unterschiedliche Schwierigkeitsstufen von Sudoku Rätseln. Wie bei allen Logikrätseln ist ein Sudoku lösbar durch das Anwenden verschiedener Lösungsstrategien. Die Lösungsstrategien variieren in ihrem Schwierigkeitsgrad.

In dieser Studienarbeit wird ein Web-Frontend entwickelt, dass einen User bei der Lösung eines schwierigen Sudoku Rätsels unterstützt. Zu Beginn kann ein User ein Rätsel in ein leeres Gitter eingeben und es wird überprüft, ob das Sudoku die Eigenschaft der eindeutigen Lösbarkeit besitzt.

Die Lösung des Sudoku soll mittels der Strategien Schritt für Schritt erklärt werden. Unter anderem soll die Technik benannt und anhand des konkreten Beispiels erläutert werden. Zudem soll auf dem Board der Bereich markiert werden, indem der User eine Technik anwenden kann. Die Hilfestellungen werden in verschiedenen Abstufungen umgesetzt, sodass dem User erst die Chance gegeben wird eine Technik selbst anzuwenden, bevor eine Erläuterung gegeben wird.

In der Studienarbeit werden auch verschiedene technische Rahmenbedingungen wie Webserver oder Framework evaluiert und mathematischen Eigenschaften eines Sudoku Rätsels beleuchtet.

Die Implementierung der Studienarbeit soll in Python und JavaScript erfolgen. Eine weitere Anforderung ist eine benutzeroptimierte Darstellungen sowohl für Mobile Endgeräte als auch auf einem Desktop PC oder Laptop.

Inhaltsverzeichnis

Abkürzungsverzeichnis VI

Abbildungsverzeichnis VII

Tabellenverzeichnis VIII

Formelverzeichnis IX

Listings X

1. Einleitung 1
1.1. Motivation . . . 2
1.2. Forschungsstand . . . 2
1.3. Forschungsfrage . . . 3

I. Wissenschaftliche Vertiefung 4

2. Rahmenbedingungen 5
2.1. Programmiersprache . . . 5
2.1.1. Python . . . 5
2.1.2. JavaScript . . . 6
2.2. Webserver . . . 7
2.2.1. Apache HTTP Server . . . 8
2.2.2. Nginx . . . 9
2.2.3. Fazit . . . 9
2.3. Framework . . . 10
2.3.1. Django . . . 10
2.3.2. Flask . . . 11
2.3.3. Fazit . . . 11
2.4. Entwurfsmuster . . . 12
2.4.1. Model View Controller . . . 13

3. Sudoku 14
3.1. Spielfeld Aufteilung . . . 14
3.1.1. Zeile . . . 15

3.1.2. Spalte . . . 15
3.1.3. Box/Block . . . 16
3.2. Regeln . . . 16
3.3. Strategien . . . 17
3.3.1. Lösungshilfen: Kandidaten-Notation . . . 18

4. Die Mathematik hinter Sudoku 19
4.1. Abzählfragen . . . 19
4.2. Komplexität . . . 21
4.3. Eindeutige Lösbarkeit . . . 22
4.3.1. Anzahl vorgegebene Ziffern für ein eindeutiges Sudoku . . . 22
4.4. Algorithmische Lösungsmethode: Backtracking . . . 23
4.4.1. Backtracking mit Brute-Force-Methode . . . 24
4.4.2. Beweis Eindeutigkeit . . . 25

II. Implementierung 27

5. Frontend 28
5.1. User-Interface . . . 28
5.1.1. HTML Struktur . . . 29
5.1.2. Responsive Design . . . 30
5.2. User-Experience . . . 31
5.2.1. Zustände . . . 32
5.2.2. User Input . . . 33
5.2.3. Funktionalität . . . 34
5.3. Abstufungen der Hilfestellungen . . . 35
5.3.1. Erste Hilfestellung . . . 35
5.3.2. Zweite Hilfestellung . . . 36
5.3.3. Dritte Hilfestellung . . . 37
5.3.4. Vierte Hilfestellung . . . 37
5.4. Weitere Funktionalitäten . . . 38
5.4.1. Fertig gelöstes Sudoku . . . 38
5.4.2. Keine Technik anwendbar . . . 38

6. Backend 39
6.1. Programmarchitektur . . . 39
6.1.1. Model . . . 39
6.1.2. View . . . 40
6.1.3. Controller . . . 40
6.2. Serverseite . . . 40
6.2.1. SocketIO . . . 41

6.3. SudokuBoard 41
6.3.1. Datenstruktur des Sudoku Boards 42
6.3.2. Funktionalität 43
6.4. Technique Manager 43
6.5. Abstrakte Klasse SolvingTechniques 44
6.5.1. Abstrakte Methoden 45
6.5.2. Hilfsfunktionen 47
6.6. Lösungsstrategien 47
6.6.1. Umgesetzte Lösungsstrategien 47
6.6.2. Weitere Lösungsstrategien 48
6.7. Beispielhafte Umsetzung von Strategien 49
6.7.1. Zahl einfügen 49
6.7.2. Kandidaten eliminieren 51

7. Testing/Validierung **53**
7.1. Frontend 53
7.2. Backend 53
7.3. End-to-End Test 54

8. Fazit **55**
8.1. Ergebnisse 55
8.2. Beantwortung Forschungsfrage 56
8.3. Ausblick 56

Literatur **A**

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
BRC	Block-Reihe-Check
BSD	Berkley Software Distribution
CE	Clear everything
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
HTTPS	Hypertext Transport Protocol Secure
MVC	Model View Controller
RBC	Reihe-Block-Check
TCP	Transmission Control Protocol
UI	User-Interface
UX	User-Experience
XML	Extensible Markup Language

Abbildungsverzeichnis

2.1. Benutzungsstatistiken von Webservern [W3t] 8
2.2. Model View Controller (MVC) Konzept 13

3.1. Darstellung des Sudokugitters mit der Markierung einer Zeile 15
3.2. Darstellung des Sudokugitters mit der Markierung einer Spalte 15
3.3. Darstellung des Sudokugitters mit der Markierung einer Box 16
3.4. Beispiel Rätsel mit Kandidatennotation 18

4.1. Markierung von drei linear verbundenen Blöcken 20
4.2. Markierung eines Dreier Sets als Kombination aus Block und Zeile 20

5.1. Hypertext Markup Language (HTML) Struktur 29
5.2. Responsive Frontend 31
5.3. Veränderung des NumPads nach Starten 32
5.4. Abbildung der möglichen Markierungsvarianten 34
5.5. Darstellung der ersten Hilfestellung 36
5.6. Darstellung der zweiten Hilfestellung 37

6.1. HTTP vs. WebSocket [PLA] 41
6.2. Struktur des Sudokuboards und Zellenidentifikation [Zam15] 42

Tabellenverzeichnis

2.1. Vergleich der Webserver Apache HTTP Server und Nginx 10
2.2. Vergleich der Frameworks Django und Flask 11

5.1. Funktionalitäten der Tastatur 33

6.1. Aufzählung aller implementierten Lösungsstrategien mit deutschem und englischem Bezeichner 48

Formelverzeichnis

4.1. Möglichkeiten vollständig ausgefüllte 9×9 Standard-Sudokus 19
4.2. Anzahl an Ergänzungen zu der ersten Reihe 20
4.3. Anzahl an Möglichkeiten zu den obersten drei Reihen 21
4.4. Zeitkomplexität Backtracking 23

Listings

4.1. Backtracking zum Lösen des Sudoku Rätsel 24
4.2. Abbruchbedingung Backtracking 25
4.3. Überprüfen auf eindeutige Lösbarkeit 26

5.1. Erste Hilfestellung 35
5.2. Zweite Hilfestellung 36
5.3. Vierte Hilfestellung 38

6.1. Initalisierung des Boards 42
6.2. Initalisierung der Kandidaten 43
6.3. Finden einer anwendbaren Lösungstechnik 44
6.4. Serverseitig help 49
6.5. Nackter Single 50
6.6. Serverseitige Unterscheidung der Auswirkung von Strategien 51
6.7. Versteckter Vierer Teil 2 52

1. Einleitung

In vielen Zeitungen ist eine Vielzahl von Logikrätseln gedruckt. Darunter eines der häufigsten Logikrätsel: das Sudoku. In der üblichen Version ist es das Ziel, ein 9×9-Gitter mit den Ziffern 1 bis 9 zu füllen. Dabei kann das Gitter in drei unterschiedliche Einheiten aufgeteilt werden. Diese Einheiten sind die Spalten, Zeilen und Blöcke des Gitters. Ein Block ist ein 3×3-Unterquadrat des Gitters. In jeder Einheit darf jede Ziffer genau einmal vorkommen. Ausgangspunkt ist ein Gitter, in dem bereits mehrere Ziffern vorgegeben sind, damit das Sudoku eindeutig lösbar ist. Sudokus gibt es in unterschiedlichen Schwierigkeitsgraden.

In der Studienarbeit soll ein **Sudoku Helper** entwickelt werden. Ziel ist es dem Nutzer verschiedene Strategien unterschiedlicher Schwierigkeitsstufen anhand eines realen Beispiels zu erklären und so den Lösungsweg für schwere Sudokus zu erläutern. [Zei12] [Sud22]

Im Verlaufe der Arbeit wird zunächst die Motivation und Aufgabenstellung der Studienarbeit beschrieben.

Danach werdend die Rahmenbedingungen, wie beispielsweise die Wahl der Programmiersprache, des Frameworks und des Webservers erläutert. Im Zug dieser wissenschaftlichen Vertiefung wird auch genauer auf die Mathematik, die hinter Sudokus steht, geschaut. Dabei werden verschiedene Themen wie die Komplexität eines Sudokus oder die eindeutige Lösbarkeit diskutiert und erläutert.

Der nächste größere Teil der Studienarbeit ist die Implementierung. Hierbei werden Designentscheidungen für Front- und Backend erläutert, sowie deren technische Umsetzung. Im Zug der Implementierung werden auch die Testing- und Validierungsstrategien beschrieben.

Am Schluss der Arbeit steht ein Fazit mit den Ergebnissen und Erfahrungen, sowie ein Ausblick, wie das Projekt noch verbessert und erweitert werden könnte.

1.1. Motivation

Sudokus gibt es mittlerweile in vielen verschiedenen Varianten, mit modifizierten Regeln. In dieser Arbeit und für den **Sudoku Helper** wird jedoch nur das klassische Sudoku, wie es in der Einleitung beschrieben wurde, betrachtet. Es gibt verschiedene Schwierigkeitsstufen von Sudokus. Meistens werden Sudokus mit vielen vorgegebenen Zahlen als einfach eingestuft und Sudokus mit wenigen vorgegebenen Zahlen als schwierig. Ein weiterer Punkt, der Einfluss auf die Schwierigkeitsstufe eines Sudokus hat, ist die Anordnung der vorgegebenen Ziffern.

Wer schon einmal versucht hat ein schweres Sudoku zu lösen, ist vielleicht auch bereits an seine Grenzen gekommen. Gerade bei schwierige Sudokus mit wenigen vorgebenden Zahlen kann es schwierig sein selbst eine passende Lösungsstrategie für das Sudoku zu finden. Oft sind anwendbare Lösungsstrategien schwierig zu erkennen und es lassen sich damit meist nur Kandidaten eliminieren. Einfachen Sudokus lassen sich oft durch simple logische Schlussfolgerungen lösen und Lösungsstrategien werden mehrheitlich unbewusst und intuitiv eingesetzt. [Mar]

In dieser Studienarbeit soll ein **Sudoku Helper** entwickelt werden, der Lösungsstrategien programmatisch erkennt und dem Benutzer bei dem Finden eines Lösungsschrittes unterstützt. Diese Unterstützung erfolgt dabei in mehreren Stufen, von der Visualisierung, über die Erklärung bis zur automatischen Umsetzung der Strategie.

1.2. Forschungsstand

Im Internet findet man immer wieder Websites mit dem Angebot Sudokus zu lösen. Hierbei können die angegebenen Zahlen eingegeben werden und das Sudoku wird korrekt vervollständigt, falls es eine eindeutige Lösung gibt. Dabei bekommt der Nutzer nur das gelöste Sudoku ohne Informationen darüber, wie man ein Sudoku als Mensch ohne die Rechenleistung eines Computers logisch lösen kann. Viele Websites erklären bereits, welche Lösungsstrategien existieren, wie sie funktionieren und wie sie sich anwenden lassen, nutzen dabei aber meist konkrete Beispiele.

Es existieren aber auch schon ähnliche Varianten zu dem in der Studienarbeit umgesetzten **Sudoku Helper**. Aber im Vergleich zur Studienarbeit wird in diesen Varianten keine schrittweise Hilfestellung angeboten.

1.3. Forschungsfrage

Für diese Studienarbeit ist die Zielsetzung, ein **Sudoku Helper** zu entwickeln. Ziel ist es nicht ein Sudoku automatisch zu lösen, sondern dem Benutzer Schritt für Schritt den Weg zur Lösung aufzuzeigen. Auf dieser Basis wird zunächst geprüft, ob das Sudoku eindeutig lösbar ist. Ist dies der Fall, muss der Nutzer die Möglichkeit haben, sich Anweisungen geben zu lassen, wie man weiter vorgehen könnte und welche Lösungsstrategie der Benutzer anwenden kann, damit man beim Lösen des Sudokus fortschreiten kann. Auf Basis der bereits eingetragenen Zahlen und den übrig gebliebenen Kandidaten, soll geprüft werden, welche Strategie momentan anwendbar ist. Dem Benutzer soll daraufhin die anwendbare Strategie genannt und anhand des konkreten Sudokus erklärt werden, damit der Benutzer das Sudoku lösen kann. Diese Anweisungen, im folgenden auch Hilfestellungen genannt, sind in vier Varianten unterteilt.

Für die erste Hilfestellung soll das Programm dem Benutzer die anzuwendende Strategie nennen und den Bereich hervorheben, auf dem die Strategie angewendet werden soll. Wenn diese Hilfestellung nicht ausreichend ist, dann markiert das Programm die konkreten Zellen und Kandidaten, die von der Strategie betroffen sind. Mit der dritten Hilfestellung soll eine Erläuterung gezeigt werden, warum bestimmte Kandidaten laut Strategie gestrichen werden können. Mit der letzten Hilfestellungen löscht das Programm die betroffenen Kandidaten. Für Ausnahmen, in denen eine Strategie eine Zahl hervorhebt, die eingetragen werden kann, erfolgt das Eintragen der Zahl anstatt das Löschen von Kandidaten.

Um ein eindeutig lösbares Sudoku vervollständigen zu können, soll der **Sudoku Helper** etwa 25 Lösungsstrategien beherrschen.

Das Programm soll auf einem Ubuntu Server mit Apache und Django laufen. Des Weiteren gibt es zusätzlich die Anforderung, dass das Web-Frontend responsive sein soll und immer eine benutzeroptimierte Darstellungen liefert.

Teil I.

Wissenschaftliche Vertiefung

2. Rahmenbedingungen

Für das Projekt werden verschiedene Rahmenbedingungen festgelegt. Dazu gehört die Programmiersprache, indem der **Sudoku Helper** programmiert werden soll, ein Webserver auf dem das Programm später laufen wird und ein Framework, dass als Gerüst für das Programm auf dem Webserver fungiert. Wichtig ist, dass das Framework auf die Programmiersprache angepasst ist, um Probleme zu vermeiden. Im Zuge der Recherchen wurden verschiedenen Webserver in Betracht gezogen und besonders die Webserver Apache HTTP Server und Nginx Server aufgrund ihrer Beliebtheit miteinander verglichen. Auch bei den Frameworks gab es mehrere Optionen, von denen zwei insbesondere in Betracht gezogen und evaluiert wurden. Im Folgenden werden die Pros und Kontras der genannten Server und Frameworks aufgezählt und die schlussendliche Entscheidung begründet.

