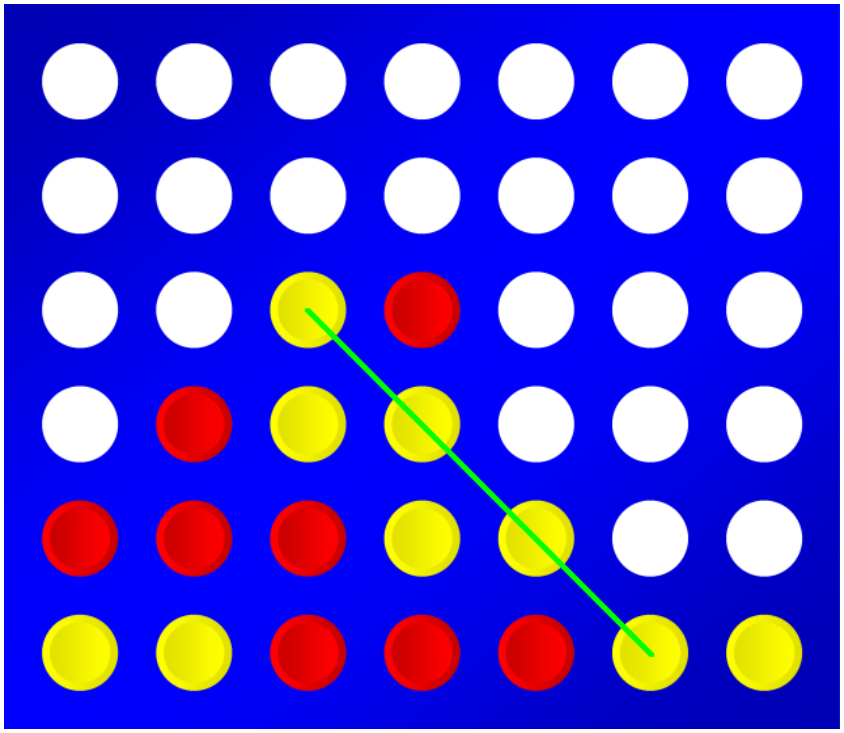
Der Negamax - Spielalgorithmus erklärt am Beispiel des deterministischen 2 - Personen Nullsummenspiels „4 Gewinnt“ mit GUI in Java



Maximilian Schneider

Informatik

Städtisches Gymnasium Rheinbach

Fachlehrer: Rolf Faßbender, IF LK 1

Schuljahr 2019/2020 – 02.04.2020

Inhaltsverzeichnis

Einleitung 3
4-Gewinnt 4
Spielerklärung 4
Eigenschaften 4
Umsetzung als digitales Spiel 4
Probleme und Herausforderungen 4
Erklärung der GUI 5
Notation 5
Programmierung in IntelliJ IDEA 6
Der Negamax-Algorithmus 7
Bedingungen 7
Spielbäume 7
Minimax-Algorithmus 8
Negamax-Algorithmus 9
Erklärung des Quellcodes 10
Bewertungsmethode 11
Beschränkter und unbeschränkter Negamax 12
Alpha-Beta Suche 12
Reflexion und Selbstkritik 13
Literaturverzeichnis 15
Bildverzeichnis 16
Anhang 17

Einleitung

Spiele waren schon immer eine faszinierende Beschäftigung für Menschen. Egal ob es Glückspiele, Legespiele, Denkspiele, Rollenspiele oder Strategiespiele sind, der Spaßfaktor ist überall vorhanden. Seitdem es elektronische Spiele gibt, hat sich jedoch einiges in der Spielewelt verändert. Vor etwa 50 Jahren, noch als technische Versuche an Universitäten angesehen, entwickelten sich die Computerspiele zu einer Freizeitgestaltungsmöglichkeit. In kurzer Zeit ist eine Spielebranche entstanden, die jährlich Umsätze in Milliardenhöhe macht. Das Spektrum an Spielen ist riesig. Hinzu kommt die Variationsbreite einzelner Spiele. So gibt es Spiele in den man allein agiert und welche die man zu zweit oder in einer Gruppe spielt. Aber was ist, wenn man bei solchen keinen zum Spielen hat? Dann ist es in vielen Spielen möglich gegen einen Computergegner (Nicht-Spieler-Charakter oder kurz NPC) zu spielen. Dieser NPC wird von einer künstlichen Intelligenz (kurz KI) gesteuert, welche durch Algorithmen eigene Züge und Verhaltensweisen berechnet und schließlich ausführt. Der „Negamax“ ist ein sogenannter Spielalgorithmus und wird in Zweipersonenspielen verwendet, wo ein Spieler durch einen NPC ersetzt werden kann. Dazu gehört auch das Spiel „4 Gewinnt“.

Der Großteil von Spielern sieht immer nur das fertige Produkt und hinterfragt nicht, welche Algorithmen im „Hintergrund“ laufen, oder wie und warum ein NPC in einer Situation nach festgelegten Schritten handelt. Wäre es da nicht verlockend selbst mal in die Welt der Spieleentwickler hineinzuschauen und versuchen ein eigenes Spiel mit NPC auf die Beine zu stellen?[1]

Mit dieser Facharbeit beabsichtige ich eine Grafische Benutzeroberfläche (kurz GUI) für das 2-Personen Rundenstrategiespiel „4 Gewinnt“ zu entwickeln und den Spielalgorithmus „Negamax“ in das Spiel zu implementieren und im weiteren Verlauf zu erläutern.

[1] Vgl. (12), „Non-Player Character“

4-Gewinnt

Spielerklärung

Das von Howard Wexler erfundene Strategiespiel „4 Gewinnt“ ist ein 2-Personen Spiel, bei dem Spieler das Ziel verfolgen, als Erster vier der eigenen Spielsteine entweder diagonal, waagerecht oder senkrecht in eine Linie zu bringen. Die Spieler werfen dabei abwechselnd Spielsteine in das senkrecht stehende Spielbrett, welches aus sieben Spalten und sechs Zeilen besteht (siehe Titelseite).[2]

Eigenschaften

Bei „4 Gewinnt“ handelt es sich um ein **endliches deterministisches Nullsummenspiel.** Das Spiel endet immer nach einer bestimmten Anzahl an Zügen und ist somit endlich. Des Weiteren ergibt die Summe der Gewinne und Verluste beider Spieler gleich „null“, weshalb „4 Gewinnt“ die Eigenschaft eines Nullsummenspiel besitzt. [3]

Unter einem deterministischen Spiel versteht man in der Informatik ein Spiel, bei dem nur reproduzierbare Zustände auftreten. Zukünftige Ereignisse und Ergebnisse können durch den Ablauf von Zuständen eindeutig festgelegt werden. „4 Gewinnt“ hat eine begrenzte Anzahl an Zuständen bzw. Stellungen, welche alle reproduzierbar sind. Aus diesem Grund ist „4 Gewinnt“ ein deterministisches Spiel.[4]

Umsetzung als digitales Spiel

Probleme und Herausforderungen

Bei der Umsetzung des Spiels müssen mehrere Dinge beachtet werden. Das Spielbrett ist nur schwer 3-dimensional umsetzbar und wird deshalb der Einfachheit als 2-dimensionales Spielbrett dargestellt. Die Möglichkeiten, wo Steine in die Reihen geworfen werden, sind durch Pfeile markiert und durch das Drücken der Pfeile werden die jeweiligen Spielsteine in die Reihe „geworfen“. Die Gravitation bzw. das Fallen der Steine wird durch eine Animation, in der der Stein das Spielbrett „hinuntergleitet“ bzw.

[2] Vgl. (1), „Vier Gewinnt“
[3] Vgl. (4), „Nullsummenspiel“
[4] Vgl. (6), „Was bedeutet deterministisch?”

konstant an Höhe verliert, veranschaulicht. Das Spielfeld besteht aus einer „Wand“ (Dunkelgraues Quadrat) mit 42 „Löchern“ (Hellgraue Kreise), die die jeweilige Position der Spielsteine anzeigen.

Erklärung der GUI

Ähnlich wie im richtigen Spiel, gibt es einen Spieler, der gelbe Steine hat (Spieler 1) und einen, der rote Steine besitzt (Spieler 2). Die verschiedenen Einstellungen werden neben dem Spielfeld mithilfe eines JLabel dargestellt. Dort findet sich die Option „Animation“, welche die Animation für die Steine aktiviert bzw. deaktiviert. Diese Funktion kann genutzt werden, um die Wartezeit, bis der Stein in seine Position gefallen ist, zu verringern. Der nächste Spieler darf nämlich erst seinen Spielstein werfen, wenn der Stein vom Gegenspieler an seiner Position ist.

Des Weiteren gibt es die Funktion „Zufälliger Einwurf“, welches einem Spieler die Möglichkeit gibt, gegen einen NPC zu spielen, der seine Steine zufällig in eine Spalte wirft. Diese Option war vor allem für Testzwecke initiiert und als Vorgänger des „Negamax“ eingebaut.

Der „Negamax“ ist schließlich eine weitere Wahlmöglichkeit, welche dem Nutzer der GUI die Möglichkeit bietet, gegen einen NPC zu spielen, der den Negamax-Algorithmus verwendet. Dazu gibt es ein extra Feld, wo der Spieler die Suchtiefe des Algorithmus bestimmen kann und somit die Stärke des NPC bestimmt.

