„Vier Gewinnt“

THOMAS WÖLLERT (DIPL.-INF. (FH))

Matrikel-Nr. 05478901-0199
Semestergruppe IG2

Entscheidungstheorie
Prof. Dr. Peters

Praktikum SS2006

Master of Science
Computergrafik und Bildverarbeitung

Fachbereich Informatik und Mathematik
Fachhochschule München

1 Aufgabenbeschreibung

Es soll eine Version des Spiels „Vier gewinnt“ programmiert werden. Die Entscheidungen des Computergegners müssen aus einer vollständigen Handlungsanweisung bestehen (z.B. den auszuführenden Spielzug komplett durchdenken und nicht nur zufällig handeln). Zusätzlich soll die Möglichkeit bestehen, mit den „Vier gewinnt“-Applikationen der anderen Studenten über eine Textdatei zu kommunizieren, damit diese gegeneinander antreten können.

2 Entwicklungsumgebung

Da zur Programmiersprache und Entwicklungsumgebung keine Vorgaben gemacht wurden, fiel die Entscheidung auf *Java* in der neuen Version 6 [1]. Als Entwicklungsumgebung kam *Eclipse* [2] in der Version 3.2 zum Einsatz.

3 Implementierung des Rahmenprogramms

Der Schwerpunkt der gestellten Aufgabe lag auf der Entwicklung der Künstlichen Intelligenz des Computergegners. Daher wird auf das restliche Programm in diesem Abschnitt nur kurz eingegangen.

Beim Start der Applikation werden mittels zwei Kommandozeilenargumenten die verschiedenen Spielmodi für beide Spieler übergeben. Zur Zeit gibt es drei verschiedene Einstellungen, die in Tabelle 1 dargestellt sind. Zur Initialisierung werden diese Spielmodi ausgewertet und die passenden Spielklassen instanziiert. Wird der Spielmodus „`e`“ gewählt muss zusätzlich pro externem Mitspieler noch der Pfad zur Textdatei übergeben werden, die benutzt werden soll, um Spielinformationen mit dem Gegner auszutauschen.

Argument	Beschreibung
`h`	Menschlicher Mitspieler
`c`	CPU gesteuerter Mitspieler
`e`	Externer Mitspieler (Kommunikation erfolgt über Textdatei)

Tabelle 1: Kommandozeilenargumente der Spielmodi

Die Implementierung der verschiedenen Spielmodi erfolgte mittels Ableitungen von der abstrakten Klasse `AbstractPlayer`. Während des Spiels wird bei jedem Spieler die Methode `doTurn(Playfield, PlayerState)` aufgerufen und damit der aktuelle Spielstand sowie die Spieleridentität („grün“ oder „rot“) übergeben.

Der Ablauf des Spiels selbst wird von der Klasse `Game` übernommen, welche dafür sorgt, dass abwechselnd bei beiden Spielern die erwähnte Methode `doTurn(...)` aufgerufen wird.

Eine grafische Darstellung wurde für das Spielfeld selbst gewählt. Dabei kamen Grafiken der verschiedenen Spielsteine sowie *Java2D* zum Einsatz (siehe Abbildung 1). Spieler 1 erhielt hierbei die grünen Spielsteine, während Spieler 2 die Farbe Rot einsetzte.

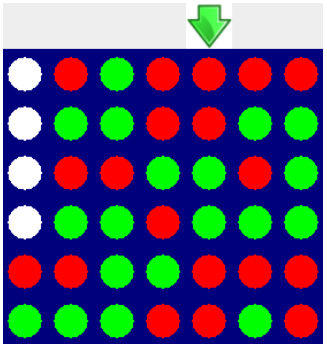

Abbildung 1: Das grafische Spielfeld

Auf die Darstellung der Textdatei wird nicht näher eingegangen, da sie bereits bei der Aufgabenstellung für alle Teilnehmer festgelegt worden ist und ebenfalls so umgesetzt wurde. Als einzige Besonderheit sei hier erwähnt, dass kein weiterer Benutzereingriff notwendig ist, um dem Spiel mitzuteilen, wann der externe Spieler seinen Zug ausgeführt hat. Hierbei wird in konstanten Zeitintervallen der Zeitstempel der betreffenden Textdatei überprüft. Sobald dieser sich geändert hat, wird angenommen, dass der Gegenspieler seinen Zug beendet hat.

