Einleitung

Die Herausforderung, einen Computer zum Schachspielen zu befähigen, beschäftigte bereits Alan Turing und seine Studenten. Was das Schachspiel besonders macht, ist die enorme Vielfalt möglicher Spielverläufe. Schätzungen zufolge gibt es nach 40 Zügen etwa 10^{120} verschiedene mögliche Stellungen auf dem Brett - eine Zahl, die sogar die geschätzte Anzahl der Atome im beobachtbaren Universum (etwa 10^{80}) übertrifft.

Die Berechnung optimaler Schachzüge ist eine immense Herausforderung, die selbst modernste Computer an ihre Grenzen bringt. Eine vollständige Durchsuchung aller möglichen Spielverläufe wäre mit heutiger Rechenleistung nicht realisierbar. Ein einfacher Brute-Force-Ansatz ist daher praktisch unmöglich.

Daher widmet sich dieses Studienprojekt der Analyse, wie moderne Schach-KIs diese Problematik bewältigen und welche faszinierenden Algorithmen und Heuristiken sie dabei verwenden. Es werden auch eine Architektur einer einfachen Schach-KI vorgestellt und umgesetzt, um einen näheren Einblick in die Funktionsweise und Programmierung einer Schach-KI zu geben.

Wie funktioniert Stockfish?

Stockfish zählt zu den bekanntesten und leistungsfähigsten Schach-Engines weltweit. Als Open-Source-Programm wird es kontinuierlich von einer aktiven Community weiterentwickelt. Es verfügt über die typischen Komponenten einer Schach-Engine wie Suchalgorithmus, eine Bewertungsfunktion, Bitboard-Repräsentation und heuristische Verfahren.

Mini-Max Algorithmus:

Die Spieltheorie bennent Spiele, bei denen die Summen der Gewinne und Verluste aller Spieler gleich null ist, als Nullsummenspiele.
Mini-Max ist eins der gängigen Algorithmen der bei Nullsummenspielen wie Schach oder Tic-Tac-Toe eingesetzt wird. Stockfish setzt Mini-Max ein um mögliche Züge als Knoten in einem Suchbaum darzustellen, und weist den Knoten mithilfe einer Bewertungsfunktion eine Zahl zu, die die Qualität des Zuges repräsentiert. Das Besondere an diesem Algorithmus ist, dass er nicht nur den aktuell besten eigenen Zug sucht, sondern auch berücksichtigt, welcher Zug den Gegner in seinen zukünftigen Optionen möglichst stark einschränkt. Somit übernimmt ein Spieler die Rolle des *Maximierers*, der darauf abzielt, Züge mit dem höchstmöglichen Bewertungswert zu erreichen. Der Gegenspieler wird im Gegensatz zum *Minimierer* und versucht, den niedrigstmöglichen Wert zu erzwingen – also die für den Maximierer ungünstigste Fortsetzung. Dadurch wird nicht nur der eigene Vorteil gesucht, sondern zugleich der Gegner

eingeschränkt.
In der Schachbewertung gilt dabei: Positive Werte deuten auf einen Vorteil für Weiß, negative Werte auf einen Vorteil für Schwarz.