Außerdem gibt es einen Knopf „Zurücksetzten“, der das Spielfeld löscht und somit die Spieler von vorne anfangen können. Außerdem gibt es die Möglichkeit, die GUI im Vollbild Modus zu starten und die Farben der Spielsteine zu ändern.

Das Programm berechnet nach jedem Zug, ob ein Spieler gewonnen hat. Wenn dies zutrifft, dann wird der Gewinner des Spiels angezeigt und es bietet sich die Möglichkeit, das Spiel neu zu starten (siehe Abbildung 9).

Notation

Um Spielsteine immer an die richtige Stelle einzusortieren und das Spielfeld mit allen Steinen im Überblick zu halten, muss man den Spielsteinen einen Wert zu schreiben, der es ermöglicht, die Spielsteine voneinander zu unterscheiden und wiederzufinden. Die Notation ergibt sich daraus, dass Reihen und Spalten durch Zahlen gekennzeichnet

werden. Spalten werden von links nach rechts nummeriert (von 0 bis 6) und Zeilen von oben nach unten (von 0 bis 5) (siehe Abbildung 1). Wenn also ein Spieler einen Spielstein in die mittlere Reihe „einwirft“, so landet der Stein bei dem Platz [3] (Spalte) [5] (Zeile) (siehe Abbildung 2).

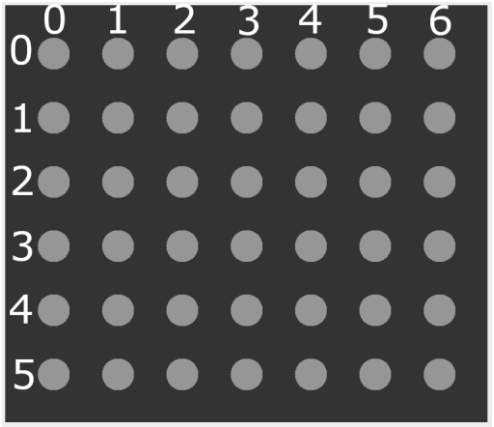


Abbildung 1: Notation des 4-Gewinnt Spielbretts

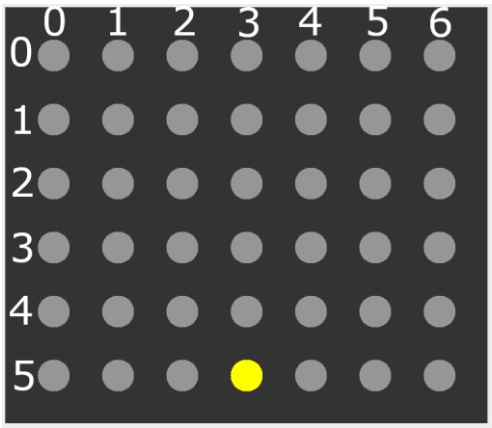


Abbildung 2: Spielstein wird in das Feld [3] [5] geworfen

Programmierung in IntelliJ IDEA

Da ich für dieses Projekt eine IDE (engl.: „integrated development environment“, dt.: „Integrierte Entwicklungsumgebung“) brauchte, habe ich mich für die IDE IntelliJ IDEA entschieden. Meine Wahl fiel auf diese IDE, da ich diese bereits bei mehreren Projekten genutzt habe und IntelliJ IDEA eine Vielzahl an vorinstallierten Plugins besitzt. Fehler und Probleme werden live im Editor angezeigt und es werden Vorschläge zur Verbessrung bzw. zur Behebung der Fehler vorgeschlagen. Nichtsdestotrotz war es nicht leicht eine funktionierte GUI aufzustellen und diese benutzerfreundlich zu gestalten.

Der Negamax-Algorithmus

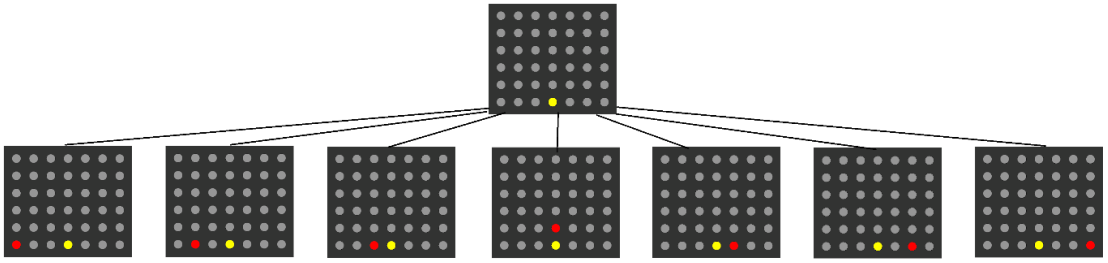
Bedingungen

Damit man in einem Spiel den Negamax-Algorithmus verwenden kann, müssen zwei Bedingen erfüllt sein. Es muss sich um ein Nullsummenspiel handeln und es muss ein Zweipersonenspiel sein. Beide Bedingungen sind bei dem Spiel „4 Gewinnt“ gegeben.[5]

Spielbäume

Bei 2-Personen Spielen führen beide Spieler abwechselnd Züge aus. Nach jedem Zug ändert sich das Spielfeld und es liegt eine andere Konstellation vor. Das Ziel beider Spieler ist es, das Spielfeld so zu modifizieren bzw. zu ergänzen, dass sie dadurch gewinnen. Problematisch ist jedoch die Bestimmung des jeweiligen bestmöglichen nächsten Zuges, der den Spieler zu seinem Ziel führt. Fast alle spielführenden Programme nutzen Spielbäume (engl.: „Game trees“) um dieses Problem zu lösen.

Betrachtet man das Spiel „4 Gewinnt, gibt es bei jeder Spielsituation bis zu 7 verschiedene Züge, die ein Spieler gehen kann. Liegt eine Spielsituation vor, kann also ein Spieler zwischen diesen Zügen wählen und so eine neue Spielsituation ermöglichen. Diese Spielsituation hat schließlich dann auch bis zu sieben weitere verschiedene Züge, die ein Spieler wählen kann. Veranschaulichen kann man dies in einem Baum. Die Knoten sind die unterschiedlichen Spielsituationen, welche jeweils bis zu sieben weitere Teilbäume besitzen. Blätter sind im Fall des Spiels „4 Gewinnt“ eine Spielsituation, bei der ein Spieler gewonnen hat, oder wenn alle Steine verbraucht wurden. Die Wurzel repräsentiert die aktuelle Spielsituation.[6]



[5] Vgl. (5), „Der Alpha-Beta-Algorithmus und Erweiterungen bei Vier Gewinnt“
[6] Vgl. (7), „Spielbaum“

Abbildung 3: Aufbau eines Spielbaums - Die Knoten besitzen genau wie die Wurzel im Bild sieben weitere Teilbäume, welche aber nicht abgebildet sind

Minimax-Algorithmus

Bei einem Spiel zwischen zwei Menschen, würde jeder Spieler versuchen, die Gewinnchancen für sich zu maximieren und die seines Gegenübers zu minimieren.

Der Minimax-Algorithmus dient zur Ermittlung des aktuellen bestmöglichen Zugs bei einer bestimmten Spielsituation. Der Algorithmus ist der Vorgänger des Negamax-Algorithmus und sichert, wie der Negamax-Algorithmus, dem Spieler immer den höchstmöglichen Gewinn. Bei dem Algorithmus werden zwei ausführende Spieler, Spieler A bzw. Spieler MAX und Spieler B bzw. Spieler MIN, benutzt. Der Minimax versucht nun die maximale Chance zu einem Sieg für Spieler MAX zu errechnen und die minimale Chance für den Spieler MIN. Die Spielstellungen werden bei dem Minimax-Algorithmus in einem Spielbaum abgespeichert. Es wird abwechselnd ein Zug ausgewählt, wobei der Algorithmus im Wechsel den bestmöglichen Zug für Spieler MAX und darauffolgend den schlechtesten Spielzug für Spieler MIN bestimmt. (In Abbildung 5 bedeutet Min, die Auswahl für den schlechtesten Zug bzw. Entscheidung und Max für die Auswahl des bestmöglichen Zuges.)

Dadurch ist gewährleistet, dass bei einem perfekten Spiel beider Spieler, der bestmögliche Zug für den Spieler Max berechnet wird, da der Gegenspieler, Spieler B, versucht, die Gewinnchancen für Spieler A weitgehend gering zu halten.

Umso höher der Wert des Zuges ist, desto besser ist dieser. Mit diesem Auswahlverfahren werden alle möglichen Züge berechnet, insofern es keine Begrenzungen gibt. Es wird davon ausgegangen, dass der Mensch, der gegen den NPC spielt, immer den besten Spielzug macht und somit keine Fehler begeht.[7] [8] [9]

Ein ausführliches Beispiel eines Minimax-Verfahrens befindet sich im Anhang.

