**Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte BeschreibungProgrammieranfänger programmieren ein Schachprogramm**

Berufsmaturiätsarbeit

Berufssmaturitätsschule Zürich

Ausrichtung Technik, Architektur, Life Sciences

Begleitende Lehrperson: Eveline Eisenring

Verfasser: Robin Corbonnois & Benjamin Wild

Klasse: BPl21a

Abgabedatum: 25.11.2024

# Inhaltsverzeichnis

Inhalt

[Inhaltsverzeichnis 2](#_Toc183187002)

[1.) Abstract 4](#_Toc183187003)

[2.) Einleitung 5](#_Toc183187004)

[2.1) Motivation 5](#_Toc183187005)

[2.1.1) Schach 5](#_Toc183187006)

[2.1.2) Programmieren 6](#_Toc183187007)

[3.) Hauptteil 7](#_Toc183187008)

[3.1) Grundlagenaufarbeitung 7](#_Toc183187009)

[3.1.1) Schach 7](#_Toc183187010)

[3.1.2) Programmieren 8](#_Toc183187011)

[3.2) Arbeitsmethoden 12](#_Toc183187012)

[3.2.1) Projekt Übersicht, Funktionsstruktur 12](#_Toc183187013)

[3.2.2) Projektspezifikation 12](#_Toc183187014)

[3.2.3) Arbeitspakete 13](#_Toc183187015)

[3.2.4) Flussdiagramme 14](#_Toc183187016)

[3.3) Code 15](#_Toc183187017)

[3.3.1) Programmiersprache 15](#_Toc183187018)

[3.3.2) Pythons Keywords und wichtigste Funktionen 16](#_Toc183187019)

[3.4) Dokumentation und Aufbau des Codes 18](#_Toc183187020)

[3.4.1) Hauptschleife 18](#_Toc183187021)

[3.4.2) Methoden aus Bibliotheken 20](#_Toc183187022)

[3.4.3) Exe Datei/Setup 22](#_Toc183187023)

[3.4.4) Ergebnis 23](#_Toc183187024)

[3.4.5) Beschreibung des Programms 24](#_Toc183187025)

[3.4.6) Herausforderungen und Lernprozesse 25](#_Toc183187026)

[3.4.7) Zukünftige Erweiterungen 26](#_Toc183187027)

[4.) Schlusswort 27](#_Toc183187028)

[5.) Quellenverzeichnis 28](#_Toc183187029)

[6.) Anhang 30](#_Toc183187030)

[6.1) Chessbot BR Software 30](#_Toc183187031)

[6.2) Erweiterungen des Codes, die nicht implementiert wurde 30](#_Toc183187032)

[6.3) Alte Versionen des Chessbots 30](#_Toc183187033)

[6.4) Screenshots des Codes (ausgewählte Teile) 31](#_Toc183187034)

[7.) Bescheinigung 33](#_Toc183187035)

[8.) Dank 34](#_Toc183187036)

# Abstract

In dieser Arbeit lautet die Hypothese: „Programmieranfänger können ein Schachprogramm programmieren.“ Im Rahmen unseres Projekts, sollen wir ein funktionierendes Produkt entwickeln und grundlegendes Programmierwissen erwerben.

Die Arbeit vermittelt Grundkenntnisse zum Schachspiel, zu Programmierumgebungen und zum Programmieren selbst. Es werden die wichtigsten Funktionen in Python sowie die Methoden der verwendeten Bibliotheken genauer betrachtet. Zentrale Arbeitsmethoden und Schritte im Projektmanagement, wie Funktionsstrukturen und Flussdiagramme, werden erläutert. Der typische Problemlösungsprozess eines Programmierers wird nachvollziehbar dargestellt. Das entwickelte Programm wird detailliert mit Flussdiagrammen und Hauptfunktionen beschrieben. Mögliche Herausforderungen und der Umgang damit werden beleuchtet. Abschliessend werden potenzielle Erweiterungsmöglichkeiten aufgezeigt.

Ob die Hypothese bestätigt werden kann, hängt von der Zielsetzung, dem Vorkenntnisstand und der verfügbaren Zeit ab. In unserem Fall wird die Hypothese bestätigt: das entwickelte Produkt funktioniert und erfüllt die gesetzten Ziele.

# Einleitung

„Entwicklung“ bildet den Rahmen unseres Projekts. Die zentrale Hypothese bzw. Leitfrage lautet: „Programmieranfänger können ein Schachprogramm entwickeln.“

Wir vermuten, dass dies möglich ist. Ein Schachprogramm ist ein typisches Projekt für fortgeschrittene Programmierer. Im Internet gibt es bereits viele Tutorials, Codebeispiele und Foren zu diesem Thema. Ebenso sind zahlreiche Bibliotheken verfügbar, in denen mühsame Aufgaben, wie die Erstellung grafischer Benutzeroberflächen, bereits existieren. (Automatetheboringstuff, 2024)

In den letzten Jahren haben sich KI-Systeme rasant weiterentwickelt. Sie sind inzwischen in der Lage, Entwickler bei anspruchsvollen Programmierproblemen zu unterstützen – sei es bei der Konzeptplanung, der konkreten Umsetzung oder der Fehlersuche im Code. Tools wie ChatGPT und ähnliche Systeme sind sehr nützlich. (ChatGpt, 2024)

Neben all diesen Hilfestellungen wurde unsere Projektplanung teilweise flexibel gestaltet. Dadurch konnten wir unseren Fokus bei unerwartet komplexen Funktionen anpassen und so unser Ziel wahrscheinlicher erreichen: ein funktionsfähiges, ansprechendes, vielseitiges und benutzerfreundliches Schachprogramm zu entwickeln.

Wir werden grundlegende Konzepte des Programmierens vorstellen, zentrale Arbeitstechniken sowie den typischen Problemlösungsprozess eines Programmierers erläutern und die erzielten Entwicklungsergebnisse präsentieren. Grundkenntnisse der Schachregeln werden hierbei vorausgesetzt.

## 2.1) Motivation

In den folgenden Kapiteln möchten wir unsere Motivation in Bezug auf beide Themengebiete ausführlich darlegen.

### 2.1.1) Schach

Schach ist ein faszinierendes Spiel, da seine Regeln einfach zu erlernen sind. Dennoch erfordert es viel Übung, um sich kontinuierlich zu verbessern und das Spiel zu meistern. Trotz der einfachen Grundregeln ist Schach äusserst komplex und bietet enormes Potenzial zur Entwicklung individueller Fähigkeiten.

Um ein guter Schachspieler zu werden, muss man seine Fähigkeit zur mentalen Visualisierung sowie sein strategisches und logisches Denken schärfen. Schach fordert den Spieler heraus, komplexe Probleme zu erkennen und zu lösen wie bei einer Projektarbeit.

Dank Schachcomputern können Spieler nach jeder Partie analysieren, warum sie gewonnen oder verloren haben. Jede Stellung bietet einen objektiv besten Zug, sodass Glück keine Rolle spielt. Trotz dieser objektiven Natur des Spiels bietet Schach Raum für Kreativität, die sich in brillanten Angriffen und raffinierten Verteidigungsmanövern zeigt.

Schach ist für Menschen aller Altersgruppen geeignet und überwindet sprachliche sowie kulturelle Barrieren. Das Spiel ist kostenlos online verfügbar und bietet zahlreiche Spielvarianten und Zeitformate.

