# Vier Gewinnt

10. Mai 2019

# Inhaltsverzeichnis

# 1 Einleitung

Das Spiel „Vier Gewinnt“ erscheint auf den ersten Blick relativ simpel. Nahezu jeder hatte schon Kontakt mit diesem Spiel und hat es sicher einmal oder auch viele Male gespielt. Egal ob auf einem Blatt Papier im Unterricht, mit kleineren oder größeren Spielen aus Plastik, es ist immer ein faszinierendes Spiel für zwei Personen. Es ist nicht so simpel wie z.B. Tic-Tac-Toe, bei dem es bei geübten Spielern immer in einem Unentschieden endet. Bei dem Spiel „Vier Gewinnt“ geht es viel um das Aufbauen von Fallen, die man dann ausnutzen kann, um zu gewinnen, wenn der Gegenspieler diese ausgelöst hat. Nachdem ich schon einmal ein „Tic-Tac-Toe“ Spiel mit unbesiegbarem Computergegner programmiert hatte [1] , war mir klar, dass ich bei der Facharbeit dies mit einem „Vier Gewinnt“ Spiel umsetzen werde.

# 2 Vier Gewinnt

## 2.1 Spielidee

Die Grundidee des „Vier Gewinnt“ Spieles ist es vier der eigenen Spielsteine bzw. Symbole in eine Reihe zu bringen. Dabei ist es egal, ob dies horizontal, vertikal oder diagonal erreicht wird. Es spielen immer zwei Spieler gegeneinander, die abwechselnd einen Spielstein von oben in das Spielfeld einfügen, der dann so lange herunterfällt, bis er auf das Ende des Spielfeldes oder einen anderen Spielstein stößt. Sobald zum ersten Mal vier Spielsteine eines Spielers eine Reihe bilden, endet das Spiel und der betreffende Spieler gewinnt. Das Spielfeld in der Grundversion ist ein 7x6 Feld. Grundsätzlich sind aber auch größere und kleinere Spielfelder möglich. [2]

## 2.2 Implementation

Zunächst habe ich ein GitHub-Repository erstellt,um effizient die Versionskontrolle zu nutzen, die den Quelltext auch für andere verfügbar macht. Darauf folgend habe ich, wie in der Softwareentwicklung üblich, die Klassendiagramme angelegt,
*Klassendiagramme*
um diese dann nach Umwandlung in ein Implementationsdiagramm umzusetzen.

[1] `https://github.com/boba2fett/TicTacToe`

[2] Lehmann, Jörg „Vier gewinnt“ `http://www.brettspiele-report.de/vier-gewinnt/` Stand: 01.05.2019

*Implementationsdiagramme*
Nach der Umsetzung versuchte ich die Laufzeit etwas zu senken, indem ich eine eigene Klasse erstellte, an deren Objekten die Simulationen durchgeführt wird, um es mehr abzutrennen und die redundante Speicherung der bisherigen Züge zu vermeiden.
*Klassendiagramme VierGame und VierLogik* Am Ende des Entwicklungsprozesses habe ich noch eine grafische Oberfläche erstellt, um das Spiel effizienter zu testen. Dabei fiel mir auf, dass die Funktion zu sehen, mit welchen Feldern man gewonnen hat, sehr praktisch ist, da sonst schnell der Überblick und die Lust am Spielen verloren geht.

# 3 Die KI

## 3.1 Taktik der KI

Die Taktik der KI (Künstliche Intelligenz) ist natürlich zu gewinnen oder eine (unmittelbare) Niederlage zu verhindern. Dazu wird ein Minimax-Algorithmus genutzt, um die beste Entscheidung zu treffen. Bei diesem Verfahren durchläuft der Algorithmus einen Suchbaum, der bei der maximalen Suchtiefe jedem Ergebnis einen Wert zuweist oder, wenn schon vorher ein Sieg oder eine Niederlage auftritt, diesen jeweils einen positiven bzw. negativen Wert zuordnet. Bei der Auswertung wird dann immer das Minimum der gegnerischen Züge gewertet und das Maximum der eigenen, so dass jeder Spieler ein „perfektes “ Spiel spielt. Des Weiteren bevorzugt der Algorithmus einen schnellen Sieg bzw. eine spätere Niederlage.
Neben dem Minimax-Algorithmus könnten auch vorberechnete Eröffnungszüge genutzt werden, aber diese unterscheiden sich für jede Spielfeldgröße. Einige Spiele gelten dadurch auch schon als gelöst:

| height | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 11 | = | | | | | | | | |
| 10 | = | = | | | | | | | |
| 9 | = | = | + | | | | | | |
| 8 | = | = | - | + | - | | | | |
| 7 | = | = | + | = | + | | | | |
| 6 | = | = | - | + | - | - | | | |
| 5 | = | = | = | = | + | + | + | | |
| 4 | = | = | - | = | - | - | - | - | |
| | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | width |