[7] Vgl. (2), „Sind Computer (un)schlagbar? Computerstrategien für Zweipersonenspiele“
[8] Vgl. (5), „Der Alpha-Beta-Algorithmus und Erweiterungen bei Vier Gewinnt“
[9] Vgl. (11), „Variable Depth Search“

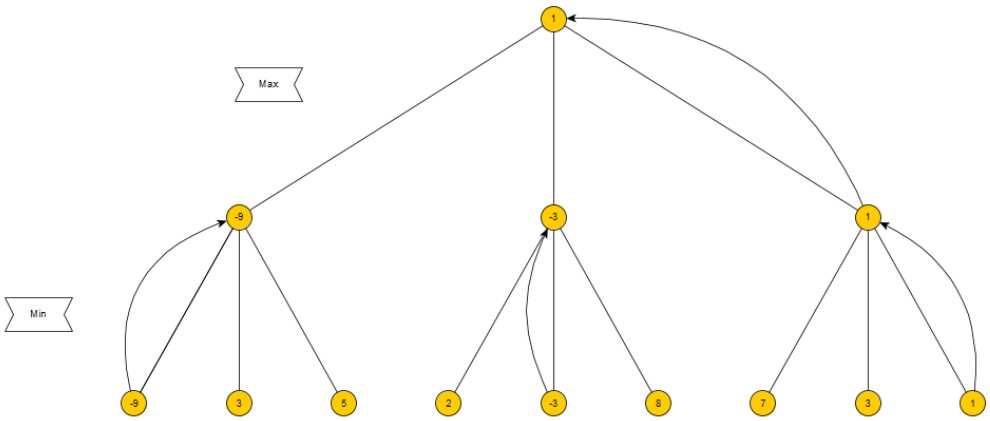


Abbildung 4: abstrakter Spielbaum (Minimax-Variante)

Negamax-Algorithmus

Der Minimax-Algorithmus ist aufwendig zu implementieren und der Quellcode ist vergleichsweise zu anderen Suchalgorithmen ziemlich groß. Deshalb hat man versucht den Code zu vereinfachen und der Negamax-Algorithmus ist daraus entstanden.

Der Negamax wird rekursiv auf alle zukünftigen Stellungen angewandt und eine Bewertung der jeweiligen Spielsituation berechnet. Der Algorithmus speichert, wie der Minimax-Algorithmus, alle Nachfolgestellungen in einem Spielbaum und kann so eine Auswahl der Züge vornehmen.

Der Negamax-Algorithmus arbeitet mit einem ähnlichen Prinzip wie der Minimax-Algorithmus. Jedoch versuchen bei dem Negamax-Algorithmus beide Spieler ihre Züge zu maximieren, um für sich selbst ein maximales Ergebnis zu erzielen. Dadurch muss nicht mehr unterschieden werden, welcher Spieler an der Reihe ist. Damit ein Spieler den besten Spielzug für sich errechnen kann, wird nach jedem Spielerwechsel die Bewertung der vorherigen Züge negiert bzw. mit -1 multipliziert, weshalb nach jedem Wechsel immer das Maximum der negierten Bewertungen berechnet wird. Dies wird gemacht, da der schlechteste Zug für den Gegenspieler, der beste Zug für den jeweiligen Spieler selbst ist. Wenn also alle Bewertungen negiert werden, dann ist der Zug mit der vorherigen schlechtesten Bewertung das Maximum. Das Maximum bzw. der schlechteste Zug für

den Gegner wird dann schließlich von dem anderen Spieler als Zugmöglichkeit gewählt. [10] [11]

Ein ausführliches Beispiel eines Negamax-Verfahrens ist im Anhang.

Erklärung des Quellcodes

Zu Beginn wird der Methode „Negamax“ eine Spielposition und eine Tiefe übermittelt.

Wenn die Tiefe gleich Null ist oder einer der beiden Spieler gewonnen hat, indem er 4 Steine in eine Reihe geworfen hat, dann soll eine Bewertung der aktuellen Spielstellung folgen. Dies passiert also nur bei Blättern des Spielbaums.

Wenn die Suchtiefe des Baums nicht „null“ beträgt oder keiner gewonnen hat, dann werden Züge für die aktuelle Spielstellung berechnet. Bei „4 Gewinnt“ können dies insgesamt bis zu sieben verschiedene Züge sein. Bei anderen Spielen wie Schach ist diese Zahl noch deutlich größer.

Diese Züge werden schließlich in einer for-Schleife ausgeführt und die Bewertung derer Spielstellung wird durch einen rekursiven Aufruf der Negamax-Methode in einer Variable gespeichert.

Im Anschluss folgt eine if-Abfrage, welche den größten Wert aus allen Bewertungen der möglichen Züge zur aktuellen Spielstellung bestimmt und speichert. Dieser wird am Ende der Methode zurückgegeben, wenn die Methode nicht schon am Anfang durch die erste if-Anweisung abgebrochen wird.[12]

[10] Vgl. (5), „Der Alpha-Beta-Algorithmus und Erweiterungen bei Vier Gewinnt“
[11] Vgl. (8), „Adversariale Suche für optimales Spiel: Der Minimax-Algorithmus und die Alpha-Beta-Suche“
[12] Vgl. (5), „Der Alpha-Beta-Algorithmus und Erweiterungen bei Vier Gewinnt“

```
public int negamax(Spielpostion, Tiefe){
        if(Tiefe==0 || Spiel Ist Vorbei){
                return Bewertung(Spielposition);
        }
        PunkteBesterZug = -Unendlich;
        Züge = GeneriereZüge(Spielposition);
        for(int i = 0; i<Züge.length; i++){
                MacheZug(Züge[i]);
                PunkteVomAktuellenZug = -negamax(Spielposition, Tiefe -1);
                if(PunkteVomAktuellenZug > PunkteBesterZug){
                        PunkteBesterZug = PunkteVomAktuellenZug;
                }
                MacheZugRückgängig(Züge[i]);
        }
        return PunkteBesterZug;
}
```

Abbildung 5: Pseudocode des Negamax-Algorithmus

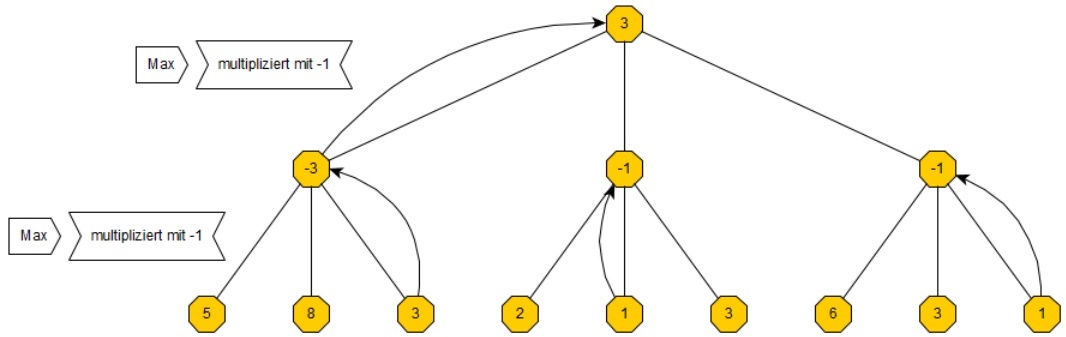


Abbildung 6: abstrakter Spielbaum (Negamax-Variante)

Bewertungsmethode

Ein wichtiger Bestandteil des Negamax-Algorithmus ist die Bewertungsmethode, welche einen Wert für die übergebene Spielstellung ermittelt und diesen zurückgibt. Umso besser die Bewertung für eine Spielstellung ist, desto besser kann der Algorithmus den bestmöglichen Zug bestimmen. Es werden dabei mehrere Kriterien untersucht. Durch viele Abfragen wird schließlich ein Wert für Spieler 1 und einen für Spieler 2 ermittelt. Dieser wird berechnet, indem den Spielern Punkte für die verschiedenen Positionen der Spielsteine auf dem Feld zu gewiesen werden. Ein einzelner Spielstein bringt nur einen Punkt, wenn zwei nebeneinander liegen, bringen diese beiden Steine insgesamt 15

Punkte. Bei drei bzw. vier Steinen gibt es dementsprechend noch höhere Werte. Wenn also der Wert für einen Zug steigt, dann ist dieser Zug besser für denjenigen Spieler. Dieser Wert fällt deutlich höher aus, wenn der Spieler versucht Spielsteine nebeneinander zu legen, da schließlich das Ziel des Spiels ist, vier Steine in einer Linie zu haben.[13]

Beschränkter und unbeschränkter Negamax

Das 4-Gewinnt Spielfeld hat insgesamt 42 Plätze für Spielsteine (7 * 6) und ist somit sehr groß im Gegensatz zu anderen Spielen wie TicTakToe. Wenn nun der Negamax-Algorithmus eine vollständige Untersuchung aller möglichen Spielzüge durchgeht, dann wird der Spielbaum üblicherweise zu groß, als dass man ihn in kurzer Zeit durchsuchen könnte. Um dieses Problem zu lösen, kann man eine Suchtiefe einbauen, welche dem Algorithmus erlaubt nur bis zu einer bestimmten Tiefe zu suchen (z.B. bei einer Suchtiefe von 3, kann der Algorithmus maximal drei Züge vorausdenken). Dadurch wird nur ein gewisser Teil des Spielbaumes aufgebaut und die Suchzeit wird erheblich begrenzt. Diese hängt aber auch immer von der Programmierung der Bewertungsmethode ab, da bei einer steigenden Anzahl an Abfragen, auch die Berechnungsdauer steigt.[14]

Alpha-Beta Suche

Bei Vier-Gewinnt gibt es Züge, die weniger Sinn machen als andere, bzw. einem Spieler im weiteren Verlauf nur Nachteile bringen. Diese Züge auch bei dem Negamax-Algorithmus zu verfolgen, kostet unnötige Zeit, denn diese Züge werden auf jeden Fall nicht von dem Algorithmus als bestmöglichen Züge ausgewählt. Aus diesem Grund wurde die „Alpha-Beta Suche“ (engl.: Alpha-Beta Pruning) entworfen.