Wir spielen beide Schach in unserer Freizeit und haben ELO von 1500 und 1950, was ein Turnier Niveau aufzeigt.

### 2.1.2) Programmieren

Für uns ist diese Arbeit die perfekte Möglichkeit, das Programmieren im Allgemeinen, aber auch speziell Python, zu erlernen. Programmieren ist in vielen Arbeitsbereichen vertreten und ein mächtiges Werkzeug, das im Alltag benutzt wird. In unserer sich schnell automatisierenden Welt wird es immer wichtiger, eine Programmiersprache zu beherrschen und Programmierorientiertes Denken anzuwenden. Da die Spielregeln von Schach sehr präzis, erwartet und gut geregelt sind ist eine Programmierung bestens geeignet.

Deshalb haben wir Schach, ein Spiel, mit dem wir uns gut auskennen, mit dem Programmieren, etwas, das wir erlernen möchten, kombiniert. Ein funktionierendes Schachprogramm ist das passende und angestrebte Endergebnis dieses Projekts.

# Hauptteil

## 3.1) Grundlagenaufarbeitung

Im folgenden Kapitel wird der Hintergrund des Schachspiels erläutert. Zudem werden die Grundlagen des Programmierens sowie die dazugehörige Terminologie verständlich erklärt.

### 3.1.1) Schach

Schach ist ein strategisches Brettspiel, bei dem zwei Spieler abwechselnd ihre Figuren bewegen. Das Ziel ist es, den gegnerischen König schachmatt zu setzen, was bedeutet, dass er im nächsten Zug geschlagen werden könnte und keine Möglichkeit mehr hat, zu entkommen.

#### 3.1.1.1) Überblick

Schach ist weltweit bekannt, das bekannteste Brettspiel Europas und gilt als Sport. Der Weltschachverband FIDE (Fédération Internationale des Échecs) ist alle zwei Jahre Gastgeber der Schachweltmeisterschaften. (Wikipedia, Schach, 2024)

Die Komplexität des Spiels ist für Anfänger schwer zu erfassen. Dennoch ist Schach mit nur acht Figuren auf dem Brett (beide Könige mitgezählt) vollständig gelöst. Auf höchstem Niveau, wie bei Schachmeistern, enden etwa 43 % aller Partien mit einem Remis (Unentschieden).

#### 3.1.1.2) Geschichte

Die Ursprünge des Schachs liegen in Nordindien mit dem Vorläufer Chaturanga. Über die islamische Expansion verbreitete es sich weiter. Im 13. Jahrhundert wurde es gesellschaftlich wichtig, im 15. Jahrhundert in Spanien modernisiert, und die heutigen Regeln entstanden. Ab dem 18. Jahrhundert gehörte Schach zur bürgerlichen Kultur, die erste Weltmeisterschaft fand 1886 statt. Von 1945 bis 1990 dominierte die Sowjetunion, mit Ausnahme von Bobby Fischers Sieg 1972. (Wikipedia, Schach, 2024)

#### 3.1.1.3) Bezug zu Computer

Computer sind im Vergleich zur langen Geschichte des Schachs relativ jung, haben aber in den letzten Jahren erheblich an Bedeutung gewonnen. 1997 schlug der IBM-Schachcomputer Deep Blue den damaligen Schachweltmeister Garry Kasparow und markierte damit den Beginn eines neuen Zeitalters im Schach (Wikipedia, Schach, 2024). Heutzutage sind Schachprogramme den menschlichen Spielern weit überlegen. Sie werden hauptsächlich genutzt, um das eigene Spiel zu verbessern, sich auf Partien vorzubereiten oder Spiele im Nachhinein zu analysieren.

Trotzdem gab es bis vor kurzem noch Spielmodi, wie zum Beispiel das Vier-Spieler-Schach, in denen Menschen gelegentlich Computer besiegen konnten.

Mittlerweile gibt es auch Schachturniere für Computer, bei denen die besten Programme gegeneinander antreten (die meisten Partien enden in einem Unentschieden). Zu den bekanntesten Schachprogrammen gehören heute Stockfish und Komodo Dragon. (Github, 2024)

In den letzten Jahren erlebte Schach, insbesondere während der COVID-19-Pandemie, einen enormen Hype auf grossen Streaming-Plattformen wie Twitch und YouTube. Auch Online-Schachplattformen wie chess.com und lichess.org verzeichneten noch nie so viele aktive Spieler.

### 3.1.2) Programmieren

Die beste Analogie zum Programmieren lautet: „Dem Computer sagen, was er tun soll.“ Ein Computerprogramm ist entsprechend einfach eine Abfolge von Befehlen, die der Computer ausführt, um eine Aufgabe zu erledigen. Das Problem dabei ist, dass der Computer unsere Umgangssprache nicht versteht und Maschinencode für den Menschen schwer lesbar ist. Um die Kommunikation zwischen Menschen und Computer zu ermöglichen, wurden Programmiersprachen entwickelt. Sie dienen als Vermittler zwischen der menschlichen Sprache und der Maschinensprache. (ETH, 2024)

Ähnlich wie bei menschlichen Sprachen gilt: Konzepte, die in einer Programmiersprache existieren, lassen sich oft auf andere übertragen. Daher ist das Erlernen der ersten Programmiersprache, ähnlich wie das Erlernen der Muttersprache, der aufwändigste Schritt. Sobald man jedoch die Grundlagen beherrscht, fällt es deutlich leichter, neue Programmiersprachen zu lernen. Wie wir Menschen verschiedene Sprachen entwickelt haben, existieren auch zahlreiche Programmiersprachen. Insgesamt gibt es etwa 700 verschiedene Programmiersprachen (Knecht, 24). Diese unterscheiden sich in ihren Eigenschaften und lassen sich nach verschiedenen Kriterien einteilen. Ein zentrales Unterscheidungsmerkmal, auf das wir hier eingehen möchten, ist die Art der Ausführung: interpretierte und kompilierte Sprachen.

Kompilierte Programmiersprachen sind maschinennäher, da der Quellcode nach dem Schreiben kompiliert (in Maschinencode umgewandelt) und dann direkt von der Hardware ausgeführt wird. Diese Programme laufen zwar schneller, sind jedoch schwieriger zu schreiben und können nur auf der passenden Hardware (z. B. spezifische CPU-Architektur) ausgeführt werden.

Interpretierte Sprachen hingegen werden nicht direkt ausgeführt. Zuerst wird der Quellcode in Bytecode umgewandelt, der auf einer virtuellen Maschine und nicht direkt auf der Hardware ausgeführt wird. Das nennt man „interpretieren“. Dadurch kann der Code plattformübergreifend genutzt werden, allerdings läuft er etwas langsamer. (ETH, 2024)

Um Programme mit interpretiertem Code zu schreiben, benötigt man einen Editor und einen Interpreter. Im Editor wird der Quellcode verfasst, der dann über den Interpreter ausgeführt wird. Eine integrierte Entwicklungsumgebung (IDE, Integrated Development Environment) kombiniert beide Funktionen in einer Anwendung. Ein Quellcode-Editor bietet oft auch hilfreiche Zusatzfunktionen, wie die farbliche Darstellung von Befehlen, um den Code übersichtlicher zu gestalten. (ETH, 2024)

Um einen Befehl in einer Programmiersprache auszudrücken, müssen wir die Schlüsselwörter (Keywords) der Programmiersprache – also das Vokabular – verwenden. Dabei ist es wichtig, die Syntax zu beachten. Wenn wir nun die passenden Schlüsselwörter mit der korrekten Syntax kombinieren, entsteht funktionierender Code.

