Entwicklung einer künstlichen Intelligenz für das Spiel “Schach” in Python

Studienarbeit

des Studiengangs Angewandte Informatik

an der Dualen Hochschule Baden-Württemberg Mannheim

von

Lars Dittert & Pascal Schröder

29.04.2019

Bearbeitungszeitraum	17.09.2018 - 29.04.2019
Matrikelnummer, Kurs	5388171 & 5501463, TINF16AI-BC
Ausbildungsfirma	IBM Deutschland GmbH, Mannheim
Betreuer	Prof. Dr. Karl Stroetmann

Unterschrift Betreuer _____

Inhaltsverzeichnis

Erklärung der akademischen Aufrichtigkeit

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema:
Entwicklung einer künstlichen Intelligenz für das Spiel "Schach" in Python
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.*

* falls beide Fassungen gefordert sind

Mannheim, 29.04.2019
Ort, Datum

Unterschrift Lars Dittert & Pascal Schröder

Abstract

1 Einleitung

1.1 Zweck und Ziel dieser Arbeit

1.2 Kriterienformulierung

2 Theoretische Hintergründe

Vor dem Erstellen einer Schach-**KI!** (**KI!**) muss zunächst betrachtet werden, wie der Aufbau von KIs für spiel-mechanische Abläufe ist, welche Funktionen diese enthalten müssen und wie diese speziell auf das Spiel "Schach" angewandt werden können. Dazu werden zunächst Grundlagen der Spieltheorie betrachtet ehe näher auf Möglichkeiten zur Nutzung dieser im Kontext des Schachspiels eingegangen wird. Im zweiten Teil werden die verwendeten Methoden für die in dieser Arbeit beschriebene KI erläutert und die Wahl begründet.

2.1 Spieltheorie

2.1.1 Geschichte

Die Geschichte der Spieltheorie im Computerumfeld beginnt mit Claude Shannon im Jahre 1949, als dieser seine Gedanken zur möglichen Realisierung eines Schach spielenden Computers veröffentlicht. Dabei begründet er zunächst seine Auswahl auf das Spiel Schach für sein langfristiges Ziel eines spielenden Computers und legt dann das Ziel an sich fest.

> "The chess machine is an ideal one to start with, since: (1) the problem is sharply defined both in allowed operations (the moves) and in the ultimate goal (checkmate); (2) it is neither so simple as to be trivial nor too difficult for satisfactory solution; (3) chess is generally considered to require 'thinking' for skillful play; a solution of this problem will force us either to admit the possibility of a mechanized thinking or to further restrict our concept of 'thinking'; (4) the discrete structure of chess fits well into the digital nature of modern computers. . . . It is clear then that the problem is not that of designing a machine to play perfect chess (which is quite impractical) nor one which merely plays legal chess (which is trivial). We would like to play a skillful game, perhaps comparable to that of a good human player."

Die wesentlichen Punkte von Claude Shannon sind dabei, dass Schach weder zu simpel noch zu komplex ist, um es von einem Computer spielen zu lassen - Es gibt ein eindeutiges Ziel und eindeutige Operationen auf dem Weg zu diesem Ziel, aber kein eindeutig perfektes Spiel. Dementsprechend kann weder dies das Ziel sein, noch kann es das Ziel sein, einfach

nur ein legales Spiel durchzuführen. Viel mehr muss es mit dem Schachspiel von Menschen vergleichbar sein. Menschen kennen den perfekten Zug nicht, können jedoch eine begrenzte Zahl an Zügen in dessen Wahrnehmungsbereich evaluieren und den in diesem Rahmen scheinbar besten Zug auswählen. Ein ähnliches Verhalten sollte auch bei einem Computer erkennbar sein.

Der vorgeschlagene Ansatz zum Evaluieren der Züge stellt dabei ein Algorithmus dar, der heute unter "Minimax-Algorithmus" bekannt ist. Dieser ruht auf dem "Minimax-Theorem" von John Vonn Neumann, der dieses 1929 beweisen konnte. Dieser Ansatz wird in Kapitel 2.1.2 erklärt.

Dieser Ansatz geht dabei davon aus, dass alle Spieler stets den optimalen Zug spielen. Das Ziel des Computers ist dabei alle möglichen Stati des Spiels zu berechnen, zu evaluieren und schlussendlich den Zug auszuwählen, der das voraussichtlich beste Endergebnis zur Folge hat.

