Implementatie Vier Op een Rij

Abstract

Vier Op 'n Rij is een spel voor twee spelers, waarbij de eerste speler die vier schijven op een rij plaatst het spel wint. Het is bij zo een simpel spel dan ook interessanter om tegen een slimmere computer te spelen. Op het eerste gezicht lijkt het creëren van een computer gestuurde speler voor Vier Op 'n Rij eenvoudig. Uit dit paper blijkt echter dat er veel obstakels zijn bij het programmeren van een efficiënte Vier Op 'n Rij speler. Er is geprobeerd inzicht te geven in de problemen waar dit project tegenaan liep en hoe deze problemen zijn opgelost.

Gerben van der Huizen
10460748
Michael Chen
10411941
David van der Kooij
10431543
Lucas Twisk
10448136
Kyllian Broers
10343865

July 5, 2013

Inleiding

In de laatste week van het project 'ZSB' moest er zelf een onderwerp verzonnen worden. Er werd eerst besloten om de robotarm te laten schrijven. Echter zijn er geen A.I. elementen gevonden die hierbij toegepast konden worden. Uiteindelijk is er gekozen voor het implementeren van het spel Vier Op 'n Rij met gebruik van de robotarm. Echter werd er naarmate de week vorderde besloten om alleen met een eigen bord representatie te werken in plaats van met de robotarm. Het bleek dat de robotarm nog niet in werkende staat was. Het spel is geïmplementeerd met Java en Prolog.
Het spel Vier Op 'n Rij wordt gespeeld met twee spelers en het is de bedoeling een rij te vormen van vier schijven op een verticaal speelbord. Een uitgebreid overzicht van de spelregels is te vinden bij het kopje 'materiaal en methode'. Het zou geen uitdaging zijn als het programma willekeurige zetten doet. Het is juist interessant om gebruik te maken van een strategisch en efficiënt algoritme. Dit brengt de volgende onderzoeksvraag naar voren:

Is het mogelijk een strategisch en efficiënt algoritme te programmeren voor het spel Vier Op 'n Rij?

Dit kan worden onderzocht door een implementatie te vinden waar bij er gebruik wordt gemaakt van de vaardigheden die er in de eerdere weken zijn vergaard. Aangezien er al andere spellen, zoals 'Tic-Tac-Toe' en schaken [5] met een efficiënt algoritme [7] geprogrammeerd zijn, zou het voor Vier Op 'n Rij ook moeten kunnen.

Materiaal en Methode

Materiaal

Zie Apendix voor de materialen die gebruikt zijn.

Spelregels Vier Op 'n Rij

Het originele spel wordt gespeeld op een verticaal bord van zeven hokjes breed en zes hokjes hoog. Doordat het bord verticaal staat, verplaatsen de schijven naar het laagst beschikbare vak. Het spel wordt gespeeld door twee spelers, waarbij de één met de rode schijven en de andere met de gele schijven speelt. Beide spelers kunnen maximaal 21 schijven plaatsen. Om de beurt moeten de spelers een schijf plaatsen op het bord tot er een winnaar bekend is. Een speler wint het spel wanneer er vier schijven van dezelfde kleur verticaal, horizontaal of diagonaal naast elkaar liggen. Het spel eindigt in gelijkspel wanneer geen van beide spelers een winnende toestand weet te bereiken. In dit onderzoek wordt Vier Op 'n Rij op een horizontaal bord gespeeld, verder zijn de spelregels hetzelfde gebleven.

Methode

Prolog

Prolog wordt gebruikt voor het genereren van een 'slimme' zet van de computer. Dit is geïmplementeerd in 'FIARmove.pl'. Hiervoor wordt het 'alpha-beta pruning' algoritme [3] gebruikt, wat een verbetering op het 'Minimax' algoritme [2] is. 'Minimax' (en alpha-beta dus ook) kijkt als het ware vooruit en gaat alle mogelijkheden af. Omdat dit heel veel mogelijkheden zijn, is gekozen om de diepte die het algoritme zoekt tot vier te limiteren. Hierdoor zal het algoritme niet verder dan vier zetten vooruit kijken.
Alle mogelijkheden op de diepte vier krijgen een waarde. Deze waarde wordt bepaald door het aantal 'Twee Op 'n Rij' en 'Drie Op 'n Rij' die elke speler heeft. Er wordt voor elke speler een sub waarde uitgerekent. Hierin telt de 'Drie Op 'n Rij' zes keer zo zwaar