Bei grösseren Programmen, die komplexe Aufgaben lösen müssen, kann der entsprechende Code sehr lang werden und sich wiederholen. Um zu vermeiden, dass man denselben Code immer wieder schreibt und das Programm übersichtlicher bleibt, können bestimmte Codeblöcke als Funktionen definiert werden. Wenn man den entsprechenden Block, der mehrere Befehle enthält, benötigt, kann man die Funktion aufrufen, anstatt alle einzelnen Befehle erneut zu schreiben.

Selbst definierte Funktionen sind zwar praktisch, aber je nach Problemstellung kann auch dies sehr aufwendig sein. Deshalb kommen bei grösseren Projekten häufig Bibliotheken zum Einsatz. Eine Bibliothek enthält neben anderen Ressourcen vorprogrammierte Funktionen. Wenn man die passende Bibliothek wählt, findet man darin oft Funktionen, die mühsame oder komplexe Teile des Codes übernehmen. Diese Funktionen wurden von Experten entwickelt, die sich intensiv mit den jeweiligen Aufgaben beschäftigt haben, und bieten daher stabile und fehlerfreie Lösungen. Sie wurden von vielen Anwendern getestet, was sie in der Regel schneller und stabiler macht als individuell erstellte Lösungen. Ausserdem spart man dadurch viel Zeit, da man diese Funktionen nicht selbst programmieren muss.

Besonders zeitaufwendig kann die Erstellung einer grafischen Benutzeroberfläche (GUI, Graphical User Interface) sein, wenn man sie selbst programmieren möchte. Eine GUI ist das, was der Benutzer sieht, und über die GUI kommuniziert das Programm mit dem Nutzer und umgekehrt. Daher greift man häufig auf Bibliotheken zurück, die vorgefertigte GUI-Elemente enthalten.

Eine Klasse (class) ist eine Schablone, mit der Objekte erstellt werden. Sie definiert Attribute und Verhalten, die diese Objekte haben. Eine Funktion ist ein eigenständiges, wiederverwendbares Stück Code, das eine bestimmte Aufgabe ausführt. Eine Methode hingegen ist eine Funktion, die in einer Klasse definiert ist und nur in Zusammenhang mit Objekten dieser Klasse verwendet werden kann. Man kann sagen, eine Methode ist eine spezielle Art von Funktion, die innerhalb einer Klasse arbeitet und oft auf die Eigenschaften des Objekts zugreift.

Stellen wir nun den Bezug zu unserem Projekt her und schauen uns an, welche konkreten Werkzeuge wir verwenden. Wir haben uns für die Programmiersprache Python entschieden. Python ist eine interpretierte Sprache, auf die die bereits erwähnten Vorteile zutreffen. Das Wichtigste ist, dass Python einfach zu erlernen ist. Python hat eine einfache Syntax, die gut lesbar ist. Da Python eine weit verbreitete Programmiersprache ist, stehen viele Bibliotheken zur Verfügung, die uns das Programmieren erleichtern. Für uns als Physiklaboranten ist Python eine kluge Wahl, da wir diese Sprache in unserem beruflichen Umfeld wieder antreffen und nutzen könnten.

Spyder haben wir als integrated development environment (IDE) gewählt. Neben einem guten Editor und einer übersichtlichen Konsole bietet Spyder einen Variablenmanager, einen eingebauten Profiler (um die Laufzeit zu optimieren) und eine Codeanalyse, um mögliche Fehler zu entdecken und die Qualität des Codes zu verbessern. Der Editor bietet automatische Vorschläge und ermöglicht es dir, einzelne Zeilen des Codes auszuführen, um sie zu testen und Fehler zu finden. Darüber hinaus ist Spyder gut dokumentiert und daher eine ausgezeichnete Wahl für Anfänger. (Contributors & License., 2024)

Spyder führen wir über Anaconda (open-source distributor von Python) aus. Anaconda hat Spyder sowie viele andere IDEs vorinstalliert (z. B. welche, die praktisch für die Datenanalyse sind, was nicht weit von unserem Berufsfeld entfernt ist), die sich einfach aktualisieren lassen. Zudem bringt Anaconda die wichtigsten Bibliotheken bereits mit, sodass die Anwendung alles zusammenfasst, was wir zum Programmieren benötigen.

In unserer App nutzen wir Tkinter, ein einfaches GUI-Tool in Python zur Darstellung von Elementen wie Buttons oder Dialogfenstern. Es gibt zwar mehrere Tools zur GUI-Erstellung, aber Tkinter ist im Vergleich zu QT einfacher.

Wir verwenden auch die „python-chess“-Bibliothek, die das Spiel repräsentiert, sowie „chess.engine“, um mit externen Schach-Engines wie Stockfish zu kommunizieren. Die Bibliothek „os“ ermöglicht die Interaktion mit dem Betriebssystem, während „threading“ dafür sorgt, dass Hintergrundoperationen parallel laufen. (Python docs, 2024)

Zusätzlich nutzen wir „comtypes.client“, um Windows-Anwendungen wie Office anzusprechen, und „webbrowser“, um URLs im Standardbrowser zu öffnen. (Python docs, 2024)

## 3.2) Arbeitsmethoden

In den folgenden Kapiteln werden wichtige Arbeitsmethoden aufgezeigt.

### 3.2.1) Projekt Übersicht, Funktionsstruktur

Grössere Projekte können einschüchternd wirken, insbesondere wenn sie eine noch nie zuvor durchgeführt wurden. Deshalb ist es wichtig, systematisch vorzugehen und komplexe Aufgaben in kleinere Teilaufgaben zu zerlegen. Eine Methode, dies zu tun und gleichzeitig das Projekt klar zu definieren, ist die Erstellung einer Funktionsstruktur. Dabei wird das Projekt in Teilfunktionen aufgeteilt, die gegebenenfalls weiter unterteilt werden können.

Ein Bild, das Text, Screenshot, Schrift, Klebezettel enthält.

Automatisch generierte BeschreibungDer Zeit- und Kostenaufwand für die einzelnen Teilaufgaben lässt sich so einfacher und präziser abschätzen. Mithilfe einer Funktionsstruktur, sowohl für das Produkt als auch für den schriftlichen Kommentar, kann die gesamte Zeitplanung erstellt werden.

Abbildung 1 Übersicht des BMA Schach Programms: Funktionen und Struktur

### 3.2.2) Projektspezifikation

Ein Chessbot ist eine Schachanwendung, die ermöglicht das Spielen von Schach auf einem Computer. Unser Chessbot ermöglicht das Spielen entweder gegen einen Bot oder gegen einen Freund, in einer benutzerfreundlichen Oberfläche. Er nutzt die Schach-Engine Stockfish und zeigt Spielzüge mit Emojis an und die Algebraische Notation.