4 Entwurf des Computergegners

Vorgaben bei der Stellung der Aufgabe gaben bereits die Richtung vor, in die sich die Entwicklung des K.I.[1]-Gegners bewegen sollte. Wichtigster Punkt war die Notwendigkeit einer *vollständigen Handlungsanweisung*, bevor ein Zug getätigt wird. Diese Bedingung lässt sich nur zufriedenstellend erfüllen, wenn vor dem Durchführen eines Spielzuges ein Baum zur Bewertung aller möglichen Spielzüge aufgebaut wird. Dabei müssen allerdings nicht nur alle eigenen Züge, sondern auch alle möglichen Reaktionen des Gegners berücksichtigt werden.

[1] K.I., Künstliche Intelligenz

Im speziellen Fall des Spiels „Vier gewinnt“ ist dabei eine *starke Gewinnstrategie*[2] möglich. Dies haben Victor Allis [5] und James D. Allen [6] unabhängig voneinander bereits in den späten 90’er Jahren bewiesen.

Als Grundlage der Implementierung wurde der *Minimax-Algorithmus* [7] gewählt. Dieser ist auf Nullsummenspiele[3], bei denen zwei Gegner abwechselnd Züge durchführen zugeschnitten. Der Algorithmus sichert bei dieser Art von Spielen den höchstmöglichen Gewinn bei optimaler Spielweise des Gegners zu [7].

Dabei muss beachtet werden, dass die angesprochenen Spiele nicht vom Zufall abhängig sind, wie z.B. ein Würfelspiel. Das bedeutet, dass in jeder Spielsituationen jedem der beiden Spieler alle Zugmöglichkeiten des Gegners bekannt sind. Hierzu zählt als komplexeres Beispiel auch Schach.

Durch die Bedingung des höchstmöglichen Gewinns ist die optimale Strategie gefunden, wenn sie zum bestmöglichen Ergebnis eines Spielers führt. Dabei wird jeweils von der optimalen Spielweise des Gegeners ausgegangen.

5 Der Minimax-Algorithmus

Die Grundlage des *Minimax-Algorithmus* besteht darin, dass jeder aktuellen Spielsituation eine Heuristik[4] in Form eines Wertes zugewiesen werden kann. Dieser Wert trifft eine Aussage darüber, wie gut oder wie schlecht die aktuelle Spielsituation für den betreffenden Spieler ist. Da der Gewinn des einen Spielers der Verlust des anderen ist, versucht daher Spieler 1 den Wert für sich zu maximieren, während Spieler 2 darauf aus ist, die Heuristik für sich zu minimieren. Daraus resultiert auch der Name *Minimax* des Algorithmus.

[2] Starke Gewinnstrategie, ein allgemeiner Algorithmus existiert, der unabhängig vom aktuellen Spielstand perfektes Spiel vorgibt, auch, wenn in den bisherigen Zügen bereits Fehler gemacht wurden. Für ein Spiel mit einer endlichen Anzahl von Zügen bzw. Positionen ist dies grundsätzlich möglich, z.B. mit Hilfe eines genügend schnellen Computers. Das Problem könnte jedoch darin liegen, einen effizienten Algorithmus zu erstellen [3].

[3] Nullsummenspiele, bei Spielen mit vollständiger Information spricht man von Nullsummenspiel, wenn jeder Teilnehmer mit einer jeweiligen Gewinnstrategie ein Unentschieden erzwingen kann. Beispielsweise ist Tic Tac Toe ein Nullsummenspiel, während bei Vier gewinnt der Spieler mit dem ersten Zug einen Sieg erzwingen kann.

