Grundpfeiler einer klassischen Schach-KI

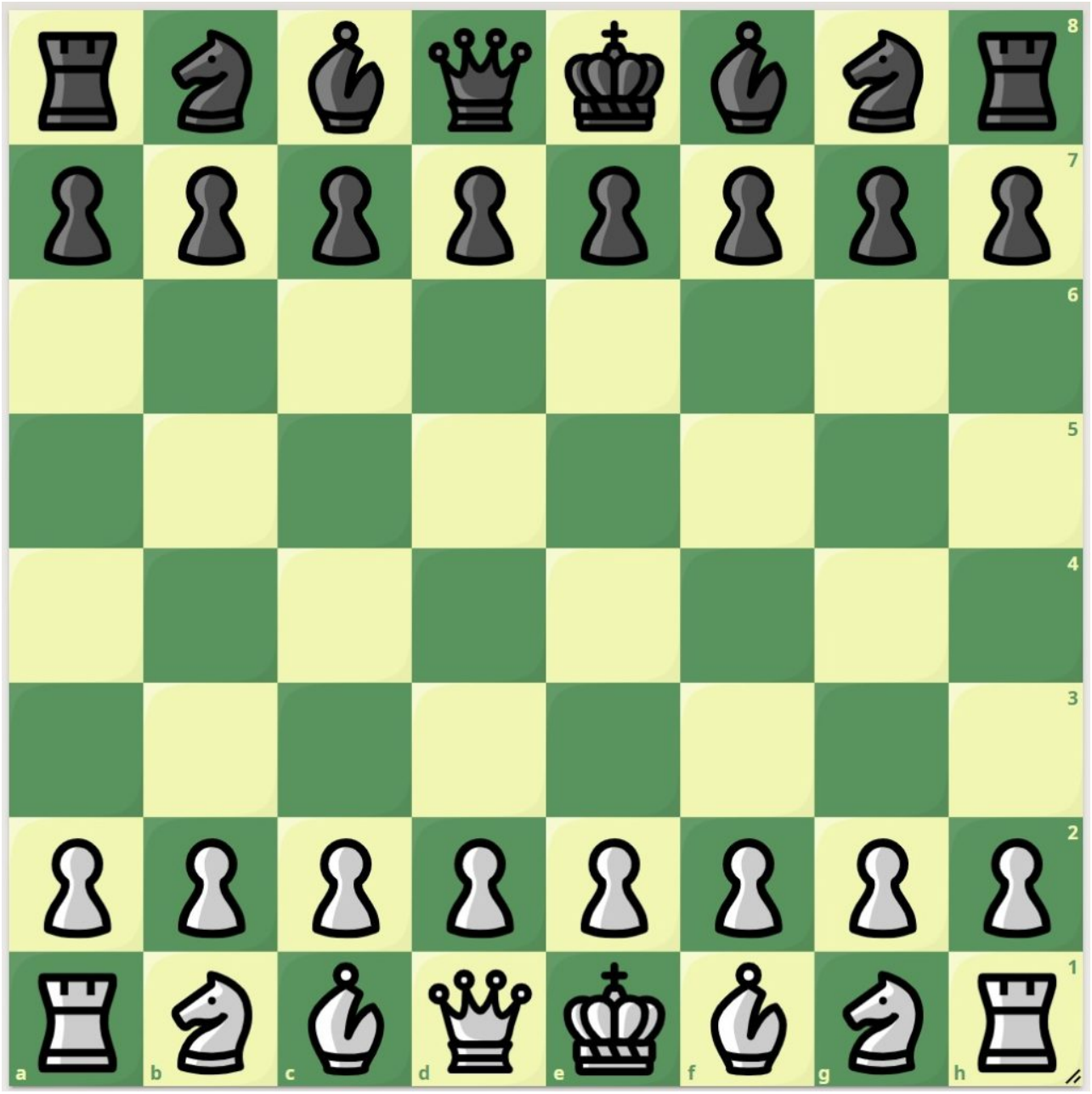
Description

Seit ein paar Monaten habe ich ein neues Hobby. Ich programmiere eine Schach-KI. Und wie das mit neuen, aufregenden Dingen so ist, erzählt man gerne darüber. Bei den Gesprächen, die eigentlich überwiegend Monologe sind, werden mir manchmal Fragen bezüglich der Denkweise einer solchen KI gestellt. Darauf möchte ich nun ein wenig eingehen. Wie funktioniert eine klassische Schach-KI?

Programmierer werfen zwei Begriffe in den Raum. „Alpha-Beta-Suche“ und „Minimax-Algorithmus“. Daraus besteht eine Schach-KI. Technisch gesehen ist das richtig, aber unvollständig. Diese beiden Mechanismen sorgen dafür, dass die KI Züge findet, ohne alle Möglichkeiten berechnen zu müssen. Ob gute Züge herauskommen, hängt von ganz anderen Dingen ab.

