Gymnasium Starnberg Abiturjahrgang 2024

SEMINARARBEIT

Rahmenthema des Wissenschaftspropädeutischen Seminars:

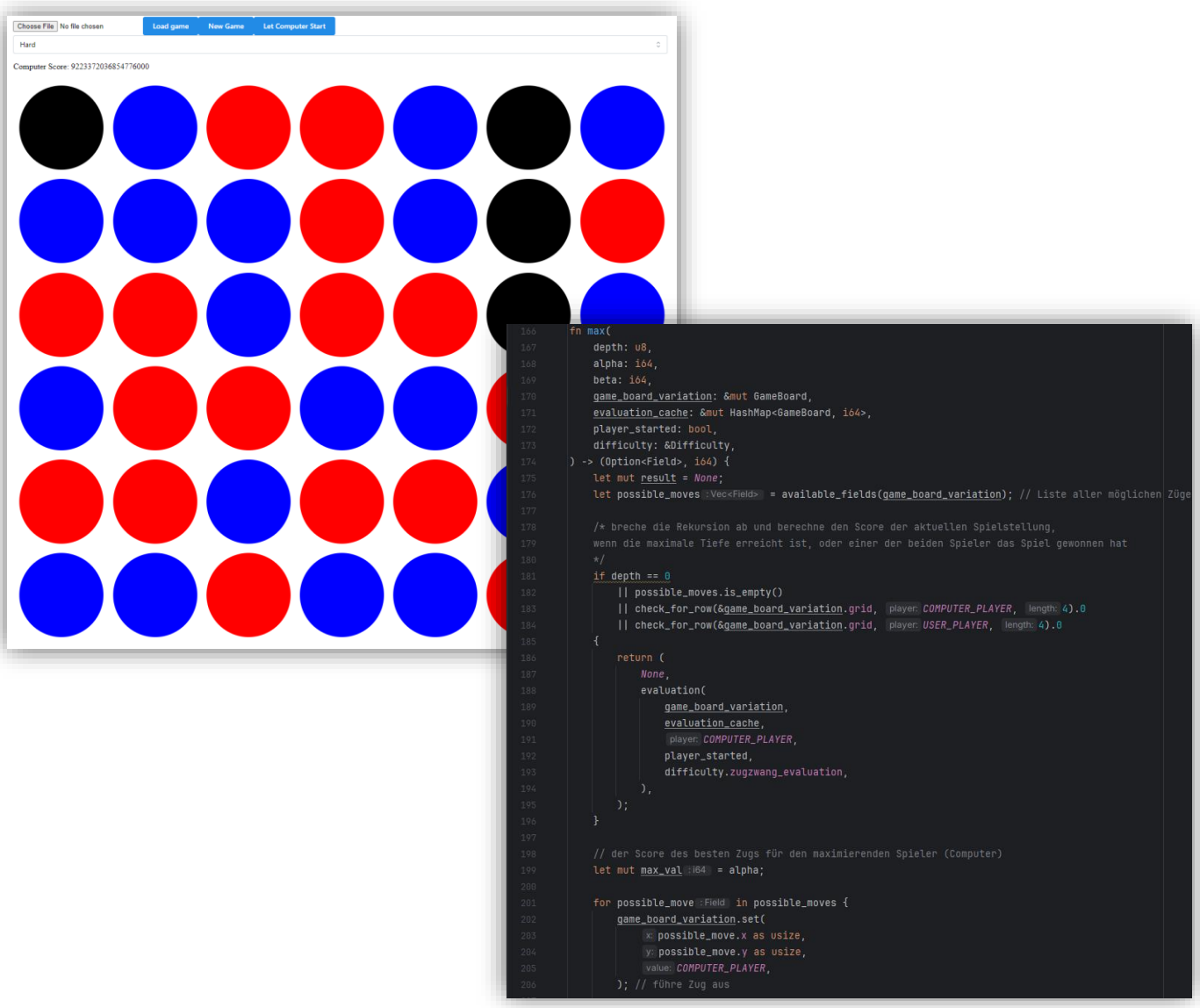
Spieltheorie

Leitfach:

Mathe

Thema der Arbeit:

Entwicklung eines Vier Gewinnt Computers



Inhaltsverzeichnis

1. Vier gewinnt erklärt

Vier gewinnt (engl. „Connect Four“) ist ein klassisches Zweipersonen-Strategiespiel, welches in den 1970-er Jahren von Howard Wexler und Ned Strongin entwickelt wurde.

Spielziel:

Das Ziel von Vier gewinnt ist es, als erster Spieler vier seiner eigenen Spielsteine in einer waagerechten, senkrechten oder diagonalen Reihe zu platzieren.

Spielbrett:

Das Spielbrett besteht aus einer rechteckigen Anordnung von sieben Spalten und sechs Reihen.

Spielablauf:

Die beiden Spieler verwenden abwechselnd ihre Spielsteine, die oben in die Spalten des Spielbretts fallen gelassen werden. Jeder Spieler benutzt dabei Spielsteine einer ihm zugeordneten Farbe, im Originalspiel sind dies Rot und Gelb.
Wenn das gesamte Spielbrett gefüllt ist, ohne dass ein Spieler vier Spielsteine in einer Reihe hat, endet das Spiel unentschieden.

Abbildung 1: Howard Wexler mit einem klassischen Vier gewinnt Brett.

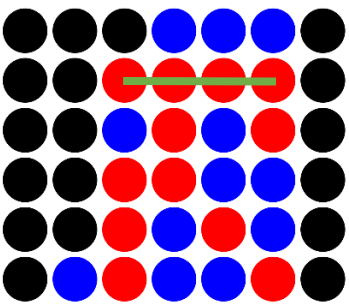


Abbildung 2: Rot gewinnt das Spiel über eine horizontale 4er-Reihe

2. Entwicklung eines Vier gewinnt Computers

Grundlagen

Bei Vier gewinnt handelt es sich um ein Nullsummenspiel (entweder gewinnt einer der beiden Spieler oder es endet unentschieden) mit vollständiger Information (es gibt keine verdeckten Karten oder ähnliches). Da es kein Element des Glücks gibt (wie z.B. Würfeln), hängt der Ausgang des Spiels ausschließlich von den strategischen Entscheidungen der Spieler ab und ist damit deterministisch. Dadurch eignet es sich für die Entwicklung einer künstlichen Intelligenz mit dem Ziel, für durchschnittlich begabte, menschliche Spieler anspruchsvoll zu sein.

Die Herausforderung liegt darin, in jeder eigenen Spielrunde den bestmöglichen Zug zu finden.
Einen ersten Überblick über das Problem bietet der sog. Spielbaum. Ein solcher Baum visualisiert die verschiedenen Zugmöglichkeiten und ihre Auswirkungen auf das Spielgeschehen, und unterstützt somit die strategische Entscheidungsfindung.