Durch die „Alpha-Beta Suche“, welche eine optimierte Variante des Minimax und des Negamax-Suchverfahrens ist, können unnötige Teilbäume weggelassen werden, ohne sie zu durchsuchen. Dazu werden zwei weitere Werte (Alpha und Beta) hinzugenommen, welche den Wert des aktuell bestmöglichen Zuges bzw. Spielbaums angeben und wodurch schließlich entschieden werden kann, ob Teilbäume des Suchbaums bearbeitet werden müssen. [15] [16]

Ein ausführliches Beispiel einer „Alpha-Beta Suche“ befindet sich im Anhang.

---

[13] Vgl. (10), „Optimierung und Laufzeitanalyse einer künstlichen Intelligenz für das Spiel Vier gewinnt“

[14] Vgl. (5), „Der Alpha-Beta-Algorithmus und Erweiterungen bei Vier Gewinnt“

[15] Vgl. (8), „Adversariale Suche für optimales Spiel: Der Minimax-Algorithmus und die Alpha-Beta-Suche“

[16] Vgl. (9), „Der Alpha-Beta-Algorithmus“

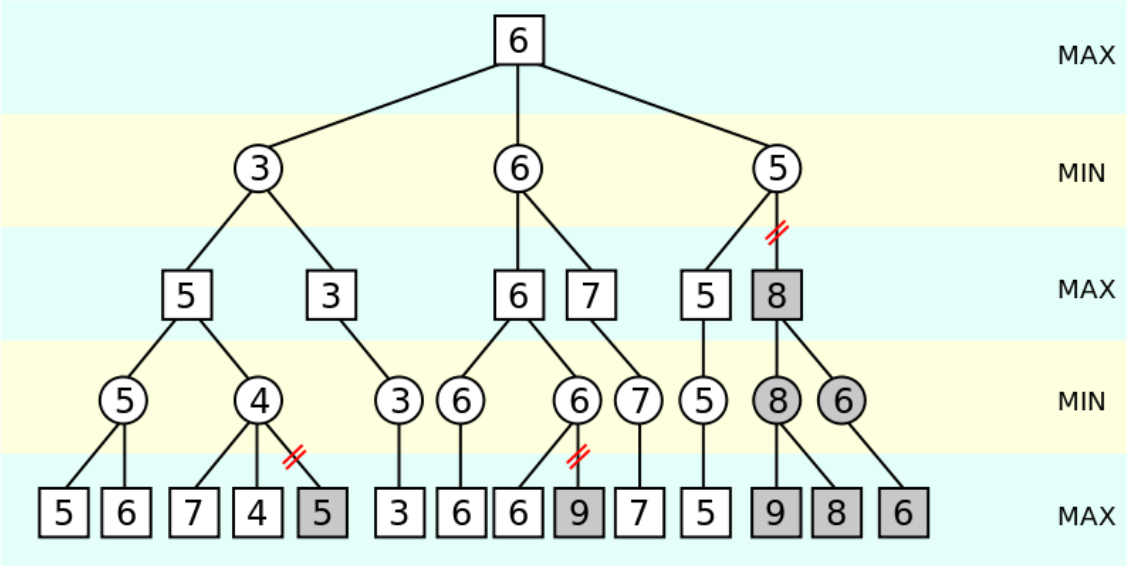


Abbildung 7: Beispiel eines Alpha-Beta Pruning (Minimax-Variante)[17]

Reflexion und Selbstkritik

Mit dieser Facharbeit wurde das Ziel verfolgt, eine GUI für das Spiel „4 Gewinnt“ zu programmieren und den Negamax-Algorithmus für das Spiel zu entwickeln. Mit der Implementierung des Algorithmus wurde ein NPC der von einer KI gesteuert wird, erschaffen.

Die Struktur und das Aussehen der GUI ist einfach gestaltet und bringt nicht viele Probleme auf. Die Skalierung hat jedoch viel Zeit beansprucht und ist in der momentanen Version nur mit einer Auflösung von 1920 * 1080 Pixel optimal spielbar. Nichtsdestotrotz ist der Negamax-Algorithmus erfolgreich implementiert und es ist möglich gegen einen NPC anzutreten, der mögliche zukünftige Züge vorausberechnet und bei einer hohen Suchtiefe nahezu unschlagbar ist. Nur selten entstehen Fehlentscheidungen, welche wahrscheinlich auf Grund der stetig ausbaufähigeren Bewertungsfunktion des Algorithmus auftreten. Zwar ist die jetzige Bewertungsfunktion schon rentabel, kann aber noch verbessert werden. Dadurch kann sich der Algorithmus sicherer für einen Zug entscheiden und es treten weniger Probleme auf.

[17]Vgl. (1), „Connect 4“ (Bildverzeichnis)

Des Weiteren ist die grenzenlose Ausbaufähigkeit des Algorithmus im Verlaufe der Facharbeit deutlich geworden. So gibt es weitere Themen wie die Principal-Variation-Suche oder die Killer-Heuristik, welche Optimierungsansätze für die hohe Suchzeit des Algorithmus zeigen. Mit dieser Facharbeit wurde zwar der Negamax weitgehend erläutert, jedoch ist dieser nur einer von vielen Varianten der Tiefensuche. Zudem gibt es noch weitere Spielalgorithmen, welche auf anderen Prinzipen aufgebaut sind. Es wird stets nach einer besseren Möglichkeit gesucht und viele Ansätze werden ausprobiert. Jedoch würden diese Themen die Facharbeit in ihrer Länge überschreiten.[18]

Ein unbegrenzter Negamax-Algorithmus ohne weiteren Optimierungsansätze ist im Hinblick auf die Laufzeitanalyse nur schwer in Spielen einsetzbar. Die langen Wartezeiten würden den Spaßfaktor ruinieren und Spieler wären abgeneigt das Spiel zu spielen. Wenn jedoch ein begrenzter bzw. optimierter Spielalgorithmus eingesetzt wird, würde dieser zwar nicht immer die perfekte Lösung haben, jedoch ist dies für ein Spiel wie „4 Gewinnt“ oder „TicTacToe“ mehr als nur ausreichend. Bei einem Spiel wie Schach gibt es jedoch eine viel größere Auswahl an Zügen und eine Bewertung eines Spielfeldes ist deutlich aufwendiger. Dadurch müsse man auch die Suchtiefe verändern bzw. verringern. Der standartmäßige Negamax.Algorithmus würde für Schach nicht brauchbar sein, denn die Suchzeit würde bei einer noch höheren Anzahl an Abfragen und möglichen Zügen bei verschiedenen Spielsituationen zu hoch sein.

Durch das Implementieren des Negamax-Algorithmus ist in dieser Facharbeit klar geworden, dass ein Computer auch nur eine begrenzte Anzahl an Rechenoperationen hat und somit bei einer zu hohen Belastung durch viele Abfragen auch einige Zeit braucht, bis dieser ein Ergebnis liefern kann. Durch die Optimierung der Suchzeit kann dies jedoch flächendeckend reduziert werden.

[18] Vgl. (5), „Der Alpha-Beta-Algorithmus und Erweiterungen bei Vier Gewinnt“

Literaturverzeichnis

Internetquellen:

[1]

Bram Schoonhoven, “Vier Gewinnt”, (Zugriff am: 01.04.2020)

https://www.brettspielnetz.de/spielregeln/vier+gewinnt.php

[2]

Daniel Garmann, 2008, „Sind Computer (un)schlagbar? Computerstrategien für Zweipersonenspiele“, (Zugriff am: 01.04.2020)

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=2ahUKEwiwiZu-6MfoAhUPzaQKHeI_Aj4QFjAAegQIAhAB&url=http%3A%2F%2Fprojekte.gymnasium-odenthal.de%2Finformatik%2Fdateien%2FInformatik%2FJahrgangsstufe%2520Q%2F003%2520Unterrichtsreihen%2520Java%2F11%2520Spielstrategien%2FDocs%2F00%2520Aufsatz%2520Spielstrategien.pdf&usg=AOvVaw2zTCxidNqEEbmzo5csItdr

[3]

Game Maker Community, Juni 2014, „Computer (KI) mit Hilfe des Minimax-Algorithmus erstellen“, (Zugriff am: 01.04.2020)

https://www.gm-d.de/wbb/index.php/Thread/22903-Computer-KI-f%C3%BCr-Tic-Tac-Toe-M%C3%BChle-Schach-4-Gewinnt-mit-dem-Minimax-Algorithmus-A/

[4]

Dr. Gudrun Stiller, „Nullsummenspiel“ (Zugriff am: 01.04.2020)

http://www.wirtschaftslexikon24.com/d/nullsummenspiel/nullsummenspiel.htm

[5]