2.1. Programmiersprache

In den folgenden Abschnitten werden die genutzten Programmiersprachen vorgestellt und die Entscheidungen für diese erläutert. Neben Python für das Backend wird für das Frontend JavaScript mit der Erweiterung von jQuery verwendet.

2.1.1. Python

Für die Implementierung der Strategien und generell im Backend wurde sich für die Programmiersprache Python entschieden. Python ist eine höhere, universelle Sprache und ist als eine sehr vielseitige Programmiersprache anerkannt. Mit Python kann ein objektorientierter, aspektorientierter oder funktionaler Programmierstil umgesetzt werden und die Typisierung funktioniert dynamisch. Des Weiteren wird Python oft bei der Datenanalyse und bei der Verarbeitung großer Datenmenge verwendet. Zusätzlich spielt die Laufzeit zunächst nur eine untergeordnete Rolle und ist nicht sicherheitsrelevant, wodurch bei der Nutzung von Python keine größeren Probleme entstehen sollten. Ein weiterer Vorteil von Python ist die große Standardbibliothek, die, falls nötig, durch

Module erweitert werden kann. Noch ein Vorteil ist die hohe Kompatibilität mit mehreren Frameworks. Im Laufe des Kapitels werden noch zwei Frameworks vorgestellt, die für das Projekt in Betracht gezogen worden.

Für das Backend ist eine objektorientierte Umsetzung vorgesehen, die mit Python umgesetzt werden kann. Zudem ist Python einfach zu lesen und klar strukturiert. Auch für die Umsetzung der Lösungsstrategien bietet sich Python durch Module für mathematische Berechnungen und Datenwissenschaft an.

In diesem Projekt wird mit der Version 3.9 von Python gearbeitet. [Ern20] [Sar20, S. 50 ff.] [Sta]

2.1.2. JavaScript

Im Frontend wurde sich für JavaScript mit jQuery entschieden. JavaScript ist eine der meist verwendeten Programmiersprachen der Welt. Als JavaScript erfunden wurde, ging man davon aus, dass es für kurze Codeschnipsel verwendet wird, eingebettet in eine Webseite. Es ist eine Skriptsprache und wurde für die Benutzerinteraktion entwickelt. Mithilfe von JavaScript lassen sich Inhalte verändern oder neu generieren. Für das Projekt ist die Interaktion mit dem Benutzer von entscheidender Bedeutung, da er ständig neue Zahlen oder Kandidaten eingeben kann.

Diese Anforderungen lassen sich mit JavaScript umsetzten. Daher wurde sich beim Frontend Devolpment für JavaScript mit der Erweiterung von jQuery entschieden. [Ste20, S. 7 ff.] [Sta]

jQuery

Neben JavaScript wird die freie Programmbibliothek jQuery für das Frontend verwendet. jQuery bietet die Funktionalität der Document Object Model Navigation und Manipulation sowie ein erweitertes Event-System. Mit jQuery wird die JavaScript-Programmierung vereinfacht und viele Methoden können in einer einzelnen Zeile von Code umgesetzt werden. Ein weiterer Vorteil ist, dass auch jQuery einfach erweitert werden kann und sehr populär ist. [W3s]

2.2. Webserver

Die Studienarbeit soll zum Schluss auf einem Webserver laufen. In diesem Kapitel wird beschrieben, welchen Anforderungen ein Webserver gerecht werden muss. Danach werden mit dem Apache HTTP Server und dem Nginx Webserver zwei beliebte Webserver vorgestellt und verglichen. Zuletzt wird die Designentscheidung aufgrund des Vergleichs begründet.

Ein Webserver ist in der Regel ein Server, der zur Verbreitung von Webinhalten im Inter- oder Intranet dient. Die jeweiligen Informationen und Dokumentationen können demnach weltweit oder firmenintern erreicht werden. Damit eine Website jederzeit erreichbar sein kann, muss der Webserver permanent online sein.

Der Rechner, auf dem der Webserver läuft, wird als Host bezeichnet. Der Webserver ist für die zuverlässige Übertragung von statischen, wie beispielsweise von unveränderlichen HTML-Dateien, aber auch von dynamischen Dateien verantwortlich. Für dynamische Dateien muss der Webserver vor der Antwort Programmcode ausführen. Dieser Programmcode wird in diesem Fall in der Programmiersprache Python geschrieben. In der HTML Datei wird die inhaltliche Gliederung definiert und in der Cascading Style Sheets (CSS) Datei die Darstellung, wie etwa Farben und Formatierung.

Für die Übermittlung wird das Übertragungsprotokoll Hypertext Transport Protocol (HTTP) oder die verschlüsselte Variante Hypertext Transport Protocol Secure (HTTPS) verwendet. Ein Webserver ist in der Lage, die Inhalte auf viele verschiedene Rechner gleichzeitig zu übermitteln. Wie viele Nutzeranfragen (Requests) ein Server bearbeiten kann, hängt von der Hardware und der Auslastung des Hosts ab.

Am meisten verbreitet sind unter anderem die Webserver Apache HTTP Server und Nginx, wie in die Statistik 2.1 dargestellt. Beide Programme sind freie Software und wurden daher für das Projekt in Betracht gezogen. [Kar19] [Sta]

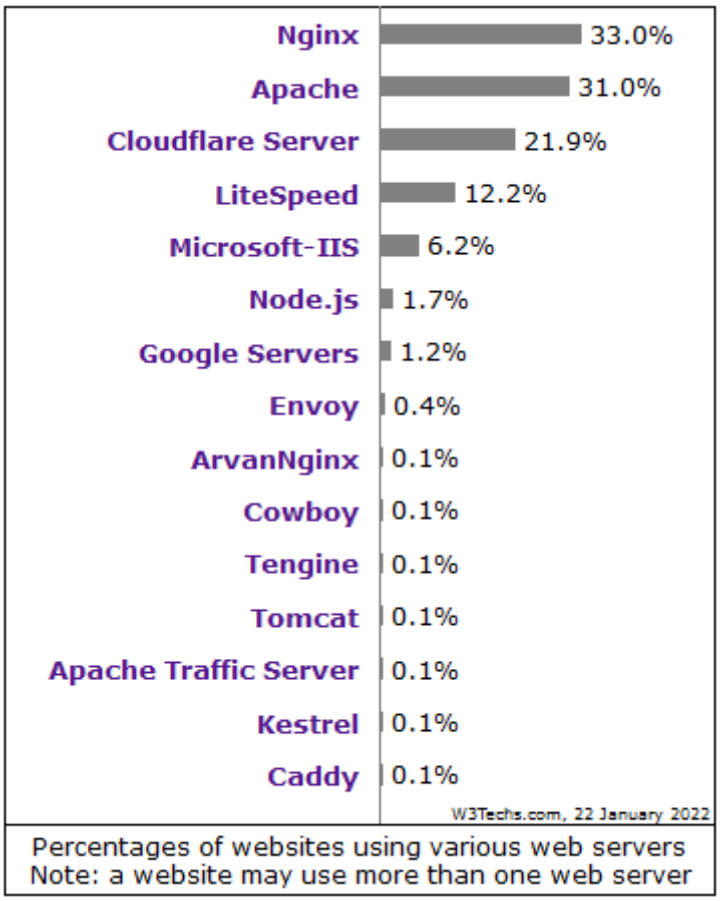


Abbildung 2.1.: Benutzungsstatistiken von Webservern [W3t]

2.2.1. Apache HTTP Server

Der Apache Webserver wurde erstmals 1995 veröffentlicht und hat sich schnell zu einem der beliebtesten Webserver entwickelt. Er unterstützt neben Unix und Linux noch eine Vielzahl an weiteren Betriebssystemen.

Der Apache Server ist modular aufgebaut, wodurch benötigte Funktionen, die der Server nicht nativ bereitstellt, durch Module importiert werden können. Das Erstellen dynamischer Webseiten wird mittels serverseitiger Skriptsprachen bewerkstelligt. Über ein Modul kann der entsprechende Interpreter in den Server integriert werden.

Beim Apache-Webserver wird ein Ansatz verfolgt, bei dem jede Clientanfrage von einem separaten Prozess oder Thread bearbeitet wird. Dadurch werden Prozesse, die Schreib- oder Leseoperationen erfordern, nacheinander abgearbeitet und es kann passieren, dass ein Request in der Warteschlange verweilen muss, bis der vorherige Request durchgeführt werden konnte. Damit man dieses Problem umgehen kann, gibt es die Möglichkeit,

mehrere Single-Threading-Prozesse gleichzeitig zu starten. Diese Strategie ist jedoch mit einem hohen Ressourcenaufwand verbunden. Um dies zu vermeiden, kommen Multi-Threading-Mechanismen zum Einsatz. Für die parallele Abfrage von Clientanfragen gibt es verschiedene Multi-Processing-Module, die integriert werden können. [Kar19] [Ion]

2.2.2. Nginx

Der Marktanteil von Nginx ist in den letzten Jahren kontinuierlich gestiegen, weshalb auch dieser Webserver für die Studienarbeit in Betracht gezogen wurde. Nginx wurde erstmals 2004 veröffentlicht und ist wie der Apache Server auch mit diversen Betriebssystemen kompatibel.

Wie Apache ist auch Nginx modular aufgebaut und verschiedene Funktionen können über Module bereitgestellt werden. Zu diesen Modulen gehört jedoch nicht die Option, Interpreter für eine Programmiersprache entsprechend in den Webserver zu integrieren. Es wird also ein weiterer Anwendungsserver benötigt. Für kleine Webprojekte ist das ein Mehraufwand, der nicht immer eingegangen werden muss.

Dieser Server zeichnet sich durch eine hohe Performance aus. Dabei können eine möglichst große Anzahl an Clients gleichzeitig bedient werden und der Ressourcenverbrauch trotzdem gering gehalten. Durch die ereignisorientierte Architektur können Client-Anfragen asynchron bearbeitet werden, wodurch Arbeitsspeicher und Zeit gespart werden kann. Die Nebenläufigkeit ist realisiert, ohne dass für jede neue Verbindung ein zusätzlicher Prozess oder Thread benötigt wird. Die Stärke dieser Architektur zeigt sich bei großen Webprojekten. [Kar19] [Ion]

2.2.3. Fazit

Die folgende Tabelle 2.1 zeigt die Unterschiede und Gemeinsamkeiten der beiden Webserver auf.

Der Apache Webserver bietet eine breite Möglichkeit an Modulen, um die Software zu erweitern. Der ausschlaggebende Grund, warum sich in diesem Projekt für den Apache HTTP Server entschieden wurde, ist die Möglichkeit Interpreter für Programmiersprachen über ein Modul direkt in den Webserver zu integrieren. Zudem muss für das Projekt nur ein Client bedient werden. Dadurch wird für das Anzeigen von dynamischen Inhalten kein separaten Anwendungsserver benötigt. Damit bietet der Apache HTTP Server eine bequemere Lösung für kleine Websites, deren Inhalt dynamisch erzeugt wird.

Merkmal	Apache	NGINX
Funktion	Webserver, Proxy-Server	Webserver, Proxy-Server, E-Mail-Proxy, Load-Balancer
Betriebssystem	Alle unixoiden Plattformen, Windows	FreeBSD, Linux, Solaris, IBM AIX, HP-UX, macOS, Windows
Lizenz	Apache License v2.0	BSD-Lizenz
Entwickler	Apache Software Foundation	Nginx, Inc.
Statische Webinhalte	Ja	Ja
Dynamische Webinhalte	Ja	Nein
Software-Architektur	Prozess-/threadbasiert	Eventgesteuert

Tabelle 2.1.: Vergleich der Webserver Apache HTTP Server und Nginx

2.3. Framework

Dieser Abschnitt stellt zwei verschiedene Frameworks vor, die für die Studienarbeit infrage kommen. Auch hier wird zunächst erläutert, was ein Framework ist und anschließend zwei vorgestellt, verglichen und die Nutzungsentscheidung darauf basierend begründet.

Ein Framework ist ein Programmiergerüst. Es wird insbesondere bei komponentenbasierten und bei der objektorientierten Softwareentwicklung verwendet. Ein Framework ist kein fertiges Programm, sondern stellt den Rahmen, in der eine Anwendung erstellt werden soll, zur Verfügung. Frameworks lassen sich in Typen gliedern, wobei es nicht immer eine strikte Trennung gibt. Zu diesen Typen gehören beispielsweise Application Frameworks oder Webframeworks.

Die beiden Frameworks, Django und Flask, die für das Projekt in Betracht gezogen wurden, gehören zu den beliebtesten Webframeworks für Python und sind für die Entwicklung von dynamischen Webanwendungen ausgelegt. [Ste20, S. 195 ff.] [Sta] [Lub22]

2.3.1. Django

Django ist ein kostenloses Open Source Webframework und kann plattformübergreifend genutzt werden. Es folgt dem MVC Entwurfsmuster, das auch in der Studienarbeit genutzt wird. Der Fokus von Django liegt in einer schnellen Entwicklung mit weniger Code. Mit dem Framework kommen viele bereits eingebaute Pakete, die genutzt werden können.

Dazu gehört unter anderem die integrierte Anbindungen an verschiedenen Datenbanksysteme. Django hat verschiedene Mechanismen integriert, um die Sicherheit der Website zu garantieren. So können SQL Injektionen oder Cross-site Scripting ausgeschlossen werden.

Ein weiterer Vorteil von Django ist die große und aktive Community, wodurch viel Fragen schnell beantwortet werden können und es gibt es eine ausführliche Dokumentation zu dem Framework. [Her22] [Lub22]

2.3.2. Flask

Webseiten, die Flask als Framework nutzen, sind meistens Single-Page-Webanwendungen. Flask ist im Gegensatz zu Django modular und eher minimalistisch aufgebaut. Daher muss in der Entwicklung tendenziell mit externen Bibliotheken gearbeitet werden. Man nennt Flask deshalb auch ein Mikroframework. Für die Sicherheit der Website gibt es die Flask-Security Bibliothek, die dieselben Mechanismen wie Django beinhaltet.

Aus diesen Gründen ist Flask eher für kleinere Projekte geeignet, die benutzerdefinierte Komponenten erfordern oder für das Prototyping. Die Anzahl der Webseiten, die Flask als Framework nutzen, steigt stetig weiter. Der Vorteil, warum Flask immer beliebter wird ist die Kontrolle, die man über das Projekt bekommt, indem man die Komponenten selbst bestimmen kann. [Her22]

2.3.3. Fazit

Beide Frameworks wären für das Projekt geeignet. Die Django Community ist größer und aktiver als die von Flask. Daher gibt es mehr Informationen über Django. Jedoch wird Flask mittlerweile in mehreren Projekten genutzt. In der Tabelle 2.2 sind die wichtigsten Eigenschaften nochmals gegenübergestellt.

Merkmal	Django	Flask
Use cases	mittelgroße bis große Projekte	kleine bis mittelgroße Projekte
Community	große, aktive Community	kleinere Community
Packages	built-in Packages	modular
Performance	gut	gut

Tabelle 2.2.: Vergleich der Frameworks Django und Flask

Aufgrund der Leichtigkeit wurde sich bei den Frameworks für Flask entschieden. Da es bei der Performance keine großen Unterschiede zwischen Django und Flask gibt, ist es als Merkmal für die Entscheidung eines Frameworks nicht relevant. Für den **Sudoku Helper** wird nur eine einseitige Anwendung ohne Datenbankanbindung benötigt, wodurch Flask das geeignetere Framework ist. Auch der modulare Aufbau spricht für Flask, da nur wenige zusätzliche Pakete genutzt werden.