Denken

Ein Anfang muss her. Am besten startet man damit, dass die KI die Regeln kennt. Ein Brett von 64 Feldern lässt sich mit Arrays gut realisieren. Die einzelnen Figuren, Bewegungsmöglichkeiten, Schlagen, Zug zurück und dann kommen die Sonderregeln wie Rochade und der Bauer mit Doppelzug, Umwandlung und en passant. Wann ist Schach? Welche Regeln gelten hier? Matt, dreifache Stellungswiederholung, 50-Züge Regel.



Man fängt mit offensichtlichen Dingen an. Brett, Aufbau und Regeln.

Eine simple KI ist damit schon beendet. Jetzt braucht man nur noch eine kleine Funktion, die zufällig einen möglichen Zug auswählt. Ende der Geschichte.

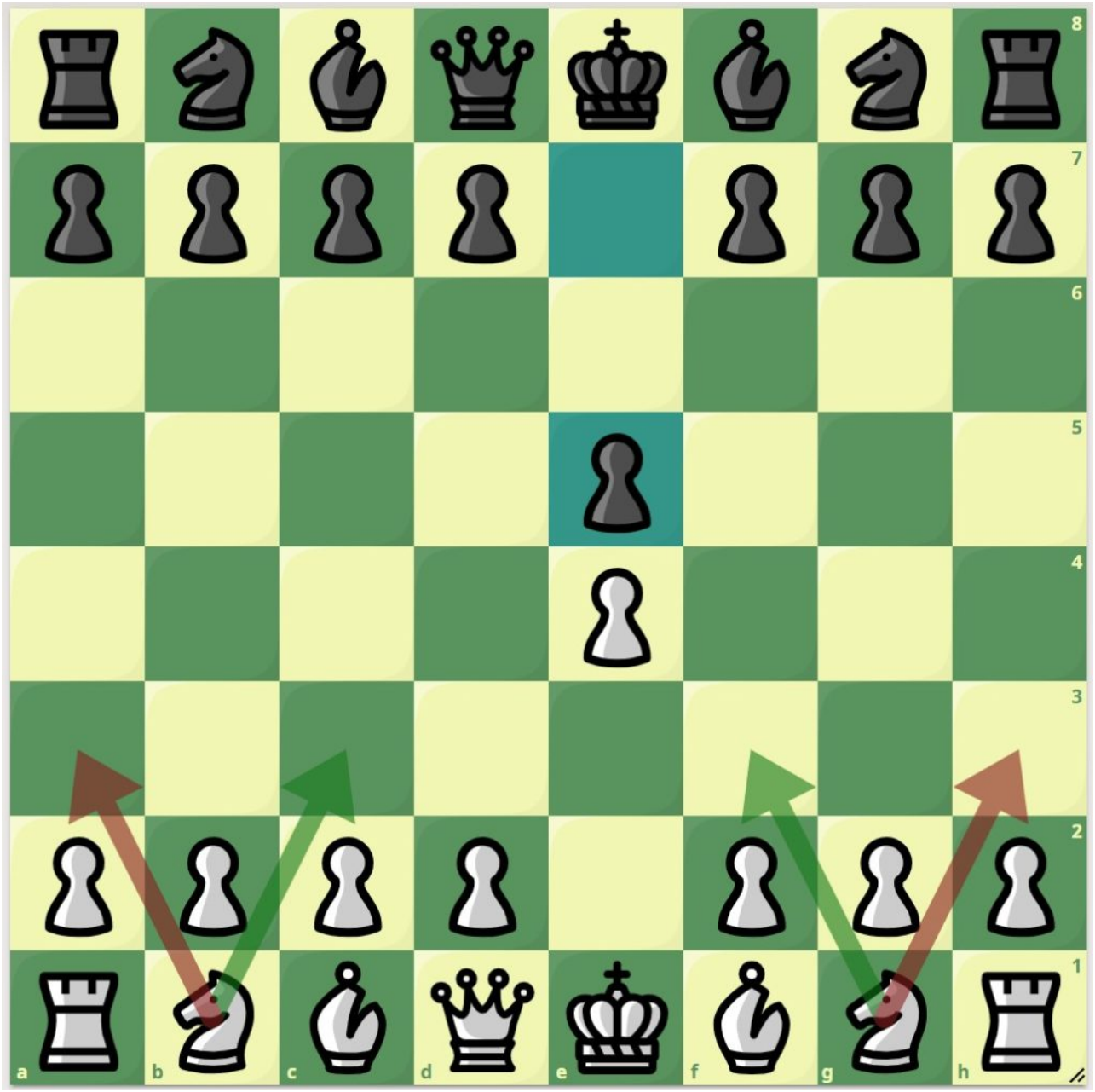
Das Resultat ist aber furchtbar. Also weiter.

Suchen

Irgendeinen Zug zu machen ist somit nicht die Lösung. Wir brauchen eine Möglichkeit, alle durchführbaren Züge zu sammeln und zu bewerten. Hier spielen die beiden o. g. Algorithmen eine wichtige Rolle. Sie sucht sämtliche Züge, die nach den Regeln erdenklich sind. Dann werden diese bewertet. Ist ein Zug gut bzw. weniger schlecht als andere, geht die KI weiter in die Tiefe.