Hendrik Baier, Mai 2006, „Der Alpha-Beta-Algorithmus und Erweiterungen bei Vier Gewinnt“ (Zugriff am: 01.04.2020)

http://www.ke.tu-darmstadt.de/lehre/arbeiten/bachelor/2006/Baier_Hendrik.pdf/

[6]

Hendrik, Mai 2010, “Was bedeutet deterministisch?” “ (Zugriff am: 01.04.2020)

https://www.pentagraphix.de/was-bedeutet-deterministisch/

[7]

H.W Lang, Oktober 2006, „Spielbaum“, (Zugriff am: 01.04.2020)

https://www.inf.fh-flensburg.de/lang/algorithmen/graph/spielbaum.htm

[8]

Julius Adorf, Dezember 2009, „Adversariale Suche für optimales Spiel: Der Minimax-Algorithmus und die Alpha-Beta-Suche“, (Zugriff am: 01.04.2020)

http://www.juliusadorf.com/pub/alphabeta-seminar-paper.pdf

[9]

Maria Hartmann, Mai 2017, „Der Alpha-Beta-Algorithmus“, (Zugriff am: 01.04.2020)

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwi-np_m7cfoAhWxVRUIHVLCB3kQFjAAegQIBRAB&url=http%3A%2F%2Fwww.inf.fu-berlin.de%2Flehre%2FSS17%2FPSThInf%2Fnotes%2F03_alphabeta.pdf&usg=AOvVaw2EFlLObTl6R1ALlevt2avS

[10]

Tobias Hilbig, Dezember 2014, „Optimierung und Laufzeitanalyse einer künstlichen Intelligenz für das Spiel Vier gewinnt“, (Zugriff am: 01.04.2020)

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=2ahUKEwjQpYjV5sfoAhWCjKQKHUYqAucQFjAAegQIBRAB&url=https%3A%2F%2Fwww.hans-riegel-fachpreise.com%2Ffileadmin%2Fhans-riegel-fachpreise%2FModule%2Fausgezeichnete-arbeiten%2Fhans-riegel-fachpreise-seminararbeit-vwa-2015-hilbig.pdf&usg=AOvVaw1TNjO-_oduhLi_ewDl0Z5Z

[11]

T.A. Marsland und Y. Björnsson, Dezember 2013, „Variable Depth Search“, (Zugriff am: 01.04.2020)

https://www.ru.is/kennarar/yngvi/pdf/MarslandB01.pdf

[12]

USK Unterhaltungssoftware Selbstkontrolle, „Non-Player Character“, (Zugriff am: 01.04.2020)

https://usk.de/alle-lexikonbegriffe/non-player-character/

Bildverzeichnis

[Titelseite]

Ghabes, 2015, „Connect4 Wins.PNG“, Zugriff am: 01.04.2020

https://de.wikipedia.org/wiki/Datei:Connect4_Wins.PNG

[1]

Jez9999, 28.Mai 2007: AB pruning.svg, Zugriff am: 01.04.2020

https://de.wikipedia.org/wiki/Datei:AB_pruning.svg

Anhang

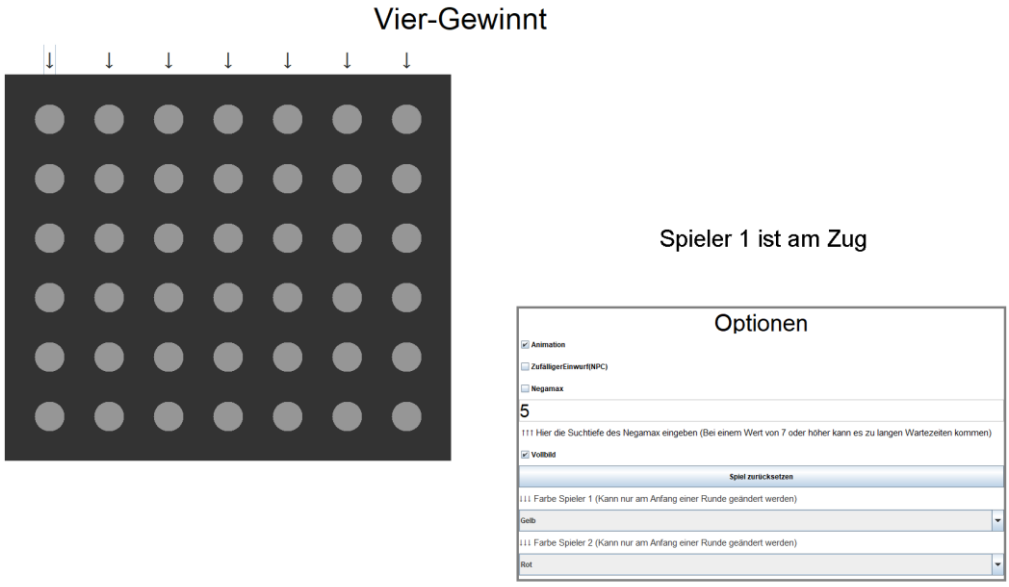


Abbildung 9: Die fertige GUI beim Starten des Programms

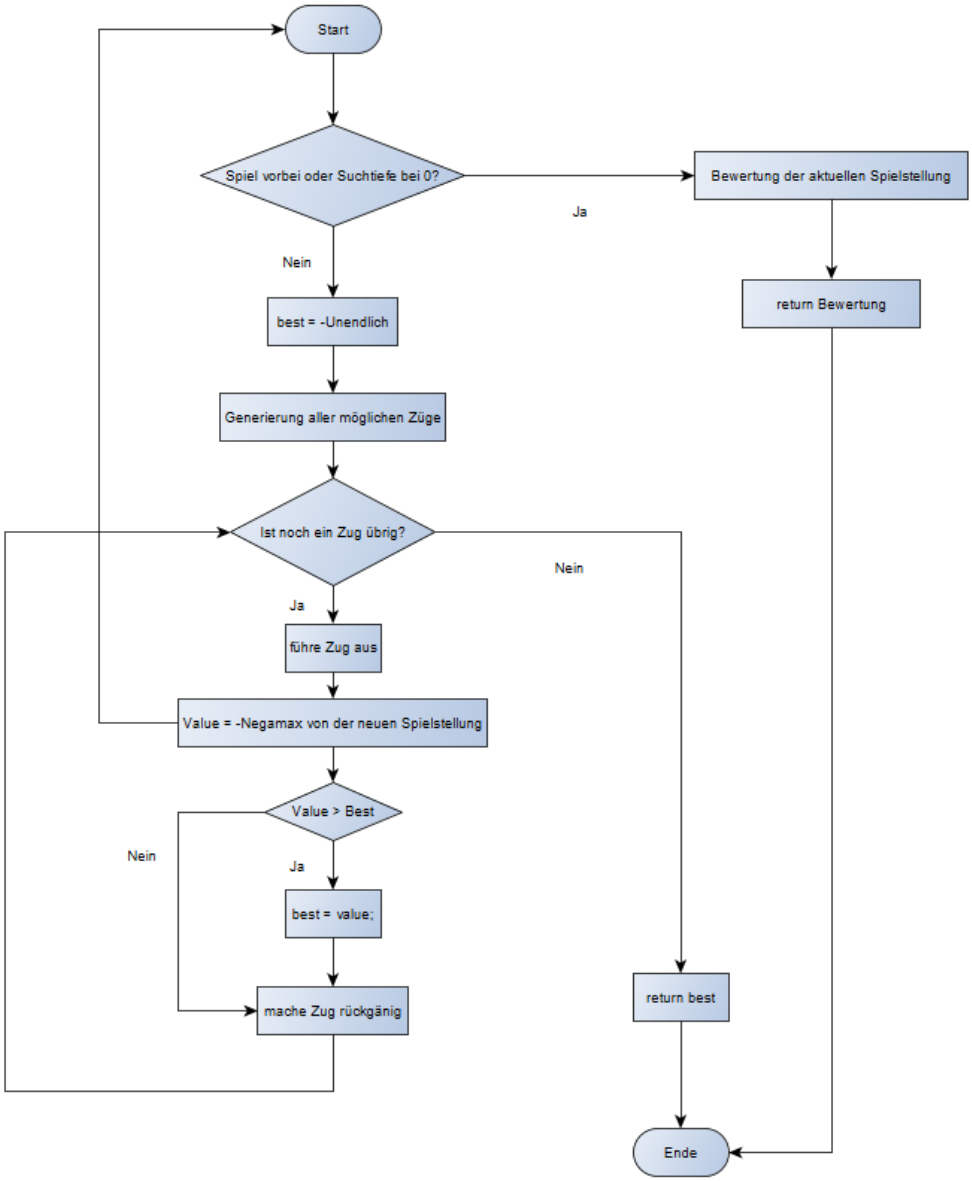
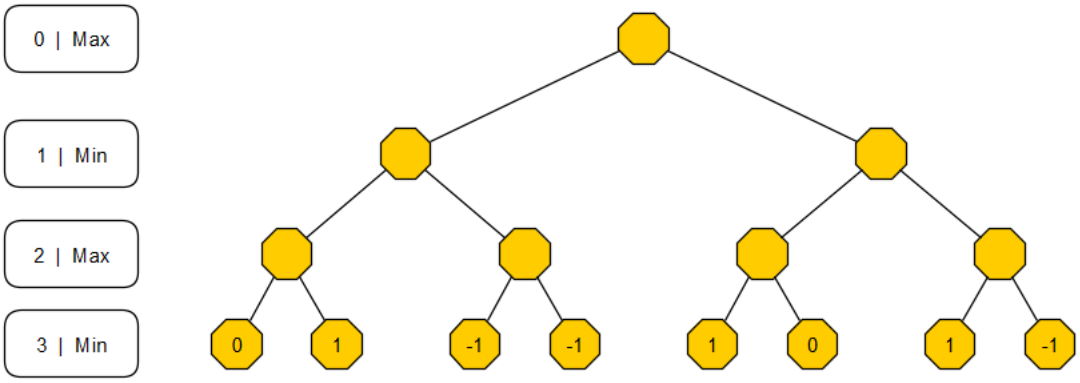