Das Entwurfsmuster MVC wird für das Framework Flask übernommen.

2.4. Entwurfsmuster

In der Studienarbeit wird für die Übersichtlichkeit ein Entwurfsmuster verwendet. Der folgende Abschnitt stellt das Entwurfsmuster und die Komponenten vor.

Wie im Abschnitt der Frameworks schon erwähnt, wird in diesem Projekt die MVC Softwarearchitektur verwendet, damit der Programmcode eine gewisse Übersichtlichkeit behält und Kopplung reduziert wird. Dadurch lässt sich durch die Möglichkeit die einzelnen Module auszutauschen, die Skalierbarkeit erhöhen. Zudem wird das Zusammenspiel der Komponenten in einem Projekt verständlicher. Damit ist es einfacher nicht-funktionale Eigenschaften wie die Modifizierbarkeit, Wartbarkeit oder Sicherheit darzustellen und zu verstehen. [Mus21, S. 57 f.]

Das Framework Django gibt das MVC Architekturmuster vor. Da aber Flask als Framework genutzt wird, muss diese Architektur selbst implementiert werden. Im Folgenden wird das Architekturmuster mit seinen drei Komponenten kurz beschrieben.

2.4.1. Model View Controller

Das Ziel dieser Entwurfsarchitektur ist es, spätere Änderungen oder Erweiterungen zu vereinfachen und die Wiederverwendbarkeit der einzelnen Komponenten zu steigern. In der Abbildung 2.2 ist die Architektur visualisiert. Durchgezogene Linien stehen hierbei für eine direkte Assoziation und gestichelte für eine indirekte Assoziation. [Mus21, S. 57 f.]

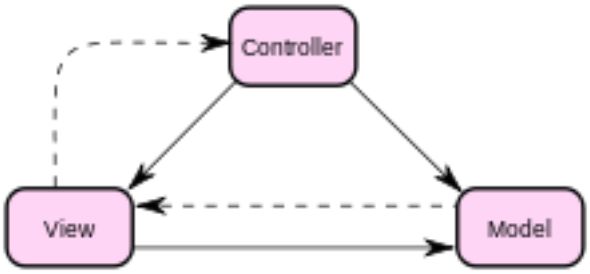


Abbildung 2.2.: MVC Konzept

Model

Im Model sind die Daten enthalten, die dargestellt werden sollen. In Falle des **Sudoku Helper**s ist hier das Sudoku Board gespeichert. Dieser Teil dient zur Verwaltung und Manipulation der Daten und kommuniziert mit der View nur über den Controller.

View

In der View oder der Präsentation werden die Daten aus dem Model dargestellt. Außerdem wird in der View die Benutzerinteraktion geregelt. Die View kennt die Daten aus dem Model, soll diese aber nicht bearbeiten oder verändern. Jegliche Kommunikation mit dem Model erfolgt über den Controller.

Controller

Der Controller ist die zentrale Steuereinheit. In ihm werden das Model und die View verwaltet. Die View unterrichtet den Controller von Benutzerinteraktionen. Im Controller wird diese dann ausgewertet und gegebenenfalls das Model und die View angepasst.

3. Sudoku

Nachdem in der Einleitung schon kurz auf das Spielfeld und die Regeln eines Sudokus eingegangen wurde, werden beides in diesem Kapitel noch einmal ausführlicher behandelt. Zudem werden auf Elemente des Spielfelds eingegangen, welche relevant für die Strategien sind. Im letzten Abschnitt wird das Prinzip der Strategien, um ein Sudoku lösen zu können, behandelt. Da für die Strategien und den **Sudoku Helper** die Lösungshilfe der Kandidaten-Notation eine wichtige Rolle spielt, wird darauf in einem weiteren Unterkapitel eingegangen und die Funktion und Bedeutung erklärt.

3.1. Spielfeld Aufteilung

Zunächst wird die Aufteilung des Spielfelds erläutert und die Units vorgestellt. Jede Unit bekommt eine eindeutige Kennzeichnung, auf den im weiteren Verlauf der Arbeit referenziert werden kann.

Das Spielfeld, auf dem ein Sudoku gespielt wird, besteht aus einem 9x9 Gitter und damit aus 81 einzelnen Zellen. Diese Gitter wird weiterhin in je neun Zeilen, Spalten und 3x3 Blöcke aufgeteilt. Jede Aufteilung, weiterhin auch Unit genannt, besteht immer aus neun Zellen. Damit kann ein Sudoku also nochmal in 27 verschiedene Units aufgeteilt werden. Wie diese Units aussehen, wird im Folgenden gezeigt. [Sud22]

3.1.1. Zeile

Eine Zeile in einem Sudoku Spielfeld sind Zellen, die horizontal zueinander ausgerichtet sind. In der Abbildung 3.1 wurde die oberste Zeile beispielhaft markiert.

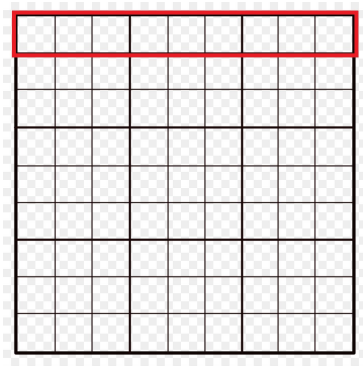


Abbildung 3.1.: Darstellung des Sudokugitters mit der Markierung einer Zeile

Auf die Zeilen wird ab jetzt und im Folgenden der Arbeit mit den Zahlen 1 bis 9 referiert.

3.1.2. Spalte

Analog zu einer Reihe sind Spalten Zellen, die vertikal zueinander ausgerichtet sind. In der Abbildung 3.2 wurde beispielhaft die erste Spalte markiert.

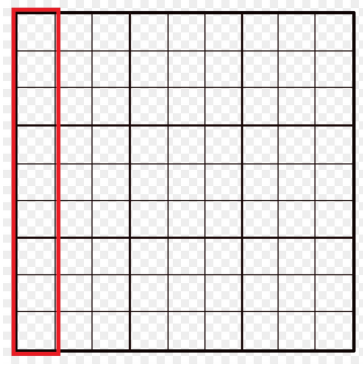


Abbildung 3.2.: Darstellung des Sudokugitters mit der Markierung einer Spalte

Die Spalten werden ähnlich zu den Zeilen von links nach rechts nicht mit Zahlen, sondern mit den Buchstaben A bis I beschrieben.

3.1.3. Box/Block

Boxen bestehen aus 3x3 Zellen. Die 3x3 Boxen sind im Sudoku Feld so angeordnet, dass jede Zelle in genau einer Box liegt und, dass es neun Boxen geben kann. In der Abbildung 3.3 wurde eine Box markiert. Wie zu erkennen ist, sind die Boxen auch im Frontend durch dickere Linien voneinander abgegrenzt.

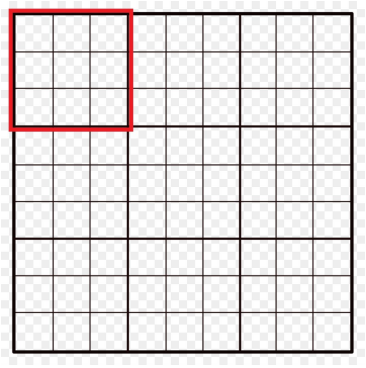


Abbildung 3.3.: Darstellung des Sudokugitters mit der Markierung einer Box

Blöcke werden durch römische Zahlen identifiziert. Dabei wird erst von links nach rechts und von dann von oben nach unten mit I bis IX nummeriert.

3.2. Regeln

Nachdem die Abgrenzungen der Units und damit die Aufteilung des Spielfelds definiert wurde, werden die Regeln auf Basis des jetzigen Wissensstandes nochmals genauer erläutert.

Die Regeln ergeben sich durch einfache Bedingungen, wenn man die Aufteilung des Spielfeldes kennt. In jeder Zeile, Spalte und Box dürfen und müssen die Zahlen von eins bis neun nur einmal vorkommen.

Am Ende kommen die Zahlen von eins bis neun also jeweils neunmal auf dem gesamten Sudoku Gitter vor, ohne dass sie sich gegenseitig sehen, also sich keine Zahl doppelt in einer Zeile, Spalte oder Box befindet.

Damit man das Sudoku Gitter ausfüllen kann und es nur eine mögliche Lösung gibt, sind zu Spielbeginn einige Ziffern vorgegeben. Mithilfe dieser Ziffern ist es möglich, mit den gerade erklärten Regeln das ganze Sudoku Gitter durch Logik und Deduktionen zu füllen. [Sud22] [Mar]

3.3. Strategien

In diesem Abschnitt wird die Bedeutung von Strategien für des Lösens eines Sudokus beschrieben. Es werden zwei verschiedene Arten von Strategien angesprochen. Für das Erkennen der Strategien wird das Hilfsmittel der Kandidaten-Notation gebraucht und daher erläutert.

Um ein Sudoku zu lösen, können verschiedene Strategien eingesetzt werden. Das Ziel einer Strategie ist es hierbei eine passende Zahl in das Sudoku einzufügen oder mögliche Kandidaten zu eliminieren. Es gibt zwei grundlegende Strategien, um Zahlen direkt einzufügen: den Hidden Single und den Naked Single. Beide Strategien können oft intuitiv und ohne eine große Erklärung gefunden und umgesetzt werden. Weitere leichte Strategien wie ein Hidden Double oder ein Naked Double, bei denen Kandidaten eliminiert werden können, sind mithilfe der Kandidaten-Notation leicht gefunden.

Insgesamt gibt es über 30 gebräuchliche Strategien, die zur Unterstützung bei schweren Sudokus angewendet werden können und noch einige weitere Modifikationen dieser. Oftmals sind anspruchsvollere Strategien für das menschliche Auge schwer zu entdecken. Wichtig bei diesen komplexeren Strategien ist eine gewissenhafte Ausführung der Kandidaten-Notation. [Mar] [Zam15, S. 9]

3.3.1. Lösungshilfen: Kandidaten-Notation

Kandidaten sind Zahlen in einer Zelle des Sudoku Gitters, die, mit dem aktuellen Informationsstand, als feste Zahl noch infrage kommen. Das Problem liegt dabei, dass meistens mehrere Zahlen in eine Zelle können. Kandidaten können durch das Eintragen von weiteren Zahlen oder das Anwenden von Strategien eliminiert werden. Wenn man dabei jede Zelle einzeln betrachtet, geben die Kandidaten keinen weiteren Nutzen. Wenn die Kandidaten jedoch über mehrerer Zellen in Bezug zueinander angeschaut werden, lassen sich Zusammenhänge erkennen. Die meisten Strategien basieren auf diesen Zusammenhängen.

4	2 8	9	7	2 6 8	1	5 6 8	5 6	3
1 3 7	3 8	5	4	6 8 9	6 9	2	6 7 9	1 7 9
1 2 7	6	1 7	5	2 8 9	3	7 8	4	1 7 9
2 3	7	8	6	1	4	9	3 5	2 5
1 6	2 3	1 6	9	5	8	4	3 7	2 7
9	5	4	3	7	2	1	8	6
5 6 7	1	3	8	4	6 7 9	5 6 7	2	5 7 9
5 6 7	4	2	1	6 9	6 7 9	3	5 6 7 9	8
8	9	6 7	2	3	5	6 7	1	4

Abbildung 3.4.: Beispiel Rätsel mit Kandidatennotation

Sobald eine Zahl eingetragen wird, müssen die Kandidaten aktualisiert werden. Wenn eine Zahl eingetragen wird, kann jeder gleichwertiger Kandidat, der diese Zahl sieht, gestrichen werden. [AV14, S. 85 f.] [Mar] [Zam15, S. 4]

4. Die Mathematik hinter Sudoku

In diesem Kapitel sollen einige mathematische Fragen über Sudokus geklärt werden. Dazu gehören Fragen zur eindeutigen Lösbarkeit und wie diese bewiesen werden kann oder wie viele mögliche fertige Sudokus es denn überhaupt gibt. Zudem wird eine mathematische Beschreibung eines Sudokus gegeben und eine algorithmische Lösungsmethode, die ohne das Anwenden von Strategien, eine Lösung für ein Sudoku findet, falls es eine Lösung gibt.

4.1. Abzählfragen

In diesem Abschnitt soll geklärt werden, wie viele mögliche Lösungen für ein vollständig ausgefülltes Sudoku Gitter existieren. Dabei werden zwei verschiedene Ansätze beschrieben. Besonders der Ansatz von Jarvis und Felgenhauer wird genauer betrachtet.

Ein trivialer Ansatz, um die Frage zu beantworten, wie viele vollständig ausgefüllten 9×9 Standard-Sudokus es gibt, könnte sein, zeilenweise von links nach rechts oder spaltenweise von oben nach unten Zahlen einzutragen. Damit würden sich pro Zeile, Spalte oder Block 9! Möglichkeiten ergeben. Daraus ergeben sich

$$(9!)^9 = 1{,}1 \cdot 10^{50} \tag{4.1}$$

Möglichkeiten, ein Sudoku Gitter auszufüllen. Von diesen ausgefüllten Gittern entsprechen aber nicht alle der Konvention, dass in den Zeilen, Reihen und Boxen jede Zahl von 1 bis 9 nur einmal vorkommen darf.

Felgenhauer und Jarvis hatten für eine einzelne Unit dieselbe Herangehensweise. Anschließend haben Sie die möglichen Lösungen von drei linear verbundenen Blöcken errechnet, wie in Abbildung 4.1 dargestellt. Für die Erklärungen werden im weiteren Verlauf exemplarisch die Blöcke I, II und III verwendet.

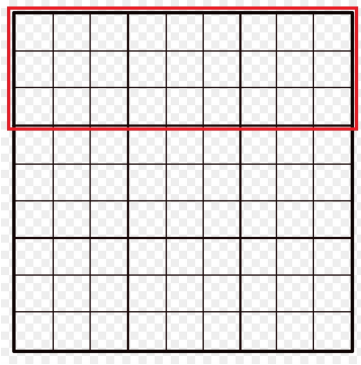


Abbildung 4.1.: Markierung von drei linear verbundenen Blöcken

Als Ausgangspunkt setzten Felgenhauer und Jarvis die Box I als ausgefüllt voraus. Zunächst berechnen sie die Anzahl von Möglichkeiten, den Rest der oberen Reihe mit zwei Sets aus jeweils drei Zahlen auszufüllen. So ein Dreier Set, in das die Zahlen eingetragen werden sollen, ist in Abbildung 4.2 markiert. Jedes Set aus drei Zahlen kann in $3! = 6$ verschiedenen Anordnungen in die oberste Zeile eingetragen werden. Dabei müssen alle sechs Sets, die die Blöcke II und III formieren, mit in Betracht gezogen werden. Dasselbe gilt für die Vertauschung der Anordnung der Sets der obersten Reihe. Dadurch ist der erste Teil $2 \cdot (3!)^6$ der folgenden Formel erklärt.

Alle anderen 18 Möglichkeiten zwei dieser dreier Sets auszufüllen verhalten sich anders. Nachdem die obere Reihe in Block II und III ausgefüllt wurde, können Zahlen in den übrigen vier Sets aufgrund von fehlenden Einschränkungen vertauscht werden. Damit haben Felgenhauer und Jarvis für den zweiten Teil der Gleichung, $18 \cdot 3 \cdot (3!)^6$ argumentiert. Insgesamt gibt es also 56 Konfigurationen, die sechs Dreier Sets weiter auszufüllen.