Bei Schach-KIs denkt man in Halbzügen. Bauer *e2* auf *e4* ist ein solcher. Wenn Schwarz *e7-e5* zieht, ist es ein weiterer. Wie viele Halbzüge eine KI schafft, hängt von zahlreichen Faktoren ab. Prozessor, Speicher, Zugzeit, aber auch die Programmiersprache. Die meisten KIs mit einer Spielstärke von 2000 Elo schaffen im Blitzschach zwischen 7 und 14 Halbzüge. Meine KI liegt bei 9 bis 12, je nach Zeit. Die Weltspitze bei 20 bis 30. Das ist durch Optimierungen und Multithreading möglich.

Es ist aber nicht automatisch so, dass eine KI mit einer höheren Suchtiefe gegen eine mit geringerer gewinnt.



Gute und schlechte Züge, aber wie entscheidet das die KI?

Bewerten

Die Bewertung ist das A und O. Ich kann 100 Halbzüge im voraus denken, wenn die Beurteilung schrott ist, dann bringt das nichts. Doch wie kann eine KI zwischen guten und schlechten Zügen unterscheiden?

Mit Zahlen. Sogar mit sehr vielen Zahlen. Man fängt zunächst damit an, die Figuren zu werten. Etwa so, wie wenn man Schach lernt. Da hat der Bauer einen, Springer und Läufer drei, Turm fünf und die Dame neun Punkte. Der König ist unsterblich und hat keine Bewertung.

Da die Zahlen zu klein sind, um praktikabel zu sein, multipliziert man sie mit hundert. Nimmt man alleine dies in die Berechnung mit auf, führt es bspw. dazu, dass die KI eine Dame nicht mehr gegen einen Bauern tauschen würde, außer es führt zum Matt oder soll eines verhindern. Durch diese zentrale Bewertung versteht die KI automatisch die taktischen Motive. Sie spielt die möglichen Kombinationen durch und merkt, dass sie am Ende Material gewinnt. Deswegen sind KIs auf taktischer Ebene so stark.

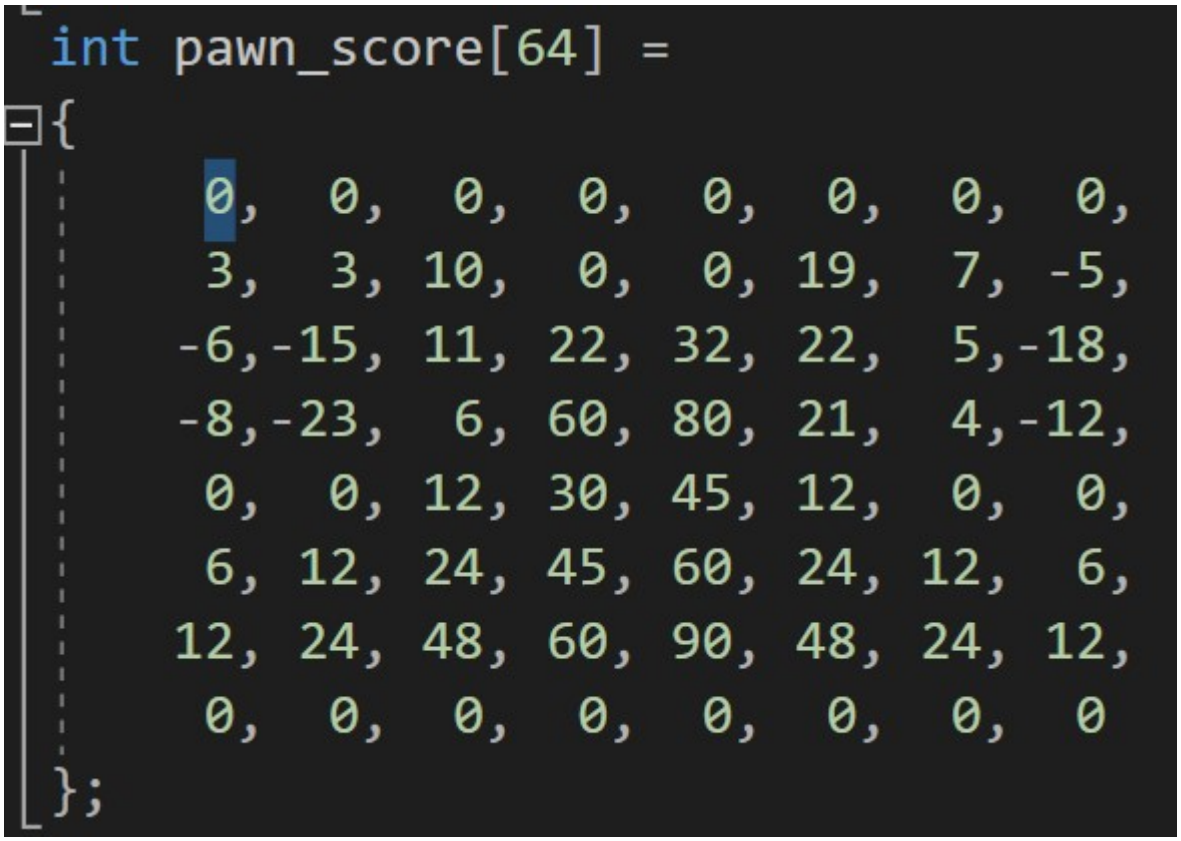
Doch woher weiß die KI, welche Felder besser und welche schlechter sind?

