1. Das Nimmspiel

1.1 Einführung

Deep Blue Kasparow, Philadelphia 1996

Deep Blue, der Supercomputer schlägt Garry Kasparow. So oder ähnlich lauteten die Schlagzeilen 1996. Die 6 Partien waren insgesamt ausgeglichen, zum Schluss gewann der Rechner. Kasparow war überrascht und gestand ein, den Rechner nicht ausrechnen zu können. Künstliche Intelligenz ist also berechenbar. Im Prinzip schon, dass vorliegende Script versucht dabei an einem einfachen überschaubaren Beispiel den Einstieg in das Thema zu verdeutlichen.
Aber was ist die entscheidende Grundlage eines Spiels, um es zu berechnen? Wesentlich ist ein gesunder Pessimismus. Man geht davon aus, das der Gegenspieler grundsätzlich den richtigen Zug macht, also einen Zug, der den Gegner in eine Position bringt, von der aus er im weiteren Spiel verlieren muss. Eine solche Position zu erkennen, ist genau die Schwierigkeit einer Strategie. Eigentlich weiß man erst dann ob ein Zug sinnvoll war, wenn man das jeweilige Spiel zu Ende gespielt hat. Für ein einfaches Nimmspiel ist das vorstellbar, für Schach kaum, da dort die Möglichkeiten zu ziehen, exponentiell wachsen.

Neben SCHACH, GO , VIER GEWINNT und TIC TAC TOE gibt es weitere 2 - Personenspiele, die beschrieben werden können. Ein relativ bekanntes und auch einfaches Beispiel, welches in unterschiedlichen Variationen auftritt, ist das Streichholzspiel.

Auf einem Tisch liegt eine endliche Menge von Streichhölzern, sagen wir zehn. Das Spiel besteht nun darin eine bestimmte Anzahl von Streichhölzern abwechselnd zu ziehen, zum Beispiel eins oder zwei. Verloren hat der Spieler, welcher den letzten Streichholz zieht. Die beiden Spieler seien Bart und Borg, welche abwechselnd eine bestimmte Anzahl von Hölzern ziehen. Ein möglicher Spielverlauf wäre der folgende, wenn man davon ausgeht, dass zehn Streichhölzer auf dem Tisch liegen und Bart beginnt.

Zug	Bart	Borg	Bart	Borg	Bart	Borg	Bart
Bart	2		1		1		1
Borg		1		1		2	verloren
Stand	8	7	6	5	4	2	1

Borg muss am Ende einen ziehen und hat verloren. Bart hatte eigentlich von Beginn an keine Chance zu gewinnen, nutzt aber einen Fehler von Borg in Runde 4 aus. Dort zieht Borg einen Stein und bringt damit Bart in Siegerlaune. Der Rest ist für Bart ein Kinderspiel. Hätte Borg dort 2 gezogen hätte Bart verloren.

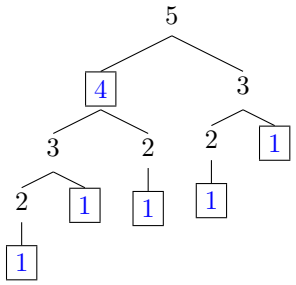
Zug	Bart	Borg	Bart	Borg	Bart	Borg
Bart	2		1		1	verloren
Borg		1		2		2
Stand	8	7	6	4	3	1

Die Struktur des Spiels ist überschaubar, wenn man sich das Ende anschaut. Liegt ein Streichholz auf dem Tisch hat der aktuelle Spieler verloren, er befindet sich in einer Verlustposition (v), liegen zwei oder drei Steine auf dem Tisch befindet sich der aktuelle Spieler in einer Siegposition, er kann gewinnen (g). Zieht er in diesem Fall ein oder zwei Hölzer zwingt er den Gegner in die Verlustposition. Liegen vier Hölzer auf dem Tisch befindet sich der aktuelle Spieler im Nachsehen, denn sein Ziehen führt nicht zum Sieg, wohl aber den Gegner in eine Siegposition. Dieses Muster kann man fortsetzen und verallgemeinern. Gelingt es dem Spieler den Gegner in eine Verlustposition zu bringen, kann er ab dort das Spiel immer wieder bestimmen. Im vorliegenden Fall bedeutet dies, dass

$$\boxed{\text{v} = \text{Anzahl der Hölzer } \% \text{ (Anzahl der zu ziehenden Hölzer} + 1) = 1}$$

Insgesamt waren zehn Streichhölzer also denkbar ungünstig, da der beginnende Spieler hiermit sofort in einer Verlustposition war.