Abbildung 1:
[3]

Zu Abbildung 1:
+ der 1. Spieler gewinnt
- der 2. Spieler gewinnt
= Unentschieden

## 3.2 Schwächen

Theoretisch sind der KI mit dem Minimax-Algorithmus nur Grenzen durch die Rechenleistung bzw. die Tiefe der Simulation gesetzt.

## 3.3 Implementation

Die KI ist in der Klasse AiMiniMax beherbergt. Die dabei wichtigen Methoden sind zum einen die Methoden zum Erzeugen von VierLogik Objekten, an denen die Simulationen durchgeführt werden, und zum anderen die Umsetzung des MiniMax-Algorithmus. Der Aufruf aus VierGame benutzt die Methode aiTurn. In dieser, wird jeder Möglichkeit auf ein Feld zu setzen, ein Wert durch den MiniMax-Algorithmus zugeordnet und danach wird auf das Feld mit dem höchsten evaluierten Wert gesetzt bzw. auf eines der Felder mit dem höchsten evaluierten Wert. Das Herz der KI ist aber der MiniMax-Algorithmus. Der Aufruf erfolgt damit, dass der Zug günstig für den Gegenspieler ist (minimierend für die KI), weil der Zug der KI ja schon für jedes der verfügbaren Felder gemacht wurde, um jedem dieser Felder einen Wert

zuzuweisen. Wenn es hier bei einer der durchprobierten Möglichkeiten für den Gegenspieler zu setzen,zu einem Sieg des Gegenspielers kommt, wird ein negativer Wert zugewiesen, der depth entspricht und zurückgegeben wird. Dies sorgt dafür, dass eine spätere Niederlage besser bewertet wird als eine frühe Niederlage. Bei einem Unentschieden ist der entsprechende Wert 0. Wenn nichts davon eingetreten ist, wird der entsprechende Zug in history gespeichert und der Algorithmus beginnt den nächsten Zug des Spielers zu bewerten. Hier gilt das umgekehrte, da es den Spieler maximiert. Ein früher Sieg bekommt eine sehr positive Bewertung und im nächsten Aufruf ist es wieder minimierend. Wenn die maximale Suchtiefe erreicht ist, wird 0 zurückgegeben und diese Reihenfolge an Zügen wird wie ein Unentschieden behandelt.

### 3.3.1 1. Beispiel

Nun betrachten wir eine Beispielsituation, wie sie vorkommen könnte und sehen uns genauer an, wie die KI ihre Entscheidung trifft. Diese werden durchgeführt auf einem kleinen Feld von der Größe 4x4 und die KI soll einen Verteidigungszug machen. Damit das Log nicht zu lang wird begrenzen wir die KI auf eine Suchtiefe von 4. Die Ausgaben werden erzeugt durch die Klasse AiMiniMaxPrint. Gegeben ist folgende Situation:


Abbildung 2:

Im folgenden setzen wir als Spieler gegen den Computer auf das Feld 3:1, um zu testen, ob er die Bedrohung erkennt.

Wir erkennen: Das tut er. Aber warum?
Dazu sehen wir uns das generierte Log an:

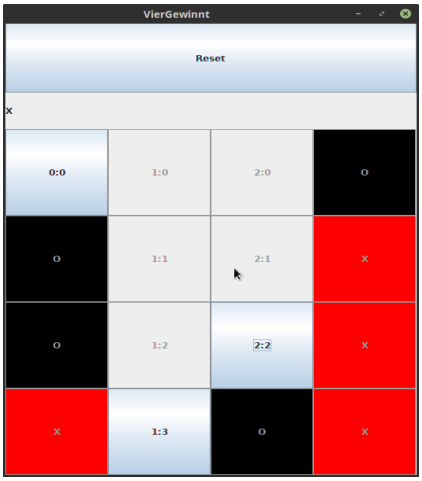
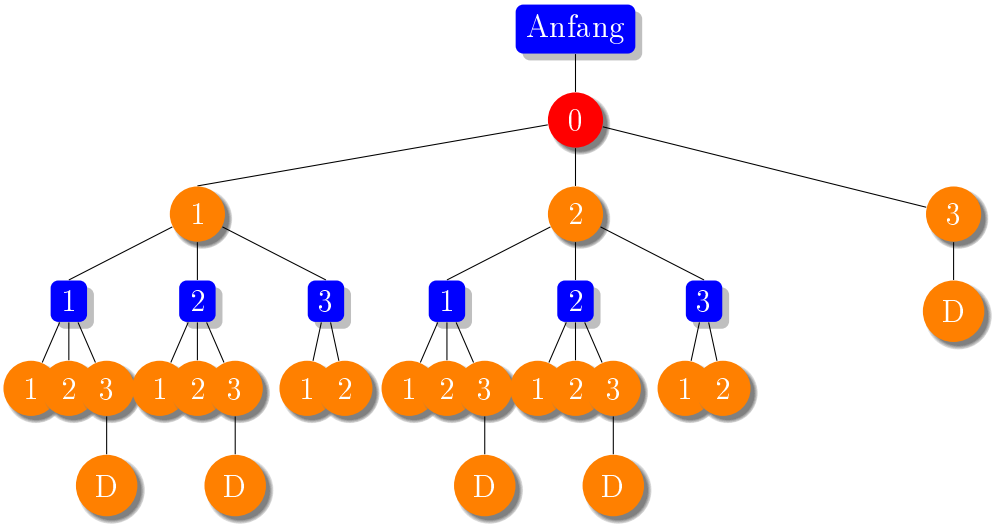



Abbildung 3:

Anhang
Also die wichtigen Zeilen:
0,3 Defeat
1,3 Defeat
2,3 Defeat
3,return 0
Evaluated Value 0
Consider 3

Aber was steht hier? Es werden alle Möglichkeiten durchprobiert, bis man in einer maximierenden Ebene auf einen Sieg trifft oder in einer minimierenden auf eine Niederlage. Zeile 1 zeigt, dass wenn der Computer auf 0 setzt, der Spieler mit dem setzen auf 3 Gewinnt. Also eine Niederlage für die KI. Zeile 2 und 3 zeigen das Selbe für den ersten Zug auf 2 oder 3. Zeile 4 hingegen zeigt, dass in den weiteren Stufen bis zur maximalen Suchtiefe keiner durch ein perfektes Spiel gewinnt




Hier ist das Baumdiagramm zu 0 als ersten Zug der KI, in dem alle Simulationen angeführt sind, die gemacht werden. Endet ein Pfad im Nichts ist die Maximale Tiefe erreicht, also wie Unentschieden. Steht ein D am Ende eines Pfades ist es eine Niederlage. Blau eingefärbte Knoten sind Maximierende und Orange eingefärbte Minimierende. Fangen wir von links an. Die Maximale Tiefe ist erreicht, also ist der Wert für den Pfad 0-1-1 (Von oben nach unten gelesen) erst mal das Maximum 0. Der nächste Wert von rechts ist wieder 0, also keine Veränderung. Der darauf folgende ist eine Niederlage aber, da es ein maximierender Knoten ist, zählt hier das Maximum, 0. Die beiden Knoten daneben liefern kein anderes Ergebnis und dadurch wird der Wert des Minimierenden Knotens 0-1 auch 0. Für den Knoten 0-2 gilt das Selbe. Der Knoten 0-3 mündet direkt in eine Niederlage und damit einen Negativen Wert (hier -2, da Wert=-Suchtiefe+aktuelle Tiefe). Das Minimum von 0-1, 0-2 und 0-3 ist also -2.
Bei dem Start mit dem Knoten 3 ergibt sich nachher ein Wert von 0, der dann den anderen (0,1,2) bevorzugt wird. Am Ende wird im Log noch niedergeschrieben, dass beim Wert 0 das Feld 3 eine Möglichkeit ist, welches dann auch ausgewählt wird.