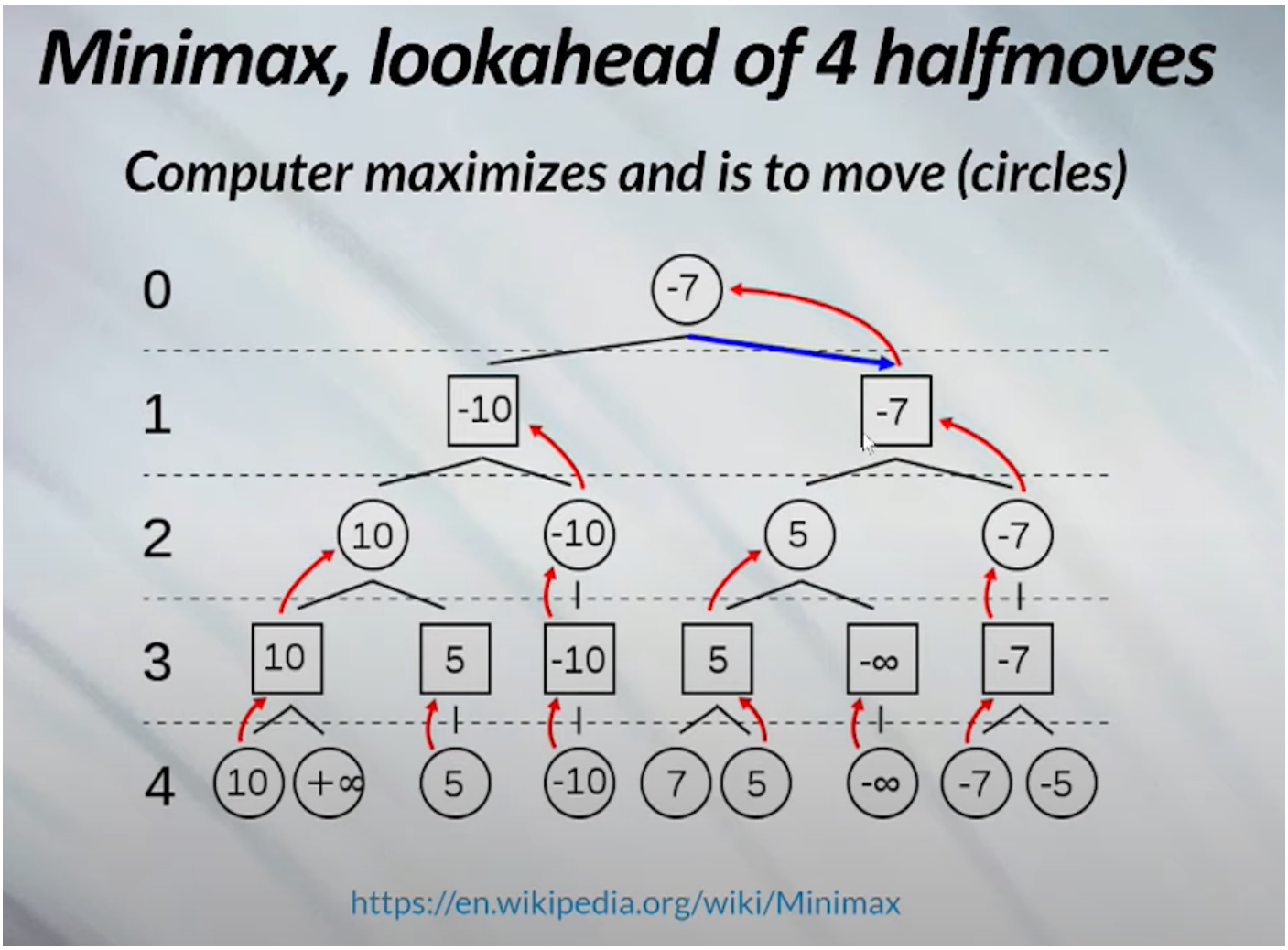

Abbildung 1.1: Minimax-Suchbaum mit Tiefe 4

Minimax erzeugt den Suchbaum, indem es eine festgelegte Tiefe vorausschaut. Mithilfe von **Alpha-Beta-Pruning** werden Zweige eliminiert, die voraussichtlich keine vielversprechenden Stellungen liefern können. **Abbildung 1.1** zeigt einen Beispielbaum, der die Berechnung des optimalen Zuges veranschaulicht. Weiß (runde Knoten) ist am Zug und strebt danach, den Wert zu maximieren. In Tiefe 4 ist ein Zug mit dem Wert **+∞** sichtbar, der Weiß wahrscheinlich den Sieg sichern würde. Allerdings blockiert Schwarz (eckige Knoten) diesen Zug als Minimierer und wählt stattdessen Knoten 10. Das Blatt mit dem Wert 10 wird daher hochgeschoben bzw. „rückpropagiert“. Dieser Prozess wird rekursiv für alle Knoten wiederholt, bis die Werte an die Wurzel des Baums zurückgegeben werden. Dort trifft der Spieler die Entscheidung, sich für den minimalen oder, wie in diesem Fall, den maximalen Wertknoten zu entscheiden. Schwarz wählt somit den Zug mit der Bewertung -7, da er dort für sich das lokale Maximum sieht.