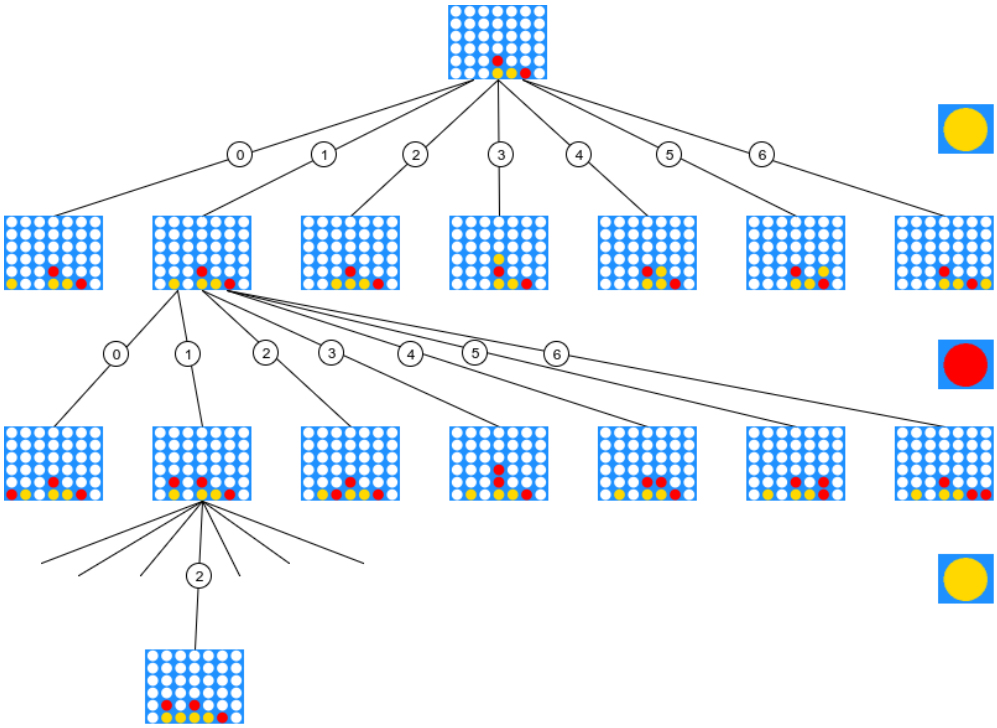


Abbildung 3: Teil eines Vier gewinnt Spielbaums. Rechts der jeweilige Spieler der am Zug ist

Bei einfachen Spielen, wie bspw. „Tic-Tac-Toe“ können Computer alle möglichen Spielabläufe berechnen und damit einen gesamten Spielbaum erstellen. Damit kennt der Computer an jeder Stelle des Spiels sein für ihn bestmögliches Ergebnis und kann den jeweiligen Zug ausführen.

Vier gewinnt ist jedoch weitaus komplexer.
Wie man in Abbildung 3 erkennt, stehen dem beginnendem Spieler 7 mögliche Züge zur Auswahl. Für jeden dieser Züge hat der andere Spieler wiederum 7 mögliche Züge. Nach bereits zwei Zügen erhalten wir somit bereits $7^2 = 49$ mögliche Spielstellungen. Auf einem klassischen Board gibt es insgesamt 4.531.985.219.092 mögliche valide Spielstellungen.[1]
Eine Berechnung des gesamten Spielbaums in Echtzeit ist damit mit der Rechenleistung aktueller PCs nicht möglich.

Tatsächlich wurde das Spiel jedoch unabhängig von Victor Allis im Jahr 1988, sowie James D. Allen im Jahr 1990 gelöst[2].[3] Beide gelangten zu der gleichen Schlussfolgerung: Der Startspieler hat die Möglichkeit, das Spiel mit oder vor dem 41. Zug gegen die stärkste Verteidigung zu gewinnen, wenn er seinen Zug in der mittleren Spalte startet. Wählt er hingegen die Spalte links oder rechts daneben, führt ein perfektes Spiel[4] beider Seiten zu einem Unentschieden. Falls er seinen ersten Spielstein in eine der vier übrigen Spalten einbringt, resultiert dies in einer Niederlage gegen einen perfekten Gegenspieler.

Diese Lösung ist jedoch äußerst komplex und entspricht auch nicht dem Ziel dieser Arbeit, soll aber der Vollständigkeit halber aufgeführt werden. James D. Allen hat darauf basierende Strategien in seinem Buch „The Complete Book of Connect 4“ dargelegt.

Der Minimax Algorithmus

Um ohne größere Verzögerung nach Eingabe des Nutzers einen Zug zurückgeben zu können, sowie die Möglichkeit zu haben, den Schwierigkeitsgrad des Programms anzupassen, eignet sich der Minimax-Algorithmus.

Der Minimax-Algorithmus ist ein Entscheidungsbaum-Algorithmus, der in vielen verschiedenen Brettspielen verwendet werden kann. Ziel des Minimax-Algorithmus ist es, die bestmögliche Spielstrategie für einen Spieler zu finden, indem er alle möglichen Spielzüge bis zu einer gewissen Tiefe durchdenkt und die Konsequenzen dieser Züge

---

[1] Berechnet von Stefan Edelkamp und Peter Kissmann: Symbolic Classification of General Two-Player Games, September 2008, S. 2
[2] Ein gelöstes Spiel ist ein Spiel, dessen Ausgang (Sieg, Niederlage oder Unentschieden) von jeder beliebigen Stellung aus korrekt vorhergesagt werden kann, vorausgesetzt, dass beide Spieler perfekt spielen
[3] Victor Allis: A Knowledge-based Approach of Connect-Four, Oktober 1988
[4] Perfektes Spielen in der Spieltheorie bezieht sich auf das Verhalten oder die Strategie eines Spielers, die unabhängig von der Reaktion des Gegners das bestmögliche Ergebnis für diesen Spieler gewährleistet

bewertet. Der Algorithmus berücksichtigt die Züge beider Spieler und versucht, den bestmöglichen Zug für den aktuellen Spieler zu ermitteln.

Die festgelegte Tiefe des Algorithmus entscheidet über seine Laufzeit sowie die qualitative Leistung bzw. den Schwierigkeitsgrad des Programms vom Gegenspieler ausgesehen.

Minimax Funktionsweise:

1. Spielbaum erstellen: Zu Beginn erstellt der Algorithmus bis zu einer festgelegten Tiefe einen Baum, der alle möglichen zukünftigen Spielzüge darstellt. Jeder Knoten im Baum repräsentiert eine Spielsituation nach einem Zug.
2. Bewertungsfunktion verwenden: Für jede Endposition, also wenn die maximale Tiefe erreicht ist, wird im Baum eine Bewertungsfunktion aufgerufen, um die Stärke der Stellung aus Sicht des Computers zu bewerten. Mehr dazu im Abschnitt „Die Bewertungsfunktion".
3. Minimax-Prinzip anwenden: Der Algorithmus durchläuft den Baum rekursiv, beginnend an der aktuellen Position. In jedem Schritt werden die besten Züge für den aktuellen Spieler und die schlechtesten Züge für dessen Gegner ausgewählt. Der Computer maximiert die Bewertung seiner Spielstellung („max"). Dennoch berücksichtigt er auch die Züge des Gegners, welcher versucht, die Bewertung der Spielstellung des maximierenden Spielers / Computers zu minimieren („min"). Er nimmt also an, dass sein Gegner ebenfalls stets den besten Zug spielt. Da die Spieler immer abwechselnd an der Reihe sind, wechseln sich „max" und „min" kontinuierlich ab.
4. Rückgabe des besten Zugs: Nachdem der Algorithmus den gesamten Baum durchlaufen hat, kehrt er zurück und gibt den besten Zug aus, den der maximierende Spieler machen sollte. Dies ist der Zug, der bei perfektem Spielen[5] beider Spieler zu der Stellung führt, die die höchste Bewertung für den Computer hat.

Beispiel der Funktionsweise

Ein gutes Beispiel, um die Funktionsweise des Minimax-Algorithmus zu veranschaulichen, ist eine Falle, die sich schnell vom beginnenden Spieler im Spiel aufbauen lässt und unerfahrene Spieler des Öfteren übersehen.