### 3.2.3) Arbeitspakete

Nachdem die Funktionsstruktur und der Zeitplan erstellt wurden, kann man sich Schritt für Schritt den einzelnen Programmierproblemen widmen. Beim Bearbeiten eines Programmierproblems folgt man stets demselben Ablauf. Zunächst Überlegt man verschiedene Lösungsansätze. Der vielversprechendste – oft der einfachste – wird gedanklich ausgearbeitet. Häufig greift man dabei auf Flussdiagramme zurück, auf die wir später noch genauer eingehen. Erst danach beginnt der Programmierprozess. Anschliessend wird der Code getestet, da er oft nicht sofort einwandfrei funktioniert. Durch mehrere Zyklen von Code-Anpassungen und Tests entsteht schliesslich funktionierender Code. Im Rahmen eines kontinuierlichen Verbesserungsprozess wird der Code ständig verbessert. (ETH, 2024)

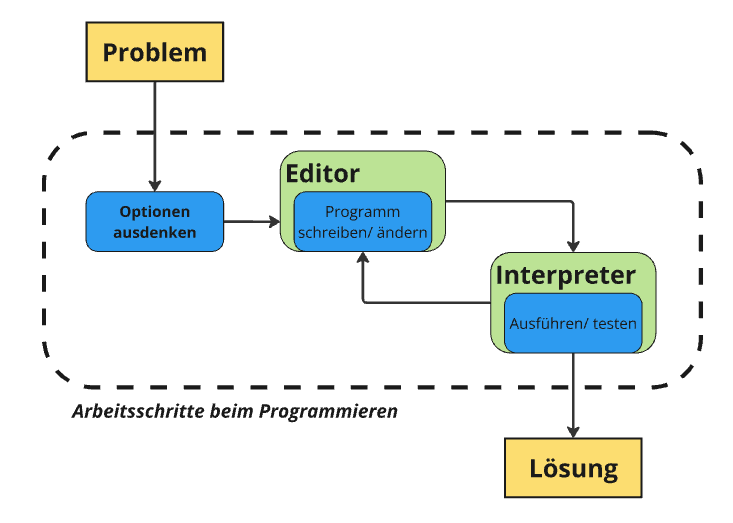


Abbildung 2 Arbeitsschritte beim Programmieren (ETH, 2024)

### 3.2.4) Flussdiagramme

Flussdiagramme sind ein bewährtes Werkzeug zur Visualisierung von Programmabläufen. Sie helfen besonders Anfängern, den Aufbau eines Programms sowie Verzweigungen und Schleifen besser zu verstehen. Durch die grafische Darstellung lassen sich Fehler und ineffiziente Abläufe leichter erkennen.

Es gibt Normen für die Gestaltung von Flussdiagrammen, die eine einheitliche Struktur gewährleisten. Wir orientieren uns an der DIN 66001 (1966), die fünf grundlegenden Elemente definiert. Diese Norm legt hauptsächlich die Formen der Symbole fest, nicht aber ihre Farben, da Farben meist nur zur besseren Übersicht verwendet werden. Ein Flussdiagramm wird von oben nach unten gezeichnet, wie der Programmablauf im Computer. (Wikipedia, Programmablaufplan, 2024)

Bei grösseren Projekten ist es sinnvoll, ein Hauptflussdiagramm für den gesamten Ablauf zu erstellen. Für komplexe oder besonders wichtige Teile, wie bestimmte Operationen oder Unterprogramme, sollten zusätzliche Detailflussdiagramme verwendet werden, um diese logisch klarer darzustellen.

Terminatoren (grüne/ rote, abgerundete Rechtecke) markieren den Anfang und das Ende eines Programms oder Unterprogramms.

Operatoren (graue Rechtecke) stehen für einfache Tätigkeiten oder Anweisungen, die in wenigen Zeilen Code ausgeführt werden.

Unterprogramme (orange Rechtecke mit Doppelstrichen an den Seiten) repräsentieren selbstgeschriebene oder importierte Funktionen, die modular, aber nicht unbedingt komplex sind. Bei komplexen Unterprogrammen ist es sinnvoll, eigene Flussdiagramme zu erstellen.

Ein- und Ausgabefelder (braune Parallelogramme) zeigen Interaktionen mit dem Benutzer oder Daten. Verzweigungen (rote Rauten) stellen Entscheidungsprozesse dar, die den weiteren Ablauf des Programms steuern

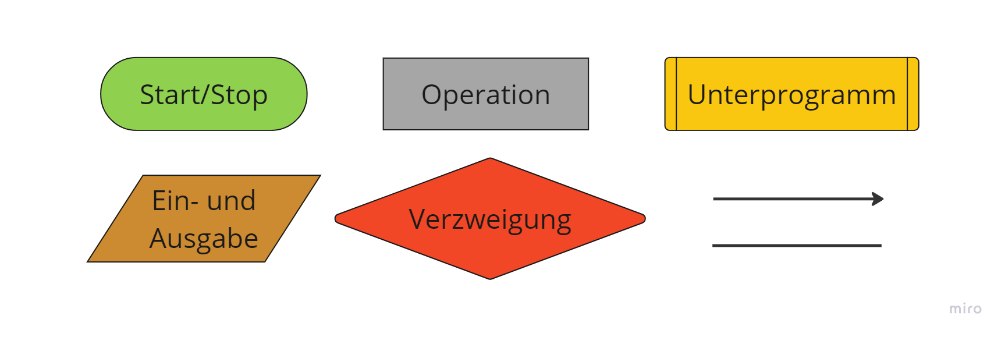


Abbildung 3 Grundlegende Blöcke eines Flussdiagramms

(Wikipedia, Programmablaufplan, 2024)

## 3.3) Code

### 3.3.1) Programmiersprache

Code wird – ähnlich wie viele Sprachen – von links nach rechts und von oben nach unten gelesen. Die Entwicklungsumgebung Spyder nutzt, wie viele andere IDEs, verschiedene Farben, um den Code übersichtlicher zu gestalten und das Schreiben zu erleichtern. Folgender beschriebener Farbcode gilt nur für Spyder im «light mode», und kann beliebig angepasst werden.

Teile des Codes sind ein- oder mehrfach eingerückt. Einrückungen machen den Code übersichtlicher und sind zugleich notwendig, um klar anzugeben, welche nachfolgenden Zeilen noch zu einer Schleife oder einer bestimmten Funktion gehören. Sie sind also auch ein wichtiger Bestandteil der Syntax.

Pythons Keywords werden in Orange dargestellt. Keywords sind reservierte Begriffe, die Python für bestimmte grundlegende Operationen und Strukturen verwendet, wie z. B. zur Definition von Funktionen, Klassen, Bedingungen oder Schleifen. Sie bilden eine zentrale Grundlage der Sprache. Zu den häufigsten Keywords gehören: def, class, if, elif, else, import, from, as, return, pass und lambda.

Unterschiedliche Datentypen haben ebenfalls eigene Farben im Hellmodus. Strings (Texte, die von Python als „Text“ erkannt werden) sind grün dargestellt. „Booleans“ (also binäre Werte wie True und False) sind violett. Numerische Werte (ganze Zahlen „Integer“ und Fliesskommazahlen „Float“) werden braun angezeigt.

Kommentare sind für Erklärungen und zusätzliche Informationen im Code gedacht. Sie stehen direkt im Quelltext und beginnen immer mit einem #. In Spyder sind sie rot markiert.

Blaue Wörter stehen für selbst definierte Namen. Zum Beispiel benötigt jede Funktion einen Namen, damit sie in anderen Teilen des Programms aufgerufen werden kann.