Abbildung 10: Programmablaufplan des Negamax-Algorithmus

Ausführliches Beispiel eines Minimax-Algorithmus anhand eines abstrakten Spielbaums

Gegeben ist folgender Baum:



In den Zeilen mit den Kästchen „Max“ wird der Spieler repräsentiert, der versucht ein maximales Ergebnis zu erzielen. Zeilen mit dem Kästchen mit dem Wort „Min“ repräsentieren den Spieler, der versucht ein minimales Ergebnis zu erzielen. Im Kontext gesehen ist Spieler Max der Computer und Spieler Min ist der Mensch.

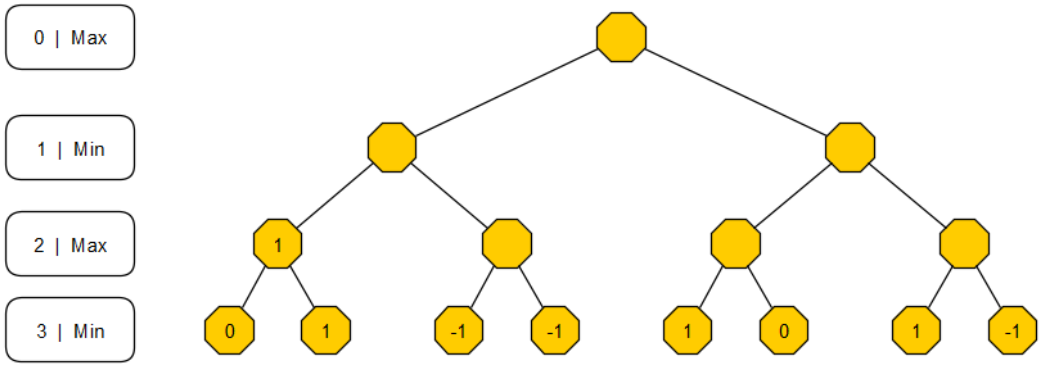
Eine Gewinnstellung für den Computer wird durch die Zahl 1 dargestellt. Die Zahl 0 bedeutet Unentschieden und -1 bedeutet eine Niederlage für den Computer.

Da der Computer versucht seinen Gewinn zu maximieren, versucht dieser zu einer 1 zu kommen.

Nun wird folgendes zuerst beachtet:

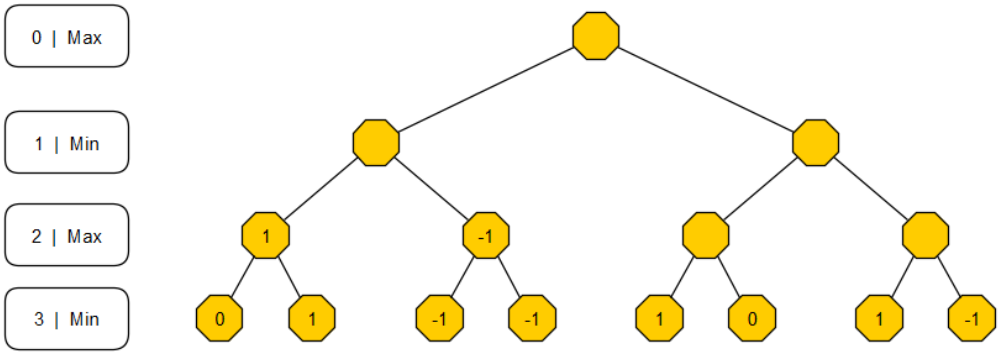
Bei der Ausgangssituation sind die untersten Werte, die minimierten Werte von Spieler Min, weshalb nun Spieler Max an der Reihe ist. Dieser schaut sich die beiden Werte unten

links an und wählt den besseren Wert aus, da er versucht, seinen Gewinn zu maximieren:



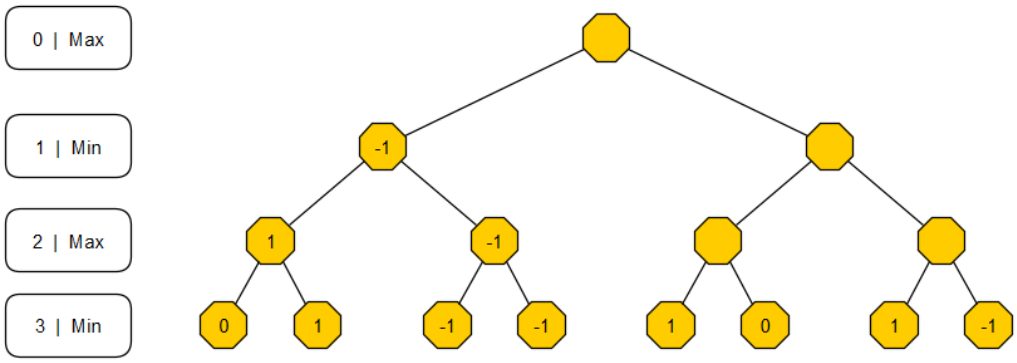
Es wird also die Stellung ausgewählt, wo der Spieler gewinnen könnte.

Diese Entscheidung wird nun auch für die Knoten daneben gefällt:



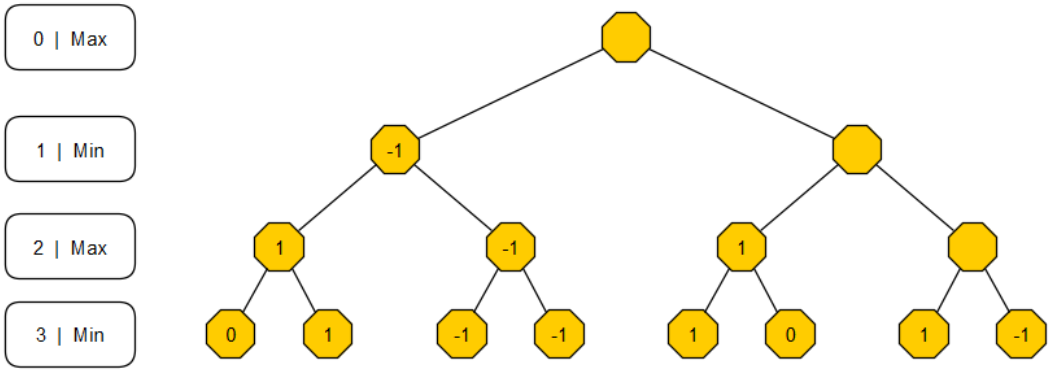
Da beide Züge zu einem Sieg für den Spieler Min bedeuten, hat Spieler Max keine andere Wahl als einen davon zu wählen.

Nun sind beide Werte bestimmt und Spieler Min ist am Zug:

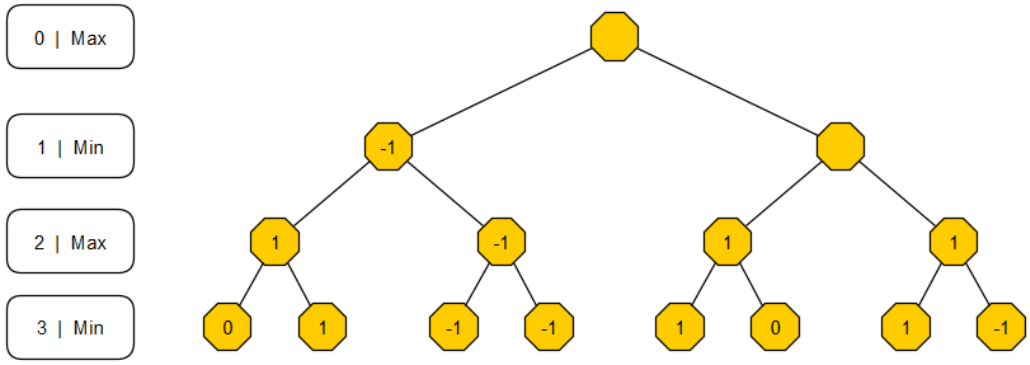


Dieser versucht die Gewinnchance für Spieler Max zu minimieren und wählt den Zug, wo dieser gewinnen würde und nicht wo Spieler Max eine Chance auf einen Sieg hätte.