Gedanken zu Zahlen

In der Schachschule lernt man darüber viele Dinge. Bauern, vor allem im Zentrum, sind besonders stark. *„Springer am Rand bringt Kummer und Schand!“* Oder *„Ein Bäuerlein, das vorgeprellt, hat oft dem Feind ein Bein gestellt.“*

Man darf jetzt aber nicht erwarten, dass man diese Sätze eingibt und sich die KI daran hält. Wir müssen es in Zahlen umwandeln.

Wie erwähnt, haben wir ja ein Array mit 64 Feldern. Also können wir jedes Feld für jede Figur mit Zahlen bewerten. Um beim Zug *e2-e4* zu bleiben: *e1* ist 0, weil der Bauer hier nicht stehen kann. *e2* ist eine relativ niedrige Nummer, *e3* etwas größer und *e4* ist hoch. Die KI rechnet Positionen mit *e2*, *e3* und *e4* durch und stellt fest, dass die Evaluierung für den Ackersmann auf *e4* am besten ist.



```
int pawn_score[64] =
{
     0,  0,  0,  0,  0,  0,  0,  0,
     3,  3, 10,  0,  0, 19,  7, -5,
    -6,-15, 11, 22, 32, 22,  5,-18,
    -8,-23,  6, 60, 80, 21,  4,-12,
     0,  0, 12, 30, 45, 12,  0,  0,
     6, 12, 24, 45, 60, 24, 12,  6,
    12, 24, 48, 60, 90, 48, 24, 12,
     0,  0,  0,  0,  0,  0,  0,  0
};
```

Werte für den Bauern bei der Eröffnung. Die markierte 0 ist a1.

Das Gleiche passiert mit allen anderen Figuren. Springer am Rand oder in der Ecke haben sehr niedrige Zahlen. Die Felder *c3* und *f3* sind deutlich stärker bewertet. Also zieht die KI vorwiegend auf diese, statt auf *a3* oder *h3*.

Die meisten guten KIs fassen diese Werte in Spielphasen zusammen. Größtenteils bilden Eröffnung und Mittelspiel eine Tabelle, Endspiel eine zweite. Ich bewerte derzeit alle drei Spielphasen einzeln. Ob das wirklich besser ist, weiß ich noch nicht.

Die Stellungsbeurteilung funktioniert im Kern wie folgt:

- Die KI macht gedanklich einen Zug.
- Sie zählt die Werte der Figuren zusammen.
- Sie zählt die Werte der Felder zusammen, auf der die Figuren stehen.
- Nun macht sie das Gleiche mit dem Gegner.
- Vom eigenen Wert wird der Wert des Gegners abgezogen.

Ist die Zahl positiv, steht die KI besser dran. Ist sie negativ, steht der Gegner besser.

```
int rook_score[64] =
{
     0,  0,  0, 20, 20,  0,  0,  0,
     0,  0,  0, 10, 10,  0,  0,  0,
     0,  0,  0, 10, 10,  0,  0,  0,
     0,  0,  0, 10, 10,  0,  0,  0,
     0,  0,  0, 10, 10,  0,  0,  0,
     0,  0,  0, 10, 10,  0,  0,  0,
    25, 25, 25, 25, 25, 25, 25, 25,
     5,  5,  5, 10, 10,  5,  5,  5
};
```

Werte des Turms für die Eröffnungsphase

Sondersituationen

Das ist der Kern. Damit kommt man relativ weit. Aber irgendwann stellt man fest, dass es nicht reicht. Also fängt man an, bestimmte Stellungstypen und Situationen einzeln zu bewerten. Hier ein paar Beispiele:

- Zwei Läufer ergeben einen Bonus, da das Läuferpaar eine besondere Kraft ausstrahlt.
- Turm auf einer halboffenen oder offenen Linie bekommt einen Bonus.
- Zwei Türme auf einer solchen Linie ergeben einen weiteren Bonus.
- Die Möglichkeit zu rochieren ergibt wieder einen Bonus.
- Doppel- und Dreifachbauern sind schlecht.