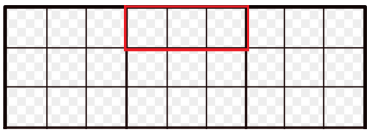


Abbildung 4.2.: Markierung eines Dreier Sets als Kombination aus Block und Zeile

Damit gibt es

$$2 \cdot (3!)^6 + 18 \cdot 3 \cdot (3!)^6 = 56 \cdot (3!)^6 = 2612736 \tag{4.2}$$

mögliche Erweiterungen einer bereits ausgefüllten Unit zu drei linear verbundenen Blöcken.

Da es für eine vollständig ausgefüllte Unit aber 9! unterschiedliche Ausführungen gibt, muss man die die Möglichkeiten eine Unit auszufüllen mit den möglichen Ergänzungen multiplizieren.

Damit gibt es

$$9! \cdot 2612736 = 948109639680 \tag{4.3}$$

Möglichkeiten, die oberen drei Reihen eines Sudokugitters auszufüllen.

Aus den 2612736 Möglichkeiten wollten Felgenhauer und Jarvis dann versuchen, die verbleibenden Blöcke des Sudoku Gitters mittels einer Brute-Force Methode aufzufüllen. Da es mit den 2612736 Möglichkeiten immer noch ein zu großer Aufwand wäre, haben sich Felgenhauer und Jarvis drei Reduktionen überlegt, um die Anzahl an Möglichkeiten die ersten drei Reihen auszufüllen zu reduzieren.

Es wurden folgende Reduktionen angewandt: die lexikographische Reduktion, die Reduktion der Permutationen und die Reduktion der Spalten. Durch diese Reduktionen werden Sudokus mit Lösungen, die sich durch Spiegeln, Drehen oder durch symmetrische Eigenschaften doppeln lassen, herausgefiltert. Aus Praktikabilitätsgründen kann in dieser Arbeit keine umfassende Übersicht der Reduktionen gegeben werden.

Nach den Reduktionen bleiben 44 Möglichkeiten, die Zeilen zwei und drei auszufüllen. Danach muss herausgefunden werden, wie viele Konfigurationen eines Sudokus es gibt, das Gitter zu vervollständigen.

Felgenhauer und Jarvis haben in ihrer Veröffentlichung von 2006 die These aufgestellt, dass es 6.670.903.752.021.072.936.960, also ca. 6,7 Trilliarden verschiedene, vollständig ausgefüllte Standard-Sudokus Rätsel gibt. Nach der Reduktion von symmetrischen Lösungen bleiben nur noch 5.5 Milliarden Lösungen. [FJ06] [AV14, S. 90 ff.]

4.2. Komplexität

Das folgende Unterkapitel gibt eine mathematische Definition eines Sudoku. Dabei wird eine allgemeine Beschreibung gegeben, die auf Sudokus, die ein Problem n-ter Ordnung sind, angewendet werden kann.

Bei Sudokus, die in den Medien veröffentlicht werden, und immer eindeutig lösbar sein müssen, kann durch logische Schlussfolgerung das Sudoku Rätsel gelöst werden. Durch die eindeutige Lösbarkeit ist es nicht notwendig Fallunterscheidungen zu treffen, also auszuprobieren.

Mathematisch gesehen ist ein Sudoku ein Problem n-ter Ordnung. Dabei ist n eine natürliche Zahl. Auf einem $N \cdot N$ Gitter, wobei $N = n^2$ gilt, sind die Zahlen 1 bis N in das Gitter einzufügen. Dabei soll in jeder $1 \cdot N$ Zeile, $N \cdot 1$ Spalte und $n \cdot n$ Block, jede der

Zahlen 1 bis N genau einmal vorkommen. N^2 Felder des Gitters können dabei vorbelegt sein. Ein $9 \cdot 9$ Sudoku ist damit ein Problem 3-ter Ordnung. [HM07]

4.3. Eindeutige Lösbarkeit

Eine wichtige Eigenschaft von Sudoku Rätseln ist die eindeutige Lösbarkeit. In diesem Abschnitt wird erklärt, was genau unter eindeutiger Lösbarkeit gemeint ist und wie viele Ziffern vorgegeben sein müssen, um das zu erreichen.

Für ein Sudoku Gitter ohne vorgegebene Ziffern gibt es 5.472.730.538 (5,5 Milliarden) unterschiedliche, richtige Lösungen. Auch wenn nur eine oder zwei Ziffern vorgegeben werden, gibt es immer noch eine sehr große Anzahl an Lösungen für dieses Sudoku.

Sudokus, die in irgendeiner Form veröffentlicht werden, sind normalerweise mit der Vorgabe einer eindeutigen Lösung erstellt. Sobald ein Sudoku nur eine korrekte Vervollständigung hat, ist es eindeutig lösbar. Daraus lässt sich folgern, dass in eindeutigen Sudokus in jede freie Zelle nur eine einzige Ziffer eingetragen werden kann, ohne die Regeln zu brechen.

Als notwendiges Kriterium für ein eindeutig lösbares Sudoku, müssen unter den vorgegebenen Zahlen mindestens acht unterschiedliche Zahlen von 1 bis 9 vorkommen. Dieses Kriterium ist gegeben durch den Fakt, dass bei nur sieben vorgegebenen unterschiedlichen Ziffern die beiden übrigen in der zugehörigen Lösung vertauscht werden können. [HM07] [AV14, S. 95 ff.]

4.3.1. Anzahl vorgegebene Ziffern für ein eindeutiges Sudoku

Es gibt die Vermutung, dass die minimale Anzahl an vorgegebenen Ziffern 17 ist. Mittels Brute-Force Methoden wurden eindeutige Sudokus mit nur 17 vorgegebenen Zahlen gesucht. Daran wurde 2011 von einem Forschungsteam von Gary McGuire geforscht. Es gibt jedoch noch keinen mathematischen Beweis für die Vermutung. Diese Vermutung basiert also hauptsächlich auf dem Generieren und Ausprobieren von vielen unterschiedlichen Sudokurätseln mit nur 17 Ziffern und einer eindeutigen Lösung. [Zei12] [AV14, S. 95 ff.]

4.4. Algorithmische Lösungsmethode: Backtracking

Um die eindeutige Lösbarkeit beweisen zu können, muss ein Sudoku Rätsel gelöst werden. In der Studienarbeit wird das Überprüfen der eindeutigen Lösbarkeit mittels der algorithmischen Lösungsmethode des Backtracking bewiesen. Dieser Algorithmus wird im Verlaufe des Abschnitts erklärt und wie damit der Beweis der eindeutigen Lösbarkeit umgesetzt werden kann.

Backtracking ist eine Problemlösungsstrategie und mit der Rekursion verwandt. Es werden alle möglichen Lösungen ausprobiert und in jedem Schritt nach einer Abbruchbedingung überprüft. Die Bedingungen an das Problem, dass mit Backtracking gelöst werden sollen, sind die Folgenden:

1. Ein Problem kann in endlich viele Teilschritte aufgeteilt werden
2. Ein Teilschritt besitzt eine Abbruchbedingung
3. Ein Teilschritt hat endlich viele Lösungsmöglichkeiten

Für jedes Element des Problems, in diesem Fall für jede freie Zelle, werden alle möglichen Zustände ausprobiert. Die Zustände sind im Fall des Sudokus die Zahlen von eins bis neun. Wenn ein Zustand zulässig ist, also in der Zeile, Reihe oder Box die Zahl nicht bereits eingetragen ist, so wird rekursiv überprüft, ob es für den aktuellen Zustand eine Lösung gibt. Wenn es diese nicht gibt, wird der vorherige Schritt rückgängig gemacht und eine neue Lösung gesucht. Damit basiert Backtracking auf dem Trial-and-Error Prinzip und versucht eine erreichte Teillösung in eine Gesamtlösung zu transferieren.

Bei z möglichen Verzweigungen jeder Teillösung und einem möglichen Verzweigungsbaum mit der Tiefe von N hat das Backtracking, sofern $z > 1$ ist, im schlechtesten Fall mit $O(z^N)$ eine exponentielle Laufzeit.

$$1 + z + z^2 + z^3 + ... + z^N \tag{4.4}$$

Wenn mittels des Backtrackings keine Lösung für das Sudoku gefunden wurde, gibt es keine Lösung. Wenn jedoch eine Lösung gefunden wurde, so ist noch nicht bewiesen, dass diese Lösung eindeutig ist. Um eine eindeutige Lösbarkeit zu garantieren, wird mittels Backtracking überprüft, dass keine weitere Lösung existiert.

Da es für das Lösen eines Sudokus keinen effizienteren Algorithmus gibt und bereits vor der Anwendung der Lösungsstrategie überprüft werden soll, ob es eine eindeutige Lösbarkeit gibt, ist dieser Ansatz am sinnvollsten. [Log14, S. 209 ff.] [Kno17]

4.4.1. Backtracking mit Brute-Force-Methode

Backtracking mit der Brute-Force Methode funktioniert wie oben beschrieben auf dem Prinzip des Backtrackings, mit dem Ausprobieren aller möglichen Fällen. Mit dem ersten freien Feld probiert man mit der 1 beginnend, ob man zu einer Lösung kommt. Als Abbruchbedingung ist das nicht singuläre Auftreten der neu eingefügten Zahl in Reihe, Zeile und Box implementiert.

Bei der Brute-Force Methode wird für jede freie Zelle von 1 bis 9 überprüft, ob diese Zahl in die Zelle eingetragen werden darf. Sobald eine Zahl eingetragen werden darf, wird das gemacht und mit der nächsten freien Zelle fortgefahren. Falls in einer Zelle die Abbruchbedingung für jede Zahl ausgelöst wurde, also keine Zahl mehr in die Zelle eingetragen werden kann, wird bei der vorherigen Zelle weiter versucht. Bis das Sudoku entweder vollständig gelöst ist, oder es keine Möglichkeiten mehr gibt und das Sudoku damit nicht lösbar ist.

Die Laufzeit des Algorithmus hängt von der Anzahl der freien Zellen ab und damit auch von der Anzahl der vorgegebenen Zahlen. Wenn viele Zahlen vorgegeben sind, dann ist die Tiefe N des Verzweigungsbaums und damit auch die Laufzeit geringer.

```
1   def solve(self, numbers=(1, SIZE + 1), board=None):
2     """
3     Solves the SudokuBoard recursively via backtracking
4     :return: False if not solvable, True if solved
5     """
6     square = self.get_empty_square(board)
7
8     if not square:
9       return True
10
11    for i in range(*numbers):
12      if self.is_valid(i, square, board=board):
13        board[square[0]][square[1]][0] = i
14
15        if self.solve(numbers=numbers, board=board):
16        return True
17
```

```
18          board[square[0]][square[1]][0] = 0
19
20      return False
```

Listing 4.1: Backtracking zum Lösen des Sudoku Rätsel

Abbruchbedingung

Die Abbruchbedingung für das Backtracking ist mit einer eigenen Funktion *is_valid* umgesetzt.

```
1   def is_valid(self, number, position, board=None):
2       ...
3       # Check row
4       for j in range(SIZE):
5           if board[position[0]][j][0] == number and j != position[1]:
6           return False
7       ...
8       # Check Box
9       box_x = (position[1] // 3) * 3
10      box_y = (position[0] // 3) * 3
11
12      for i in range(box_y, box_y + 3):
13          for j in range(box_x, box_x + 3):
14              if board[i][j][0] == number and (i, j) != position:
15              return False
16
17  return True
```

Listing 4.2: Abbruchbedingung Backtracking

Wie in dem Code implementiert, wird überprüft, ob eine *number* an einer bestimmten *position* eingesetzt werden darf, oder ob diese Zahl bereits in der entsprechenden Reihe, Zeile oder Box vorhanden ist. Wenn die Zahl gültig ist, gibt die Funktion *True* zurück, anderenfalls *False*, womit die Abbruchbedingung als erfüllt gilt. [Kno17]

4.4.2. Beweis Eindeutigkeit

Der Algorithmus des Backtrackings bricht nach dem Finden einer vollständigen Lösung für ein Sudoku ab. Damit ist bestätigt, dass es eine Lösung für das Sudoku Rätsel gibt, jedoch

nicht, dass diese Lösung eindeutig ist. Um die Eindeutigkeit eines Sudokus zu beweisen, müsste nach dem Finden einer Lösung erfolglos versucht werden, eine weitere Lösung zu finden.

Um die Eindeutigkeit eines Sudoku Rätsels zu beweisen, wird nach dem Finden einer Lösung das Backtracking nochmals von hinten begonnen. Damit ist gemeint, dass bei der ersten Lösungssuche in den freien Zellen die Zahlen von 1 bis 9 eingetragen werden und nach dem Abbruchkriterium überprüft wird. Beim Suchen nach einer zweiten Lösung beziehungsweise bei dem Beweis der Eindeutigkeit wird das Backtracking nochmals ausgeführt, dabei aber die Zahlen von 9 bis 1 in die freien Zellen eingetragen.

```
1   def is_uniquely_solvable(self):
2     temp_board = deepcopy(self.board)
3     self.solve(numbers=(9, 0, -1), board=temp_board)
4     return self.solved == temp_board
```

Listing 4.3: Überprüfen auf eindeutige Lösbarkeit

Nach dem Ausführen der beiden Backtracking Algorithmen werden die beiden gefundenen Lösungen miteinander verglichen. Wenn die dabei gefundenen Lösungen übereinstimmen, kann mit Garantie gesagt werden, dass das Sudoku eindeutig lösbar ist.

Teil II.

Implementierung

5. Frontend

Nachdem in den bisherigen Kapiteln eher die technischen und theoretischen Grundlagen diskutiert wurden, wird nun die eigentliche Implementierung des **Sudoku Helper**s erläutert.

Zunächst wird das Frontend mit allen Funktionalitäten des User-Interfaces und der daraus resultierenden User Experience, sowie die Umsetzung der einzelnen Hilfestellungen beschrieben.

Das Frontend ist letztlich alles, was der User auf der Website erfahren kann. Alles was der Nutzer nicht erleben kann, gehört zum Backend. Trotzdem ist das Frontend sehr entscheidend, denn es sollte möglichst benutzerfreundlich sein. Dazu gehört eine intuitive Nutzung der Oberfläche und im Falle des **Sudoku Helper**s das übersichtliche Anzeigen der Lösungsstrategie und die Erklärung dazu, sowie die entsprechenden Markierungen auf dem Sudoku Board.

5.1. User-Interface

Das User-Interface (UI) ist die Nutzerschnittstelle zum Backend. An dieser Stelle beeinflusst der Nutzer durch seine Handlung die Reaktion im Backend und wie das Frontend letztlich wieder auf diese Handlung reagiert. An dieser Stelle der Ausarbeitung werden die Designentscheidungen für das UI vorgestellt. Daher wird im Folgenden die HTML Struktur und das darauf basierende Responsive Design beschrieben.

Ein wichtiger Teil des UIs ist es, dem Nutzer alle Funktionalitäten übersichtlich und klar zur Verfügung zu stellen. Wenn das UI schon unübersichtlich und schlecht zu verstehen ist, dann wird auch die UX keine gute sein, egal wie gut die Software im Backend letztlich ist. Eine weitere Anforderung an das Frontend ist das Responsive Design. Die Website soll sich möglichst allen Formaten anpassen und auf verschiedenen Endgeräten entsprechend skaliert werden.

5.1.1. HTML Struktur

Die HTML Datei ist unterteilt in *<head>* und *<body>*.

Im Head werden Informationen über die Seite, wie beispielsweise der Titel oder die Stylesheets definiert. Ebenso werden die benutzten Skripts aus den externen Dateien in das Dokument eingebunden.

Im Body sind die darzustellenden Inhalte eines HTML Dokuments enthalten und weil das Produkt hauptsächlich auf einem Desktop PC genutzt wird, ist die HTML Struktur im Body darauf angepasst.