Nun ist der linke Teilbaum bestimmt, was heißt, dass nun der rechte folgt:

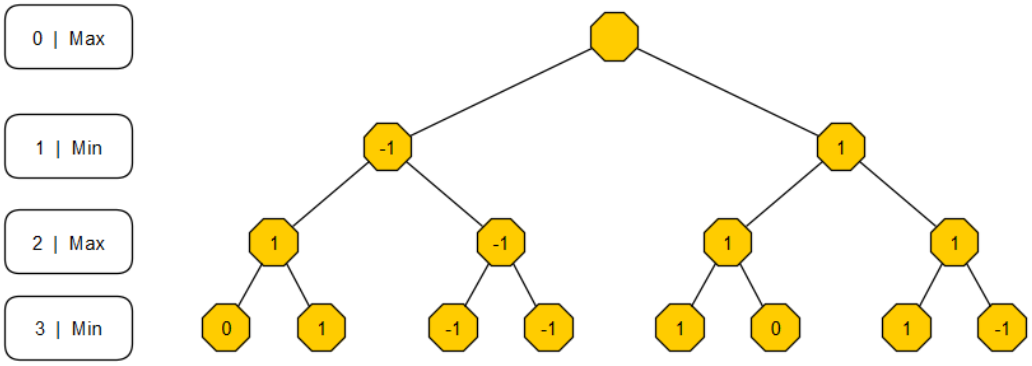


Es wird wie beim linken Teilbaum vorgegangen. Spieler Max wählt seine bestmögliche Chance auf einen Gewinn.



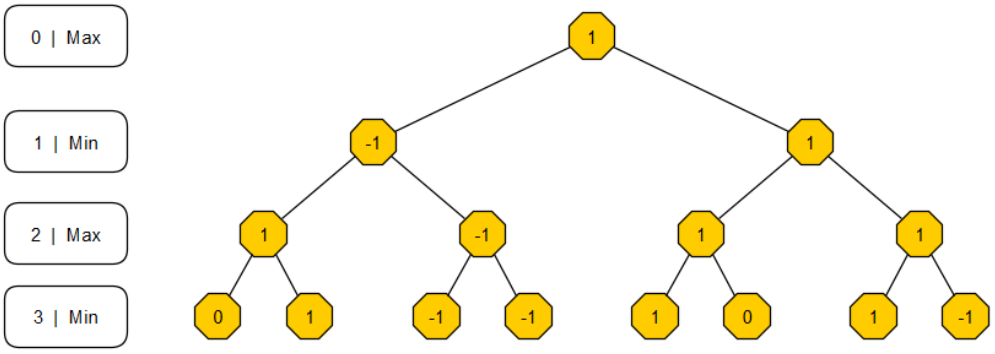
Dies passiert ebenfalls bei der Entscheidung unten rechts.

Nun ist Spieler Min dran:



Dieser kann sich aber nur zwischen zwei Gewinnchancen für Spieler Max entscheiden und nimmt deswegen eine von beiden.

Nun folgt der letzte Zug für Spieler Max:



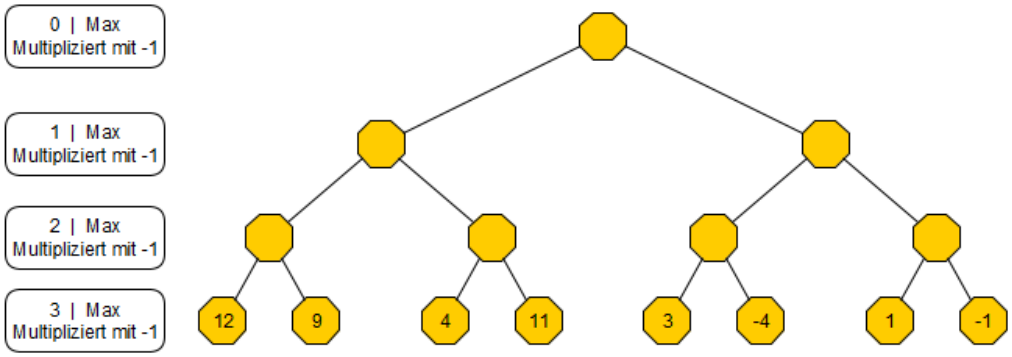
Er kann entscheiden zwischen dem linken Zug, wo Spieler Min (Mensch) wahrscheinlich gewinnen wird und dem rechten Zug, wo er gewinnen wird. Also wählt er den rechten.

Somit ist der bestmögliche Zug für die aktuelle Spielstellung von Spieler Max gefunden.

Dieser Spielbaum gibt jedoch lediglich die Auswahl für zwei Auswahlmöglichkeiten. Bei „4 Gewinnt“ sind dies bis zu 7 verschiedene.

Ausführliches Beispiel eines Negamax-Algorithmus anhand eines abstrakten Spielbaums

Gegeben ist folgender Baum:

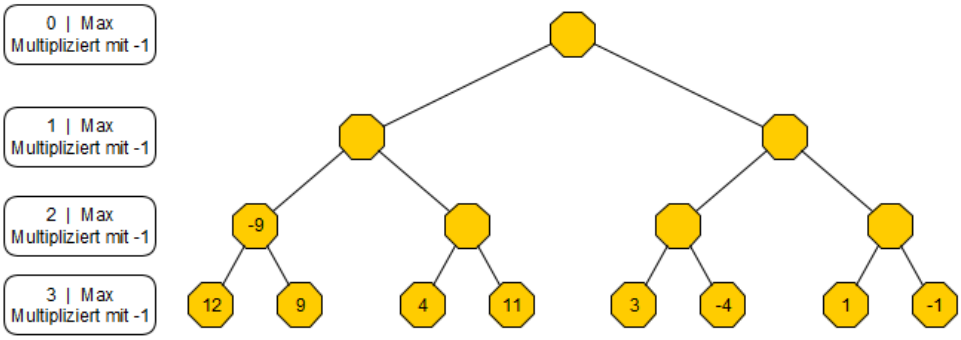


Die Blätter des Baums sind bereits bewertet. Im Gegensatz zu dem Minimax-Beispiel, gibt es in diesem Baum Zahlen die größer bzw. kleiner sind als 1 bzw. -1. Da nun die Negamax-Suchmethode angewendet wird, gibt es keinen Spieler mehr, der den

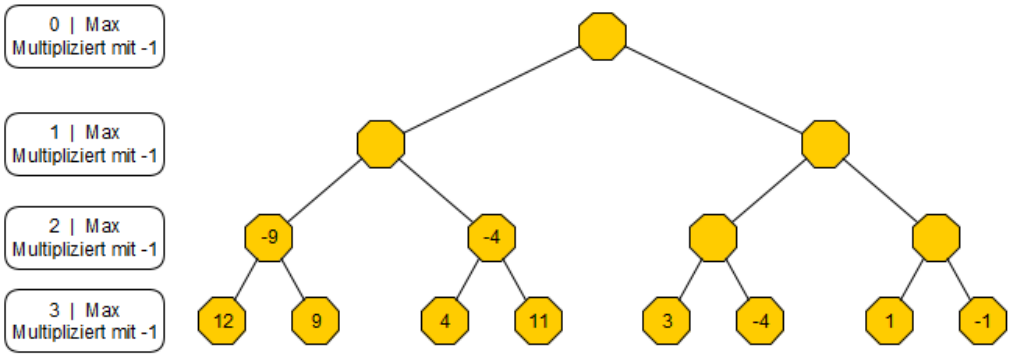
minimalsten Wert sucht, sondern beide Spieler versuchen den maximalsten Wert für sich zu suchen. Zudem werden bei jedem Spielerwechsel die Werte negiert bzw. mit -1 multipliziert.

Nun werden bei dem Algorithmus zuerst die beiden Blätter links unten betrachtet. Beide Werte werden mit -1 multipliziert und schließlich wird der größere Wert übernommen:

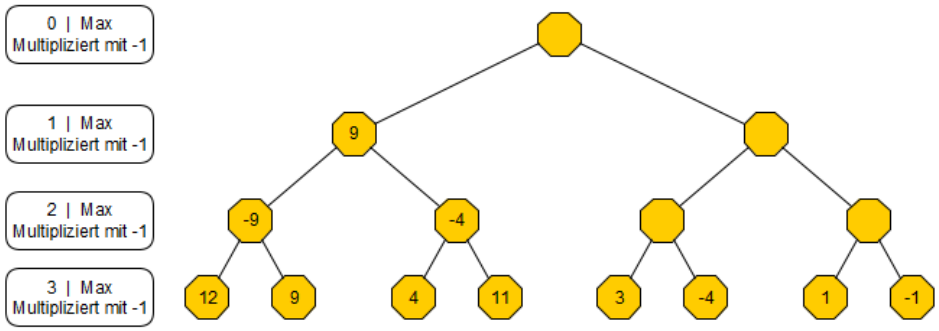
12 wird zu -12 und 9 zu -9. Da -9 > -12 ist, wird der Wert -9 übernommen.