Mit diesem Wissen lässt sich eine Spielstrategie ableiten. Es geht darum den Gegenspieler immer in eine Verlustposition zu zwingen. Nicht alle Spiele sind so einfach wie das Nimmspiel, die Vorgehensweise ist jedoch identisch und lässt sich mit einem Baum gut darstellen. Allerdings wird für die Darstellung eine Hölzeranzahl von fünf verwendet. Die Verlustpositionen sind demzufolge eins und vier.



Im Baum ist erkennbar, das ein sofortiges Ziehen von eins sinnvoll ist, um den Gegner in eine Verlustposition zu zwingen. Zwar wird das Bestreben das Gegners ein analoges Vorgehen sein, was jedoch nicht mehr gelingt, denn auf das Ziehen eines Streichholzes wird mit dem Ziehen von zwei Hölzern und auf das Ziehen von einem Holz mit dem Ziehen von Zweien reagiert. Diese Strategie lässt sich verallgemeinern, wenn man die Positionen am Ende bewertet. In unserem Fall sei die Bewertung für ein Streichholz am Ende -1, der Spieler hat verloren.

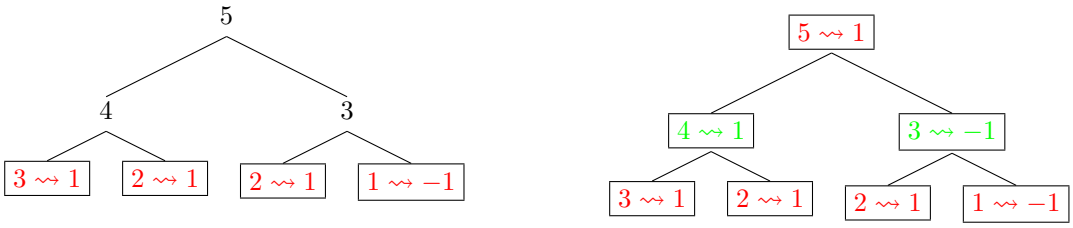
1.1 Min - Max Spieler und Negamax

Das oben stehende einfache Beispiel besitzt einen vollständigen Spielbaum, der in der Praxis oft nicht erstellt werden kann. Da die Menge der Verzweigungen mit zunehmender Tiefe ansteigt, ist es notwendig, den Baum in einer vorher festgelegten Tiefe abzuschneiden. Diese Tiefe des Schnittes bezeichnet man als Suchtiefe des Baumes. Für das nachfolgende Beispiel sei die Suchtiefe zwei, der Spieler, der am Zug ist, schaut zwei Züge voraus. Ist dies erfolgt, müssen die Züge bewertet werden, in wie fern der weitere Weg in eine Verlust - oder Gewinnsituation führt. Gerade die Bewertung der Züge stellt im Weiteren den wohl kompliziertesten Teil der Theorie dar.

Nach dem Min-Max Algorithmus versucht jeder Spieler ein für sich optimales Ergebnis zu erzielen. Ein Spieler bemüht sich die maximal bewertete Stellung zu erreichen, der Gegenspieler den Spieler in eine minimal bewertete Stellung zu zwingen. In unserem Beispiel wissen wir, das Verlustpositionen (v) berechnet werden können. Vom bewerteten Blatt beginnend versucht Spieler MIN den am geringsten bewerteten Zug zu wählen, Spieler MAX im Anschluss den maximal bewerteten Zug. Eine Gewinnsituation wird mit 1 gewertet, eine Verlustsituation mit -1

Spieler MAX wird also wie wir schon wissen, einen Streichholz ziehen und damit seine Siegchancen festigen.