„Self“ ist der Parameter einer Klasse und sorgt dafür, dass ein Programm auch auf andere Teile des eigenen Codes zugreifen kann, zum Beispiel wenn verschiedene Bereiche Informationen gemeinsam nutzen oder austauschen. „Self“ ist stets grau dargestellt.

Der Rest des Codes (z. B. Funktionen aus importierten Bibliotheken) wird schwarz angezeigt.

### 3.3.2) Pythons Keywords und wichtigste Funktionen

Wir möchten die wichtigsten Funktionen von Python und unsere an den häufigsten verwendeten Funktionen kurz erläutern.

Def (kurz für „Define“) ist ein Befehl, mit dem wir Variablen oder ganze Funktionen definieren. In unserem Programm befinden sich die meisten Funktionen in der Hauptklasse. Nach dem Schlüsselwort def folgt in der Regel der Funktionsname, und die darauffolgenden Codezeilen sind eingerückt, um anzuzeigen, dass sie zur Funktion gehören.

„Import“, „from“ und „as“ sind drei zentrale Schlüsselwörter, die oft zusammen verwendet werden. „Import“ lädt bestimmte Funktionen oder Module aus zugreifbaren Bibliotheken (durch das Schlüsselwort from). Mit as können wir Namen umbenennen, was hilfreich ist, wenn Bibliotheksnamen oder Funktionsnamen abgekürzt werden sollen. Beispiel: import tkinter as tk oder from tk import messagebox.

„If“ ist ein zentrales Schlüsselwort in Python und bedeutet, wenn. Es prüft, ob eine bestimmte Bedingung erfüllt ist. Wenn das der Fall ist, wird der folgende Codeblock ausgeführt. Else ist das Gegenstück: Wenn die Bedingung nicht erfüllt ist, wird der Code im „else“-Block ausgeführt. So lassen sich Entscheidungen im Programmablauf steuern. Elif (kurz für „else if“) fügt weitere Bedingungen hinzu, die geprüft werden, wenn mehrere mögliche Zustände den Programmablauf beeinflussen sollen.

Das Programm läuft weiter, da es sich in einer ereignisgesteuerten Schleife („root.mainloop()“) befindet, die ständig auf Benutzeraktionen wartet. Gleichzeitig stellen Threads sicher, dass Hintergrundprozesse laufen, ohne die Reaktion der GUI zu beeinträchtigen. Beendet wird das Programm erst, wenn die Beenden-Funktion „on\_closing()“ ausgeführt oder das Fenster geschlossen wird.

## 3.4) Dokumentation und Aufbau des Codes

### 3.4.1) Hauptschleife

Die Hauptaufgabe der Hauptschleife des Programms besteht darin, das GUI-Fenster zu erstellen und das Chessbot-Objekt zu initialisieren, um die Benutzeroberfläche und die zugrunde liegende Spielmechanik zu steuern. Zunächst wird mit dem Objekt „tk.Tk()“ ein neues Hauptfenster erzeugt, das als zentrales Anzeigefenster der Anwendung dient. Dieses Fenster enthält alle wesentlichen Elemente wie das Schachbrett, Spielinformationen und Steuerelemente. Anschliessend wird das „Chessbot“-Objekt initialisiert, wodurch die gesamte Logik des Schachspiels geladen und die Benutzeroberfläche konfiguriert wird. Dies umfasst unter anderem das Einrichten eines leeren Schachbretts, der verschiedenen Spielmodi, der Timer sowie die Anzeige von Informationen über den Spieler und den Bot. Ausserdem wird ein zufälliger Startspieler bestimmt. (Python docs, 2024)

Nach dem Start der Hauptschleife durch Aufruf von „root.mainloop()“ bleibt das Fenster geöffnet und alle Benutzeraktionen werden kontinuierlich verarbeitet. Während das Fenster geöffnet ist, kann der Spieler Züge machen, Modi auswählen und das Spiel steuern. Alle Interaktionen, wie das Klicken auf das Schachbrett oder das Wechseln des Modus, werden von der Schleife zuverlässig überwacht und verarbeitet. Abschliessend wird festgelegt, wie sich das Programm beim Schliessen des Fensters verhält: Durch den Aufruf „root.protocol(„WM\_DELETE\_WINDOW“,„app.on\_closing)“ wird sichergestellt, dass die Schach-Engine sicher beendet wird und keine Hintergrundprozesse aktiv bleiben. (Python docs, 2024)

Die Hauptschleife gewährleistet also, dass die Anwendung immer interaktiv bleibt und sämtliche Elemente in Echtzeit reagieren. Sie sorgt dafür, dass das Benutzererlebnis reibungslos und benutzerfreundlich ist, sodass alle GUI-Elemente und Spielprozesse optimal ablaufen (Miro, 2024)

Ein Bild, das Text, Screenshot, Diagramm, Design enthält.

Automatisch generierte Beschreibung

Abbildung 4 Hauptflussdiagramm

### 3.4.2) Methoden aus Bibliotheken

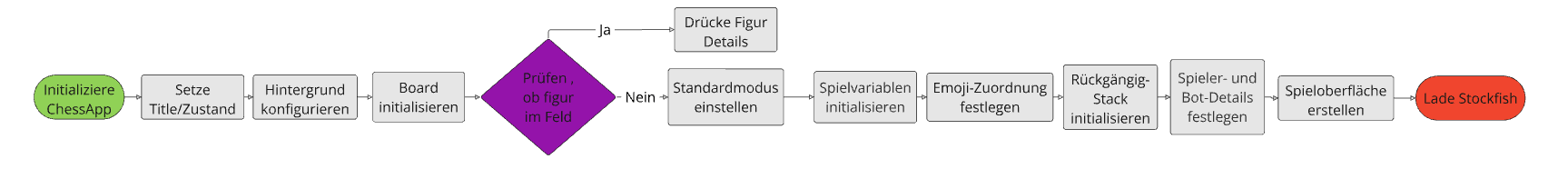
Die „\_init\_“-Methode (Zeilen im Code: 22-87) initialisiert die Schach-App, lädt die GUI und setzt das Schachbrett in die Startposition. Sie lädt zudem die Stockfish-Engine im Hintergrund.

Abbildung 5 Flussdiagramm def\_Init

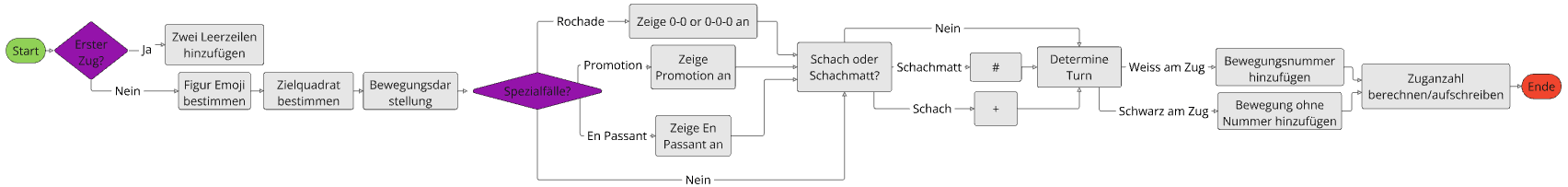
Die „display\_move\_in\_output“-Methode (Zeilen 867–925) ist der führende Bildschirm für das Spiel und protokolliert die Züge im Ausgabefenster. Sie verarbeitet Spezialfälle wie Rochaden, Bauernumwandlungen sowie Schach- und Mattsituationen und sorgt für eine klare Darstellung der Spielzüge. (Python docs, 2024)

Abbildung 6 Flussdiagramm def\_Display\_move\_in\_output

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte BeschreibungDie „handle\_square\_click“-Methode (Zeilen 515-552) verarbeitet Klickaktionen auf dem Schachbrett. Sie prüft, ob eine gültige Figur ausgewählt wurde, und führt Züge oder spezielle Aktionen wie Bauernumwandlungen aus.

