

Tic Tac Toe mit Computergegenspieler

für Maxima

von Jonathan Wendt

Einleitung:

Das Programm befasst sich mit dem klassischen Tic-Tac-Toe-Spiel. Jedoch wurde es um eine KI erweitert, die es dem Spieler durch einen rekursiv ablaufenden Minimax-Algorithmus im Grunde unmöglich macht zu gewinnen.

Vorbereitung:

Zur Nutzung des Programms muss das CAS Maxima (Quelle: <http://maxima.sourceforge.net/>) auf dem Rechner installiert sein. Man sollte nun xMaxima öffnen und über File -> Batch File Silently die Datei tictactoe.mac ins das Programm laden. Nach einer kurzen Ladezeit erscheint der Schriftzug „Start mit der Eingabe von "tictactoe;"“. Wenn sie nun „tictactoe;“ eingeben, startet das Programm.

Regeln:

Das Ziel des Spiels besteht darin, als erster 3 eigene Symbole in einer Reihe zu haben. Vertikal, horizontal aber auch diagonal. Dazu setzen beide Spieler immer abwechselnd eines ihrer Symbole auf ein beliebiges freies Feld.

Erklärung des MiniMax-Algorithmuses:

Der Algorithmus besteht aus zwei Funktionen kiZugMax (Z. 66-74) und kiZugMin (Z. 77- 85). Beide Funktionen sind fast identisch,

unterscheiden sich aber in einem wichtigen Detail. Die eine Funktion versucht immer die maximale Bewertung zu erreichen, die andere immer die minimale. Wie man auch am Namen erkennen kann.

Diese Funktionen arbeiten nach dem Prinzip eines Baumes.

Der Computer (Funktion `kiZugMin`) geht alle, von seiner Position aus, möglichen Züge durch. Dabei möchte er natürlich die kleinstmögliche Bewertung erreichen (-1 = der Computer gewinnt oder 0 = Unentschieden) Dafür übergibt er nun das neuentstandene Feld, also das Feld in dem sein Zug eingetragen ist, weiter an `kiZugMax`, die Funktion, die den Spieler repräsentiert. Diese Funktion berechnet dann, so wie der Spieler wahrscheinlich handeln würde, den Zug mit der besten Bewertung für den Spieler (also, 1 = Spieler gewinnt oder 0 = Unentschieden). Nach diesem System rufen sich die beiden Funktionen immer weiter gegenseitig auf, bis das Feld entweder voll ist, also wahrscheinlich ein Unentschieden erreicht ist, oder einer der Spieler gewonnen hat. Nun bekommt die Funktion `ki`, den für den Computer bestmöglichen Zug zurück und setzt nun das Symbol des Computers auf dieses Feld

Der Computer berechnet also alle möglichen Züge und damit verbunden alle Gewinn- und Verlustmöglichkeiten und sucht sich den für ihn besten Zweig heraus.

Dazu durchläuft er eine Schleife aller möglichen Züge.

```
for zug in moeglicheZuege(M) do(
```

Man kopiert zuerst die Matrix um Seiteneffekte auszuschließen und Fehler zu vermeiden.

```
tmpM: copymatrix(M),
```

Und setzt nun das Feld des entsprechenden aktuellen Zuges auf 1 bei `kiZugMax` (dem Funktionsteil, der davon ausgeht, dass der Spieler am Zug ist) oder auf -1 bei `kiZugMin` (dem Funktionsteil, der davon ausgeht, dass der Computer am Zug ist).

tmpM[zug[1], zug[2]]: 1,

Um Rechenzeit zu sparen, wird die Funktion `gewinnAbfrage` ausgeführt und gespeichert. Sie ermittelt, ob bei dem so erzeugten Spielstand ein Gewinner existiert.

sieger: gewinnAbfrage(tmpM),

Wenn als Sieger nun 1 bei `kiZugMax` zurückgegeben wird, das heißt der Spieler gewonnen hätte, wäre dies der Zug, den der Spieler wählen würde. Also werden die Variablen `besterZ` (Zug) und `besterW` (Wert) gesetzt und die Schleife mit `return` verlassen, da für den Spieler kein besseres Ergebnis möglich ist. Bei `kiZugMin` funktioniert das ebenso, nur dass der Computer seinen eigenen Sieg also `sieger = -1` bevorzugen würde. Sollte das Feld voll sein, muss die Rekursion natürlich abgebrochen werden, nur diesmal wird 0 zurückgegeben.

if sieger=1 or feldVoll(tmpM) then return([besterW, besterZ]: [sieger, zug]),
Nun folgt der rekursive Aufruf. Da, wenn man sich in der Funktion `kiZugMax` befindet, der Spieler am Zug war, als nächstes der Computer am Zug sein muss ruft man die jeweils andere Funktion auf. In diesem Fall also `kiZugMin` und speichert die Bewertung des Zuges, also wer gewinnen würde, zur weiteren Verarbeitung. Dieser Wert befindet sich in der zurückgegebenen Liste an zweiter Position.

bewertung: kiZugMin(tmpM)[1],

Wenn diese Bewertung nun besser als der bis jetzt gespeicherte Wert des besten Zuges ist, wird er gespeichert. Ansonsten nicht weiter verwendet. Besser bedeutet für `kiZugMax` also größer als der gespeicherte Wert und für `kiZugMin` kleiner als der Gespeicherte. Daher kommen auch die Bezeichnungen.

if bewertung>besterW then (besterZ: zug, besterW: bewertung),

Hier wird nun die Schleife wieder geschlossen, damit noch die anderen möglichen Züge überprüft werden können.

Wenn die gesamte Schleife durchlaufen ist, also alle möglichen Züge überprüft wurden, wird der beste Zug und damit verbunden auch die Bewertung für die Rekursion in einer Schleife zurückgegeben. Das funktioniert in Maxima einfach dadurch, dass man das Argument als letztes in den Block schreibt.

[besterW, besterZ])\$

Anmerkung: Für diese Erklärungen wurden nur Programmcodeausschnitte aus der Funktion kiZugMax verwendet

Quellen:

<http://dkicomp.dki.tu-darmstadt.de/~robert/maxima-einfuehrung.html>

<http://maxima.sourceforge.net/docs/manual/en/maxima.html>

<http://de.wikipedia.org/wiki/Minimax-Algorithmus>

<http://maxima.sourceforge.net/docs/tutorial/de/Casting-SPELs-Teil-I.pdf>

<http://maxima.sourceforge.net/docs/tutorial/de/Casting-SPELs-Teil-II.pdf>

Die Funktion Zeilen 106-116 habe ich von meinem Informatiklehrer und sie dienen lediglich zur Beschleunigung des Programms und der Umlenkung der daraus resultierenden Compilerausgaben. An der Funktionsweise des Programms ändern sie allerdings nichts.