Dies ist für einen Computer leicht durchführbar. Dafür verlangt es folgende Informationen:

- Anfangszustand
- Alle erreichbaren Zustände
- Bewertung jedes einzelnen Zustandes

Um die erreichbaren Zustände zu erhalten, benötigt es einen Algorithmus, der an Hand der Spielregeln und eines Ausgangszustands s_0 alle erreichbaren Zustände nextStates berechnet. Von jedem Zustand enthalten in nextStates wird dann wiederum jeder mögliche erreichbare Zustand berechnet. Diese Schleife wird fortgeführt, bis die berechneten Zustände den Status eines Endzustandes erreichen, von dem aus keine Änderungen des Zustands mehr möglich sind.

Zudem braucht es eine Funktion, die die einzelnen Zustände evaluiert. Am Ende wird der Zug ausgewählt, der den erfolgreichsten Endzustand verspricht.

Dies ist solange umsetzbar, wie der Computer ohne Probleme alle möglichen Zustände berechnen und evaluieren kann. Um dies an einem Beispiel zu zeigen - Beim Spiel "Tic Tac Toe", bei dem in einem 3x3 Feld zwei Spieler gegeneinander antreten mit dem Ziel drei aneinander angrenzende (vertikal, horizontal oder diagonal) Felder mit ihrer Figur zu belegen, gibt es insgesamt 255.168 verschiedene Spielverläufe. Diese sind von heutigen Computern in akzeptabler Zeit berechenbar und evaluierbar.

Dazu bildet der Computer einen Minimax-Baum, wie in Kapitel 2.1.2 beschrieben, und wählt dann den erfolgversprechendsten Zug aus. Nach jedem getätigten Zug werden die

erreichbaren Zustände nur noch vom neuen Zustand aus berechnet. Im Spätspiel kann dies bei “Tic Tac Toe” beispielsweise aussehen wie in Figur 2.1.1.1

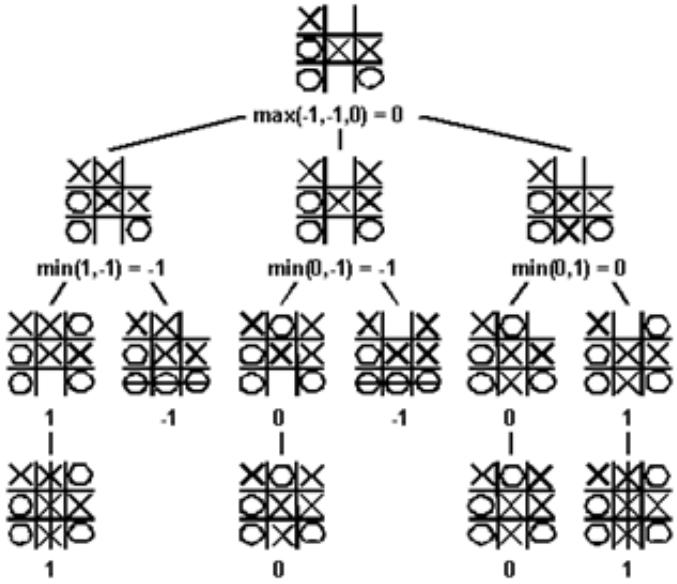


Figure 2.1.1.1 Tic Tac Toe Minimax Baum []

Dabei werden vom Ausgangszustand, der den Zustand des aktuellen Spiels widerspiegelt, aus alle möglichen Folgezustände berechnet. Von diesen werden wiederum alle möglichen Folgezustände berechnet. Dies wird solange fortgeführt, bis alle neu berechneten Zustände Endzustände sind. Dann werden alle Endzustände evaluiert. Eine 1 stellt dabei einen Sieg dar, eine 0 ein Unentschieden und eine -1 eine Niederlage. Da davon ausgegangen wird, dass der gegnerische Spieler stets den besten Zug auswählt, wird jeder Zustand, der noch Folgezustände besitzt, mit dem für ihn schlechtesten Wert aller seiner möglichen Folgezustände bewertet. Der Computer wählt dann den Zustand mit der für ihn besten Bewertung.

In diesem Beispiel wird sich der Computer somit für den dritten Zug von links entscheiden, da bei beiden anderen ein Sieg des Gegenspielers bevorsteht, sollte dieser jeweils die perfekten Züge spielen. Beim dritten Zug kann der Gegner maximal noch ein Unentschieden erreichen.