Diese und weitere Zahlen fließen in die Berechnung mit ein und können sogar an Spielphasen gekoppelt sein. Die Kunst besteht darin, diese Werte (bei mir sind es derzeit rund 1400) so auszubalancieren, dass am Ende ein guter Zug rauskommt.

```
// Opening and Middlegame values
#define PAWNVALUEOP   220
#define KNIGHTVALUEOP 780
#define BISHOPVALUEOP 780
#define ROOKVALUEOP   1100
#define QUEENVALUEOP  2145

// Endgame values
#define PAWNVALUEEG   220
#define KNIGHTVALUEEG 735
#define BISHOPVALUEEG 802
#define ROOKVALUEEG   1150
#define QUEENVALUEEG  2180
```

Im Endspiel verschiebt sich die Bewertung der Figuren ein wenig

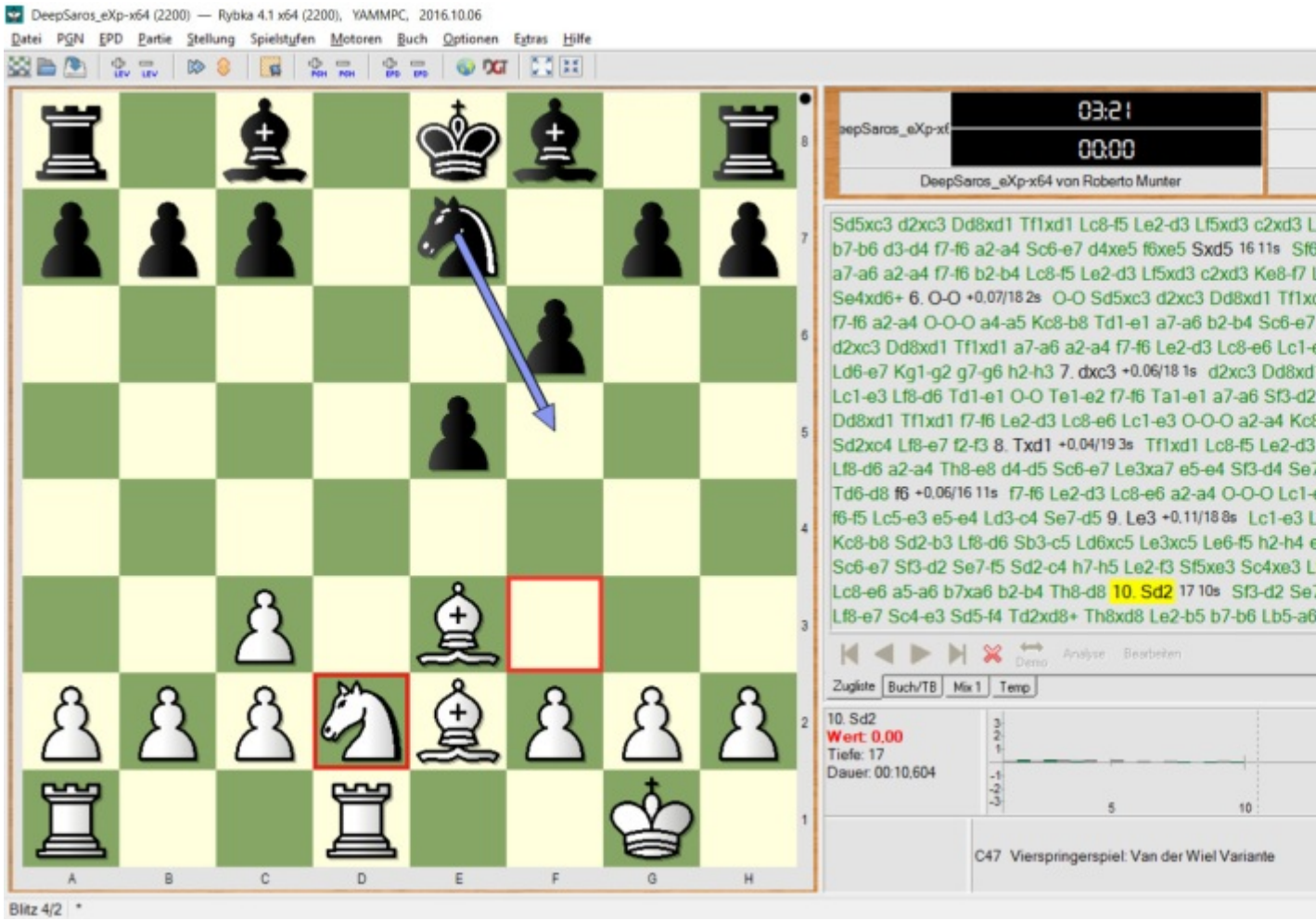
Kommunizieren

Warum das so viele Zahlen sind? Ein Spieler hat sechs verschiedene Figuren. Alleine für die Eröffnung brauchen wir 6 x 64 Werte. Damit sind wir bei 384. Ziehen wir die erste Reihe beim Bauern ab, bleiben 376. Bei drei Spielphasen sind wir bei 1128 Einzelwerten. Dann kommen die Figurenwerte, Sondersituationen und vielleicht noch Mobilität hinzu.