Hierbei platziert man 2 Steine mittig nebeneinander auf die unterste Reihe, sodass links und rechts von ihnen jeweils 2 Spalten frei sind. Wenn der Gegner nun keinen Stein

[5] Perfektes Spielen ist hier limitiert von der Tiefe des Algorithmus

horizontal daran angrenzt wie im rechten Teil des Baumes, hat der beginnende Spieler gewonnen. Er kann nun eine horizontale 3er-Reihe aufbauen, die links und rechts frei ist. Damit hat er zwei Möglichkeiten zu gewinnen (links oder rechts zu einer 4er-Reihe vervollständigen), sein Gegner kann jedoch nur eine davon im nächsten Zug verhindern.

Eine grundlegende künstliche Vier-Gewinnt-Intelligenz sollte also die Gefahr einer horizontalen 3er-Reihe frühzeitig erkennen und abwenden.

Die detaillierte Erklärung der Bewertungsfunktion erfolgt im Abschnitt "Die Bewertungsfunktion". Für den aktuellen Kontext genügt es jedoch zu wissen, dass der Wert $-\infty$ einen Sieg für den Gegner repräsentiert, während der Wert $+x$ eine noch nicht definierte positive Zahl darstellt.

Die Funktionsweise kann wie folgt visualisiert werden, wobei einige uninteressante Spielstellungen der Übersicht halber weggelassen werden.

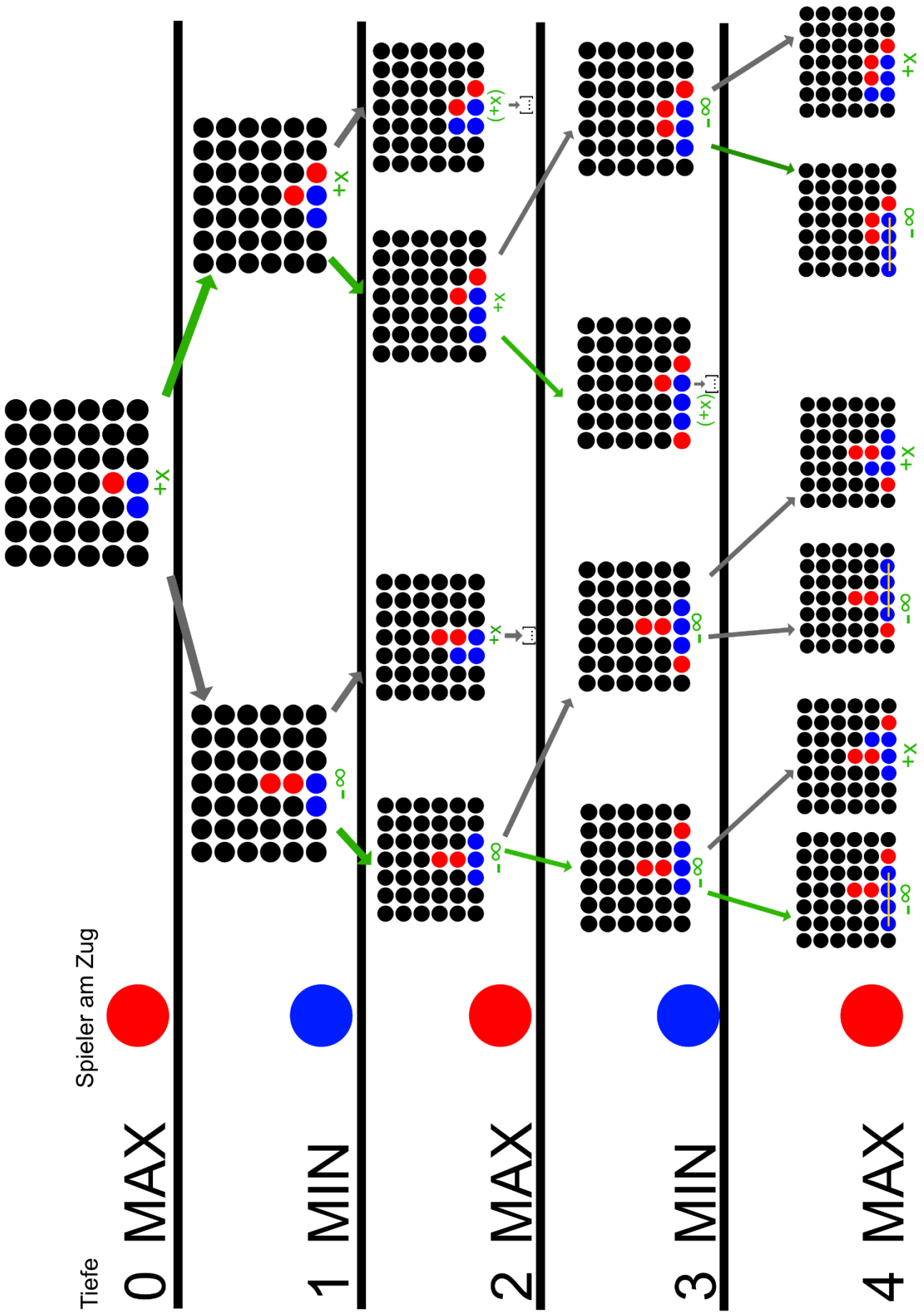


Abbildung 4: Veranschaulichung des Minimax-Algorithmus

Wie sich in der Visualisierung erkennen lässt, wird nach dem Minimax-Prinzip in einer max-Spalte stets der beste Zug für den Computer (rot) ausgewählt, in einer min-Spalte der beste Zug für blau.

Bei einer maximalen Tiefe von 4 erkennt die KI also bereits die entstandene Falle, und entscheidet sich, wie man in der ersten Zeile erkennt, für einen Zug, der sie abwendet.

Alpha-Beta-Pruning

Um bei hoher Tiefe eine dennoch schnelle Antwortmöglichkeit des Programms gewährleisten zu können, ist es sehr erstrebenswert, den Minimax-Algorithmus zu optimieren. Dies lässt sich über die sog. Alpha-Beta-Pruning Technik erreichen.

Alpha-Beta-Pruning reduziert die Rechenzeit um einen großen Faktor, indem es Zweige im Spielbaum abschneidet, die nicht durchsucht werden müssen, weil es bereits einen besseren Zug gibt. Es wird Alpha-Beta Pruning genannt, weil es zwei zusätzliche Parameter in der Minimax-Funktion übergibt, nämlich Alpha und Beta.

Diese sind wie folgt definiert:

- Alpha ist der beste Wert, den der **Maximierer** derzeit auf dieser Ebene oder darüber garantieren kann.
- Beta ist der beste Wert, den der **Minimierer** derzeit auf dieser Stufe oder darunter garantieren kann.

Das „Pruning“ lässt sich wie folgt an einem minimalistischen Spielbaum veranschaulichen:

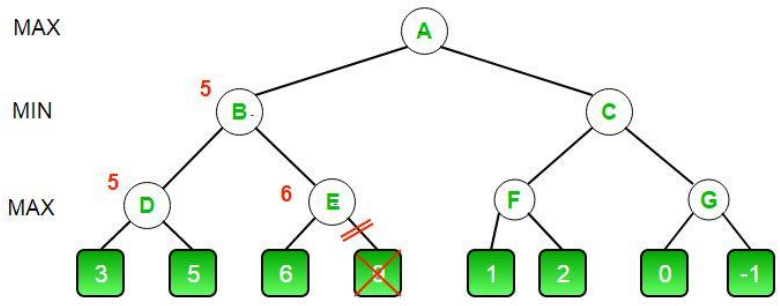


Abbildung 5: Alpha Beta Pruning

- Der erste Aufruf beginnt bei A. Der Wert von alpha ist hier laut Definition -∞ und der Wert von beta ist +∞. Diese Werte werden an die nachfolgenden Knoten im Baum weitergegeben. Bei A muss der Maximierer das Maximum von B und C wählen, also ruft A zuerst B auf.