In der Abbildung 5.1 ist das Frontend mit eingezeichneter Aufteilung zu sehen. Das Sudoku Board, das NumPad und die Erläuterung der Strategie liegen jeweils nebeneinander. Das Sudoku Board wird links und in einer zur Browsergröße passenden Höhe abgebildet, damit alle Zahlen gut zu sehen sind. Rechts daneben wird das NumPad platziert, sodass zwischen den zwei Elementen, die dem Benutzer Interaktion ermöglichen, kein Freiraum liegt. Um das NumPad nicht zu verschieben, wenn ein Titel einer Strategie angezeigt wird, hat das Feld für Titel und Erklärung einen festen Platzhalter rechts neben dem NumPad.

Die Lösung ist dabei so aufgebaut, dass das Sudoku Board in einem * * liegt und das NumPad mit dem Textfeld in einem weiteren. Dieses * * von NumPad und Textfeld ist nochmal in zwei * *s unterteilt. In der folgenden Abbildung 5.1 ist die Unterteilung der Elemente nochmals visualisiert.

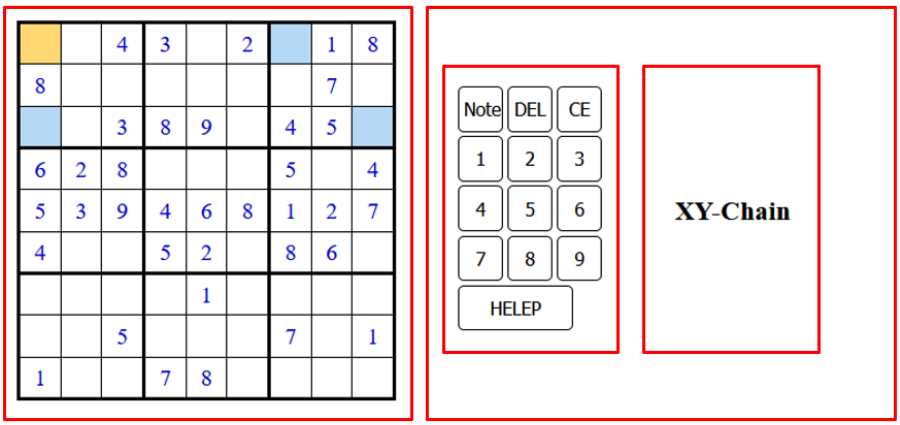


Abbildung 5.1.: HTML Struktur

5.1.2. Responsive Design

Die in der HTML Struktur beschriebenen Designentscheidungen wurden hauptsächlich durch das Responsive Design beeinflusst.

Die Elemente verhalten sich entsprechend dem Flexbox Layout-Modell. Damit werden Elemente automatisch gestreckt, wenn es mehr Platz zum Füllen gibt und sie werden gestaucht, wenn es weniger Platz gibt. Mit den Flexboxen ist es daher relativ einfach möglich, das Responsive Design umzusetzen.

Wenn das Fenster kleiner wird, dann skaliert sowohl das Board als auch das NumPad und das Textfeld mit. Falls das Fenster zu schmal wird, sodass Board und Kombination aus NumPad und Textfeld nebeneinander kein mehr Platz haben, dann wird das Element von NumPad und Textfeld unter dem Board angezeigt.

Insgesamt wird das Board so skaliert, dass der Nutzer in der horizontalen nicht scrollen muss, um das gesamte Board zu sehen. Das ist aber nur bis zu einer gewissen Breite sinnvoll, da sonst das Board zu klein wird. Diese Breite wurde basierend auf der Bildschirmbreite gängiger mobiler Endgeräte bestimmt.

Damit die Position des NumPads nicht abhängig vom Inhalt des Textfelds ist, werden die * *s des NumPads und Textfelds jeweils als eigene Flexbox behandelt. Durch diese beiden Flexboxen ist auch das Anzeigen bei schmaleren Fenstern einfacher geworden. In diesem Fall wird das NumPad und die Erläuterung nebeneinander unter dem Board angezeigt, wie in Abbildung 5.2 dargestellt. Die Flexbox von NumPad und Textfeld ist dabei genauso breit wie das Board selbst.

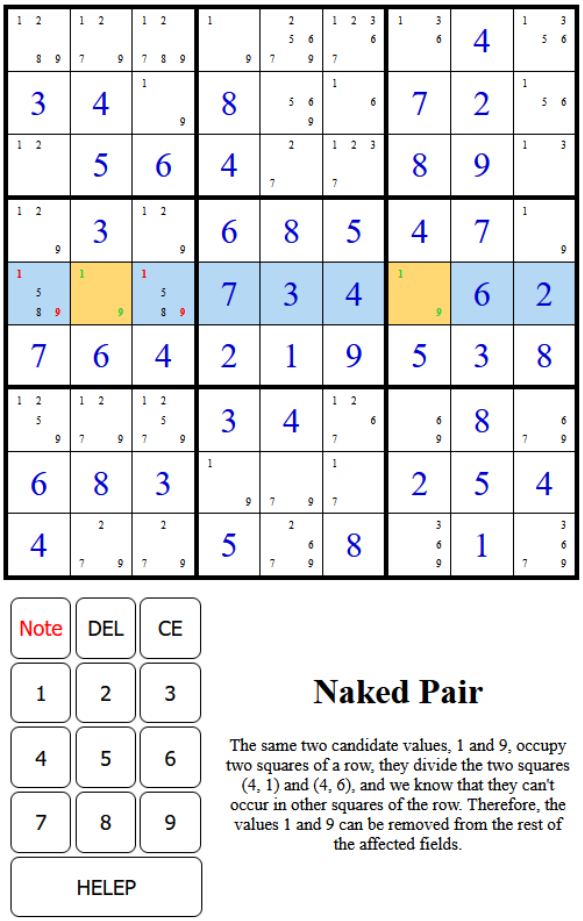


Abbildung 5.2.: Responsive Frontend

Ein weiterer Punkt, der beim Responsive Design beachtet werden muss, ist die Schriftgröße. Eine Anpassung wird deshalb bei den Zahlen und Kandidaten auf dem Board und auch bei der Größe der Buttons des NumPad und die Schrift darauf vorgenommen. Das Gleiche gilt für das Textfeld mit Lösungsstrategie und Erläuterung. Die Skalierung der Schriftgröße wird in der CSS Datei über Viewports designt. Damit die Schrift am Ende nicht zu klein ist, wird eine *min-font-size* definiert.

5.2. User-Experience

Der folgende Abschnitt ist eine Ausführung über verschiedene Funktionalitäten, die für eine gute User-Experience implementiert sind. Im Zuge dessen werden die beiden Zustände in denen sich der **Sudoku Helper** befinden kann, der User Input und die Elemente auf dem Frontend erläutert.

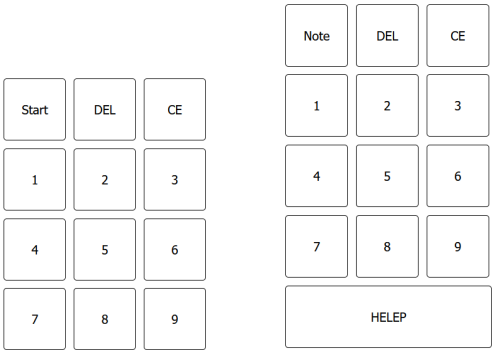
Neben einem übersichtlichen und verständlich aufgebauten UI muss noch eine gute User-Experience (UX) für ein gutes Frontend garantiert werden. Dazu gehören die Funktionalitäten der Knöpfe und eine angenehme Auswahl an Farben zur Markierung, um dem User die Nutzung der Website möglichst angenehm zu gestalten. Zum Beispiel gibt es für das Eintragen der Zahlen auf dem Sudoku verschiedene Varianten um eine gute UX zu erreichen.

5.2.1. Zustände

Der **Sudoku Helper** kann sich in zwei Zuständen befinden. Zu Beginn, oder nach dem Zurücksetzten des Boards, befindet sich die Seite in einem Anfangszustand, in dem der Benutzer das zu lösende Sudoku eingeben kann. Per Knopfdruck auf den *Start*knopf wird das Sudoku auf eindeutige Lösbarkeit geprüft. Falls das Sudoku eindeutig lösbar ist, ändert sich der Zustand zum „Spielzustand“, die bereits eingegeben Zahlen werden farblich markiert und der Nutzer kann diese nicht mehr verändern. Zudem verändert sich das NumPad, mit dem weitere Funktionalitäten des Boards freigeschaltet werden.

Es gibt auf dem UI ein NumPad mit verschiedenen Tasten. Wenn das Sudoku noch nicht gestartet wurde, unterscheidet sich die Anordnung der Knöpfe. In diesem Zustand sind in der obersten Reihe auf dem NumPad die Tasten *Start*, Delete (*Del*) und Clear everything (*CE*) zu sehen.

Beim Drücken des *Start*knopfes wird, wie bereits beschrieben, zuerst die eindeutige Lösbarkeit des Rätsels geprüft. Wenn ein Sudoku nicht eindeutig lösbar ist, dann wird das dem Nutzer angezeigt. Falls es aber eindeutig lösbar ist, wechselt der **Sudoku Helper** in den „Spielzustand“ und der *Start* Button verschwindet, beziehungsweise wird durch einen *Note* Knopf ersetzt. Zudem erscheint unter dem NumPad ein *Help* Knopf.



(a) (b)

Abbildung 5.3.: Veränderung des NumPads nach Starten

5.2.2. User Input

Der User Input kann sowohl über das auf der Website abgebildete NumPad geschehen, sowie über definierte Tasten auf der Tastatur. In der Tabelle 5.1 sind die Keys mit ihren zugeordneten Funktionalitäten aufgelistet. Die Codierung wird im Hintergrund verwendet, um herauszufinden, welche Taste gedrückt wurde.

Key	Code	Funktionalität
1..9 (Numpad)	97..105	Zahlen einfügen
1..9	49..57	Zahlen einfügen
left arrow	37	Pointer nach links bewegen
up arrow	38	Pointer nach oben bewegen
right arrow	39	Pointer nach rechts bewegen
down arrow	40	Pointer nach unten bewegen
shift	16	Notes anzeigen
space	32	Zelle markieren
tab	9	Pointer zur nächsten Zelle bewegen
delete	46	Löschen der markierten Zellen
backspace	8	Löschen der markierten Zellen

Tabelle 5.1.: Funktionalitäten der Tastatur

Mit der Taste *Clear everything (CE)* wird das komplette Sudoku Board zurückgesetzt. Dabei werden alle Zahlen auf dem Board gelöscht und der Zustand des Boards in den Anfangszustand zurückgesetzt. Mit der *Del* Taste wird in den ausgewählten Zellen der Eintrag gelöscht. Über die *Note* Taste wird der aktuelle Stand der Kandidaten ein- oder ausgeblendet. Durch das mehrmalige Drücken des *Help* Knopfes werden die abgestuften Hilfestellungen angezeigt.

5.2.3. Funktionalität

Auf dem Sudoku Board befindet sich eine markierte Zelle, im folgenden auch Cursor genannt. Diesen Cursor kann man mit den Pfeiltasten und der Shift Taste bewegen. Wenn eine Zahl eingegeben wird, dann wird diese in der Zelle des Cursors eingetragen. Sobald die Zahl eingetragen wird, springt der Cursor in die nächste Zelle.

Dem Nutzer ist es zudem möglich, durch Anklicken von Zellen mehrere Zellen auszuwählen. Eine weitere Möglichkeit Zellen auszuwählen, bietet die Leertaste auf dessen Knopfdruck die Zelle des Cursors ausgewählt wird. Die ausgewählten Zellen sind dann grau gefärbt. Wenn man die Markierung einer Zelle wieder rückgängig machen möchte, ist dies durch nochmaliges auswählen möglich. Für den Fall, dass Zellen markiert sind, wird die einzutragende Zahl in alle grau hinterlegten Zellen eingefügt.

In der Abbildung 5.4 sieht man den Cursor ganz oben links auf der ersten Zelle (A, 1). Sowie einige ausgewählte Zellen.

2	6							
1			8					
			9	6	2			
			3			7		
				5				8
		2	7	4			9	
4							2	
7		3			6			
				3			8	1

Abbildung 5.4.: Abbildung der möglichen Markierungsvarianten

Neben dem Löschen einer Zahl mittels des *Del* Knopfes ist es zudem möglich eine Zahl durch nochmaliges Eintragen zu löschen. Wenn man eine andere Zahl eintragen möchte, dann kann man die vorherige Zahl überschreiben, ohne dass ein vorheriges Löschen notwendig ist.

Mit dem *Note* Knopf können Kandidaten angezeigt oder ausgeblendet werden. Wenn zu einem Zeitpunkt, indem die Kandidaten angezeigt werden, eine inkorrekte Zahl eingetragen wird, so wird diese rot markiert.

5.3. Abstufungen der Hilfestellungen

Die Anforderung an die verschiedenen Abstufungen der Hilfestellung wird über das Frontend umgesetzt. Dafür werden im Backend die relevanten Informationen gesammelt. Im Folgenden wird die Abstufung und Umsetzung der Hilfestellungen erläutert und anhand von Beispielen gezeigt. Durch das wiederholte Drücken des *Help* Knopfes, werden die Informationen über die Strategie durch die nächste Hilfestellung erweitert.

Falls der Benutzer sich zu einem Zeitpunkt während einer Hilfestellung dazu entscheiden sollte, selbst mit dem Lösen fortzufahren, wird der Fortschritt der Hilfestellung gelöscht und die Markierungen zurückgesetzt.

5.3.1. Erste Hilfestellung

Die erste Hilfestellung besteht daraus, die anzuwendende Technik auf dem Frontend anzuzeigen und relevante Zellen zu markieren. In dem Codebeispiel 5.1 ist die Umsetzung dafür beschrieben.

```
1  socket.on('help0', function (technique_result) {
2    $("#technique-name").text(technique_result['name']);
3    $("#technique-explanation").text("");
4    colorCells(technique_result['primaryCells'], 'rgb(255,216,115)');
5    colorCells(technique_result['secondaryCells'], 'rgb(181,216,244)');
6  });
```

Listing 5.1: Erste Hilfestellung

In das Element der CSS-Klasse *technique-name* wird der Name der Technik als Text gesetzt. Das vorgesehene Feld für die Erklärung wird zurückgesetzt, für den Fall, dass die letzte Erklärung noch angezeigt wird. Über *colorCells* können betroffene Zellen auf dem Board in verschiedenen Farben markiert werden. Diese Markierungen und das Anzeigen des Techniknamens sind in Abbildung 5.5 dargestellt. Die Zellen werden unterschiedlich eingefärbt entsprechend ihrer Bedeutung für eine Lösungstechnik. In den meisten Fällen haben die *primaryCells* (gelb) eine besondere Bedeutung für eine Technik und sind darauf ausgelegt zuerst aufzufallen. Die *secondaryCells* (blau) weisen dabei eher auf den Grund hin, weshalb eine Technik angewendet werden kann oder in welchen Units die Technik gefunden wurde.