mee als de 'Twee Op 'n Rij'. Voor de computer wordt een negatieve waarde berekent, voor de gebruiker een positieve. Deze waardes worden opgetelt en dat is de waarde van de toestand. Als de computer in het voordeel is in die toestand, is de waarde negatief. Als de gebruiker in het voordeel is, is de waarde positief. Bij een waarde van tien heeft de gebruiker gewonnen en bij de negatieve waarde tien heeft de computer gewonnen. Deze waardes worden door het 'alpha-beta' algoritme vergeleken en uiteindelijk vindt de computer zo de beste zet.
Het alpha-beta algoritme heeft twee start waardes nodig, alpha en beta. Deze zijn gedefiniëerd als negatief oneindig voor alpha en postief oneindig voor beta. In deze implementatie is gekozen voor -100 en 100, omdat de mogelijke waardes voor een toestand gelimiteerd zijn tussen minus tien en tien.

In het aparte prolog bestand 'checkwin.pl' wordt onderzocht of de huidige toestand een winnende toestand is voor één van de spelers. Eerst worden de horizontale rijen bekeken, daarna de verticale met behulp van de 'transposed' van de matrix en als laatste de diagonale rijen.

In 'readBoard.pl' staat het predicaat 'readFIAR' dat het bord kan inlezen uit 'board.gch'. Eerst worden alle characters één voor één in een lijst gezet tot het 'end of file' character, de 'e', gelezen wordt. Daarna wordt die lijst omgezet in een matrix van zes bij zeven. Deze matrix wordt dan doorgegeven aan 'FIARmove.pl'.

Java

Nadat er een move is gevonden door 'FIARmove.pl' gaat het java bestand 'FourInRowPP.java' het 'move.txt' bestand uitlezen. De methode FiRPath kan een damstuk op veilige hoogte, zodat obstakels worden vermeden, naar zijn eindpositie verplaatsen. Om dit te realiseren zijn er tien stappen beschreven voor de robotarm over hoe het een damstuk van het punt waar de damstukken liggen naar het doel kan verplaatst worden. Om de stappen aan de robot door te geven, worden er 'GripperPos' objecten gebruikt die attributen bevatten zoals de positie van de arm, de hoogte van de arm en of de arm open of dicht staat. De 'GripperPos' worden dan toegevoegd aan een vector, en deze vector wordt weer opgeslagen in 'positions.txt'. Tenslotte wordt ook nog het bestand 'board.gch' geupdate naar de huidige spelsituatie.

Het bestand 'IK.java' [6] is hergebruikt uit de vorige opdracht. Dit bestand berekent aan de hand van de vector in 'positions.txt' de bijbehorende 'Joint Values'. De 'Joint Values' bestaan uit waardes van de hoeken van de gewrichten van de robotarm die weer berekent worden met behulp van inverse kinematica. [4] Hieronder worden de belangrijke methoden uit het bestand beschreven:
De methode 'handJointCalculation' geeft de juiste 'roll', 'pitch', en 'yaw' mee aan de 'gripper'. Het geeft ook de juiste waarde van de 'gripper' zelf met 'pos.grip'. De methode 'wristCoordinatesCalculation' geeft de pols coördinaten van de robotarm terug van een bepaalde positie. Alle gebruikte formules komen van blz. 100 uit het bestand 'Introduction to Robotics for the Computer Sciences.pdf'. [4] De methode 'armJointCalculation' kan met deze formules alle hoeken van de gewrichten berekenen. Om rekening te houden met de arm switch wordt sinus2 positief of negatief, afhankelijk van het x-coördinaat.

Shell Script

Om te zorgen dat de verschillende java en prolog bestanden achter elkaar uitgevoerd kunnen worden, wordt er gebruik gemaakt van een 'batch file' geschreven in 'shell script'. Hiermee is het mogelijk om meerdere commando's achter elkaar uit te voeren in de terminal. Het verloop van het spel wordt geïmplementeerd met behulp van een 'while loop'. In deze loop wordt eerst de beurt van de gebruiker beschreven en daarna de beurt van de computer. De beurt van de gebruiker ziet er als volgt uit: De gebruiker moet een zet invoeren en vervolgens wordt deze zet in fourInRowPP.java omgezet naar de bijbehorende 'Gripperpositions'. Deze 'Gripperpositions' worden weer omgezet in 'Joint Values' waarmee de robotarm een beweging kan maken. De beurt van de computer lijkt veel op die van de gebruiker, alleen wordt de zet van te voren berekend door 'FIARmove.pl' en via een tekst bestand door gegeven aan 'fourInRowPP.java'.