Im Baum stellt sich der Wechsel wie folgt dar.

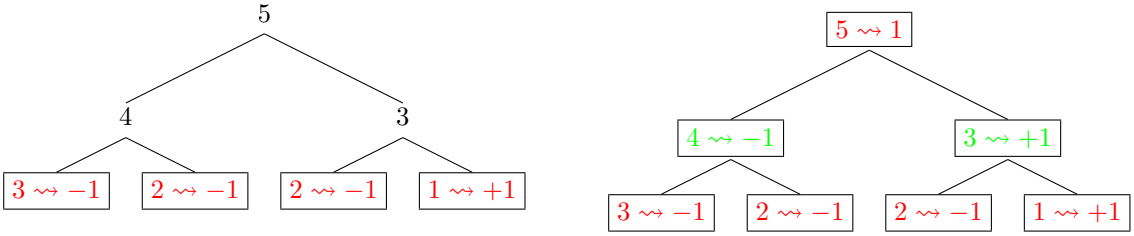


Der Nachteil dieses Vorgehens besteht in der Implementierung. Es ist hier das Vorgehen für beide Spieler zu programmieren. da ein rekursiver Durchlauf des Baumes notwendig ist, bläht dies die Implementierung unnötig auf.
Die Idee den Algorithmus zu vereinfachen, besteht in einer Negation der Minposition. Der Minspieler wählt die geringste Bewertung, um seinen Gegner in die Verlustposition zu zwingen. Negiert man die Bewertungen und wählt das Maximum, kann durchgängig mit dem Maximum gearbeitet werden, um den nächsten sinnvollen Zug des aktuellen Spielers herauszufinden. Die Logik dabei ist Folgende. Werden beide Knoten in der Suchtiefe mit 1 gewertet, bedeutet dies eine Ebene darüber befindet sich der Spieler in einer Verlustposition, er kann nicht gewinnen bzw. die Wahrscheinlichkeit dafür ist eingeschränkt, die Bewertung für den Knoten in der darüber liegenden Ebene ist -1, rechnerisch $max(-1 \cdot 1, -1 \cdot 1) = 1$.

Die möglichen Fälle sind folgende:

Knoten 1: Bewertung $+1$	Knoten 2: Bewertung $+1$	$\rightsquigarrow max(-1 \cdot +1, -1 \cdot +1) = -1$
Knoten 1: Bewertung -1	Knoten 2: Bewertung $+1$	$\rightsquigarrow max(-1 \cdot -1, -1 \cdot +1) = +1$
Knoten 1: Bewertung $+1$	Knoten 2: Bewertung -1	$\rightsquigarrow max(-1 \cdot +1, -1 \cdot -1) = +1$
Knoten 1: Bewertung -1	Knoten 2: Bewertung -1	$\rightsquigarrow max(-1 \cdot -1, -1 \cdot -1) = +1$

Der zugehörige Baum stellt sich entsprechend wie folgt dar.



1.2 Implementierung

Der Ausgangspunkt für eine Implementierung des Nimmspiels sei im Folgenden vorgestellt. Dazu werden als Ansatz Klassen erzeugt, die nur die Möglichkeit bieten zwei Spieler ohne Strategie gegeneinander spielen zu lassen. Die wesentlichen Methoden und Attribute sind im Folgenden dargestellt. Die Klasse Position ist dabei auf eine Idee von Daniel Garmann zurückzuführen.

Listing 1: Startprogramm

```
class position :
    def __init__(self,zielzahl=1,augen = 0, stand=5, spieler = 1):

        self.anzahl   = stand
        self.augen    = 0
        self.zielzahl = zielzahl
        self.am_zug   = spieler

    def folgeposition (self,zug):
        neue = position (self.zielzahl,zug,self.anzahl,self.am_zug)
        neue.zug_ausfuehren(zug)
        return neue

    def siegposition (self)      :        ...
    def zug_ausfuehren (self,zug):        ...
    def zug_moeglich (self,zug)  :        ...
```

Die Klassen Mensch und nimmspiel dienen der einfachen Umsetzung des Spiels und wurden in Python 3.0 umgesetzt. Mit diesem Gerüst ist es nun möglich, im Anschluss einen Spieler durch den Rechner zu ersetzen, um entsprechende Strategien zu testen.