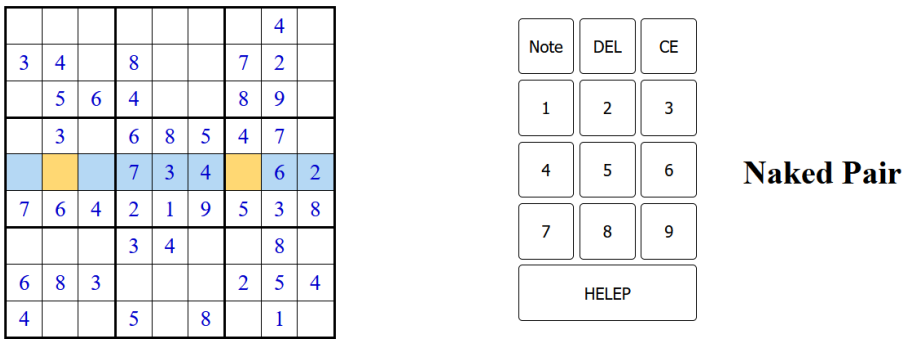


Abbildung 5.5.: Darstellung der ersten Hilfestellung

5.3.2. Zweite Hilfestellung

Mit der zweiten Hilfestellung werden relevante Kandidaten markiert. Im Falle der Kandidaten gibt es erneut zwei unterschiedliche Markierungen. In den Zeilen 2-5 wird sichergestellt, dass die Kandidaten angezeigt werden.

```
1  socket.on('help1', function (technique_result) {
2    if (!candidatesVisible) {
3      candidatesVisible = !candidatesVisible;
4      $('#candidate').css('color', 'red');
5    }
6    colorCandidates(technique_result['crossOuts'], 'red');
7    colorCandidates(technique_result['highlights'], 'limegreen');
8  });
```

Listing 5.2: Zweite Hilfestellung

Kandidaten, die für eine Lösungstechnik Relevanz haben, können mittels der Funktion *colorCandidates* markiert werden. Kandidaten, die aufgrund der Technik herausgestrichen werden können, werden auf dem Sudokuboard rot markiert. Kandidaten, auf denen eine Technik aufbaut, werden grün hervorgehoben. Die Umsetzung wird in dem Codebeispiel 5.2 beschrieben und das Ergebnis in der Abbildung 5.6 dargestellt.

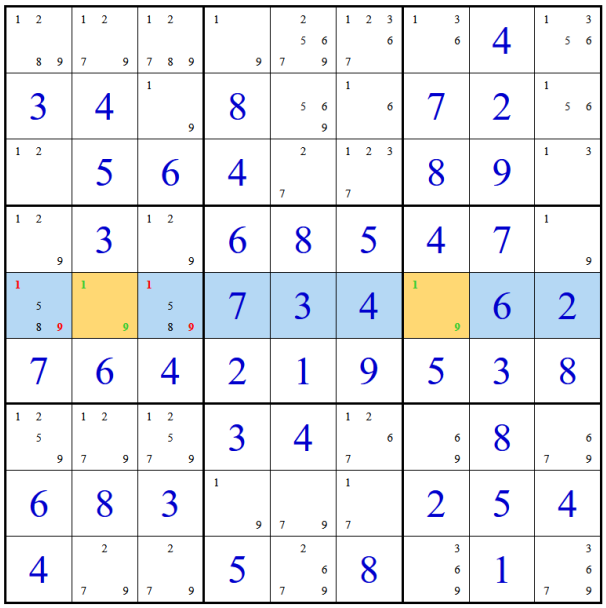


Abbildung 5.6.: Darstellung der zweiten Hilfestellung

5.3.3. Dritte Hilfestellung

Die dritte Hilfestellung besteht aus einer Erklärung der Technik anhand des aktuellen Sudokus. Die Erläuterung wird dabei unter dem Techniknamen in der CSS-Klasse *technique-explanation* in einem <p> Element angezeigt. Diese Hilfestellung wurde analog zu dem Technikname in dem Codebeispiel 5.1 umgesetzt.

5.3.4. Vierte Hilfestellung

Nachdem das Programm dem Nutzer über mehrere Schritte geholfen hat, die Technik zu finden und gegebenenfalls nachzuvollziehen, wird mit der letzten Hilfestellungen die Technik ausgeführt. Rot eingefärbten Kandidaten vom Board entfernt. Außerdem werden alle Markierungen mit der Funktion *resetCellColor()*, wie in dem Codebeispiel 5.3 gezeigt, zurückgesetzt und der Techniktitel und die Beschreibung entfernt.

```
1  socket.on('help3', function () {
2    if (!candidatesVisible) {
3      candidatesVisible = !candidatesVisible;
4      $('#candidate').css('color', 'red');
5    }
6    $("#technique-name").text("");
7    $("#technique-explanation").text("");
8    resetCellColor();
9  });
```

Listing 5.3: Vierte Hilfestellung

5.4. Weitere Funktionalitäten

In diesem Abschnitt werden zwei weitere Situationen und die Reaktion des Programms darauf beleuchtet. Zum einen wird die Reaktion beschrieben, wenn der User das Board korrekt beendet hat und zum anderen was passiert, wenn der **Sudoku Helper** keine der umgesetzten Strategien erfolgreich anwenden kann.

5.4.1. Fertig gelöstes Sudoku

Dem Nutzer wird, sobald alle Zellen auf dem Board korrekt ausgefüllt wurden, über das Textfeld angezeigt, dass er das Sudoku korrekt gelöst hat. In dem Textfeld der CSS-Klasse *technique-name* erscheint der Schriftzug *Victory* und als Erklärung in *technique-explanation* die Nachricht *The Sudoku is correctly solved!*.

5.4.2. Keine Technik anwendbar

Für den Fall, dass keine Strategie auf das Board angewendet werden kann, dann wird sich beim Benutzer über das Textfeld der CSS-Klasse *technique-name* entschuldigt, durch die Nachricht *Sorry*. Im Feld der *technique-explanation* erscheint die Erklärung *No suitable technique found!*.

6. Backend

Im Backend des **Sudoku Helper**s findet die Datenverarbeitung statt. Das Backend stellt das Gegenstück zum Frontend dar. In diesem Kapitel wird die Programmarchitektur beschrieben und die wichtigsten Klassen des Backends vorgestellt. Hier werden auch alle umgesetzten Lösungstechniken aufgezählt und bereits einen kleinen Ausblick gegeben welche Strategien noch weiter implementiert werden könnten. Danach wird die Umsetzung der Strategien beispielhaft an zwei Strategien erklärt.

6.1. Programmarchitektur

Wie in Kapitel 2.4.1 erwähnt, ist die Programmarchitektur des **Sudoku Helper** in drei Komponenten unterteilt, um eine geringe Kopplung und eine hohe Skalierbarkeit zu erreichen.

6.1.1. Model

Aufgrund einer geringen Menge an Daten wird keine Anbindung an eine Datenbank benötigt. Bei den Daten selbst handelt es sich nur um teilweise verschachtelte Listen, die in der *SudokuBoard* Klasse gehalten werden. Für die Manipulation dieser Daten verfügt die Klasse *SudokuBoard* über einige Funktionen, die dem Controller eine Verarbeitung und Nutzung der Daten ermöglicht.

Erweitert wird das Model durch die Implementierung der Lösungsstrategien und deren benötigten Komponenten, die das Anwenden der Strategien ermöglichen. Die Strategien werden über den Controller aufgerufen.

6.1.2. View

Die View ist für die Darstellung der Daten und für das Empfangen der Nutzereingaben zuständig. Sie dient dem Nutzer als Steuerung, welcher über die im vorherigen Kapitel 5.2.3 beschriebenen Funktionalitäten das Programm beeinflussen kann. Die Eingaben des Benutzers werden an den Controller weitergegeben, im Gegenzug liefert der Controller Informationen über die Daten, die dargestellt werden sollen.

Die Struktur der Darstellung ist vorgegeben durch eine HTML Datei, welche mittels einer CSS Datei gestaltet und formatiert wird. In der *client.js* Datei wird der Client initialisiert, die Verbindung zum Server aufgebaut. Sie dient zur Kommunikation mit dem Controller. Die *style.js* Datei dient zur DOM Manipulation sowie zur Änderung CSS Eigenschaften, um die Repräsentation für den Nutzer anzupassen.

6.1.3. Controller

Die Datei *app.py* repräsentiert den Controller, welcher als zentrale Schnittstelle dient. Er startet die Applikation durch das Starten des Servers und definiert einige Endpunkte für den Client. Alle Aktionen werden durch den Benutzer ausgelöst, wobei die von der View empfangenen Befehle überprüft und entsprechend die Daten an das Model weitergeleitet werden. Im Spezialfall des Betätigens der Hilfefunktion wird die oben erwähnte Erweiterung des Models der Lösungsstrategien gebraucht. Die entsprechenden Daten zur Darstellung der Hilfefunktion werden der View übergeben, für den letzten Hilfsschritt, dem Anwenden der Strategie, werden die Informationen dem Model übergeben. Die Antwort des Models wird über den Controller der View übergeben und die Darstellung aktualisiert.

6.2. Serverseite

Um die **Sudoku Helper** Anwendung zu realisieren, müssen Server und Client miteinander kommunizieren. Im Falle von standardmäßigen HTTP Requests, die synchron verlaufen wird die Seite bei jedem Request neu geladen. Dieses Verhalten beeinflusst die UX negativ, daher sollte von einer synchronen Übertragung abgesehen werden.

Eine naheliegende Alternative ist Asynchronous JavaScript and XML (AJAX), welches ein Konzept der asynchronen Datenübertragung darstellt und eine Datenübertragung ohne Neuladen der Seite ermöglichen sollte. Dennoch bieten WebSockets einige Vorteile gegenüber AJAX. WebSockets werden benutzt, um full-duplex Kommunikation zwischen Server und

Client zu definieren. Sie konzentrieren sich auf echte Parallelität und Leistungsoptimierung. Wie in Abbildung 6.1 zu erkennen ist, lassen sich über WebSockets dauerhafte Verbindungen herstellen, wobei Overhead eingespart werden kann.

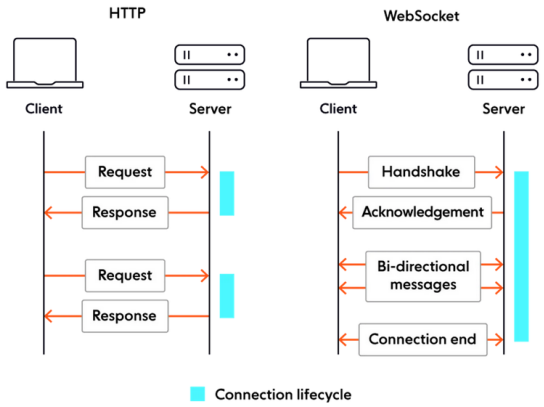


Abbildung 6.1.: HTTP vs. WebSocket [PLA]

In jedem Fall lassen sich mit einer WebSocket-Verbindung Bandbreite und CPU Zyklen einsparen. Für die Studienarbeit wird für die Kommunikation Serverseitig die Flask Erweiterung Flask-SocketIO verwendet.

6.2.1. SocketIO

Socket.IO ist eine Bibliothek, die eine bidirektionale und ereignisbasierte Kommunikation mit niedriger Latenz zwischen einem Client und einem Server ermöglicht. Die Hauptidee hinter Socket.IO ist, dass beliebige Ereignisse mit beliebigen Daten gesendet und empfangen werden können. Es baut auf dem WebSocket Protokoll auf und erweitert es mit Absicherungen wie die automatische Wiederverbindung von Clients und Verwendung von HTTP long-polling im Falle von Kompatibilitätsproblemen.

6.3. SudokuBoard

Ein wichtiger Part des Backends ist die Klasse *SudokuBoard*. In dieser Klasse wird das Sudoku Board erstellt und eine Datenstruktur erschaffen, in der die Informationen gespeichert werden. Aufgrund dieser Datenstruktur werden die Lösungsstrategien angewendet. In diesem Abschnitt wird die Datenstruktur erläutert und einige Funktionalitäten der *SudokuBoard*-Klasse vorgestellt.

Die Zahlen und Kandidaten werden in unterschiedlichen Listen gespeichert. Das Board besteht aus einer zweidimensionalen List und die Kandidatennotation wurde mithilfe einer dreidimensionalen Liste umgesetzt.

6.3.1. Datenstruktur des Sudoku Boards

Die Zellenidentifikation ist in Abbildung 6.2 dargestellt. Der Inhalt der Zellen wird in einer Zweidimensionalen Liste gespeichert. Eine Null steht dabei für eine leere Zelle. Mit dem ersten Index wird eine Zeile ausgewählt und mit dem zweiten Index die Spalte.

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8

Abbildung 6.2.: Struktur des Sudokuboards und Zellenidentifikation [Zam15]

Wenn noch kein Board initialisiert wurde, dann wird die Datenstruktur erstellt und das gesamte Board mit Nullen befüllt. Diese Funktion ist in dem Codebeispiel 6.1

```
1  def init_board(self):
2    self.board = []
3    for i in range(SIZE):
4      self.board.append([])
5      for j in range(SIZE):
6        self.board[i].append(0)
```

Listing 6.1: Initalisierung des Boards

Die Kandidaten sind in einer dreidimensionalen Liste gespeichert. Mit den ersten beiden Indizes werden, genau wie auf dem Board, die Zellen identifiziert. Die dritte Liste gibt Informationen über die Kandidaten in der entsprechenden Zelle. Die Liste hat eine Länge von neun, wobei jeder Eintrag entweder eine 0 oder eine 1 ist. Eine 1 bedeutet, dass ein Kandidat in einer Zelle ist. Der nullte Index steht dabei für den Kandidaten eins.

```
1  def init_candidates(self):
2    for i in range(SIZE):
3      self.candidates.append([])
4      for j in range(SIZE):
5        self.candidates[i].append([1] * SIZE)
```

Listing 6.2: Initalisierung der Kandidaten

Da in einem leeren Sudoku Board, ohne Zahlenwerte, jede Zelle jeden Wert annehmen kann, werden zunächst alle Kandidaten gesetzt. Die Initialisierung lässt sich im Code 6.2 nachvollziehen. Sobald Zahlen eingetragen wurden und die Kandidaten angefragt werden, aktualisieren sich die Kandidatenwerte. Um bereits angewandte Strategien, die Kandidaten eliminiert haben, nicht rückgängig zu machen, werden Kandidaten ausschließlich entfernt.

6.3.2. Funktionalität

Neben der Initialisierung des Boards und der Kandidaten gehört das Prüfen der eindeutigen Lösbarkeit eines Sudoku Rätsel zu den wichtigsten Funktionalitäten der *SudokuBoard*-Klasse. Die Umsetzung dieser Funktionen wurde in dem Abschnitt 4.3 und dem darauf folgendem bereits vorgestellt. Eine weiter wichtige Funktionalität dieser Klasse ist das updaten des Boards und der Kandidaten, wenn Zahlen eingetragen sind. Auf diesen Datenstrukturen bauen alle Umsetzungen der Lösungstechniken auf.

Des Weiteren findet im *SudokuBoard* die Umsetzung der Strategien statt, da es die Datenrepräsentation des Sudoku Boards beinhaltet. Dafür wurden die Funktionen *remove_candidates(self, cross_outs)* und *update_numbers(self, number, cells)* implementiert. Hier wird eine weitere Unterscheidung zwischen Kandidaten und Zahlen getroffen. Kandidaten lassen sich nur entfernen, während es bei Zahlen sowohl das Löschen als auch das Hinzufügen möglich ist. Die Funktion des Löschens einer Zahl ist zwar umgesetzt, wird aber nur von Benutzern eingesetzt, da keine Strategie eine Zahl löschen kann.

6.4. Technique Manager

In der Datei *technique_manager.py* wird das Aufrufen der Strategien geregelt.

Alle Lösungsstrategien sind als Klassenrepräsentation der umgesetzten Technik in einer Liste gespeichert. Weil die Techniken über eine abstrakte Klasse implementiert wurden,

können alle Objekte unabhängig von der konkreten Implementierung im Technique Manager aufgerufen werden. Es wird über die Liste iteriert und die jeweilige Klasse mit dem aktuellen Sudoku Board der Kandidaten initialisiert. Danach wird die Methode *execute_technique()* für die Lösungstechnik ausgeführt. Der Rückgabewert dieser Methode ist *True*, wenn eine Technik anwendbar ist. In diesem Fall wird die Methode *get_result()* aufgerufen, welches ein Dictionary zurückgibt, in dem alle benötigten Informationen für das Umsetzten der Hilfestellungen einer Lösungsstrategie enthalten sind. Wenn eine Technik nicht erfolgreich ist, dann wird mit der nächsten Lösungsstrategie aus der Liste weiter gemacht.