Testen

Om Vier Op 'n Rij te kunnen spelen zijn de bestanden 'FourInRowPP.java', 'FIARmove.pl', 'move.txt', 'positions.txt', 'board.gch', 'checkwin.pl', 'readBoard.pl', 'fourinarow.sh' en 'IK.java' [6] nodig. Als deze bestanden in dezelfde map zijn geplaatst, kan het spel gespeeld worden. Eerst moeten alle bestanden gecompileerd worden. Vervolgens moet er in de terminal (Linux centOS 6) 'fourinarow' getypt worden. In de terminal komt dan te staan wat de gebruiker moet gaan doen om het spel te spelen. De gebruiker kan een

zet doen door een positie te geven van het bord, bijvoorbeeld 45. Het eerste getal komt overeen met een kolom van het bord en het tweede getal komt overeen met een rij van het bord. Na op 'enter' gedrukt te hebben, zal er een nieuw bord toestand verschijnen in de terminal. De stukken die de gebruiker plaatst, wordt met een '1' aangegeven en die van de computer met een '2'. Een leeg vakje wordt aangegeven met een '0'. Om vervolgens de computer een zet te laten doen, dient de speler op een willekeurig toets te drukken op het toetsenbord.

Resultaten

Om de resultaten weer te geven is er gekozen om een gedeelte van het spel te tonen. De bestanden die de robotarm gebruikt zijn overbodig, aangezien de robotarm niet werkt.Hieronder staan een aantal spelsituaties met onder andere een computer gegenereerde move en een move die gekozen is door de gebruiker:

FIAR: Welcome to Four In A Row!

FIAR: Press a key to start the program...
FIAR: Present board setting (board.gch) is:
0000000
0000020
0001220
0012110
0021220
0121210e

Dit is een representatie van het bekende bord, de 'e' aan het eind geeft het einde van file aan.
FIAR: wanna play (p), reset board (r), or quit (q):r
Nadat het spel gestart is worden de verschillende opties getoond.
FIAR: Board is reset
FIAR: Present board setting (board.gch) is:
Het resetten van he bord is vrijwel altijd nodig, aangezien men een nieuw spel moet kunnen starten.
0000000
0000000
0000000
0000000
0000000
0000000e

FIAR: wanna play (p), reset board (r), or quit (q):p
Het spel spelen gebeurt hier.

FIAR: OK! let's go
FIAR: Start up the program...
De speler kan nu een eigen move bepalen.
FIAR: Enter move in (columnrow):15
De coördinaten voor 'column' lopen van boven naar beneden met grenswaarden van 0 tot 6. De coördinaten van 'row' lopen van links naar rechts met grenswaarden van 0 tot 5.
FIAR: Running FourInRowPP.java for move: 15
**** THIS IS THE FOURINAROW PP MODULE IN JAVA
De java module wordt aangeroepen om het pad van de robotarm te bepalen.
**** In high path
Het pad wordt aangeroepen en uitgevoerd.
FIAR: Running IK.java
'IK.java' berekent de 'Joint Values'.
FIAR: Present board setting (board.gch) is:
0000000
0000000
0000000
0000000
0000000
0100000e

FIAR: Your move was: 15

FIAR: Press any key to generate a computer move..
De computer doet een move.
FIAR: Generating computer move in prolog
Met prolog berekent het programma een move.
FIAR: Running FourInRowPP.java
Met de nieuwe move wordt nu hetzelfde gedaan als met de menselijke move.
**** THIS IS THE FOURINAROW PP MODULE IN JAVA
**** In high path
FIAR: Running IK.java

**** THIS IS THE STUDENT IK MODULE IN JAVA
FIAR: Present board setting (board.gch) is:
0000000
0000000
0000000
0000000
0000000
0102000e

FIAR: The computer move was:
3 5
Wanneer er door beide partijen een move is gekozen begint de beurt opnieuw en mag de speler weer een move bepalen.