- Bei B muss der Minimierer das Minimum von D und E wählen und ruft daher zuerst D auf.
- Bei D (maximale Tiefe) wird zuerst der linke Knoten angeschaut, welcher einen Wert von 3 zurückgibt. Der Wert von alpha bei D ist also max(-∞, 3), was 3 ist.
- Um zu entscheiden, ob es sich lohnt, den rechten Knoten zu betrachten oder nicht, wird die Bedingung beta<=alpha geprüft. Diese ist falsch, da beta = +∞ und alpha = 3. Also wird die Suche fortgesetzt.
- D schaut sich nun seinen rechten Knoten an, welcher einen Wert von 5 zurückgibt. Alpha = max(3, 5) ist 5. Der Wert des Knotens D ist nun 5.
- D gibt einen Wert von 5 an B zurück. Bei B ist beta = min(+∞, 5), was 5 ist. Der Minimierer hat jetzt garantiert einen Wert von 5 oder weniger. B ruft nun E auf, um zu sehen, ob er einen niedrigeren Wert als 5 erhalten kann.
- Bei E sind die Werte von alpha und beta nicht -∞ und +∞, sondern -∞ und 5, weil der Wert von beta bei B geändert wurde und B diesen Wert an E weitergegeben hat
- Jetzt schaut E auf seinen linken Knoten, welcher 6 ist. Bei E ist alpha = max(-∞, 6), was 6 ergibt. Hier wird die Bedingung wahr. Beta ist 5 und alpha ist 6. Also ist beta<=alpha. Folglich wird die Suche abgebrochen und E gibt 6 an B zurück.
- Der Punkt ist, dass es keine Rolle spielt, welchen Wert der rechte Knoten von E hat. Er hätte +∞ oder -∞ sein können, es wäre immer noch egal gewesen. Er musste nicht angesehen werden, da der Minimierer einen Wert von 5 oder niedriger garantiert hat. Sobald der Maximierer also die 6 sah, wusste er, dass der Minimierer niemals so weit kommen würde, weil er eine 5 auf der linken Seite von B bekommen kann (der Maximierer kann nur eine Zahl >=6 zurückgeben, der Minimierer eine Ebene darüber hat jedoch bereits eine 5 als Möglichkeit). Auf diese Weise musste der Algorithmus die 9 nicht ansehen bzw. berechnen und sparte somit Rechenzeit.

Die Bewertungsfunktion

Wie bei den Erläuterungen zum Minimax-Algorithmus erklärt, ist für dessen Funktionsweise wesentlicher Bestandteil die Bewertungsfunktion. Die im Folgenden beschriebene Heuristik machte damit auch den mit Abstand größten Teil der Entwicklungsdauer aus. Hierbei habe ich ständig experimentiert, neue Ideen implementiert und nach etwas Zeit den Großteil der neuen Implementierungen wieder entfernt. Daher

stellt das hier beschriebene, finale Bewertungsschema einen nur sehr kleinen Teil einer langwierigen Entwicklung dar.

Grundlagen

Zweck einer Bewertungsfunktion ist es, eine einzelne Spielposition sowie einen Spieler zu erhalten und daraus einen numerischen Zahlenwert zu generieren, der angibt, wie gut die übergebene Spielposition für jenen Spieler ist. Die Heuristik betrachtet dabei nicht in Zukunft mögliche Züge, sondern bezieht sich einzig auf eine Momentaufnahme des Spiels.

Die implementierte Bewertungsfunktion funktioniert wie folgt:

$$ev = ev_{game-position-player} - ev_{game-position-opponent}$$

Hat der Gegner also sehr gut positionierte Felder und der betrachtete Spieler eher schlecht gesetzte Felder wird ein hoher numerischer Wert von einem kleineren Wert abgezogen und das Ergebnis ist negativ. Beim Aufruf von Minimax wird jedoch die Bewertung maximiert. Somit werden schlecht gesetzte Felder des Gegners und gut gespielte Felder des Spielers erreicht. Die Bewertung $ev_{game-position-player}$ bzw. $ev_{game-position-opponent}$ setzt sich aus einzelnen „Scores“ zusammen. Ein Score stellt ein erkanntes Muster und seine Bewertung dar. Je höher die Bewertung des Musters, desto stärker trägt es zu einem Sieg bei. Ein Muster mit einem hohen Score wird also einem Muster mit einem niedrigen Score bevorzugt, wenn der Computer die Wahl zwischen beiden hat.

Maximaler Score bei einem Sieg

Der erste Teil der Heuristik ist selbstverständlich die Erkennung eines Siegs oder einer Niederlage. Ein Sieg wird durch eine Reihe von 4 Steinen erreicht, diese kann vertikal, horizontal oder diagonal erfolgen. Wird eine solche Reihe in der Spielposition erkannt, wird ∞ zurückgegeben, da kein anderer Teil der Heuristik von Bedeutung ist, wenn das Spiel bereits durch einen Sieg beendet wurde.

Steine in der Mitte sind zu bevorzugen

Mit nur der Siegesheuristik implementiert, würde der Computer besonders am Anfang des Spiels sehr „planlos“ sein. Da es bei Vier gewinnt schließlich um das Verbinden langer Viererreihen geht, will man möglichst viel Raum bzw. Richtungsmöglichkeiten haben, solche zu bilden. Diese Richtungsmöglichkeiten gibt es für zentralere Felder mehr als für Felder am Rand.

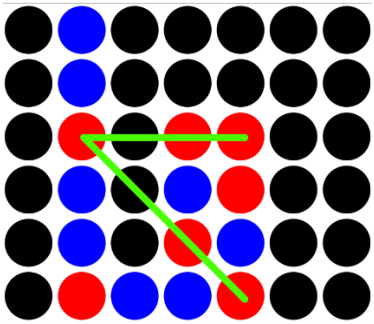
Der zweite Teil der Heuristik berechnet für jedes Feld des Spielers, wie horizontal mittig es ist, multipliziert dies mit 1000 und summiert diese Zahlen. Ein Feld ganz links oder rechts am Rand bekommt damit den Wert 0, ein Feld eine Spalte daneben den Wert 1000, ein Feld in der 3. Spalte den Wert 3000, etc.
Dadurch erfolgt auch automatisch ein Spielen in die Höhe.

Die Gewichtung von 1000 ist bewusst niedrig gewählt, wodurch dieser Aspekt der Heuristik vor allem in den frühen Spielzügen eine entscheidende Rolle spielt, was genau der Intention entspricht.

4er-Reihe mit einer Lücke

Dieser Teil der Heuristik soll den Computer dazu „verleiten“, gewinnbringende 4er-Reihen zu entwickeln. Dafür werden unvollständige, horizontale oder diagonale 4er-Reihen mit einer Lücke (leeres Feld) bewertet. Vertikale 3er-Reihen können immer vom Gegenspieler geblockt werden und sind damit nutzlos.

Eine bekannte Falle von Vier gewinnt ist die Figur der „7“[6]. Durch diese Heuristik wird diese nach Möglichkeit aufgebaut, ohne dass die Figur explizit einprogrammiert wurde.



Zwei unvollständige 4er-Reihen bilden eine „7“ -> Rot hat bereits gewonnen

Zugzwang

Definition eines Zugzwangs

Die meisten Siegesmöglichkeiten erkennt das Programm früh genug, dank einer hohen Rechentiefe.

Eine sehr anspruchsvolle Technik aus Vier gewinnt kann es jedoch nicht durch die bisherigen Implementierungen beherrschen, die sogenannten Zugzwänge[7].