Ein großer Nachteil bei der vollständigen Analyse **aller** möglichen Spielzüge ist der rapide ansteigende Rechenaufwand, da der Suchraum mit jedem Zug enorm zunimmt. Geht man beispielsweise von durchschnittlich 30 möglichen Zügen pro Stellung aus und nimmt an, dass ein Programm 50.000 Stellungen pro Sekunde berechnen kann, ergeben sich folgende beispielhafte Suchzeiten:

Tiefe (ply)	Anzahl Stellungen	Suchzeit
2	900	0.018 s
3	27,000	0.54 s
4	810,000	16.2 s
5	24,300,000	8 Minuten
6	729,000,000	4 Studen
7	21,870,000,000	5 Tage

Es ist ersichtlich, dass der Suchbaum schnell extrem groß wird trotz einer schnellen Bewertungsfunktion. Eine zuverlässige Suche der Tiefe 6 bis 7 ist jedoch Voraussetzung für eine gute Schach-KI. Dies verdeutlicht die Notwendigkeit, die Größe des Suchbaums erheblich zu reduzieren. Hierfür wird das Alpha-Beta-Suchverfahren angewendet.

Alpha Beta Pruning:

Alpha-Beta Pruning ist ein Algorithmus, der dazu dient, Zweige eines Suchbaums zu ignorieren, die nicht zu einem vorteilhaften Zug führen. Dabei werden die Parameter **α (Alpha)** = `-∞` und **β (Beta)** = `+∞` initialisiert.
Im Verlauf der rekursiven Suche repräsentiert **Alpha** den aktuell **besten (höchsten)** Wert, den der **Maximierer** garantieren kann, während **Beta** den **niedrigsten** Wert darstellt, den der **Minimierer** noch zulässt.
Sobald Beta ≤ Alpha gilt, kann der entsprechende Teilbaum "abgeschnitten" werden, da er keine bessere Entscheidung mehr ermöglichen kann bzw. weil in einem anderen Zweig ihm ein besseres Ergebnis garantiert werden kann.