Abbildung 7 Flussdiagramm def\_handle\_square\_click

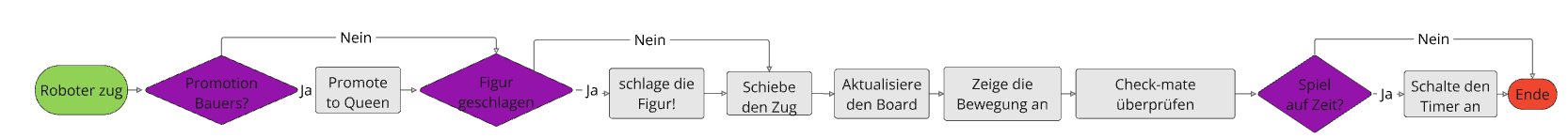
Die „bot\_move“-Methode (Zeilen 635-648) lässt das Bot seinen Zug mit der Schach-Engine berechnen und führt diesen durch. Die Methode aktualisiert das Schachbrett und behandelt spezielle Fälle wie Bauernumwandlungen.

Abbildung 8 Flussdiagramm def\_bot\_move

Ein Bild, das Diagramm, Plan enthält.

Automatisch generierte BeschreibungDie „open\_promotion\_window“-Methode (Zeilen 554-589) öffnet ein Fenster zur Auswahl der Figur bei einer Bauernumwandlung. Der Benutzer kann Dame, Turm, Springer oder Läufer auswählen, und der Zug wird dann ausgeführt.

Abbildung 9 Flussdiagramm def\_open\_promotion\_window

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

Automatisch generierte BeschreibungDie „update\_board“-Methode (Zeilen 384-416) aktualisiert die visuelle Darstellung des Schachbretts nach jedem Zug. Sie zeigt neue Figurenpositionen, markiert bedrohte Könige und stellt sicher, dass alle Felder korrekt eingefärbt sind.

Abbildung 10 Flussdiagramm def\_update\_board

Ein Bild, das Diagramm, Reihe, Schrift, Screenshot enthält.

Automatisch generierte BeschreibungDie „push\_move“-Methode (Zeilen 591-611) führt einen Zug aus und aktualisiert das Schachbrett. Sie registriert geschlagene Figuren, speichert den Zug für spätere Rücknahmen und prüft auf Endspielbedingungen.

Abbildung 11 Flussdiagramm def\_push\_move

### 3.4.3) Exe Datei/Setup

Eine .exe-Datei ist eine ausführbare Datei, die auf Windows-Computern dazu dient, ein Programm zu starten. Sobald man eine .exe-Datei öffnet, führt der Computer den darin enthaltenen Code aus (kompilierter Code), wodurch die Anwendung gestartet wird. Diese Dateien enthalten den Programmcode und alle notwendigen Ressourcen, die das Programm benötigt, wie Bilder oder Bibliotheken. Da .exe-Dateien auf Windows ausgelegt sind, funktionieren sie normalerweise nicht auf anderen Systemen ohne spezielle Software.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungEin Setup wird im Deutschen als Installationsprogramm bezeichnet und wird in Pascal Programmiersprache geschrieben. Es handelt sich um eine Software, die genutzt wird, um Programme oder Treiber auf einem Computer zu installieren. In unserem Fall dient es als Grundlage dafür, dass Stockfish und die GUI zusammen installiert und automatisch gefunden werden können, ohne dass der Benutzer einen Pfad manuell angeben muss. (Ionos, 2024)

Abbildung 12 Code-Setup für die Installation des ChessBot

### 3.4.4) Ergebnis

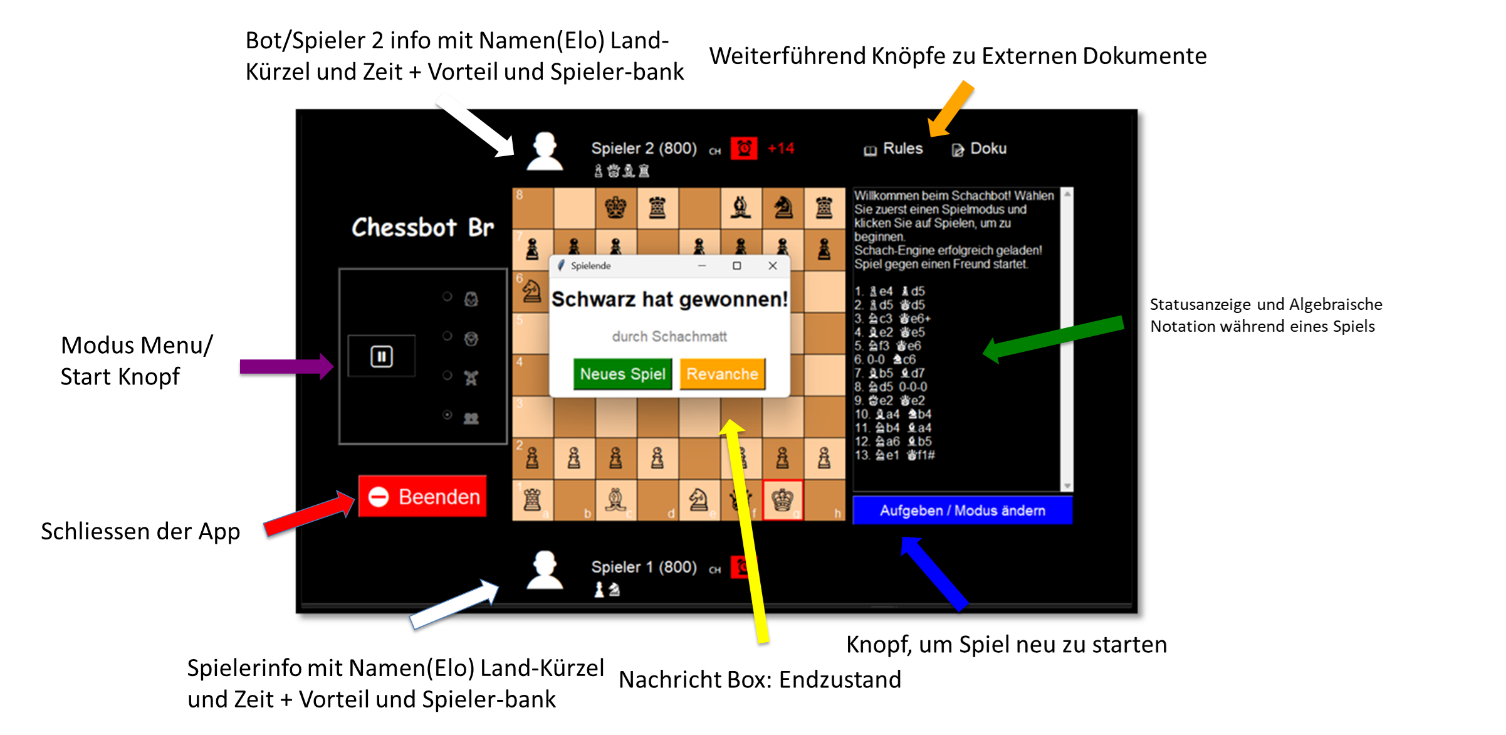
Das Ergebnis des Projekts ist eine voll funktionsfähige Schach-Applikation, genannt BR Chessbot. Diese Anwendung bietet mehrere Spielmodi (einfach, mittel, schwer, gegen einen Freund) und ermöglicht es, sowohl gegen ein Bot mit unterschiedlichen Schwierigkeitsgraden als auch gegen einen Freund zu spielen. Das Programm lädt den Stockfish-Schach-Engine, um intelligente Züge durchzuführen, und verfügt über eine grafische Benutzeroberfläche, die das Schachbrett und die gespielten Züge klar darstellt. Dies entspricht unserer Projektspezifikation.