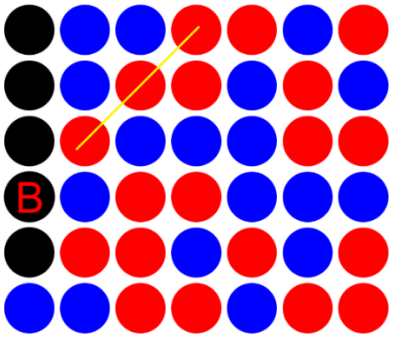
Ein Zugzwang charakterisiert sich dadurch, dass ein Spieler (A) keine anderen Zugmöglichkeiten hat, als unter eine unvollständige horizontale oder diagonale Viererreihe

---

[6] Vgl. Jennifer Mueller: Always Win at Connect 4: The Strategy That Will Keep You on Top, 24. März 2023
[7] Vgl. 2Swap: Threat Analysis, 13. August 2023

seines Gegners (B) zu setzen. Spieler A würde also lieber passen, als die Erfüllungsposition der Bedrohung von Spieler A aufzudecken.

Unter „Bedrohung“ versteht sich eine unvollständige diagonale oder horizontale Viererreihe eines Spielers, dessen Erfüllungsposition (der fehlende Stein) nicht spielbar ist, bis sie von einem Spieler aufgedeckt wird (der Spieler spielt direkt unter die Lücke).

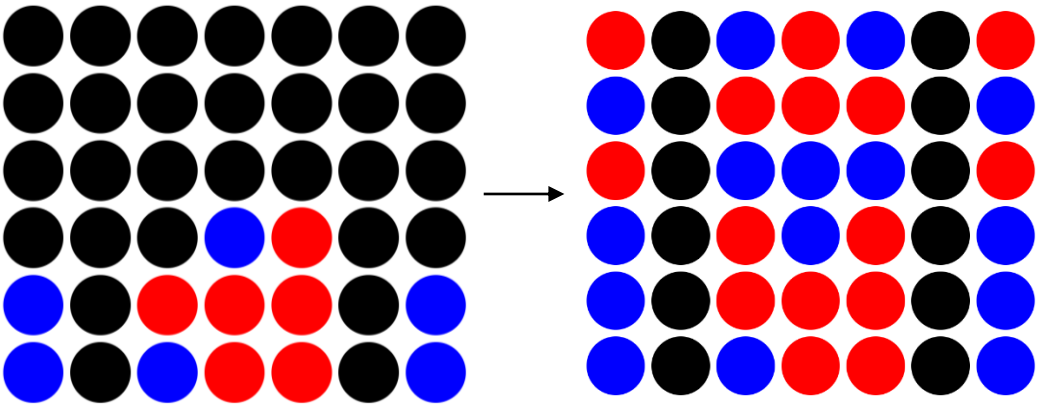


Rot begann das Spiel, Blau ist am Zug bzw. im Zugzwang und muss damit die Bedrohung von Rot aufdecken -> Rot gewinnt („B“ markiert die Erfüllungsposition)

Gerade und ungerade Zugzwänge

Die Schwierigkeit an Zugzwängen ist, dass man sie früh im Spiel vorbereiten kann, sie aber womöglich erst sehr viel später von Bedeutung sind.

Dazu betrachten wir folgende Situation, diesmal hat Blau das Spiel begonnen:
Mehr als die Hälfte des Bretts ist noch frei, Rot hat jedoch bereits zwei Bedrohungen etabliert.
Wenn nun die restlichen Spalten aufgefüllt werden, ist der startende Spieler (Blau) wieder am Zug und Rot gewinnt sicher:

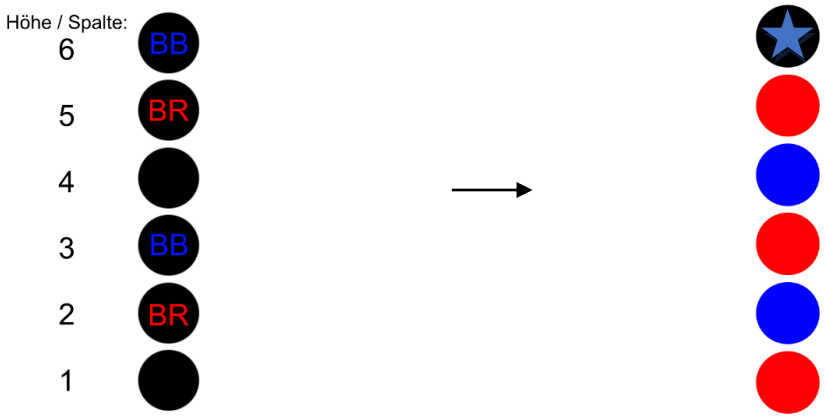


Blau im Zugzwang -> Rot gewinnt

Da es viele Züge benötigt, bis die finale Situation eintritt, kann diese Technik aufgrund der begrenzten Tiefe des Minimax-Algorithmus mit der bisherigen Implementation nicht erkannt werden.

Es stellt sich also die Frage, wie man gewinnbringende Zugzwänge aufbauen bzw. verhindern kann. Entscheidend dafür ist die Parität einer Bedrohung. Diese besagt, ob die Y-Koordinate der Erfüllungsposition (startend bei 1 am Boden) gerade oder ungerade ist.

Dazu betrachten wir folgende Spalte.
Angenommen, rot begann das Spiel, macht es keinen Unterschied wie die restlichen aufgefüllten Spalten aussehen, es wird rot sein, welcher als erster in diese Spalte setzen muss. Dies liegt an der geraden Anzahl an senkrechten Feldern des Standardbretts. Dies führt zu folgendem Ergebnis:



Wie man sieht, sind jeweils die ersten beiden Bedrohungen von Rot und Blau nichtig, aber Rot gewinnt durch Zugzwang mit seiner Bedrohung in der 5. Zeile. Man erkennt folgendes:

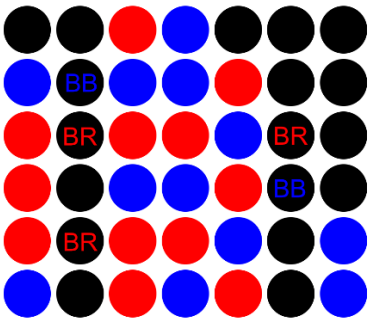
- Der beginnende Spieler (Erster Spieler) bekommt standardmäßig die ungeraden Felder.
- Der nicht-beginnende Spieler (Zweiter Spieler) bekommt standardmäßig die geraden Felder.

Daraus lässt sich schließen, mit welcher Parität ein Spieler seine Bedrohungen aufbauen will.

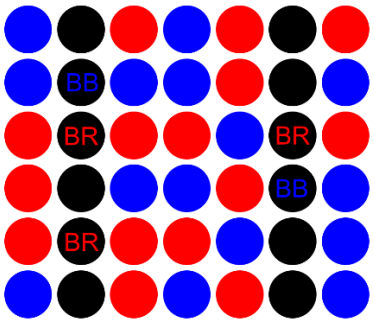
Beide Spieler mit Zugzwängen

Was passiert, wenn beide Spieler ein oder mehrere Bedrohungen ihrer bevorzugten Parität haben?
Dazu ein weiteres Beispiel, Blau hat das Spiel begonnen.

Blau ist am Zug, wer wird das Spiel gewinnen?