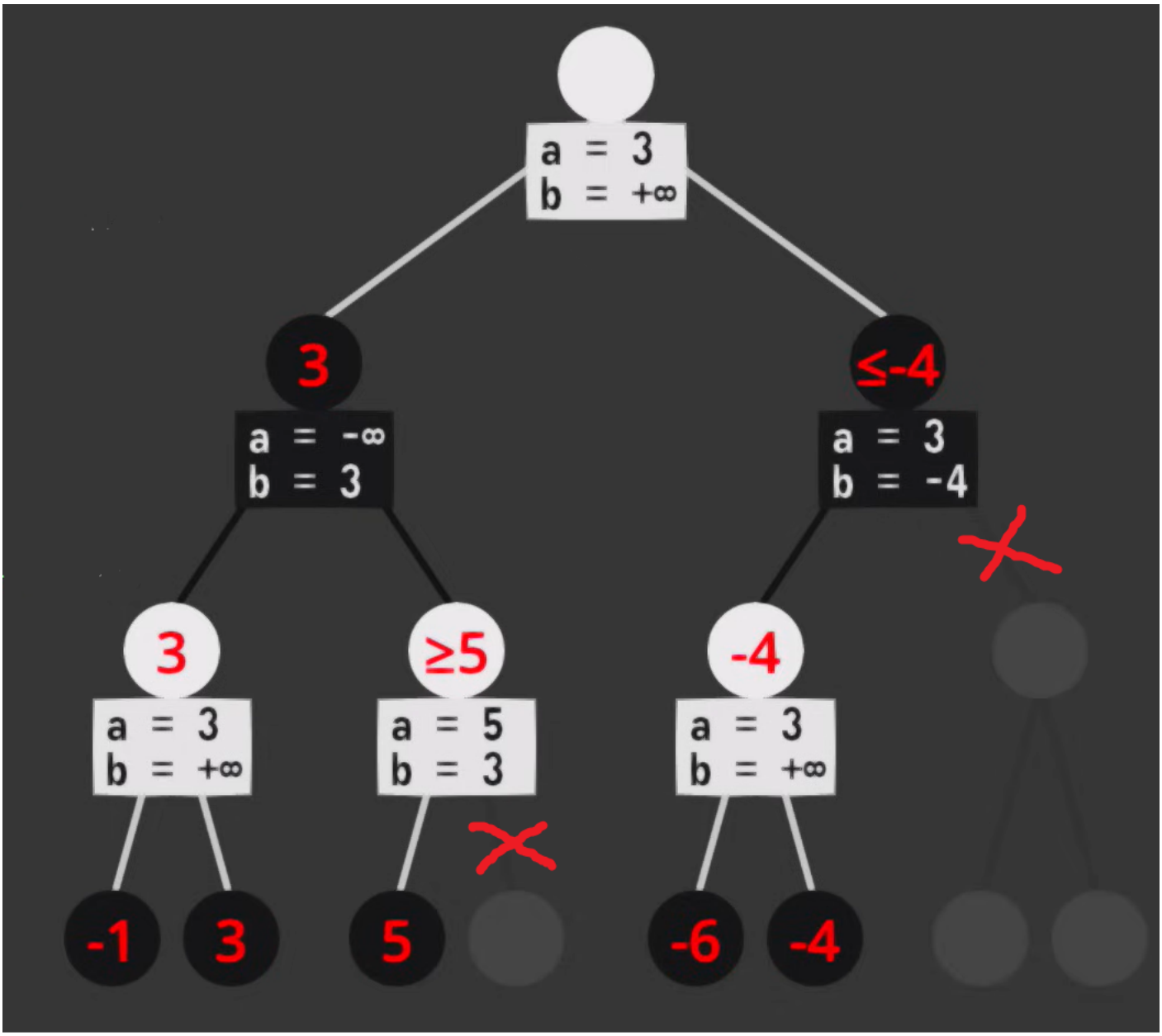

Abbildung 1.2: Alpha-Beta Suchbaum mit eliminierten Zweigen

Abbildung 1.2 verdeutlicht, wie eine Eliminierung eines Teilbaums (*Pruning*) aussehen kann. Zunächst betrachtet der Algorithmus die Blätter:

- Das erste Blatt hat den Wert **-1** → ist `-1 > α`? **Ja!** → also wird `α = -1`, `β = +∞` bleibt unverändert.
- Dann wird das Blatt mit dem Wert **3** betrachtet → ist `3 > α`? **Ja!** → also wird `α = 3`, `β = +∞` bleibt ebenfalls unverändert.
  Jetzt wird geprüft, ob die Pruning-Bedingung `β < α` erfüllt ist → hier nicht erfüllt ❌, also kein Pruning.

Die Werte `α = -∞` und `β = 3` werden an den Vaterknoten übergeben, und der rechte Teilbaum wird untersucht.

- Das Blatt hat den Wert **5**, und da `5 > 3`, wird `α = 5` gesetzt.
  Jetzt ist `β < α` **erfüllt**: ✅
  Der Zweig kann abgeschnitten werden, da Schwarz im linken Zweig bereits eine bessere Option (**3**) garantiert hat.

Dieser Zweig wird also eliminiert, da der Minimax-Algorithmus davon ausgeht, dass der Gegner den bestmöglichen Gegenzug spielt.