Abbildung 13 Chessbot Software Beschreib

Am Ende jedes Spiels erscheint ein Fenster, das den Ausgang des Spiels klar und deutlich zeigt. Dieses Fenster bietet drei mögliche Szenarien: «Weiss hat gewonnen!» wenn der weisse Spieler durch Schachmatt gewinnt. «Schwarz hat gewonnen!» wenn der schwarze Spieler durch Schachmatt gewinnt. «Unentschieden» wenn das Spiel in einem Patt endet. Zusätzlich gibt es zwei Schaltflächen: «Neues Spiel» für den Start einer neuen Partie und «Revanche» für ein sofortiges Rematch. Diese klare Darstellung sorgt für einen unkomplizierten Übergang und lädt die Spieler ein, das Spiel direkt fortzusetzen.

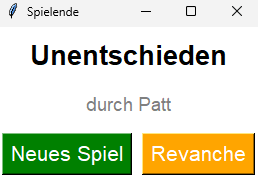
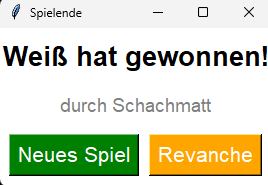


Abbildung 14 Chessbot Spielende-Bildschirme

### 3.4.5) Beschreibung des Programms

Die Initiale Einrichtung begann mit der Erstellung der Klasse Chessbot, um sowohl die grafische Benutzeroberfläche (GUI) als auch die Spielmechaniken zu implementieren. Die Benutzeroberfläche basiert auf der Tkinter-Bibliothek und bildet das Spielbrett mit einer 8x8-Anordnung von Feldern ab, darauf stehen auch Schachfiguren. Darüber hinaus wurden verschiedene Elemente wie Timer, Spielerinformationen und sogenannte Stückbänke für geschlagene Figuren hinzugefügt. (Python docs, 2024)

Für die unterschiedlichen Spielmodi - darunter die Kategorien leicht, mittel, schwer sowie der Spielmodus «Freund» - wurden Radiobuttons (ein visuelles Element) integriert, mit denen die gewünschte Einstellung leicht gewählt werden kann. Die künstliche Intelligenz des Bots wird durch die Stockfisch-Engine realisiert, wobei die Schwierigkeitsgrade durch unterschiedliche Spieltiefen (bot\_depth) variiert werden.

Die Logik für die Spielzüge umfasst Funktionen, die die Auswahl und Bewegung der Figuren ermöglichen, Züge des Bots automatisch ausführen und auch Spezialfälle wie die Umwandlung von Bauern korrekt behandeln. Für die optimale Berechnung der Bot-Züge wird die Schach-Engine Stockfisch im Hintergrund geladen. Diese Berechnungen laufen in einem separaten Thread, damit die GUI stets reaktionsfähig bleibt. (Python docs, 2024)

Für beide Spieler wurden Timer eingeführt, die automatisch die verbleibende Zeit überwachen und sicherstellen, dass das Spiel fair bleibt. Zudem wurde die Überprüfung auf Schachmatt, Patt und unzureichendes Material hinzugefügt, um die Endbedingungen des Spiels sicherzustellen.

Schliesslich wurde auch eine Funktion eingebaut, um das Spiel jederzeit aufzugeben oder einen Modus Wechsel vorzunehmen, wodurch eine besonders flexible Spielerfahrung ermöglicht wird.

### 3.4.6) Herausforderungen und Lernprozesse

Während der Entwicklung unseres Schachbots standen wir vor verschiedenen Herausforderungen und konnten wertvolle Lernerfahrungen sammeln. Eine der ersten Hürden war das Festhalten des Versionsverlaufs. Anstatt viele Dateien mit unterschiedlichen Versionen (v1, v2, v3 usw.) zu erstellen, haben wir uns letztendlich für GitHub entschieden, um den Versionsverlauf mit einem sauberen und nachvollziehbaren Commit-Log zu dokumentieren. (Github, 2024)

In der Anfangsphase haben wir das Platzieren der GUI-Elemente mit «place» umgesetzt, ohne die Möglichkeiten des «grid»-Layouts zu kennen, das sich im Nachhinein als wesentlich effizienter erwies. Ein weiteres wichtiges Feedback kam aus der Nutzererfahrung: Unsere erste Version war durch zu viele Eingaben und Auswahlmöglichkeiten gekennzeichnet, was den Einstieg erschwerte und die Nutzererfahrung beeinträchtigte. Dies hat uns dazu veranlasst, das Design zu vereinfachen und die Bedienung intuitiver zu gestalten.

Auch im Bereich Debugging und Entwicklungsumgebung gab es Lernkurven. Bei Spyder, gab es einen nicht funktionierenden Debugger. Das Visual Studio Code Debugger war intuitiv und gut geeignet. Das Verwenden dieses neue System hat uns Zeit gekostet.

Zudem mussten wir uns mit dem Erstellen einer ausführbaren Datei (.exe) und einer Setup-Funktion vertraut machen. Ohne diese .exe Funktion war das Installieren des Systems nicht auf jedem PC möglich, vor allem nicht auf Computer ohne IDE. Somit war unser System nicht zugänglich für alle.

Durch diese Erfahrungen haben wir nicht nur unsere technischen Fähigkeiten erweitert, sondern auch ein besseres Verständnis für sauberes Software-Design und effizientes Projektmanagement entwickelt.

### 3.4.7) Zukünftige Erweiterungen

Unser Programm ist stabil und voll funktionsfähig. Da die Kernfunktionen implementiert sind, konzentrieren wir uns nun auf Erweiterungen, um das Spielerlebnis weiter zu verbessern, ohne das bestehende System zu verändern.

Ein nächster Schritt wäre der Wechsel zur Qt-Bibliothek, die eine modernere und flexiblere Benutzeroberfläche bietet. Dadurch könnten wir das Layout ansprechender gestalten und die Bedienung intuitiver machen.

Zusätzlich wäre die Entwicklung einer eigenen Bot-Engine möglich, die unterschiedliche Spielstile unterstützt und so ein individuelleres Spielerlebnis schafft. Über eine Bereitstellung als Cloud-Service könnte zudem eine lokale Installation entfallen, was die Nutzung komfortabler macht.

Für eine gezielte Vertiefung des Spiels planen wir die Einführung eines Analyse-Modus, der das Überprüfen vergangener Partien und das Finden besserer Züge ermöglicht. Ergänzend könnten Schachpuzzles das taktische Verständnis fördern und die Lernerfahrung steigern.