[4] Heuristik, als Heuristik bezeichnet man Strategien, die das Finden von Lösungen zu Problemen ermöglichen sollen, zu denen kein mit Sicherheit zum Erfolg führender Algorithmus bekannt ist. Man bezeichnet sie im Kontext von Problemlöseverfahren daher auch als Faustregeln. Heuristische Prinzipien bezeichnen entsprechend Hilfsmittel bzw. vorläufige Annahmen der Forschung, von denen man sich neue Erkenntnisse erhofft.

Ablauf des Algorithmus [7]:

- Jeder Spielsituation ist eine Heuristik zugeordnet:
  - Positionen, die günstig für Spieler 1 sind, erhalten positive Werte.
  - Positionen, die günstig für Spieler 2 sind, erhalten negative Werte.
  - Spieler 1 maximiert im Laufe des Spiels den Zahlenwert, Spieler 2 minimiert ihn.
- Der Algorithmus baut einen Suchbaum auf, der alle Folgesituationen der aktuellen Situation enthält, alle Folgesituationen dieser Folgesituationen usw. bis zu einer vorgegebenen maximalen Tiefe.
- Für die Blätter des Suchbaums wird eine Heuristik bestimmt.
- Nun wird die Bewertung von den Blättern zur Wurzel übertragen:
  - Ist Spieler 1 am Zug, so wird einer Situation der Maximalwert der Bewertungen aller Folgesituationen zugeordnet.
  - Ist Spieler 2 am Zug, so wird einer Situation der Minimalwert der Bewertungen aller Folgesituationen zugeordnet.
  - Dieser Vorgang geschieht rekursiv, bis die Wurzel des Spielbaums erreicht ist.
- Abhängig vom Spieler, entscheidet sich der Algorithmus schließlich für den Zug zur Folgesituation mit maximaler bzw. minimaler Bewertung.

Würde der Suchbaum dabei bis zur Endstellung, die darstellt, wer gewinnt, aufgebaut könnte der Algorithmus ein perfektes Spiel durchführen. Dies ist aber bereits im vorliegenden Fall zu rechenzeitaufwendig, weshalb eine maximale Tiefe vorgegeben wird.

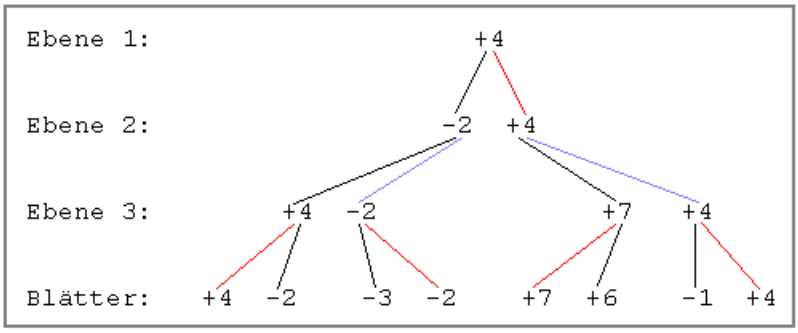

Abbildung 2: Beispielhafter Suchbaum (Quelle: [7])

Beispielhaft sei hier der in Abbildung 2 dargestellte Suchbaum beschrieben. Dabei wurde die maximale Suchtiefe 3 gewählt. Die Knoten in den Ebenen 1 und 3 entsprechen Spielsituationen,

in denen Spieler 1 am Zug ist, d.h. hier wird jeweils der maximale Wert der untergeordneten Knoten ermittelt. Die Knoten der Ebene 2 entsprechen dementsprechend Situationen in denen Spieler 2 am Zug ist, welcher jeweils den minimalen Wert der untergeordneten Knoten ermittelt. Auf Blätterebene werden die Knotenwerte jeweils über die Heuristik ermittelt. Rote Kanten im Bild führen zum Max-Wert, wohingegen blaue Kanten zum Min-Wert führen.