Listing 2: Startprogramm

```
class Mensch :
    def __init__(self):
        self.name = "␣Bart␣"      .... bzw. variabel

    def ziehen (self,pos):
        a = int(input ("Bart␣ziehe␣>>␣"))
        pos.zug_ausfuehren(a)
        return pos

class nimmspiel :

    def __init__(self):
        self.spieler  = [Mensch(), Mensch()]

    def spielen(self):
        p = position()
        while not p.siegposition():
            p = self.spieler[p.am_zug].ziehen(p)
            print ("Spielerwechsel")

        self.sieger = self.spieler[1-p.am_zug].name
        print (self.sieger)

if __name__=="__main__":

    spiel   = nimmspiel()
    spiel.spielen()
```

Die Klasse Computer ist im Grunde genauso aufgebaut, wie die Klasse Mensch mit dem Unterschied, dass vor dem Ziehen der optimale Zug, wie oben beschrieben gesucht wird. Dazu ist dem Ziehen eine Methode negamax vorzuschalten, die den optimalen Zug und die entsprechende Bewertung liefert.

Listing 3: Methode negamax

```
def negamax (self,pos,c_zug = -1 , bewertung = 2):

        if (pos.siegposition()) : return -1
        for zug in range (1,3):
                folgepos = pos.folgeposition(zug)
                folgebewertung = self.negamax(folgepos)
                if folgebewertung < bewertung :
                        bewertung = folgebewertung
                        c_zug = zug

        self.zug = c_zug
        return (-1) * bewertung
```

Auf dieser Grundlage ist es bereits möglich, strategisch sinnvoll zu spielen. Ein Abbruch bei einer entsprechenden Suchtiefe kann relativ einfach eingebaut werden. Eine Bewertungsfunktion ist dann notwendig, da der Baum in diesem Fall nicht bis zum Ende ausgewertet wird. Die Rückgabe -1 kann dann durch eben eine solche Bewertungsfunktion ersetzt werden. Hier kann für dieses Beispiel die oben beschriebene Eigenschaft einer Verlustposition (v) entsprechend ausgenutzt werden. Eine sehr gute Möglichkeit einzelne Bäume mit etwas differenteren Bewertungen nach dem Min-Max oder - Negamaxverfahren auszuwerten, stellt dieses toll visualisierte Applet zur Verfügung.