Um das Bot noch dynamischer zu gestalten, würden wir gerne Reinforcement Learning einsetzen, damit er aus gespielten Zügen lernt und sich eigenständig weiterentwickelt.

Mit diesen Erweiterungen könnten wir ein vielfältiges und bereicherndes Spielerlebnis bieten, das das Schachbot langfristig ansprechend und lehrreich gestaltet.

(Stack overflow, 2024)

# Schlusswort

Zusammenfassend lässt sich die Hypothese „Ein Programmieranfänger kann ein Schachprogramm entwickeln“ bejahen.

Der Zeitaufwand hängt jedoch stark von den Vorkenntnissen, dem Umfang des Projekts und der genauen Zieldefinition ab. In unserem Fall wurde der Grossteil der Zeit darauf verwendet, sich mit bestehenden Bibliotheken, Programmiersprachen und Methoden vertraut zu machen, um Kenntnisse zu erwerben. Unsere Zielsetzung und Projektspezifikation waren klar, was ermöglichte ein gutes Projekt- und Zeitmanagement. Das Hauptteil der Arbeit bestand darin Problemlösungen zu beheben, sie zusammen zu besprechen und nachhaltige Ergebnisse zu entwickeln.

# Quellenverzeichnis

*Automatetheboringstuff*. (25. 10 2024). Von Tutorial: https://automatetheboringstuff.com/#toc abgerufen

*ChatGpt*. (3. 11 2024). Von K.I.: https://chatgpt.com/ abgerufen

Contributors, S. D., & License., M. (18. 11 2024). *Spyder*. Von Working with Spyder: https://docs.spyder-ide.org/current/videos/working-with-spyder.html abgerufen

ETH. (18. 11 2024). *E.Turtorial*. Von Python Tutorial (Schnelleinstieg): https://et.lecturers.inf.ethz.ch/viewer/course/5TLRsNXvLcMaw229P?course\_locale\_key=de abgerufen

*Github*. (1. 11 2024). Von Stockfisch: https://github.com/official-stockfish/Stockfish](https://github.com/official-stockfish/Stockfish abgerufen

*Github*. (1. 11 2024). Von Python Chess Github Repository: https://github.com/niklasf/python-chess abgerufen

*Ionos*. (1. 11 2024). Von Was ist .exe?: [https://www.ionos.de/digitalguide/server/knowhow/was-ist-exe/](https://www.ionos.de/digitalguide/server/knowhow/was-ist-exe/) abgerufen

Knecht, S. (20. 11 24). *Einführung: Unterschiedliche Programmiersprachen.* Von Programming Workshop.: https://stekhn.github.io/programming-workshop/einfuehrung/unterschiedliche-programmiersprachen.html#:~:text=Wikipedia%20z%C3%A4hlt%20%C3%BCber%20700%20verschieden,M%C3%B6glichkeiten%20Programmiersprachen%20in%20Kategorien%20einzuordnen. abgerufen

*Miro*. (25. 10 2024). Von Whiteboard: [https://miro.com/app/board/uXjVLH\_xjIw=/](https://miro.com/app/board/uXjVLH\_xjIw=/) abgerufen

*Python docs*. (26. 10 2024). Von Tkinter.messagebox: [https://docs.python.org/3/library/tkinter.messagebox.html](https://docs.python.org/3/library/tkinter.messagebox.html) abgerufen

*Python docs*. (29. 10 2024). Von Tkinter: [https://docs.python.org/3/library/tkinter.html](https://docs.python.org/3/library/tkinter.html) abgerufen

*Python docs*. (2. 11 2024). Von Threading: https://docs.python.org/3/library/threading.html abgerufen

*Python docs*. (28. 10 2024). Von Python chess: https://python-chess.readthedocs.io/en/latest/ abgerufen

*Python docs*. (28. 10 2024). Von Python chess.engine: https://python-chess.readthedocs.io/en/latest/engine.html abgerufen

*Python docs*. (27. 10 2024). Von Webbrowser: https://docs.python.org/3/library/webbrowser.html abgerufen

*Python docs*. (31. 10 2024). Von Os: https://docs.python.org/3/library/os.html abgerufen

*Stack overflow*. (30. 11 2024). Von Coding questions: https://stackoverflow.com/questions abgerufen

Wikipedia. (18. 11 2024). Von Schach: https://de.wikipedia.org/wiki/Schach abgerufen

Wikipedia. (18. 11 2024). *Programmablaufplan*. Von Programmablaufplan: https://de.wikipedia.org/wiki/Programmablaufplan abgerufen

# Anhang

## 6.1) Chessbot BR Software

[Code Chessbot app (Setup)](https://github.com/rcsdss/this-is-my-first-project): https://github.com/rcsdss/this-is-my-first-project

## 6.2) Erweiterungen des Codes, die nicht implementiert wurde

Analyse der Position:

Code Fen and position viewer v 9.py: https://github.com/rcsdss/this-is-my-first-project

## 6.3) Alte Versionen des Chessbots

Ein Bild, das Text, Screenshot, Kreuzworträtsel, Quadrat enthält.

Automatisch generierte Beschreibung



**Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung**Ein Bild, das Screenshot, Text, Spiele, Hallensportarten enthält.

Automatisch generierte Beschreibung

Abbildung 15 Chessbot v4

Abbildung 16 Chessbot v2

Abbildung 15 Chessbot v1

Abbildung 17 Chessbot v3

## 6.4) Screenshots des Codes (ausgewählte Teile)

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung Für spezifische Informationen siehe den vollständigen beigefügten Quellcode unter https://github.com/rcsdss/this-is-my-first-project

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 17 Code Abschnitt importieren der Bibliotheken

Abbildung 18 Code Abschnitt def update\_advantage

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 19 Code Abschnitt Ende vom Code

# Bescheinigung

Name: Robin Corbonnois & Benjamin Wild

Klasse: BPL21A

Hiermit bestätigen wir, die vorliegende Berufsmaturitätsarbeit mit dem Titel

«Programmieranfänger programmieren ein Schachprogramm»

selbst verfasst zu haben. Das heisst:

* Alle Informationen aus fremden Quellen sind durch die entsprechenden Angaben (Zitate, Quellenverzeichnis) in der Berufsmaturitätsarbeit nach den Vorgaben des Leitfadens zum Zitieren und Belegen gekennzeichnet.
* Die Verwendung generativer KI-Systeme wie ChatGPT ist in der Prozessdokumentation transparent gemacht. Dabei sind folgende Angaben aufgelistet:
  + Datum der Verwendung
  + Name des KI-Tools
  + Zugang zu den Chat-Verläufen
  + Referenz auf Erarbeitungsprozess, schriftliches Produkt und Präsentation
  + Ergänzend zum jeweiligen Nachweis ist eine eigene Einschätzung hinsichtlich Plausibilität und Verwendungszweck aufgeführt.

Ort und Datum: Zürich, 22.11.2024

Unterschrift:

# Dank

Unser besonderer Dank gilt Robins Lehrmeister für seine wertvolle Unterstützung beim Erlernen von Python und Nachlesen der Dokumentation. Ebenso danken wir unsere Lehrerin Frau Eisenring für ihre wertvollen Tipps bei der Projektdurführung sowie unseren Familien für ihren täglichen Rückhalt.