### 3.3.2 2.Beispiel

Beim zweiten Beispiel wollen wir nun eine Situation herbei führen, bei dem wir in der Zwickmühle sitzen.
Nun setzen wir als Spieler auf 2:0 um nicht zu verlieren.
Und die KI setzt auf 3:1 und wir haben keine Chance mehr zu Gewinnen. Setzen wir auf 4:1 setzt die KI im nächsten Zug auf 4:0 und gewinnt. Setzten

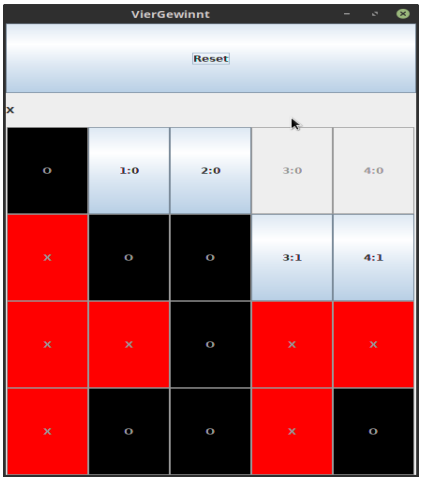
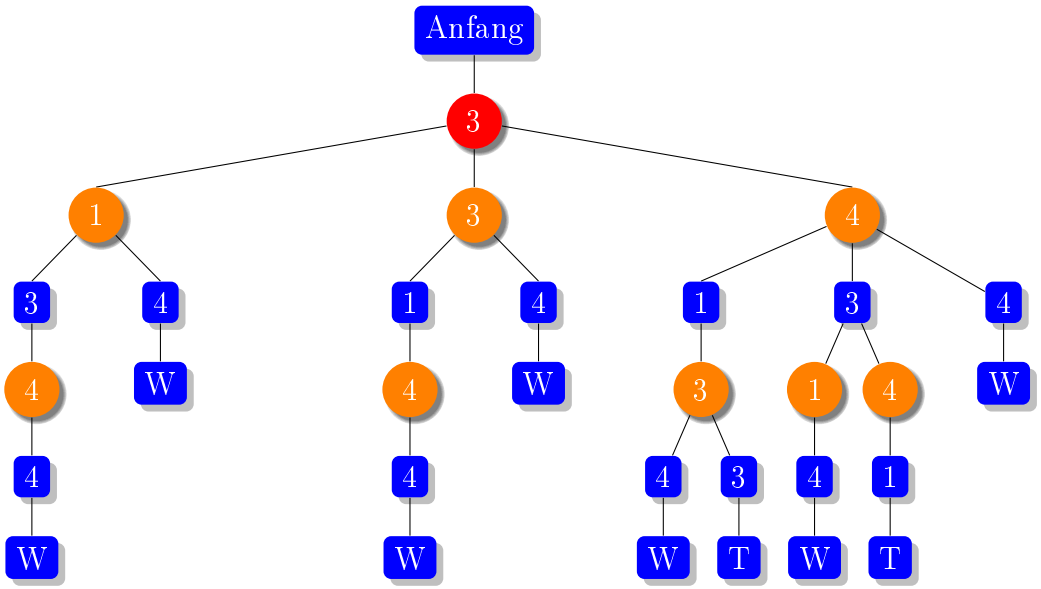



Abbildung 4:

wir hingegen nicht auf 4:1 setzt die KI auf dieses Feld und gewinnt ebenfalls.
Schauen wir uns nun wieder den Suchbaum an. Er geht hervor aus fach2-3




Oben haben wir nun die Wahl des Feldes 3 die nächste Ebene ist minimierend. Auf der nächsten Ebene ist aber wieder alles Maximierend und es tritt bei jeder Möglichkeit, durch einen schlauen Zug, ein Sieg für die KI ein. Der errechnete Wert ist dabei dann 3 (Suchtiefe - aktuelle Suchtiefe) und dieser Wert wird von keiner der anderen möglichen Felder übertroffen.
Hier das weitere Spiel bis zum Sieg der KI:

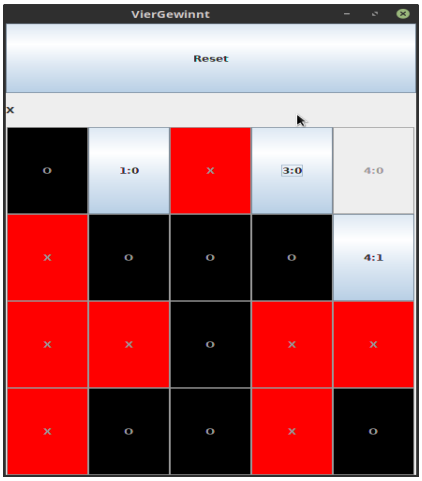



Abbildung 5:

## 3.4 Effizienzbetrachtung

Um mehr Züge zu simulieren, ist erheblich mehr Rechenleistung erforderlich. Die Komplexität steigt nahezu exponentiell, aber je weiter das Spiel fortgeschritten ist und je weniger Möglichkeiten es gibt, desto schneller wird der Algorithmus. Ich habe Simulationen mit verschiedenen Feldgrößen durchgeführt und habe den ersten Zug von der KI berechnen lassen. Dabei habe ich die Zeit gemessen und darauf basierend passende Suchtiefen gewählt, um die Zeit bei weniger als 10 Sekunden zu belassen. Hätte ich die vordefinierten Pfade genommen, wäre deutlich an Rechenzeit gespart worden, jedoch bräuchte man zum Erstellen dieser sehr lange bzw. würde es den Rahmen dieser Facharbeit sprengen. Auch wäre es möglich anstatt der VierLogik Klasse auch einfach nur kopierte Arrays zu verwenden, aber dabei ist es schwer darauf zu achten, dass keines dieser Arrays verändert wird.
Aber nun zur Effizienz bei verschiedenen Feldgrößen. Da die Höhe nicht so ausschlaggebend ist, wird hier nur betrachtet wie sich die Ausführungszeit verändert, wenn die Suchtiefe und die Feldbreite variiert werden. Die KI versucht in jedem Fall, den ersten Zug eines Spieles zu berechnen.
Im Folgenden sind die Graphen der Ausführungszeit im Bezug auf die Suchtiefe dargestellt. Von oben nach unten mit den Feldbreiten 9,7,5. Es fällt deutlich auf, dass bis zu einer Suchtiefe von 5 kaum Unterschiede in der Rechenzeit auftreten. Von da an werden die Unterschiede pro weiterer Suchtiefe immer größer und steigen exponentiell. Der Wert für die Feldbreite 9 und Suchtiefe 9 fehlt, da ich die Messung nach 2 Stunden abgebrochen habe. Die Tests wurden durchgeführt mit der Klasse Testing.

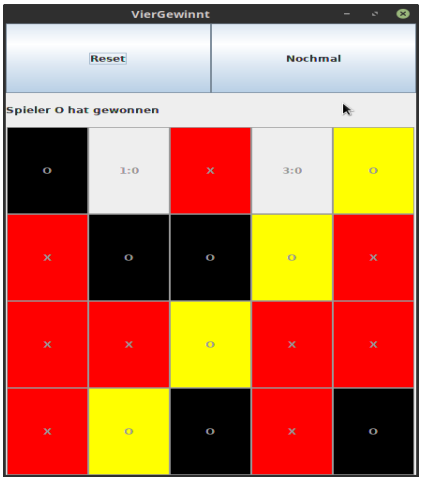
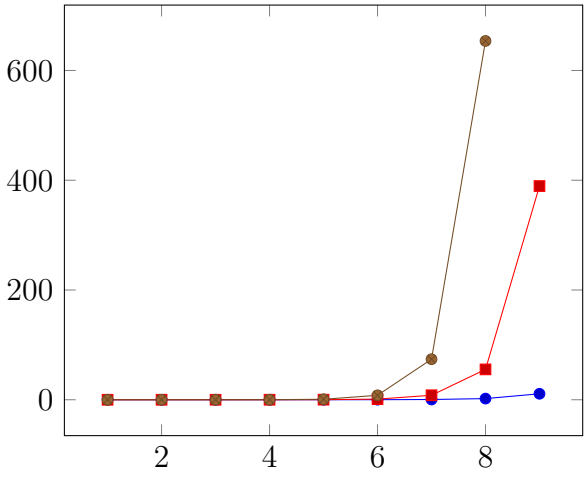



Abbildung 6:

Ausführungszeit




# 4 Künstliche Intelligenz

*Definition*

## 4.1 Aktueller Stand

*tensorflow,machine learning*

## 4.2 Ausblick

*Filme usw. (Wargames, Terminator, 2001, I Robot)*