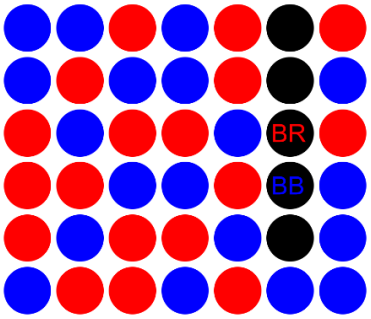


Füllen wir zuerst einmal die restlichen Felder auf:

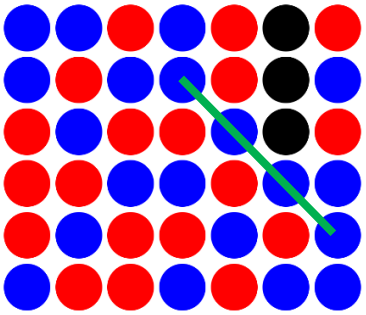


Blau ist damit wieder am Zug. In die 2. Spalte kann er nicht setzen (da sonst Rot seine Bedrohung erfüllt), also setzt er in die 6. Spalte. Rot kann daraufhin nicht ebenfalls in die 6. Spalte setzen, da Blau sonst nach den Regeln der Parität seine ungerade Bedrohung darüber erfüllt.

Damit muss Rot seine gerade Bedrohung links aufgeben und die 2. Spalte füllt sich auf.



Rot ist wieder am Zug und muss nun in die 6. Spalte spielen, was zum Sieg für Blau führt.



Damit ist die ungerade Bedrohung der geraden überlegen.

<u>Der Grund dafür ist folgender:</u>

Eine ungerade Bedrohung "kappt" eine Spalte, sodass sie eine ungerade Anzahl an Feldern hat und kehrt im Wesentlichen um, wer welche Reihe bekommen kann, wenn in die restlichen Spalten gespielt wird. Dadurch kann Rot seine gerade Bedrohung nicht erfüllen. Sobald der Rest des Spielfeldes aufgefüllt ist und Rot in die 6. Spalte setzen muss, ist hier die ursprüngliche Reihenfolge (Parität) wiederhergestellt, da Blau bereits einen Stein in dieses Feld gesetzt hat. Blau bekommt damit wieder die ungeraden Felder und gewinnt durch seine Bedrohung.

Das bedeutet, dass eine ungerade Bedrohung "stärker" ist als eine gerade Bedrohung, und führt zum Prinzip der Zugzwang-Kontrolle.

Zugzwang-Kontrolle

Ungerade Bedrohungen sind auch dem zweiten Spieler von Vorteil, sobald er zusätzlich ein oder mehrere gerade Bedrohungen aufgebaut hat:

Spieler B hat eine ungerade Bedrohung und setzt zwei Positionen unter ihre Erfüllungsposition seinen Stein. Wenn nun A am Zug ist und in eine andere Spalte spielen kann, verändert sich die Dynamik des Spiels: Beide Spieler bekommen nun die Felder der jeweils anderen Parität.

-> Spieler B erlangt Kontrolle über den Zugzwang.

Dies führt häufig dazu, dass Spieler A seine Bedrohungen in der anderen Spalte aufgeben muss.

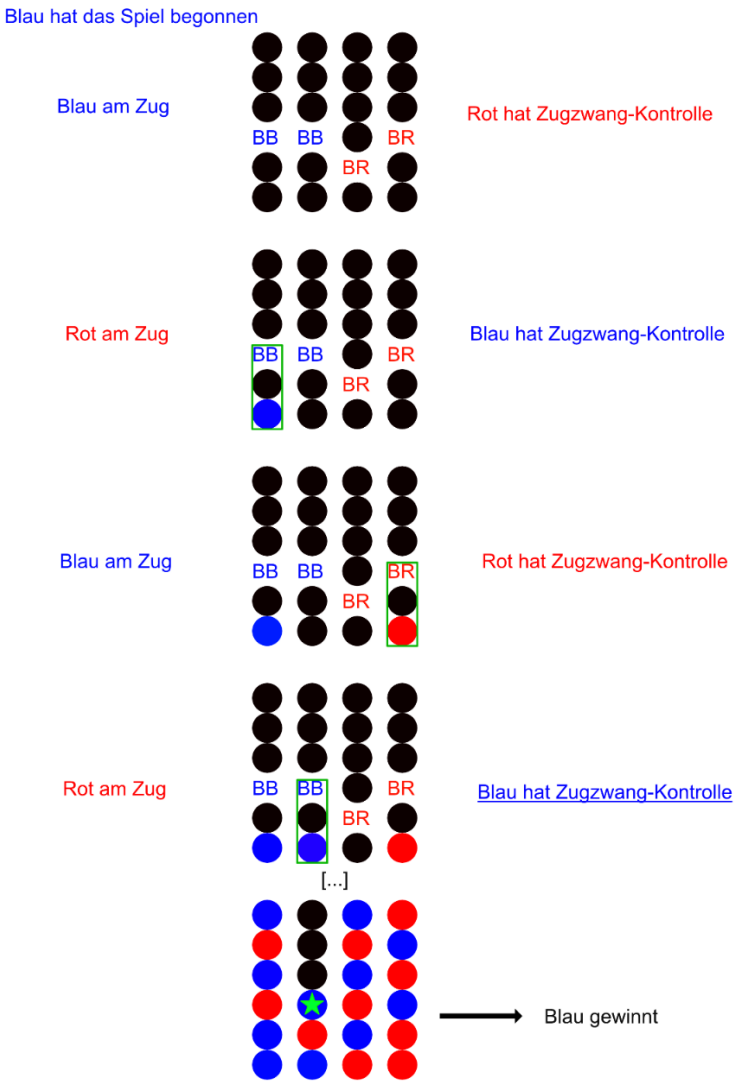
Kann jedoch Spieler A ebenfalls zwei Positionen unter eine eigene, ungerade Bedrohung setzen, erlangt er die Zugzwang-Kontrolle zurück (die zwei nun ungeraden Spalten heben sich gegenseitig auf).

Außerdem gilt: Kann der zweite Spieler entscheiden, ob er eine eigene gerade oder ungerade Bedrohung aufdecken will, sollte er auf jeden Fall die ungerade Bedrohung aufgeben, da er nur durch gerade Bedrohungen gewinnen kann (ungerade Bedrohungen dienen dem zweiten Spieler lediglich dem Erlangen der Zugzwang-Kontrolle).

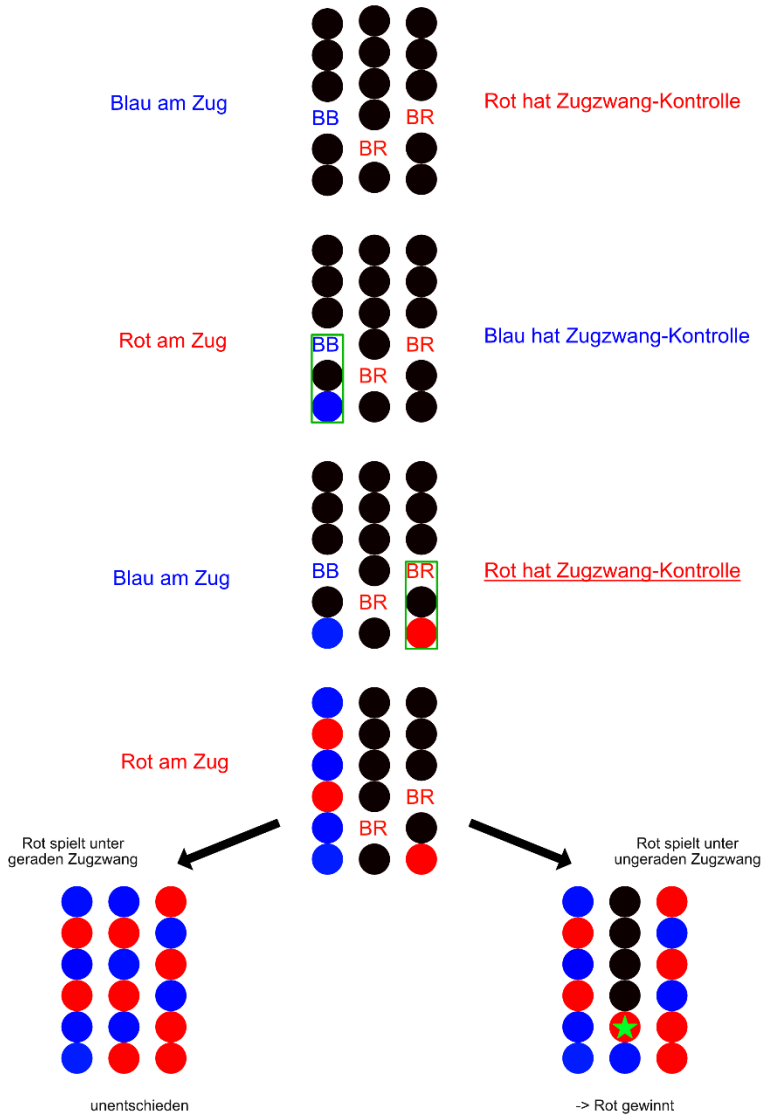
Haben beide Spieler mehrere ungerade Bedrohungen, entsteht eine komplexe Situation, in der die Zugzwang-Kontrolle ständig wechselt.