Aber wie will man das testen? Früher setzten sich gute Schachspieler dran und die haben gegen die KI gespielt. Anschließend wurden die schlimmsten Fehler analysiert, um dann zu versuchen, mit anderen Zahlen zu einem besseren Resultat zu gelangen. Wenn man mit einer KI beginnt, sollte man ebenso anfangen. Ich habe hierfür Brett und Züge im Textmodus ausgegeben, um gegen die KI zu spielen. Doch irgendwann kostet es zu viel Zeit. Oder die KI ist besser als man selbst. Die Lösung lautet „Kommunikationsprotokoll“.

Im Schach haben sich zwei Protokolle durchgesetzt. XBoard und UCI. Damit ist es möglich, dass eine KI mit einer beliebigen GUI interagiert. Das bedeutet, dass man die KI in eine grafische Oberfläche einbinden kann, ohne sich um die Grafik zu kümmern. Das alleine ist nicht die Lösung. Einige dieser Programme bieten aber die Möglichkeit, dies mit zwei KIs zu tun. Diese können sich dann gegenseitig

die Köpfe einschlagen, ohne das man selbst etwas tun muss. Man kann sogar ganze Turniere austragen. Einmal alles eingestellt schaut man nur noch zu, bis der Wettbewerb vorbei ist. Die Spiele können gespeichert, exportiert und analysiert werden.



Arena Chess GUI 3.5.1

Testfeld

Viele Wege führen nach Rom. Manche sind der Meinung, man sollte eine KI nur gegen sich selbst spielen lassen. Bei neuronalen Netzwerken ist das sogar die Norm. Andere behaupten, es sei am schlausten, sie nur gegen die beste KI antreten zu lassen.

Ich habe mehrere Dinge versucht. Zunächst gegen sich selbst, weil ich dadurch die ganz groben Fehler finden konnte. Schließlich wertet man mit jeder Partie beide Farben aus. Anschließend stellte ich ein Testfeld mit KIs zusammen, die in etwa gleich gut oder etwas besser waren. Die Turniere erfolgten nur im Blitzschach, fünf Minuten pro Seite. So bekommt man schnell Resultate. Die Partien wurden auf lichess.org importiert und analysiert. Ich schrieb dann die groben Fehler in Worten auf.

„Die KI rochiert viel zu spät.“
„Der König wandert sinnlos herum.“
„Turm a2 ist ein unsinniger Zug!“

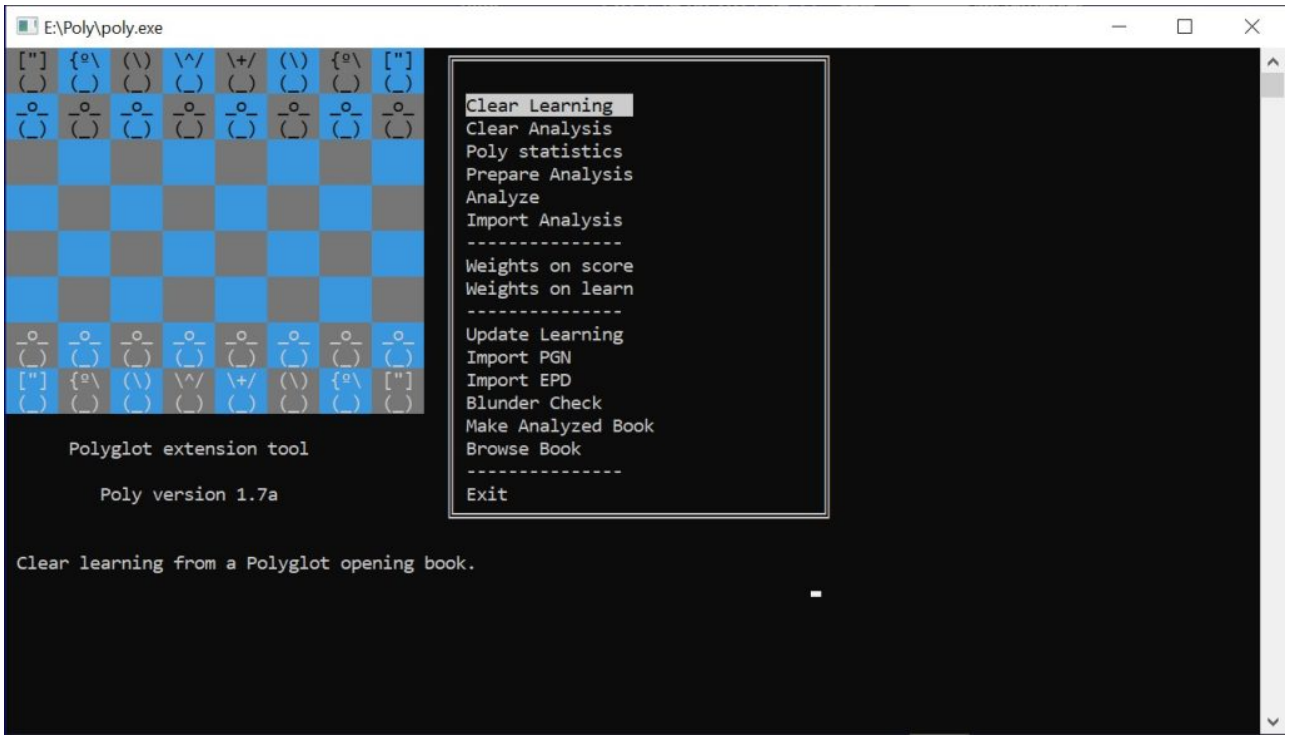
Daraufhin bewertete ich diese Probleme und schaute, welche am häufigsten vorkommen. Anschließend kam das „Warum tut die KI das?“. Manchmal sind die Fehler offensichtlich, jedoch nicht immer. Und dann passieren merkwürdige Dinge. Man ändert ein paar Werte für den Bauern ab und am Ende macht der Turm komische Züge. Das fühlt sich teilweise so an, als würde man am Kofferraum des Autos eine Schraube stärker anziehen und daraufhin eiert das rechte Vorderrad.