Diese Strategie gerät jedoch dann an ihre Grenzen, wenn nicht mehr alle möglichen Zustände berechnet werden können. Bei dem Spiel Schach beispielsweise belaufen sich Schätzungen schon nach den ersten 40 Spielzügen auf 10^{115} - 10^{120} verschiedene Spielverläufe. Dies ist auch für einen Computer in tolerierbarer Zeit unmöglich berechenbar. Aus diesem Grund muss die Strategie für solch komplexere Spiele abgewandelt werden. Die verschiedenen Ansätze dazu sind in Kapitel 2.1.3 beschrieben.

2.1.2 Minimax-Algorithmus

2.1.3 Problematik Minimax-Algorithmen komplexerer Spiele

2.2 Verwendete Methoden

- Ealuierungsfunktionen - weiß + schwarz - - Minimax/alpha beta

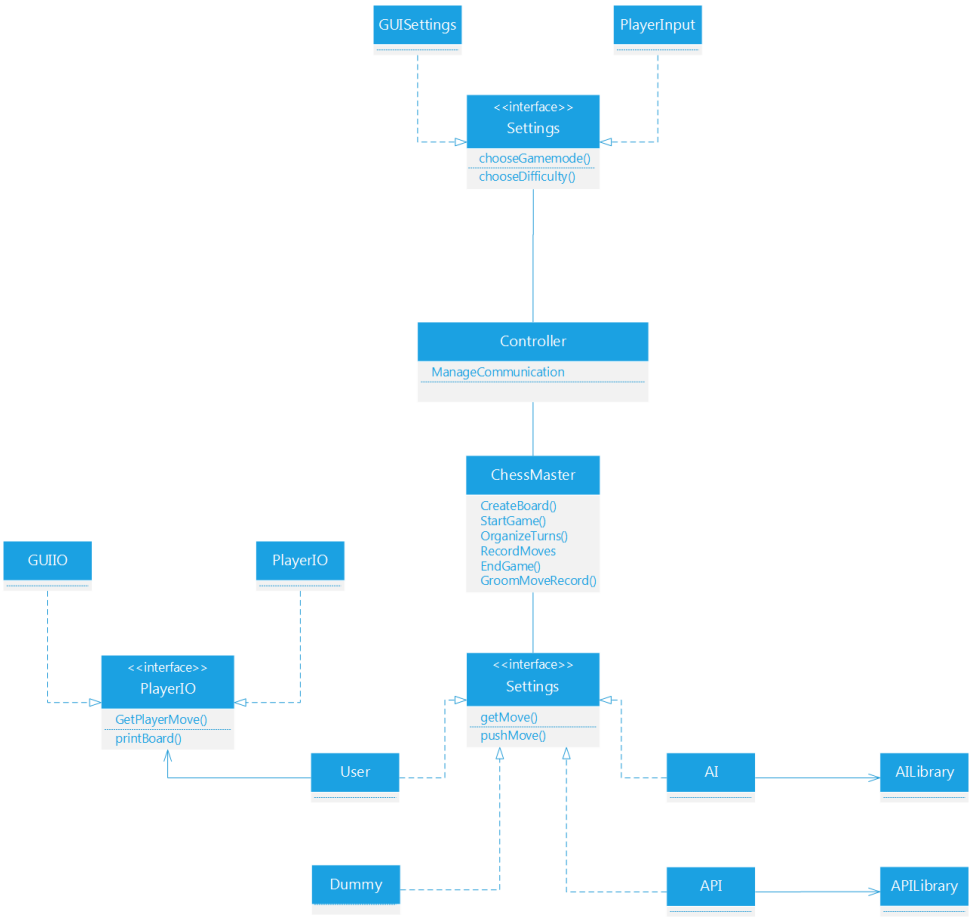
3 Technische Grundlagen

3.1 Verwendete Bibliotheken

3.2 Architektur

Um die Ziele der Erweiterbarkeit, Wartbarkeit und einfachen Verständlichkeit zu erfüllen, ist eine modulare Aufbauweise des Projektes von Nöten. Die auf diesen Prinzipien beruhende Architektur wird im folgenden Kapitel beschrieben.

Zunächst wurde das Programm der Aufgaben entsprechend in einzelne Klassen unterteilt, die verschiedene Aufgaben zugewiesen bekommen haben. Diese Aufteilung kann Figur 3.2.1 entnommen werden.



Figur 3.2.1 Klassendiagramm Architektur ChessAI

Als zentrales, Verbindungselement steht zunächst der “Controlle”. Dieser übernimmt die Abfrage aller Optionen, falls diese nicht beim Start des Programms mit übermittelt werden. Zudem initialisiert der Controller die Spielteilnehmer sowie den “ChessMaster”. Bei letzterem übergibt der Controller diesem die initialisierten Spielteilnehmer. Die Aufgaben des “ChessMaster” wird später genauer beschrieben.