Der Bewertungsfunktion zur Bestimmung der Heuristik kommt im Algorithmus entscheidende Bedeutung zu. Nur wenn vorteilhafte bzw. negative Spielsituationen erkannt werden können, ist es möglich, eine optimale Entscheidung zu treffen. Tests mit sehr einfachen Heuristikbestimmungen führten bereits zu guten Ergebnissen. Dabei wird die gegebene Spielsituation nach entstandenen 2er, 3er und 4er Gruppen durchsucht. Diese können in Spalten, Zeilen oder ansteigenden bzw. abfallenden Diagonalen angeordnet sein. 2er wurden mit 10 Punkten, 3er mit 100 und 4er mit 10000 Punkten bewertet (diese Werte sind zu für Spieler 2 zu negieren, da dieser sein Ergebnis minimieren will). In den Blättern wurde die globale Heuristik einer Spielsituation benutzt, die sich aus der Summe der Heuristiken für Spieler 1 und Spieler 2 zusammensetzt. Ohne diese Summe würden beide Spieler nur versuchen, ihr Ergebnis zu maximieren bzw. minimieren, ohne mögliche Gewinnsituationen des Gegners zu erkennen, die verhindert werden müssen.

Geringe Laufzeitoptimierungen konnten bereits in den wechselseitigen Aufrufen von `min(...)` und `max(...)` durchgeführt werden. Dabei wurde die Heuristik nicht nur in den Blättern berechnet, sondern auch vorher überprüft, ob der gegnerische Spieler (bei `max(...)` wäre dies der Minimierer Spieler 2 und vice versa) das Spiel bereits gewonnen hat. Ist dies der Fall, wird dessen Siegheuristik zurückgegeben. Dadurch verfolgt der „Verlierer“ diesen Teil des Baumes nicht weiter.

Interessante Probleme ergaben sich nach bestimmten Durchläufen des Algorithmus. Für den K.I.-Spieler, der am Zug ist, gibt es maximal sieben verschiedene Möglichkeiten seinen Stein einzuwerfen (nach der Spielfelddefinition mit sechs Zeilen und sieben Spalten). Die Evaluierung mittels Suchbaum ergab nun, dass zwei oder mehr Züge zu einer gleichen Heuristik und damit einer gleich guten Spielsituation führen. Eine dieser besten Möglichkeiten musste nun intelligent ausgewählt werden. Dazu wurde die lokale Heuristik herangezogen. Die errechneten „besten“ Spielzüge wurden nacheinander ausgeführt, die entstandene Spielsituation mittels Heuristik bewertet und dann der betreffende Zug wieder zurückgezogen. Dies macht natürlich nur bei einer Suchbaumtiefe Sinn die grösser als 1 ist. Der „beste“ Zug, der auch in diesem Abschnitt die beste Heuristik hatte, wurde ausgewählt. Natürlich können auch nach dieser zweiten Auswahl mehrere Züge mit gleicher 1. und 2. Heuristik entstehen. Trat dies ein, wurde einer der „besten“ Züge mittels Zufallsverfahren ausgewählt. *Java* stellt hierfür die Methode `Math.random()` zur Verfügung, die gleichverteilte Werte zwischen 0 und 1 erzeugt.

6 Mögliche Erweiterungen

Verbesserungen des Algorithmus sind insbesondere in der Laufzeit möglich. Tests ergaben, dass die Berechnungszeit bereits bei einer Suchbaumtiefe von 4 auf Zeiten ansteigt, die beim Spielen Geduldsprobleme verursachen können.

Eine Verkleinerung des Suchbaumes resultiert in einer Verkürzung der benötigten Rechenzeit. Einen möglichen Ansatz stellt die *Alpha-Beta-Suche* [8] dar. In dieser Variante des *Minimax-Algorithmus* wird durch zwei Werte α und β entschieden, welche Teile des Suchbaumes nicht untersucht werden müssen, weil sie das Ergebnis der Problemlösung nicht beeinflussen können.

Die *iterative Tiefensuche* [9] wird bei eingeschränkten Zeiten für die Suche (z.B. im Turnierschach) eingesetzt. Ausgehend von bereits untersuchten Stellungen wird die Suche wiederholt gestartet und dabei die Suchtiefe schrittweise erhöht. Bereits untersuchte Situationen werden gespeichert, und nur für neu erreichte Stellungen muss eine Heuristik berechnet werden. Ohne diese Tiefensuche wäre im *worst-case* Fall am Ende des Zeitlimits nur ein einziger Zug untersucht worden, aber dieser bis in sehr große Tiefe. Weitere Züge, die vielleicht schneller zum Sieg geführt hätten, wären gar nicht untersucht worden.