Das Spiel kann nur von dem Spieler gewonnen werden, der zuletzt die Zugzwang-Kontrolle hatte. Dadurch lässt sich außerdem schließen, dass zum Beginn des Spiels der zweite Spieler die Zugzwang-Kontrolle besitzt, da er mit nur einer einzigen geraden Bedrohung das Spiel gewinnen kann, sollte der erste Spieler keine Bedrohungen besitzen.

Folgendes Beispiel zeigt den Spielverlauf einer komplexen Stellung an Bedrohungen. Beachten Sie, dass die einzelnen Spalten nicht auf dem tatsächlichen Spielbrett nebeneinander liegen, sondern jede Spalte mit einer Bedrohung isoliert betrachtet wird. Mit grün sind die Züge markiert, mit der ein Spieler die Zugzwang-Kontrolle erlangt hat.



Angenommen, Blau hätte seine Bedrohung in der 2. Spalte nicht, hätte Rot zuletzt die Zugzwang-Kontrolle und würde über seine gerade Bedrohung gewinnen. Hier bestätigt sich auch nochmal, wieso es besser ist, eine ungerade Bedrohung aufzugeben als eine gerade:



Bewertung von Bedrohungen

Um eine Stellung mehrerer Bedrohungen auszuwerten, bedarf es einer Funktion, die ein Spiel mit den Positionen der Bedrohungen simuliert und daraufhin ein Ergebnis zurückgibt. Dafür werden nur die Erfüllungspositionen der Bedrohungen übergeben, und es werden nur die Spalten betrachtet, in denen jene zu finden sind.

Bei der Simulation wird der Zug eines Spielers regelbasiert ausgewählt. Diese Regeln lassen sich von den bisherigen Erkenntnissen ableiten. Es wird der Zug ausgewählt, welcher die am besten bewertete (hier höchste) Regel erfüllt, dabei jedoch keine sog. „essenzielle Regel“ (rot markiert) erfüllt. Es sollte also bspw. nie unter die gegnerische Bedrohung gespielt werden, auch wenn mit dem Zug die Zugzwang-Kontrolle erlangt werden würde.

Die Regeln lauten wie folgt:

1. Erfülle eigene Bedrohung
2. Blocke Bedrohung des Gegners
3. Fülle eine nach oben leere Spalte auf
4. Spiele 2 Felder unter eigene ungerade Bedrohung (erlange Zugzwang-Kontrolle)
5. Spiele 2 Felder unter eigene gerade Bedrohung
6. Spiele 2 Felder unter gegnerische / geteilte[8] Bedrohung
7. Spiele in eine Spalte, dessen unterste 3 Felder frei sind
8. Spiele ein Feld, auf und über dem keine eigene Bedrohung liegt
9. Decke eigene Bedrohung auf
10. Decke Bedrohung des Gegners auf

Am Ende gibt die Funktion zurück, ob ein Spieler bei der übergebenen Stellung an Bedrohungen gewinnen oder verlieren würde (-1 oder 1), oder alle Spalten aufgefüllt würden und damit ein Unentschieden entstehen würde (0). Diese Zahl wird mit 10^8 multipliziert. Die Gewichtung ist deutlich höher als bei den anderen Mustern, da sich das fortgeschrittene Spiel von Vier gewinnt beinahe nur auf Zugzwänge konzentriert, welche dann am Ende des Spiels über Sieg oder Niederlage entscheiden.

Caching

Gleiche Spielstellungen können im Spielbaum mehrmals auftreten. Daher ist als letzte Performance-Optimierung Caching implementiert. Um zu verhindern, dass die Bewertung für dieselbe Spielstellung mehrmals berechnet wird, werden sowohl die berechnete Bewertung als auch die zugehörige Spielstellung nach der Berechnung gespeichert. Beim Aufruf der Bewertungsfunktion wird zunächst überprüft, ob bereits ein Ergebnis vorliegt. Wenn dies der Fall ist, wird der gespeicherte Wert zurückgegeben, ansonsten erfolgt die Berechnung.

Anpassung der Schwierigkeitsstufe

Die Fähigkeit, den Schwierigkeitsgrad einzustellen, ist ein wesentlicher Aspekt, der die Spielerfahrung maßgeblich beeinflusst. Das Programm soll sowohl für einen Anfänger, der die Grundlagen des Spiels erlernen möchte, als auch für einen erfahrenen Spieler, der nach einer Herausforderung sucht, attraktiv sein. Daher sind drei verschiedene Schwierigkeitsgrade implementiert. Diese unterscheiden sich in der Tiefe des Minimax-Algorithmus und darin, ob die Zugzwang-Taktik angewandt wird:

[8] Geteilte Bedrohung = Bedrohung von beide Spielern mit derselben Erfüllungsposition

Leicht:

Tiefe: 4

Zugzwang-Taktik: Nein

Mittel:

Tiefe: 6

Zugzwang-Taktik: Ja

Schwierig:

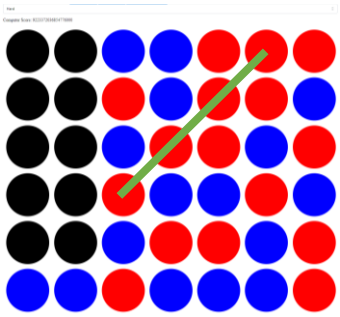
Tiefe: 8

Zugzwang-Taktik: Ja

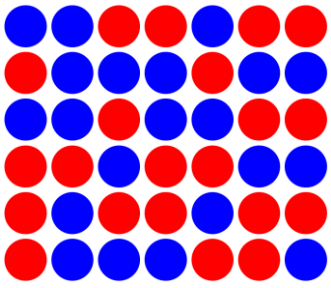
3. Test des Programms

Gegen sich selbst

Wenn man das Programm auf Schwierigkeitsgrad „Schwierig“ gegen sich selbst spielen lässt, gewinnt das nicht-beginnende Programm nach 32 Zügen:



Sehr überraschend ist, dass das Programm auf Stufe „Mittel“ beginnend gegen „Schwierig“ als zweiten Spieler ein Unentschieden herbeiführen kann:



Blau = Programm auf „Mittel“, Rot = „Schwierig“

Stand jetzt habe ich dafür noch keine allgemeine Erklärung gefunden, außer, dass es anscheinend anfängliche Züge gibt, die bei Tiefe 8 besser als bei Tiefe 6 bewertet werden, jedoch insgesamt schlechter sind.