Die Lösungsstrategien sind nach Schwierigkeitsgrad sortiert, wobei sie immer komplexer werden. Bei der Reihenfolge wurde sich bei Martin Simon und seiner Website orientiert. [Mar]

```
1  def try_techniques(board, candidates):
2    for tech in techniques:
3      technique = tech(board, candidates)
4      successful = technique.execute_technique()
5      if successful:
6        return technique.get_result()
7    return False
```

Listing 6.3: Finden einer anwendbaren Lösungstechnik

6.5. Abstrakte Klasse SolvingTechniques

Wie im vorherigen Abschnitt des Technique Manager schon erwähnt, gibt es eine abstrakte Klasse namens *SolvingTechniques*, deren Funktionsweise und Vorteile im weiteren Verlauf des Abschnittes erläutert wird.

In Python können abstrakte Klassen erstellt werden, indem sie von dem ABC-Modul erben. Mit einer abstrakten Klasse kann ein gemeinsames Verhalten definiert werden. Dieses Verhalten wird von mehreren Unterklassen geerbt. Der Unterschied zwischen Klassen und abstrakten Klassen ist, dass von abstrakten Klassen keine Objekte erstellt werden können. Der Vorteil einer abstrakten Basisklasse ist die Nutzung mehrerer Unterklassen an einer gemeinsamen Programmierschnittstelle, in diesem Fall in dem *TechniqueManager*. Zudem wird das gesamte Projekt durch das Einführen einer abstrakten Klasse übersichtlicher, da alle Unterklassen von dieser einen Klasse erben.

Diese abstrakte Klasse macht es dem Technique Manager möglich, alle Klassen der Lösungstechniken auf dieselbe Weise zu instanziieren und zu verwenden. Mithilfe der abstrakten Klasse wird die Struktur der Lösungstechniken verallgemeinert, ohne dass jede Methode vollständig implementiert ist.

Neben abstrakten Methoden sind noch weitere Hilfsfunktionen in dieser Klasse implementiert, die in mehreren Unterklassen verwendet werden können. Wenn eine Klasse von einer abstrakten Klasse abgeleitet ist, so müssen alle abstrakten Methoden implementiert und überschrieben werden.

6.5.1. Abstrakte Methoden

Die Verallgemeinerung der Lösungstechniken wird über abstrakte Methoden umgesetzt. Diese Methoden bieten die Standardfunktionalitäten der Basisklassen und müssen, wie schon erwähnt, implementiert werden.

In der abstrakten Klasse *SolvingTechniques* sind unter anderem die drei verschiedenen abstrakten Methoden *execute_technique(self)*, *configure_highlighting(self)* und *update_explanation(self)* deklariert. Diese haben in der abstrakten Klasse selbst keine Implementierung, werden aber in jeder erbenden Klasse der Lösungstechniken implementiert.

Mithilfe dieser drei abstrakten Methoden können alle Lösungstechniken übersichtlich implementiert werden. Nach Ausführen dieser Methoden wurde die Lösungstechnik auf das Sudoku angewendet. Dabei wurden verschiedene Informationen in Variablen hinterlegt, die für den User mit der weiteren Benutzung der Hilfestellungen relevant sind. Die Funktion der einzelnen Methoden wird in den folgenden Abschnitten erläutert. Im Zuge dessen werden auch die für das Frontend bereits erwähnten und wichtigen Variablen vorstellt, sowie der Grund des Erstellens.

execute_technique(self)

In der Methode *execute_technique(self)* werden die Algorithmen für die spezifischen Techniken implementiert. Dabei wird zunächst nur überprüft, inwiefern eine Technik anwendbar ist.

Wenn ein Algorithmus ein Muster für eine Technik erkennt und in der nächsten Methode Kandidaten zum Rausstreichen gefunden werden, wird der Wert *True* zurückgegeben. Wenn keine passende Konstellation gefunden wird oder die Technik keine Auswirkung auf das aktuelle Board hat, wird *False* zurückgegeben.

configure_highlighting(self)

Wenn ein Algorithmus eine Technik gefunden hat, dann wird in *execute_technique(self)* die abstrakte Methode *configure_highlighting(self)* aufgerufen.

In dieser Methode werden die spezifischen Zellen und Kandidaten, in denen die Technik gefunden wurde, markiert. Dafür wurden in *SolvingTechniques* Klassenvariablen angelegt.

- *self.cross_outs = []*
- *self.highlights = []*
- *self.primary_cells = []*
- *self.secondary_cells = []*

Diese Variablen wurden schon im Frontend im Abschnitt 5.3 erwähnt und deren Bedeutungen beschrieben. In den *self.cross_outs* werden Kandidaten bestimmter Zellen gespeichert, die laut Strategie entfernt werden können und in den *self.highlights* werden Kandidaten aus bestimmten Zellen gespeichert, die eine besondere Bedeutung für die Strategie haben. Die *self.primary_cells* und *self.secondary_cells* enthalten nur Zellen, die für die Strategie relevant sind. Dabei sind die *self.primary_cells* tendenziell relevanter für Strategien und werden deutlicher hervorgehoben im Gegensatz zu den *self.secondary_cells*. In den meisten Fällen lässt sich anhand der Variable *self.cross_outs* erkennen, ob das Ausführen einer Strategie Auswirkung auf das Board hätte. In Sonderfällen, je nach Strategie, muss *self.highlights* in Betracht gezogen werden.

update_explanation(self)

In *update_explanation(self)* wird eine Erklärung der Strategie gegeben. Diese Erläuterung wird in der Klassenvariable *self.explanation* gespeichert. Bei der Erklärung wird darauf geachtet, die Strategie an dem gefundenen Beispiel zu erläutern und den Grund zu nennen warum man diese Technik anwenden kann.

6.5.2. Hilfsfunktionen

In der abstrakten Klasse gibt es zusätzlich zu den abstrakten Methoden noch weitere Hilfsfunktionen. Diese Funktionen können in den Unterklassen aufgerufen werden, um die Implementierung zu erleichtern. Zu den wichtigsten Hilfsmethoden gehört *get_influential_cells_unit(cell, unit)*. Diese Methode gibt zu einer Zelle und einer Unit, alle Zellen, die von der übergebenen Zelle in der spezifischen Unit gesehen werden, zurück. Für viele Strategien dient diese Funktion als Ausgangspunkt um bestimmte Muster auf dem Board zu erkennen.

Eine weitere sehr wichtige Methode ist *get_result(self)*. In dieser Methode werden die in *configure_highlighting(self)* gesammelten Informationen, der Technikname und die in der Methode *update_explanation(self)* angepasste Beschreibung in einem Dictionary gesammelt und zurückgegeben.

6.6. Lösungsstrategien

In diesem Abschnitt wird ein Überblick über die implementierten Strategien gegeben. Die Lösungsstrategien sind auf der Website von Simon Martin [Mar] betitelt und beschrieben. Anhand dieser Beschreibungen wurden einige Lösungsstrategien implementiert. Zunächst wird ein Überblick über die umgesetzten Strategien gegeben, bevor ausblickend die Lösungsstrategien, die noch nicht implementiert wurden, aufgezählt werden. Es gibt noch weitere Strategien, die auf der Website nicht beschrieben werden. Diese werden im Folgenden aber nicht weiter beachtet.

6.6.1. Umgesetzte Lösungsstrategien

Da in der Ausarbeitung nur exemplarisch auf einige wenige Lösungsstrategien eingegangen wird, sind in der Tabelle 6.1 alle implementierten Lösungsstrategien aufgezählt.

Technikname	Technikname engl.	Technikname	Technikname engl.
Versteckter Single	HiddenSingle	Nackter Single	Naked Single
Verstecktes Paar	Hidden Pair	Nacktes Paar	Naked Pair
Versteckter Dreier	Hidden Triple	Nackter Dreier	Naked Triple
Versteckter Vierer	Hidden Foursome	Nackter Vierer	Naked Foursome
Reihe-Block-Check	Line-Block-Interaction	Block-Reihe-Check	Block-Line-Interaction
X-Wing	X-Wing	Steinbutt	Turbot
Drittes Auge	Third Eye	Wolkenkratzer	Skyscraper
Schwertfisch	Swordfish	Drachen	Dragon
Verbotenes Rechteck Typ 1	Forbbiden Rectangle Type 1	Verbotenes Rechteck Typ 2	Forbbiden Rectangle Type 2
Verbotenes Rechteck Typ 3	Forbidden Rectangle Type 3	Verbotenes Rechteck Typ 4	Forbidden Rectangle Type 4
XY-Wing	XY-Wing	XYZ-Wing	XYZ-Wing
X-Kette	X-Chain	XY-Kette	XY-Chain
Schwertfisch mit Flosse	Swordfish with fin	Doppelkette	Double Chain

Tabelle 6.1.: Aufzählung aller implementierten Lösungsstrategien mit deutschem und englischem Bezeichner

6.6.2. Weitere Lösungsstrategien

In der Studienarbeit wurden nicht alle auf der Website [Mar] vorgestellten Lösungstechniken umgesetzt. Weitere Strategien, die noch implementiert werden könnten, sind die folgenden:

- Erweiterter Block-Reihe-Check (BRC)
- W-Wing
- Geklonte Paare
- Leeres Rechteck
- Forcing Chain

Der Erweiterte BRC wurde nicht umgesetzt, weil es eine Kombination aus dem BRC und einem Versteckten Single ist. Außerdem wurde die Forcing Chain nicht implementiert, weil sie von der XY-Kette beinhaltet wird.

6.7. Beispielhafte Umsetzung von Strategien

Die Lösungsstrategien können zwei unterschiedliche Wirkungen auf ein Sudoku Rätsel haben. Entweder es ist möglich eine Zahl einzufügen oder es können Kandidaten aus verschiedenen Gründen eliminiert werden. Diese beiden Wirkungen werden im Backend unterschiedlich behandelt. In den nächsten zwei Unterkapiteln wird die Funktionsweise und die Unterschiede beider Varianten anhand zweier Beispiele erklärt.

Für beide Umsetzungen wird die abstrakte Klasse *SolvingTechniques* genutzt.

6.7.1. Zahl einfügen

Der Fall, dass aufgrund einer Lösungstechnik direkt eine Zahl in das Sudoku Gitter eingetragen werden kann, gibt es in den umgesetzten Techniken nur in drei Fällen. Die anderen Techniken führen lediglich zu dem Eliminieren von Kandidaten, woraufhin eventuell eine Zahl eingefügt werden kann.

- Versteckter Single
- Nackter Single
- Drittes Auge

Für das Beispiel wird die Strategie des Nackten Singles erklärt. Bei einem Nackten Single steht in einer Zelle nur noch ein einziger Kandidat. Da in jeder Zelle eine Zahl stehen muss, kann diese damit in die Zelle eingetragen werden.

Sobald auf dem Numpad der *Help* Button das erste Mal gedrückt wird, wird, nach dem Aktualisieren der Kandidaten, die Funktion *try_techniques()* im *TechniqueManager* ausgeführt. Der dazugehörige Vorgang des Controllers ist im Code 6.4 abgebildet.

```
1 @socketio.on('help')
2 def help():
3   global help_step
4   global technique_result
5   ...
6   if help_step == 0:
7     sudoku.update_candidates()
8     technique_result = technique_manager.try_techniques(sudoku.board,
          sudoku.candidates)
9     if not technique_result:
```

```
10        print("No suitable technique found!")
11        return
12    emit('help0', {'name': technique_result['name'], 'primaryCells':
          technique_result['primary_cells'], 'secondaryCells':
          technique_result['secondary_cells']})
```

Listing 6.4: Serverseitig help

Zunächst wird über die Liste der Techniken im *TechniqueManager* iteriert. An der Stelle des Nackten Singles wird bei passenden Voraussetzungen von der abstrakten Methode *execute_technique(self) True* zurückgegeben. Die Funktion ist in dem Codebeispiel 6.5 abgebildet. Der Algorithmus iteriert über alle Zellen auf dem Board. Wenn in einer Zelle eine Zahl steht, die nicht Null ist, das heißt bereits eine Zahl eingetragen ist, dann wird diese übersprungen. Da ein Kandidat über eine 1 dargestellt wird, lässt sich über die Summe aller Kandidaten in dieser Zelle auf die Anzahl an Kandidaten zurückschließen. Durch die Bedingung in Zeile 6 werden also weiterhin alle Zellen übersprungen, in denen mehr als ein Kandidat steht. Wenn eine Zelle nur einen Kandidaten hat, dann wird in dieser Zelle ein Nackter Single erkannt und zu den *self.primary_cells* hinzugefügt. Als Nächstes wird die abstrakte Funktion *configure_highlighting(self)* aufgerufen, in der das weitere Highlighting konfiguriert wird.

```
1   def execute_technique(self):
2     for i in range(9):
3       for j in range(9):
4         if self.board[i][j] != 0:
5           continue
6         if sum(self.candidates[i][j]) != 1:
7           continue
8         self.primary_cells = [(i, j)]
9         self.configure_highlighting()
10        return True
11    return False
```

Listing 6.5: Nackter Single

Auf das erfolgreiche Erkennen einer Strategie wird die Hilfsmethode *get_result()* für das Objekt des Nackten Singles aufgerufen. Diese Ergebnisse sind nun in der globalen Variable *technique_result* gespeichert. Für jede weitere Hilfestellung wird der *help_step* inkrementiert. Über *emit* kann ein Event, dessen Callback in diesem Fall bei 'help0' definiert ist, ausgeführt werden. Die 'help0' Funktion wurde im Frontend in dem Abschnitt 5.3 vorgestellt. Mit dem *emit* werden zudem die gebrauchten Informationen übergeben.

Die nächsten beiden Hilfestellungen funktionieren nach dem Selben Prinzip. Erst bei der letzten Hilfestellung, dem Anwenden der Strategie, wird die einzig nötige Unterscheidung getroffen. Wie in dem Codebeispiel 6.6 gezeigt, wird der Name der Technik abgefragt. Wenn es eine der drei Techniken ist, bei denen eine Zahl auf dem Gitter eingefügt wird, dann wird die Funktion *new_number()* aufgerufen. Dieser Funktion wird die Zelle und die Zahl übergeben. Da die einzutragende Zahl zuvor grün hervorgehoben werden soll, lassen sich die entsprechenden Werte aus der Variable *self.highlights* auslesen.

```
1  elif help_step == 3:
2    if technique_result['name'] in ['Naked Single', 'Hidden Single', '
         Third Eye']:
3      data = {'number': technique_result['highlights'][0]['value'], '
           checkedCells': [technique_result['highlights'][0]['cell']]}
4      emit('help3')
5      new_numbers(data)
6      help_step -= 1
7    else:
8      sudoku.remove_candidates(technique_result['cross_outs'])
9      candidates()
10     emit('help3')
```

Listing 6.6: Serverseitige Unterscheidung der Auswirkung von Strategien

6.7.2. Kandidaten eliminieren

Die Funktionsweise von Strategien, die Kandidaten eliminieren ist ähnlich, wie gerade beschrieben. Der einzige Unterschied ist, dass anstatt *new_number()* die Funktion *remove_candidate()* aufgerufen wird. Die Reaktion des Frontends wird wieder über *emit*s ausgelöst. Aus diesem Grund wird für die Lösungsstrategie des Versteckter Vierers nur die *execute_technique(self)* erläutert.