1.3 Alpha - Beta pruning

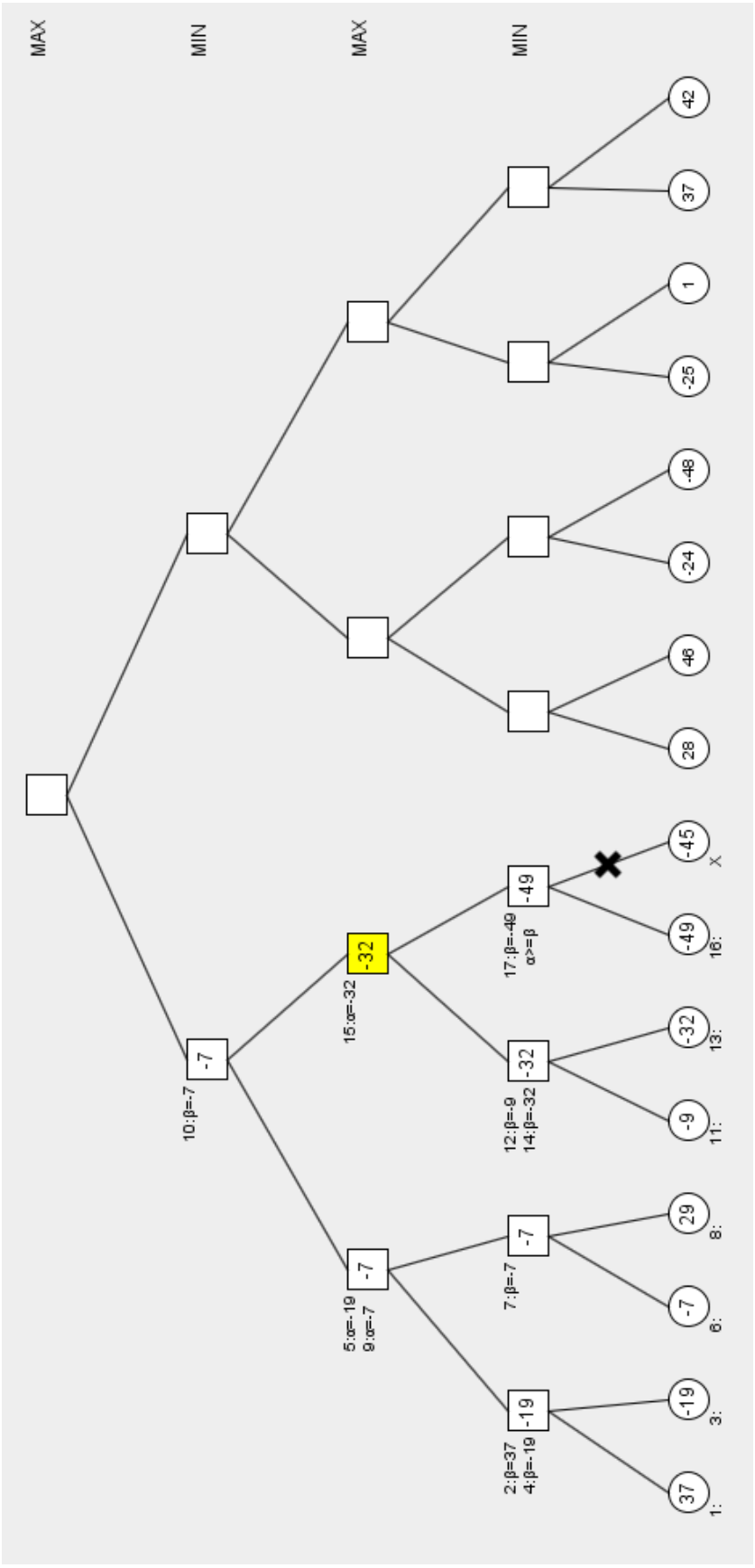
Mittels des negamax Verfahrens ist es nun möglich, einen Baum bis zu einer bestimmten Tiefe so zu analysieren, dass eine Entscheidung, welcher nächste Zug möglich ist, sinnvoll wird. Dieses Verfahren lässt sich allerdings erheblich optimieren. Der entscheidende Nachteil besteht in der Tatsache, dass der gesamte Baum durchsucht wird, was nicht notwendig ist.

Ein anschauliches Beispiel liefert ein Vergleich. Ein Spieler MIN wählt aus jeder Tasche den Gegenstand mit dem geringsten Wert. Der Spieler MAX wählt dann aus dieser Menge den wertvollsten. Dieses Vorgehen würde nach dem Min-Max Verfahren erfordern, alle Taschen zu durchsuchen. Dies ist aber nicht notwendig. Ist der Gegenstand mit minimalem Wert aus der ersten Tasche bekannt, stellt dieser Wert für den Max Spieler eine untere Grenze dar, denn einen wertvolleren Gegenstand nimmt er gern, eine wertloserer interessiert ihn nicht. Trifft er nun in Tasche zwei auf einen solchen wertloseren Gegenstand kann die Suche abgeschnitten (pruning) werden, ein cut erfolgen, da wertvollere der Minspieler kaum rausrücken würde und wertlosere nicht interessieren. Dieses Verfahren auf einen Baum zu übertragen, bedeutet eine erhebliche Verkürzung der Suche.