In allen anderen Situationen verliert erwartungsgemäß der niedrigere Schwierigkeitsgrad gegen den höheren.

Gegen die Lösung

Gegen die Lösung von Vier gewinnt (https://connect4.gamesolver.org/de/) verliert mein Programm jedes Spiel erwartungsgemäß, sowohl beginnend als auch als zweiter Spieler.

Gegen Online KIs

Im Folgenden die Ergebnisse (aus Sicht meines Programmes auf Stufe „Schwierig“) gegen andere im Internet gefundene KIs, welche nicht perfekt/gelöst spielen:

Website	**Website startet**	**Mein Programm startet**
https://kenrick95.github.io/c4/	**Sieg**	*wird von der Website nicht unterstützt*
https://www.mathsisfun.com/games/connect4.html (höchster Schwierigkeitsgrad „Hard“)	**Sieg**	**Sieg**
https://kevinshannon.com/connect4/	*wird von der Website nicht unterstützt*	**Niederlage**
http://faculty.otterbein.edu/wittman1/games/connect4/	**Sieg**	*wird von der Website nicht unterstützt*
https://codesandbox.io/s/connect-four-game-ai-fq1oz	*wird von der Website nicht unterstützt*	**Sieg**
https://papergames.io/en/connect4 (Bot)	**Sieg**	**Sieg**
https://www.cbc.ca/kids/games/play/connect-4	**Sieg**	**Sieg**
https://www.4-gewinnt.de/sehr_schwer.html (höchster Schwierigkeitsgrad)	**Sieg**	**Sieg**

Gegen zufällige Gegner auf papergames.io

Auf https://papergames.io/en/connect4 kann man online gegen zufällige (meistens) menschliche Spieler Vier gewinnt spielen.

Von 40 Spielen hat meine KI 32 Spiele (80%) gewonnen, zweimal Unentschieden gespielt und 6 Spiele verloren.

Gegen Freunde/Familie

Zum Abschluss meiner Untersuchung lud ich vier Personen aus meinem Freundes- und Familienkreis ein, gegen das Programm jeweils zwei Spiele pro Schwierigkeitsstufe zu spielen. Der erste Eintrag pro Schwierigkeitsgrad gibt das Spiel an, welches die Person

begonnen hat, während der zweite Eintrag das vom Computer begonnene Spiel aufführt. Die Ergebnisse sind wieder aus der Sicht meines Programms:

Person	Einfach	Mittel	Schwierig
A	Niederlage, Sieg	Sieg, Sieg	Sieg, Sieg
B	Niederlage, Sieg	Sieg, Sieg	Unentschieden, Sieg
C	Sieg, Sieg	Sieg, Sieg	Sieg, Sieg
D	Niederlage, Sieg	Sieg, Sieg	Sieg, Sieg

Hier bestätigt sich die Bedeutsamkeit bzw. Effektivität der Einführung eines Schwierigkeitsgrades, da die Anzahl der verlorenen Spiele mit steigender Schwierigkeit abfällt.

Ich persönlich konnte bereits auf jedem Schwierigkeitsgrad gegen meine KI gewinnen, kann dies jedoch aktuell auf Mittel oder Schwer nicht verlässlich. Auf höchster Schwierigkeitsstufe einen Sieg zu erringen, gelang mir nur durch oftmaliges Ausprobieren von verschiedenen Taktiken, wobei ich sehr von der deterministischen Funktionsweise meines Programms profitieren konnte.

4. Einschätzung der Ergebnisse

Meine künstliche Intelligenz stellt nicht die Lösung von Vier gewinnt dar und spielt auch nicht perfekt. Dies war jedoch nicht mein Ziel dieser Arbeit, und ich bin sehr zufrieden mit den erreichten Ergebnissen. Wie in den Tests gezeigt, gewinnt meine KI auf höchstem Schwierigkeitsgrad und mit Echtzeit Berechnungen gegen die meisten Personen aus meinem Bekanntenkreis, gegen andere Programme aus dem Internet und gegen einen Großteil von Online-Gegnern auf papergames.io. Von Menschen lässt sich mein Programm nur schlagen, wenn man seine deterministische Funktionsweise analysiert, oder ein sehr erfahrener Vier-gewinnt-Spieler ist.

Außerdem könnte sich mein Programm durch ein wenig Anpassungen auch für andere Spielbrettgrößen mit einer geraden Anzahl an senkrechten Spielfeldern eignen, für welche derzeit noch keine Lösungen existieren. Bei Spielbrettern mit einer ungeraden Höhe würde die Zugzwang-Taktik nicht funktionieren.

Während der kontinuierlichen Weiterentwicklung meiner künstlichen Intelligenz sind fortwährend neue Ideen aufgetaucht, während ältere Implementierungen aufgegeben

wurden. Selbst jetzt, kurz vor Abschluss meiner Arbeit, gibt es noch Ideen, die es nicht in die finale Version des Programms geschafft haben. Einige davon sind:

- Unterstützung anderer Spielbrettgrößen mit gerader Höhe (z.B. 8x8)
- Eine variable Tiefenanpassung (je weiter das Spiel, desto höher die Tiefe, da die verbleibenden möglichen Spielstellungen immer geringer werden, wodurch eine höhere Rechentiefe weniger Zeit beansprucht)
- Eine Vorberechnung und anschließende Speicherung von Eröffnungszügen in einer Datenbank
- Einsetzung von Machine Learning zur Vergabe von Scores

Da ich zu Beginn dieser Arbeit nur die Grundlagen von Vier gewinnt kannte, bin ich sehr stolz darauf, nun eine künstliche Intelligenz entwickelt zu haben, welche fortgeschrittene Vier-gewinnt-Taktiken gewinnbringend verwendet und in den meisten Fällen auch für mich unbesiegbar bleibt.

Literaturverzeichnis

2swap: Threat Analysis | Connect 4 Strategy: https://www.youtube.com/watch?v=uY5_juNHqyo, 13. August 2023, aufgerufen am 3. November 2023

James D. Allen (Hrsg. Sterling Publishing Company, Incorporated): The Complete Book of Connect 4: History, Strategy, Puzzles, New York City 2010

Victor Allis: A Knowledge-based Approach of Connect-Four, Oktober 1988, http://www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf, Oktober 1988, aufgerufen am 1. November 2023

Stefan Edelkamp, Peter Kissmann: Symbolic Classification of General Two-Player Games, https://fai.cs.uni-saarland.de/kissmann/publications/ki08-ggpsolver.pdf, September 2008, S. 2, aufgerufen am 1. November 2023

Jennifer Mueller: Always Win at Connect 4: The Strategy That Will Keep You on Top, https://www.wikihow.com/Win-at-Connect-4, 24. März 2023, aufgerufen am 5. November 2023

Abbildungen

Abbildung 1: Ryan Stellabotte: Toy Story: Catching Up with Howard Wexler, Inventor of the Classic Game Connect 4, Stand: 11. Juli 2018, aufgerufen am 01.11.2023 https://news.fordham.edu/fordham-magazine/toy-story-catching-up-with-howard-wexler-inventor-of-the-classic-game-connect-4/

Abbildung 3: Gilles Vandewiele: Creating the (nearly) perfect connect-four bot with limited move time and file size, Stand: 28. November 2017, aufgerufen am 01.11.2023 https://towardsdatascience.com/creating-the-perfect-connect-four-ai-bot-c165115557b0

Abbildung 5: Akshay L Aradhya: Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning), Stand: 16. Januar 2023, aufgerufen am 01.11.2023 https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

Weitere Abbildungen sind (evtl. bearbeitete) Screenshots meines Programms.

Programmierung

1. KI: Rust: https://www.rust-lang.org/
2. Oberfläche: React: https://react.dev/