Bei einem Versteckten Vierer wird darauf überprüft, ob vier Kandidaten in genau vier unterschiedlichen Zellen einer Unit vorkommen. Wenn dies der Fall ist, müssen diese vier Kandidaten in den vier Zellen stehen, daher können aus diesen vier Zellen alle anderen Kandidaten entfernt werden.

Da diese Technik in jeder Unit vorkommen kann, wird zu Beginn über jede Art von Unit iteriert, da es jede Unit neunmal gibt, müssen für jede Art von Unit neun Iterationen durchgeführt werden. Für jede Unit werden alle vorkommenden Kandidaten aufgelistet.

Aus der Liste mit den Kandidaten werden alle vierer Kombinationen ohne Wiederholungen gebildet. In den Zellen einer Unit wird mittels einer List comprehension überprüft, in welchen Zellen die Kandidaten der Kombination vorkommen. Diese werden zur Liste *matches* hinzugefügt. Nachdem über alle Zellen einer Unit iteriert wurde, wird die Länge der Liste auf vier überprüft. Dieser Teil der Strategie ist in dem Codebeispiel 6.7 implementiert.

Nach dem Finden eines solchen Musters werden die Highlightings konfiguriert. Dafür wird hier wie zuvor beschrieben die abstrakte Methode *configure_highlighting(self)* aufgerufen. Als Letztes wird überprüft, ob Kandidaten herausgestrichen werden können. Wenn die Liste der *self.cross_outs* größer als Null ist, dann wird der Wert *True* zurückgegeben. Wenn keine Kandidaten herausgestrichen werden können, dann wird mit der nächsten Unit auf die zuvor beschriebene Art und Weise verfahren.

```
1   combos = set(combinations(occurring_candidates, 4))
2   for self.combo in combos:
3     matches = []
4     for cell in self.unit_cells:
5       ...
6       candidates_num = SolvingTechniques.format_candidates(self.
            candidates[x][y])
7       if any(c in self.combo for c in candidates_num):
8         matches.append(cell)
9     if len(matches) == 4:
10      self.primary_cells = matches
11      self.configure_highlighting()
12      if len(self.cross_outs) != 0:
13        return True
```

Listing 6.7: Versteckter Vierer Teil 2

Nach dem erfolgreichen Finden einer solchen Strategie, wird, wie sich im Codebeispiel 6.6 erkennen lässt, die entsprechenden Kandidaten entfernt und mittels der Funktion *candidates()* die Darstellung der Kandidaten auch im Frontend aktualisiert.

Wenn in keiner Unit ein Versteckter Vierer entdeckt wird, gibt die Funktion *False* zurück und das Programm macht mit der nächsten Lösungsstrategie weiter.

7. Testing/Validierung

Zum Testen und Validieren des **Sudoku Helper** wurden zwei verschiedenen Techniken angewandt. Das Frontend wurde mittels eines Akzeptanztests von den Entwicklern getestet und das Backend mit einem End-to-End-Test.

In diesem Kapitel wird kurz der jeweilige Test erläutert und dann die Umsetzung im **Sudoku Helper** beschrieben.

7.1. Frontend

Das Frontend wurde mittels eines Akzeptanztests überprüft. Hierbei findet eine Funktionsprüfung einer User-Story statt. Dabei werden nicht funktionale Anforderungen, wie die Benutzerfreundlichkeit und Leistung der Software getestet. Diese nicht funktionalen Anforderungen wie die Antwortzeit zwischen Front- und Backend oder das Verhalten der Oberfläche wurde früh in der Entwicklung durchgeführt.

Im Zuge dieses Tests wurden Fehler abgefangen, die beispielsweise beim Neu laden der Website entstanden sind. In Meetings mit dem Kunden sind weitere Unstimmigkeiten aufgetreten, die aufgrund der Anmerkungen des Kunden verbessert wurden, wie beispielsweise die Auswahl der Farbgebung.

Ein abschließender Akzeptanztest sollte jedoch von dem Kunden der Software durchgeführt werden.

7.2. Backend

Im Backend muss die Korrektheit der Strategien überprüft werden. Diese Überprüfung findet während der Entwicklung statt.

Zu den Erklärungen der Strategien sind visuelle Beispiele gegeben. Die Beispiele dienen als Grundlage für den Test einer Strategie. Wenn eine Strategie auf ein Beispiel angewendet werden kann, dann wird diese Strategie in weiteren Sudokus gesucht. Beim Finden wird von Entwicklern, mittels des analogen Anwendens überprüft, inwiefern eine Strategie auch außerhalb der Grundlage korrekt angewandt wird. Diese Überprüfung findet im Optimalfall mehrere Male in unterschiedlichen Situationen, wie beispielsweise das Finden einer Strategie in verschiedenen Units, statt. Wenn eine Lösungsstrategie in unterschiedlichen Situationen von den Entwicklern nachvollzogen werden kann, so ist diese validiert.

7.3. End-to-End Test

Zu guter Letzt wurde ein End-to-End Test durchgeführt. In diesem Test wird nicht nur die Grundfunktionalität der Software überprüft, sondern auch die erweiterte Funktionalität im Falle von Grenzfällen wie zum Beispiel das Nichtfinden einer passenden Strategie. Damit wird der gesamte Prozess geprüft.

Im Zuge des **Sudoku Helper**s wurde ein Sudoku eingegeben und die Hilfestellungen der Strategien auf ihre Richtigkeit überprüft. Auch in diesem Test abgedeckt ist die Reaktion des Frontends, wenn im Backend keine passende Technik gefunden wird oder wenn das Sudoku korrekt gelöst wurde.

Mittels des End-to-End Tests wird sichergestellt, dass alle Prozesse, in diesem Fall das Anwenden verschiedener Strategien, korrekt ausgeführt wird und die entsprechenden Informationen ans Frontend gelangen.

8. Fazit

In diesem Kapitel wird abschließend geklärt, ob die Forschungsaufgabe mit Erfolg beendet werden konnte. Zunächst werden die wichtigsten Designentscheidungen in Kürze nochmals vorgestellt und daraufhin ein Fazit gezogen. Zum Abschluss wird hinausblicken nochmals auf den **Sudoku Helper** geschaut und weitere Verbesserungsmöglichkeiten genannt.

8.1. Ergebnisse

In der Studienarbeit soll ein **Sudoku Helper** entwickelt werden. Dieser soll einem User über verschiedenen Abstufungen dabei helfen unterschiedliche Lösungstechniken auf einen Sudoku Rätsel anzuwenden und das Rätsel zu lösen. Dabei wird davon ausgegangen, dass ein Rätsel mit etwa 25 Lösungstechniken vollständig gelöst werden kann.

In den Rahmenbedingungen werden die Programmiersprachen Python und JavaScript festgelegt. Neben den Programmiersprachen werden zwei Webserver verglichen, wobei am Ende entschieden wurde, dass die Software auf einem Apache HTTP Server laufen soll. Bei den Frameworks wurde sich für das leichtgewichtigere Flask entschieden, wobei sich an dem MVC Entwurfsmuster orientiert wurde.

Danach wurden einige mathematische Eigenschaften von Sudokus beleuchtet, wobei die eindeutige Lösbarkeit von besonderer Relevanz ist. Denn wenn ein Sudoku nicht eindeutig lösbar ist, dann können der **Sudoku Helper** und der User unterschiedliche Endergebnisse bekommen. Zudem basieren einige Strategien auf der Prämisse, dass ein Sudoku eindeutig lösbar ist.

Beim Frontend wurde besonders auf die UX geachtet sowie auf ein Responsive Design. Der Benutzer kann durch die Abstufungen der Hilfestellungen problemlos iterieren und das Frontend reagiert mit den jeweiligen Informationen darauf. Die benötigten Informationen dazu werden im Backend gesammelt. Über die Lösungstechniken, die alle identisch aufgebaut sind, da sie von einer abstrakten Klasse erben, wird nacheinander von einfach

bis schwierig iteriert und auf den aktuellen Zustand des Boards ausprobiert. Insgesamt wurden 27 Lösungsstrategien implementiert.

8.2. Beantwortung Forschungsfrage

Als Fazit lässt sich sagen, dass die Aufgabe zum größten Teil umgesetzt wurde. Die meisten Anforderungen, wie ein Responsive Design und das Implementieren von 25 Lösungsstrategien konnten umgesetzt werden. Mit den Hilfestellungen kann ein User die Anwendung der Techniken nachvollziehen und verstehen. Die Hilfestellungen teilen sich in unterschiedliche Stufen auf. Als Erstes wird nur ein bestimmter Bereich auf dem Board markiert und die Technik genannt, die angewendet werden kann. Im nächsten Schritt werden auch die relevanten Kandidaten in unterschiedlichen Farben hervorgehoben. Anschließend bekommt der Nutzer eine auf das aktuelle Rätsel angepasste Erläuterung, wie und vor allem warum eine Technik anwendbar ist. Zuletzt, kann der Nutzer das Programm die Technik ausführen lassen.

Die Studienarbeit hat gezeigt, dass auch mit 27 Lösungsstrategien nicht jedes Sudoku gelöst werden kann. Die vorgestellten Ergebnisse werfen die weiterführende Fragen auf, ob eine andere Reihenfolge der Lösungstechniken zu einer Lösung führen könnten. Eine andere weiterführende Forschungsfrage wäre, welche Techniken weiter implementiert werden müssten, damit jedes Sudoku Rätsel vom **Sudoku Helper** über logische und nachvollziehbare Methoden gelöst werden kann.

8.3. Ausblick

Der **Sudoku Helper** wurde zwar erfolgreich umgesetzt, trotzdem gibt es noch weiteres Potential für Verbesserungen.

Ein Problem des **Sudoku Helper**s ist, dass trotz der Implementierung von 27 Lösungsstrategien gerade schwierige Sudoku Rätsel noch nicht gelöst werden können. Man könnte also noch weitere Lösungsstrategien aus anderen Quellen implementieren. Des Weiteren ist die Reaktion darauf, keine anwendbare Lösungstechnik zu finden, nur suboptimal geregelt. Eine weitere Option wäre es dem User erst anzuzeigen, dass kein passender Algorithmus für das Sudoku Rätsel implementiert ist und daraufhin eine passende Zahl in das Gitter einfügt oder einen falschen Kandidaten eliminiert. Auch das Beenden des Sudokus könnte durch den optischen Effekt von Konfetti weiter zelebriert werden.

Eine weitere spannende Ergänzung wäre das eigenständige Festlegen der Reihenfolge der Lösungstechniken. Damit könnte der User bestimmten Techniken eine höhere Priorität zuordnen, um eine Technik des Öfteren zu finden, anzuwenden und zu verstehen. Dafür müssten in den Implementierungen der Lösungsstrategien weitere Anpassungen vorgenommen werden, da manche Strategien aufeinander aufbauen und daher zum Teil davon ausgegangen wird, dass bestimmte Konstellationen von Kandidaten bereits von einer früheren Technik gefunden werden.

Neben den funktionalen Erweiterungen gibt es auch noch optisch Verbesserungsmöglichkeiten, da die Oberfläche in speziellen Situationen noch sehr leer aussieht oder größer sein könnte. Das UI wurde zwar auf Basis des Empfindens der Entwickler benutzerorientiert gestaltet, mit weiterem Input des Kunden könnten jedoch weitere Optimierungen umgesetzt werden.

Literatur

[AV14] I. Althöfer und R. Voigt. *Spiele, Rätsel, Zahlen: Faszinierendes zu Lasker-Mühle, Sudoku-Varianten, Havannah, EinStein würfelt nicht, Yavalath, 3-Hirn-Schach, ...* Springer Berlin Heidelberg, 2014. ISBN: 9783642553011. URL: https://books.google.de/books?id=5vxXBAAAQBAJ.

[Ern20] P. Ernesti J. und Kaiser. *Python 3, Das umfassende Handbuch.* 5., aktualisierte Auflage. Bonn: Rheinwerk Computing, 2020. ISBN: 978-3-8362-7926-0.

[FJ06] Bertram Felgenhauer und Frazer Jarvis. „Mathematics of Sudoku I“. In: *Mathematical Spectrum* 39 (Jan. 2006).

[Her22] Michael Herman. *Django vs. Flask in 2022: Which framework to choose.* Website. Verfügbar unter https://testdriven.io/blog/django-vs-flask/; eingesehen am 25.05.2022. 2022.

[HM07] Agnes Herzberg und Ram Murty. „Sudoku squares and chromatic polynomials“. In: *Internationale Mathematische Nachrichten* 54 (Juni 2007).

[Ion] *Nginx vs. Apache: Die beliebtesten open-source-webserver im Vergleich.* Website. Verfügbar unter https://www.ionos.de/digitalguide/server/knowhow/nginx-vs-apache-ein-webserver-vergleich/; eingesehen am 25.05.2022. 2017.

[Kar19] Florian Karlstetter. *Was ist ein webserver?* Website. Verfügbar unter https://www.cloudcomputing-insider.de/was-ist-ein-webserver-a-884026/; eingesehen am 25.05.2022. 2019.

[Kno17] Simon Knott. *Backtracking.* Website. Verfügbar unter https://simonknott.de/articles/backtracking; eingesehen am 25.05.2022. 2017.

[Log14] D. Logofătu. *Grundlegende Algorithmen mit Java: Lern- und Arbeitsbuch für Informatiker und Mathematiker.* Springer Fachmedien Wiesbaden, 2014. ISBN: 9783834819727. URL: https://books.google.de/books?id=GBcyngEACAAJ.

[Lub22] Stefan Luber. *Was ist ein Framework?* Website. Verfügbar unter https://www.cloudcomputing-insider.de/was-ist-ein-framework-a-1104630/; eingesehen am 25.05.2022. 2022.

[Mar] Simon Martin. *Sudoku*. Website. Verfügbar unter https://www.thinkgym.de/rätselarten/sudoku/; eingesehen am 26.05.2022.

[Mus21] O. Musch. *Design Patterns mit Java: Eine Einführung*. Springer Fachmedien Wiesbaden, 2021. ISBN: 9783658354916. URL: https://books.google.de/books?id=nreUzgEACAAJ.

[PLA] THE ABLY PLATFORM. *The realtime web: evolution of the user experience*. Website. Verfügbar unter https://ably.com/blog/the-realtime-web-evolution-of-the-user-experience; eingesehen am 10.06.2022.

[Sar20] Dejan Sarka. *A Python Data Analyst's Toolkit, Learn Python and Python-based Libraries with Applications in Data Analysis and Statistics*. 6., aktualisierte Auflage. Berkeley, CA: Apress, 2020. ISBN: 978-1-4842-7172-8.

[Sta] *Stack overflow developer survey 2021*. Website. Verfügbar unter https://insights.stackoverflow.com/survey/2021; eingesehen am 25.05.2022.

[Ste20] R. Steyer. *JavaScript Grundlagen*. HERDT, 2020. ISBN: 978-3-86249-957-1.

[Sud22] Sudopedia. Website. Verfügbar unter http://www.sudopedia.org/; eingesehen am 25.05.2022. 2022.

[W3s] *jQuery Introduction*. Website. Verfügbar unter https://www.w3schools.com/jquery/jquery_intro.asp; eingesehen am 25.05.2022.

[W3t] *Usage statistics of web servers*. Website. Verfügbar unter https://w3techs.com/technologies/overview/web_server; eingesehen am 25.05.2022.

[Zam15] G. Zambon. *Sudoku Programming with C*. Apress, 2015. ISBN: 9781484209950. URL: https://books.google.de/books?id=xb4ICAAAQBAJ.

[Zei12] Frankfurter Allgemeine Zeitung. *Mathematik: Der heilige gral der sudokus*. Website. Verfügbar unter https://www.faz.net/aktuell/wissen/physik-mehr/mathematik-der-heilige-gral-der-sudokus-11682905.html; eingesehen am 27.05.2022. 2012.