Um gewissen Problemen auf die Spur zu kommen, spiele ich dann auch wieder selbst gegen die KI. Mit Unterstützung von Stockfish. So kann ich explizite Stellungen hervorrufen und schauen, wie die KI reagiert, wenn ich eine oder mehrere Zahlen verändere. Das hilft, ein Gefühl für die Werte zu erhalten. Durch die Abtrennung in Spielphasen kann man sich zudem auf bestimmte Bereiche konzentrieren. Zunächst muss sie die Eröffnung gut überstehen, dann das Mittelspiel und zuletzt das Endspiel. Danach beginnt man wieder bei null.

Erweiterungen

Es gibt viele Möglichkeiten, eine Schach-KI zu erweitern. Mitunter um andere Spielmodi. Für mich ist das uninteressant, da ich fast zu 100% nur klassisches Schach spiele. Aber auch hier finden sich Gelegenheiten zur Optimierung, wie etwa das Multithreading.

Eine oft verwendete Erweiterung sind Eröffnungsbücher. Das sind kleine Datenbanken mit zahlreichen Positionen und die besten Antworten darauf. Früher war ein Eröffnungsbuch mit 20.000 Stellungen bereits eine beeindruckende Sache. Heute haben die Bücher ein paar hunderttausend, manche sogar mehr als eine Million Stellungen. Diese Dateien bieten zwei Vorteile. Die KI macht nicht schon am Anfang einen groben Fehler und sie spielt nicht immer die gleichen Züge. Um etwas Vielfalt in die Sache zu bringen, wählte meine erste Version der KI mit Weiß zwischen *e4*, *d4* und *c4* aus. Das half beim KI-Vergleich, da somit unterschiedliche Stellungen auf das Brett kamen.



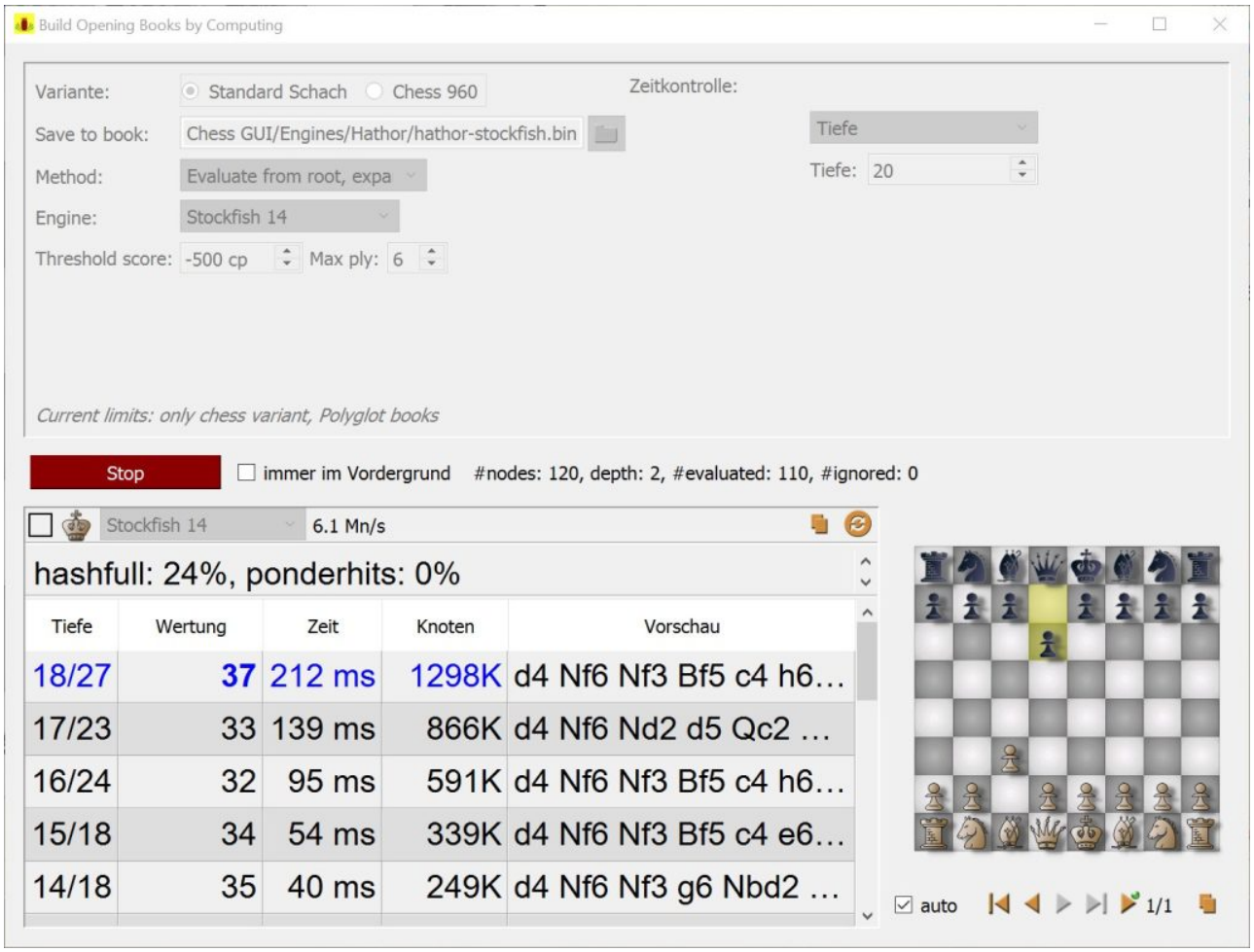
Es gibt viele Tools zur Erstellung und Analyse von Eröffnungsbüchern