Zur Abfrage der Optionen steht dem Controller eine ‘Settings’ Schnittstelle zur Verfügung. Diese wird für Eingaben des Nutzers einerseits über eine Konsole als auch optional über eine grafische Nutzeroberfläche implementiert. Dabei wird abgefragt, welchen Typ die jeweiligen Spielteilnehmer annehmen sollen und es können zudem zusätzliche Optionen für die einzelnen Spieler definiert werden. Als Beispiel kann für Spieler 1 der Typ “User” gewählt werden und für Spieler 2 der Typ “AI”. Dies ermöglicht ein Spiel des Nutzers gegen die im Rahmen dieser Arbeit entwickelte KI. Für die KI wird darauf folgend noch der Schwierigkeitsgrad abgefragt. Zusätzlich kann für jeden Spieler ein Name festgelegt werden.

Die Aufgaben des “ChessMaster” erstrecken sich über die Verwaltung des Schachspiels an sich, das Ansprechen der jeweiligen Spieler zum Ermitteln ihrer Züge sowie dem Durchführen des gewählten Spielzugs auf dem Schachbrett. Zusätzlich speichert es jedes Schachbrett, das sich im Laufe des Spiels ergibt, und fügt diese zur “board_history.” Datei hinzu. Diese speichert alle Schachbretter gemeinsam mit einem numerischen Wert. Dieser gibt Aufschluss über Erfolgsaussichten der jeweiligen Akteure des Spiels. Dazu wird nach jedem Spiel zu dem entsprechenden Eintrag in der Datei eine eins zu dem alten Wert aufaddiert, wenn der Spieler der weißen Figuren das Spiel gewonnen hat und eine eins subtrahiert, wenn der Spieler der schwarzen Figuren das Spiel gewonnen hat. Bei einem Unentschieden bleibt der alte Wert bestehen. Dies hilft der KI bei der Bewertung eines Schachbretts unter zur Hilfenahme von statistischen Werten.

Dieser spricht die vom Controller erstellten Spieler an. Diese können, wie bereits angedeutet, von verschiedenen Typen sein. Zur Auswahl stehen

- User - Ein menschlicher Akteur kann Züge über eine Nutzerschnittstelle eingeben
- AI - Die künstliche Intelligenz versucht den best möglichsten Zug zu berechnen
- Dummy - Ein zufälliger Zug wird ausgewählt
- (Optional) API - Ein Zug wird über eine Schnittstelle zu einer Online Schachplattform, auf der menschliche sowie künstliche Spieler teilnehmen dürfen, bestimmt