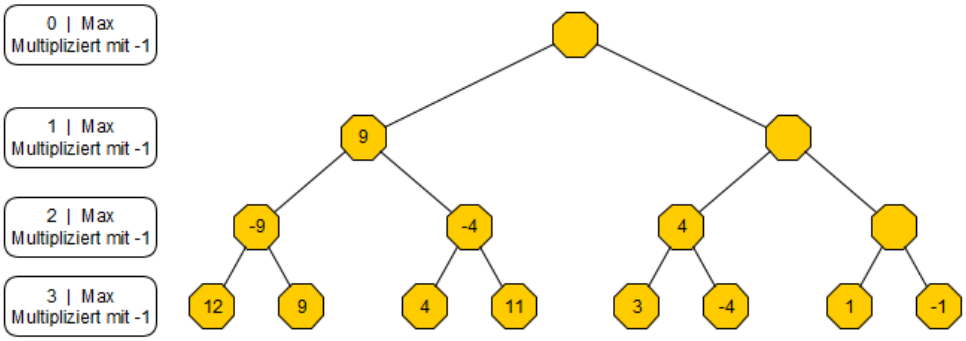
Dies erfolgt auch bei den Blättern daneben:



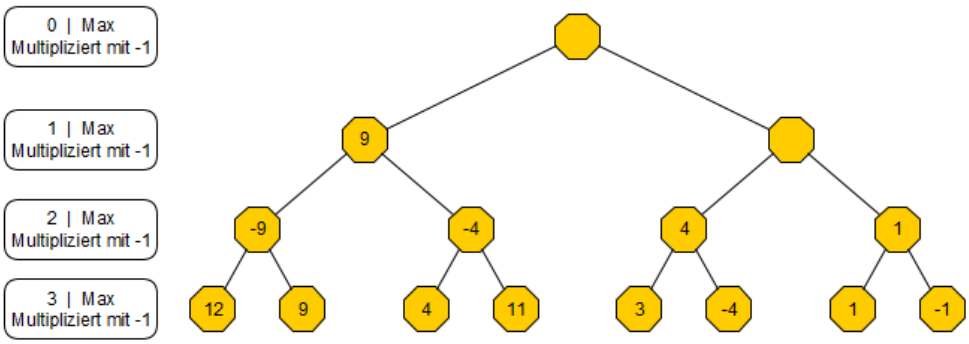
Nun ist der andere Spieler dran und versucht ebenfalls das beste Ergebnis für sich zu erzielen. Die Vorgehensweise bleibt also gleich:



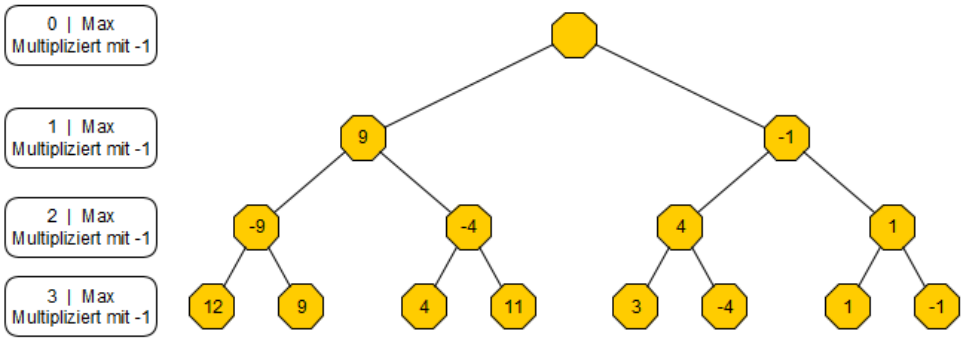
Im weiteren Verlauf wird nun auch der rechte Teilbaum der Wurzel bestimmt:



Aus 3 wird -3 und aus -4 wird 4. Da 4 > -3 ist, wird 4 übernommen.

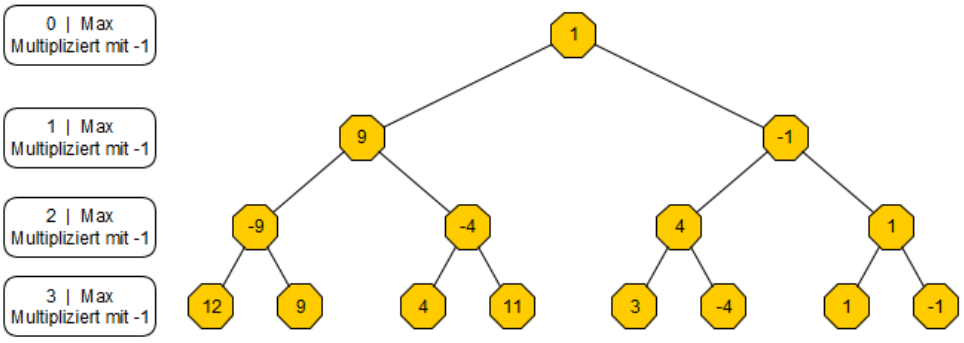


Aus 1 wird -1 und aus -1 wird 1. Da 1 > -1 ist, wird 1 übernommen.



Aus 4 wird -4 und aus 1 wird -1. Da -1 > -4, wird -1 übernommen.

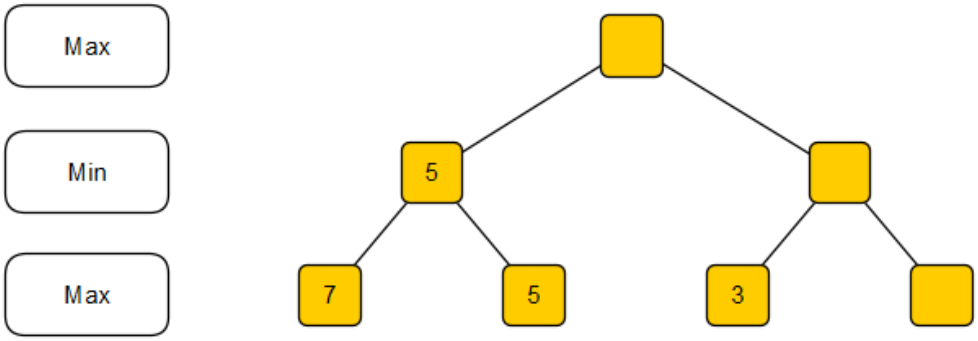
Nun wurden alle Teilbäume durchsucht und die Wurzel kann bestimmt werden:



Das Ergebnis für diesen Zug hat also ein Wert von 1.

Beispiel eines Pruning

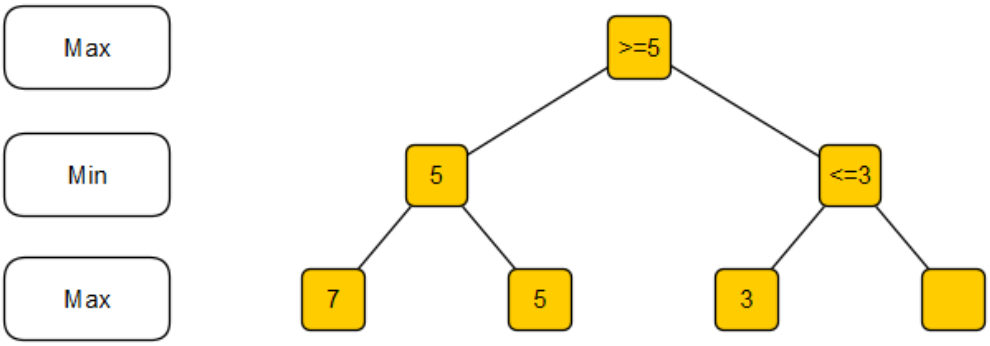
Gegeben ist folgender Spielbaum:



Drei Blätter sind bereits von dem Algorithmus bewertet worden (Minimax-Algorithmus) und der Spieler Min hat bereits seine Auswahl zwischen dem linken (7) und rechtem Teilbaum (5) getroffen. Der Spieler MAX weiß nun auf jeden Fall, dass der Ausgangszug für ihn mit einer Bewertung mit 5 oder höher ausgeht, da er von dem linken Teilbaum einen Wert von 5 zurückbekommt und für sich immer den höheren Wert aussucht. Es könnte nun aber auch möglich sein, dass der rechte Teilbaum von der Baumwurzel einen höheren Wert zurückgibt, weshalb zu mindestens ein Teil davon auch berechnet werden muss.

Für das erste Blatt kommt eine Bewertung von 3 heraus. Aus diesen Informationen kann Spieler MAX schließen, dass der rechte Teilbaum (von der Wurzel) einen niedrigeren Wert zurück gibt als der linke Teilbaum, denn Spieler Min würde in der Suchtiefe 1 den niedrigeren Wert auswählen und dieser wäre auf jeden Fall kleiner oder gleich drei. Da Spieler Max den bestmöglichen Wert aussucht, entscheidet dieser sich für den linken Teilbaum, da dieser einen höheren Wert besitzt (5) und nicht für den rechten Teilbaum, weil dieser höchstens einen Wert von 3 besitzt. Aus diesem Grund muss das eine Blatt nicht bestimmt werden, sondern der Algorithmus kann dieses einfach überspringen, was

zu einer kürzeren Suchzeit führt.



Versicherung der selbständigen Erarbeitung

Ich versichere, dass ich die vorliegende Arbeit einschließlich evtl. beigefügter Zeichnungen, Kartenskizzen, Darstellungen u. ä. m. selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter genauer Angabe der Quelle deutlich als Entlehnung kenntlich gemacht.

Rheinbach, den 02.04.2020

(Ort) (Datum)

(Unterschrift)

Gez. Maximilian Schneider