Wer es „richtig“ machen will, setzt auf ein großes Buch. Das wird meist per PGN oder mit PolyGlot realisiert. Letzteres war für mich zugegebenermaßen eine ziemliche Herausforderung, zumal die KI mein erstes C++-Projekt ist.

Endspieltabellen sind eine ebensolche Erweiterung. Während Eröffnungsbücher nur wenige Megabyte groß sind, gehen Endspieltabellen in den Terabytebereich. Ab einer bestimmten Figurenanzahl kann man in diesen Tabellen nach der Position suchen und erhält anschließend auf jeden Zug die perfekte Antwort. Das hat für die KI u. a. den Vorteil, dass sie keine 50 oder 100 Halbzüge im Voraus denken muss.

Aktueller Stand meiner KI

PolyGlot war ein großer Sprung, auch wenn es, während ich diese Zeilen schreibe, noch nicht 100%ig mit meinen Protokollen funktioniert. Sobald das geht, starten die nächsten Turniere.



Eröffnungsbücher können über Datenbanken oder – wie hier – rein durch Engine-Berechnungen erstellt werden (BanksiaGUI V. 0.52b Oktober 2021)

Die ganz groben Patzer, vor allem im taktischen Bereich, sind behoben. Die meisten Fehler lassen sich derzeit auf positionelle Probleme zurückführen. D. h. die KI weiß nicht richtig, was zu tun ist, wenn sie keine Taktik findet und sich die Position mit den aktuellen Zahlen nicht verbessern lässt. Dann kommen dumme Züge, etwa sinnlose Turmzüge, heraus.

Wenn die KI verliert, dann meistens im Mittelspiel und das genau aus solchen Gründen. Oder weil sie eine Taktik nicht sieht, da die Suchtiefe zu gering ist.

Im KI-Vergleich liegt sie derzeit zwischen 1800 und 1900 Elo. Da ist Luft nach oben. Im Schnell- und Blitzschach besiegt sie Menschen mit circa 2000 Wertungspunkte. Auch hier ist noch Luft nach oben.

Da es die perfekte KI nie geben wird, ist es eine endlose Aufgabe. Immer wieder erwische ich mich dabei, wie ich bei den Turnieren fiebere, als wäre es mein eigenes Kind. Ich freue mich über Siege, hadere mit Patzern und an manchen Tagen sehe ich nur die vielen Zahlen vor dem geistigen Auge.

Mein Ziel ist es, irgendwann auf etwa 2400 Elo zu kommen. Dann ist Version 1.0 fertig. Und danach? Vielleicht 2.0, als neuronales Netz.

Spiele gegen die KI

Ab und zu spielt meine KI online. Hier kannst Du diese Spiele sehen, ebenso eine Vielzahl der Spiele gegen andere KIs. Irgendwann, wenn sie ausgereift genug ist, will ich einen eigenen Rechner für die KI besorgen, damit sie rund um die Uhr erreichbar ist. Mal sehen, was die Zukunft bringt.

Ach ja, die KI hat auch einen Namen: **Hathor**.

Das Logo meiner KI

Date Created
3. Dezember 2021
Author
sven