Eine Verbesserung der Heuristikfunktion kann zu einem „besseren“ Algorithmus führen, welcher nicht schneller rechnet, sondern die Spielsituationen besser einschätzt. Verschiedenste Ansätze sind hier möglich. Bei der Suche nach 2er, 3er und 4er Gruppen kann zum Beispiel berücksichtigt werden, ob aus einem 3er ein 4er werden kann, wenn das 4. Feld noch leer ist. Ein Einflussfaktor wäre in einer Zeile des Spielfeldes, wieviele Steine in die betreffende Spalte des Spielfelds gefüllt werden müssen, damit der 4er gesetzt werden kann.

A Programmstart und Bedienung

Der Start des Spiels kann über die vorhandene Datei `start.bat` erfolgen. Diese liegt im Hauptverzeichnis der Applikation. Vor dem ersten Start muss sichergestellt werden, dass mindestens die *Java Runtime* in der Version 5 installiert ist. Die Version kann in der Kommandozeile mittels `java -version` abgefragt werden. Falls nötig kann diese unter [10] heruntergeladen werden.

Ein Spiel wird durch Ausführen der Datei `start.bat` begonnen. Mehrere Kommandozeilenparameter ermöglichen die Konfiguration (siehe Tabelle 2). Wurde dabei ein menschlicher Mitspieler gewählt, kann dieser seine Züge durch einfaches Anklicken der entsprechenden Spalte im Spielfenster durchführen.

Argument Nummer	Beschreibung
1	Anzahl der Spalten im Spielfeld
2	Anzahl der Zeilen im Spielfeld
3	Anzahl der zu kombinierenden Steine um zu gewinnen.
4	Spieler 1: ’`h`’ Mensch, ’`c`’ CPU, ’`e`’ Externer
5	Spieler 2: ’`h`’ Mensch, ’`c`’ CPU, ’`e`’ Externer
6	Falls Spieler 1 ’`e`’: Angabe der zu nutzenden Textdatei
7	Falls Spieler 2 ’`e`’: Angabe der zu nutzenden Textdatei

Tabelle 2: Kommandozeilenargumente zum Spielstart

Beispiele:

```
start.bat 7 6 4 h c
start.bat 7 6 4 c e spieler2.dat
start.bat 10 10 5 e e spieler1.dat spieler2.dat
```

Literatur

[1] Sun Microsystems, *Mustang: Java SE 6*, Mai 2006, https://mustang.dev.java.net/

[2] Eclipse.org, *Homepage*, Mai 2006, http://www.eclipse.org

[3] Wikipedia, *Gewinnstrategie*, Mai 2006, http://de.wikipedia.org/wiki/Gewinnstrategie

[4] L. Victor Allis, *Homepage (archived)*, Mai 2006, http://web.archive.org/web/20040930215211/www.cs.vu.nl/~victor/

[5] J.W.H.M. Uiterwijk, H.J. van den Herik, L.V. Allis: *A Knowledge-Based Approach to Connect-Four. The Game is Solved! Heuristic Programming in Artificial Intelligence: the first computer olympiad* (eds. D.N.L. Levy and D.F. Beal), pp. 113-133. Ellis Horwood Limited, Chichester, 1989.

[6] James D. Allen, *Expert Play in Connect-Four*, Mai 2006, http://homepages.cwi.nl/~tromp/c4.html

[7] Wikipedia, *Minimax-Algorithmus*, Mai 2006, http://de.wikipedia.org/wiki/Minmax-Algorithmus

[8] Wikipedia, *Alpha-Beta-Suche*, Mai 2006, http://de.wikipedia.org/wiki/Alpha-Beta-Suche

[9] Wikipedia, *Iterative Tiefensuche*, Mai 2006, http://de.wikipedia.org/wiki/Iterative_Tiefensuche

[10] Sun Microsystems, *Download Java 2 Plattform, Standard Edition 5.0*, http://java.sun.com/j2se/1.5.0/download.jsp