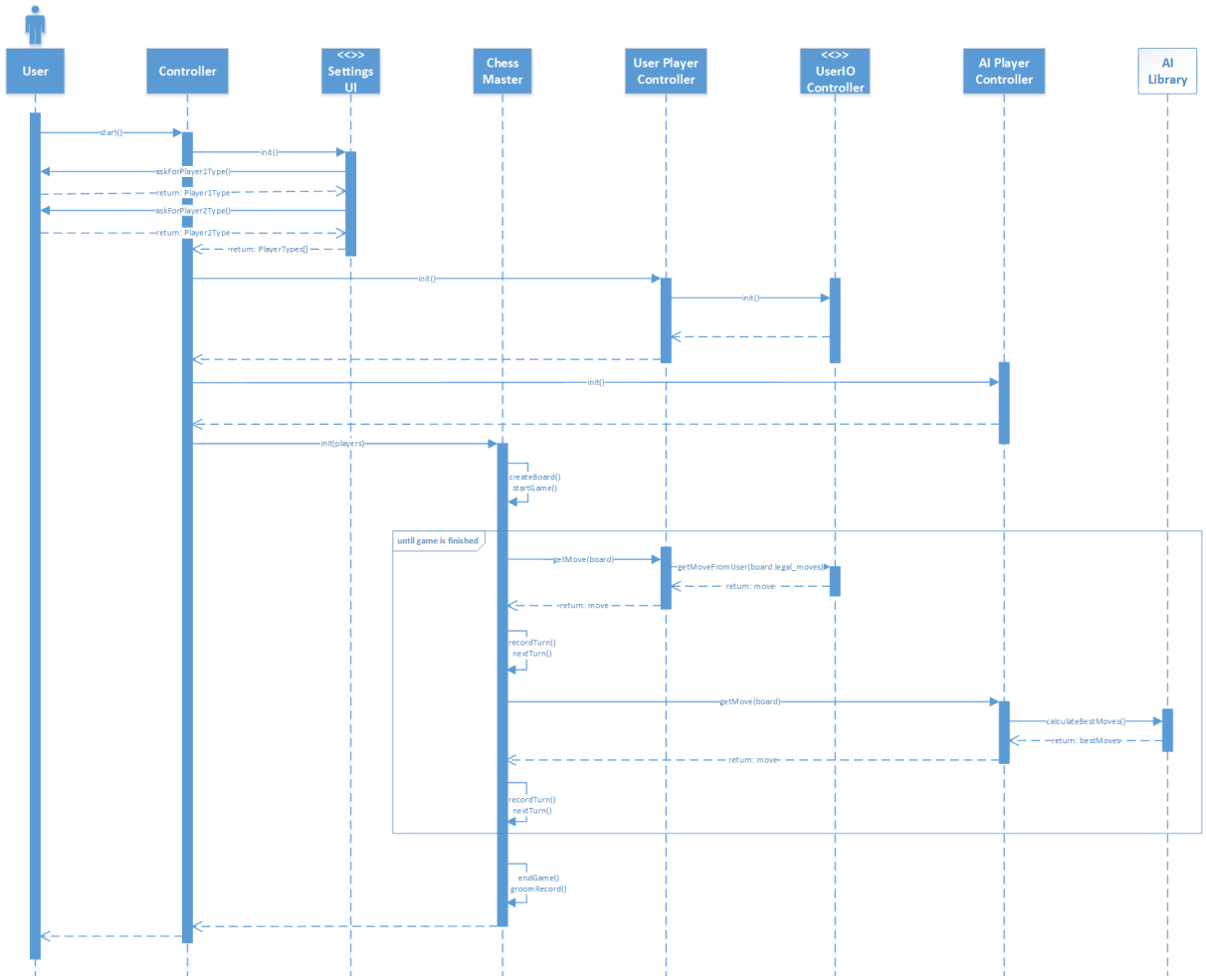
Dazu existiert eine “Player” Schnittstelle, die für jeden dieser Spieler implementiert werden muss.

Die “User” Implementation greift dabei zur Ermittlung des Zuges auf eine “PlayerIO” Schnittstelle zurück. Diese wiederum kann - ähnlich zur “SettingsUI” Schnittstelle - sowohl für Eingaben über eine Konsole als auch für Eingaben über eine grafische Nutzeroberfläche implementiert werden.

Die “AI” sowie die “API” Implementation greifen für die Ermittlungen ihrer Züge nochmals auf eigens erstellte Bibliotheken zurück, die elementare Funktionen erhalten.

Der genaue Ablauf der Ermittlung der Züge sowie der andere Operationen des Programmes werden in Kapitel XY.ZY näher erläutert.

In Figur 3.2.2 ist der sequenteille Ablauf der Funktionsaufrufe erkennbar.



Figur 3.2.2 Sequenzdiganagramm Architektur ChessAI

Dabei wird zunächst über die “Settings” Schnittstelle nach den Spielertypen sowie Namen gefragt. initialisiert der “Controller” die Spieler, in diesem Fall einerseits den “User”, der wiederum die “UserIO” Schnittstelle implementiert, und außerdem den “AI” Player.

Darauffolgend spricht der “Controller” den “ChessMaster” an. Dieser startet nun das Spiel und fragt solange wie das Spiel nicht vorbei ist immer abwechselnd erst zu Spieler 1 - dem

Nutzer - und dann zu Spieler 2 - der KI - nach dem nächstne Zug. Dabei übergibt der “ChessMaster” stets das aktuelle Schachbrett. Der Nutzer ermittelt den durchzuführenden Zug über eine Abfrage an den Nutzer über die entsprechende Schnittstelle, der “AI” Spieler, indem dieser Funktionen aus der entsprechenden Bibliothek zur Hilfe nimmt.

Nach jedem Zug fügt der “ChessMaster” diesen zum Schachbrett hinzu und speichert dieses in einen Spielverlauf. Nach Ende des Spiels wird dieser in das “board_history.csv” eingepflegt wie weiter oben bei der Beschreibung der Klasse bereits erläutert.

Abschließend beendet der “ChessMaster” das Spiel, wenn keine Revance gewünscht ist, und so wird auch das Programm im Anschluss daran geschlossen.

4 Implementation

5 Evaluation

5.1 Kriterienerfüllung

5.2 Einordnung Intelligenz

Abkürzungsverzeichnis

KI Künstliche Intelligenz

Literatur