Im Prinzip kann dieses Vorgehen auch mit Regeln zusammengefasst werden :

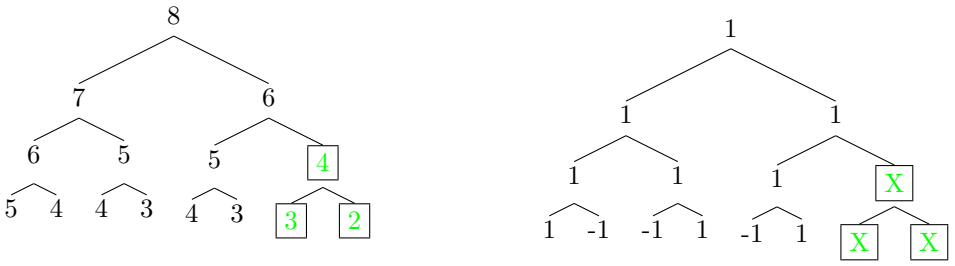
Regel 1 ($\alpha - cut$) *Die Grenze für die Min - Knoten wird α genannt.*
Wird der aktuelle Wert eines Min Knotens kleiner oder gleich α also gilt: $\alpha \geq \beta$ kommt es zum cut.

Regel 2 ($\beta - cut$) *Die Grenze für die Max - Knoten wird β genannt.*
Wird der aktuelle Wert eines Max Knotens größer oder gleich β also gilt: $\alpha \geq \beta$ kommt es zum cut.



An dem obenstehenden etwas komplexeren Beispiel ist das Vorgehen nachvollziehbar und kann am entsprechenden bereits oben erwähnten visualiserten Applet nachvollzogen und zu Ende geführt werden.

Die Übertragung des Alpha - Beta Algorithmus auf unser Nimmbeispiel gelingt relativ einfach, da sowohl Bewertungsfunktion, als auch Umfang wieder überschaubar sind. Im Beispiel kann der grün markierte Teil abgeschnitten werden, da der Wert an der Wurzel dem Wert des Knotens im rechten Unterbaum entspricht, also $\alpha \geq \beta$ gilt.



Die Übertragung auf den Negamax Algorithmus ist nun nachvollziehbar. Statt abwechselnd α und β zu betrachten, werden bei jedem rekursiven Aufruf α und β gemeinsam übergeben. α speichert den besten maximalen Wert. Am Start der Rekursion werden α mit einem möglichst kleinen, β mit einem möglichst großen Wert initialisiert. Da der Negamax - Algorithmus beim Durchlaufen des Baumes stets die Negation der Werte bildet, wird dies auch bei α und β getan. Es kommt zu einer Änderung der Vorzeichen von α und β und einem Vertauschen der Werte. Gilt analog dem vorher beschriebenen Verfahren $\alpha \geq \beta$, wird die Bearbeitung des Baumes abgebrochen und es kommt zum cut.

Bezieht man diesen Algorithmus in das Nimmspiel ein, stellt sich eine mögliche Implementierung wie folgt dar.

Listing 4: Methode alpha_beta

```
def alpha_beta (self,pos, alpha = -100, beta = 100):
        if (pos.siegposition()) : return -1
        for zug in range (1,3):
                folgepos = pos.folgeposition(zug)
                folgebewertung = -self.alpha_beta(folgepos,-beta,-alpha)
                if folgebewertung > alpha :
                      alpha = folgebewertung
                      if folgebewertung >= beta : return beta

                      self.zug = zug
        return alpha
```

.. weiterführende Anregungen zum Thema (Quellen)

(a) http://projekte.gymnasium-odenthal.de/informatik/

(b) http://www.ke.tu-darmstadt.de/bibtex/publications/show/1310

(c) http://wolfey.110mb.com/GameVisual/launch.php

(d) http://upload.wikimedia.org/wikipedia/en/7/79/Minmaxab.gif

(e) http://www-i1.informatik.rwth-aachen.de/ algorithmus/algo14.php

(f) ....