Discussie

Nadat alle programma's zodanig voltooid waren dat er getest kon worden, werd er gekozen om eerst elk programma op zich te testen, te beginnen met 'FIARmove.pl'. Deze kwam niet met het verwachte resultaat, een 'move', maar gaf een 'out of local stack' error in prolog. Dit bleek te komen doordat het door ons gebruikte alpha-beta pruning zoek algoritme[3] alle mogelijke spellen moest doorlopen om een waarde aan een node toe te kennen. Dit bleek zo dus geen eficiënt algoritme zoals in de onderzoeksvraag gesteld is. De oplossing hiervoor was een heuristiek toe te voegen die bij een diepte van 4 in de search tree van het alpha-beta pruning zoek algoritme[3] een waarde toekent aan de nodes door een score te geven voor de hoeveelheid lijnen van twee en drie schijven van elke speler.

Na verder testen bleek de 'fourinarow.sh' niet goed te werken, dit doordat het command om de 'FIARmove.pl' uit te voeren niet goed werkte. Uiteindelijk bleek dit te komen doordat in 'checkwin.pl' die in 'FIARmove.pl' wordt aangeroepen de functie 'tranpose' gebruikt word die uit een externe aangeroepen bibliotheek komt. Dit werkte niet in combinatie met 'fourinarow.sh' en hoewel dit probleem niet zozeer terugslaat op de ondezoeksvraag moest er toch een oplossig gevonden worden. De makkelijkste optie bleek uitendelijk om zelf een gelijknamige functie transpose te schrijven die hetzelfde resultaat gaf.

Kortom, gezien de resultaten van het programma doet de computer efficiënte zetten en zal het winnen van het spel wel eens een uitdaging kunnen zijn. Het is dus inderdaad mogelijk een strategisch en efficiënt algoritme te programmeren voor het spel Vier Op 'n Rij.

Logboek

Datum	Aantal Uren	Waar	Wat
24-6-13	±4	Science Park	Project keuze gemaakt, 'readBoard.pl' gemaakt
25-6-13	±5	Science Park	Begin gemaakt aan 'FourInRowPP.java', begin gemaakt aan 'FIARmove.pl'
26-6-13	±5	Science Park	Begin gemaakt aan layout en inleiding van Paper, FourInRowPP.java werkend gekregen, 'checkwin.pl' gemaakt, verder aan 'FIARmove.pl gewerkt'
27-6-13	±6	Science Park	Verder Materiaal en methode, logboek en apendix geschreven voor paper, 'shell script' gemaakt, alle prolog bestanden werkend gekregen en Vier Op 'n Rij getest/gedraaid
28-6-13	±5	Science Park	Demonstratie Vier Op 'n Rij en grotendeels het paper geschreven.

Apendix

Gebruikte materialen

- Dell Optiplex 790 Intel core i3 Windows XP/Linux centOS Version 6
- Latex editor Texworks
- Java 1.6
- Gedit
- SWI-Prolog 6.2.1
- Search Actuate and Navigate Lab Course [5]
- Beoordelingsmodel Onderzoeksverslag Zoeken, Sturen en Bewegen [8]
- Handleiding Wetenschappelijke Verslaglegging [1]
- FourInRowPP.java
- FIARmove.pl
- move.txt
- positions.txt
- board.gch
- checkwin.pl
- readBoard.pl
- fourinarow.sh
- IK.java [6]

References

[1] Coördinator Academische Basisvaardigheden, *Handleiding wetenschappelijke verslaglegging* (2013).

[2] I. Bratko, *24.2 the minimax principle*, Prolog programming for artificial intelligence, 2012.

[3] Ivan Bratko, *24.3 the alpha-beta algorithm: an efficiënt implementation of minimax*, Prolog programming for artificial intelligence, 2012.

[4] Leo Dorst, *An introduction to robotics for the computer sciences* (2001).

[5] Robrecht Tim Elise Michael, *Search, actuate and navigate lab course (zsb)* (2013).

[6] Nikos Massios Matthijs Spaan, *Ik.java: Assignment for the inverse kinematics part of the zsb lab course* (2008).

[7] Marc Schoenauer Michle Sebag David Silver Csaba Szepesvri Olivier Teytaud Sylvain Gelly Levente Kocsis, *http://cacm.acm.org/magazines/2012/3/146245-the-grand-challenge-of-computer-go/fulltext* (2012).

[8] UVA, *Beoordelingsmodel onderzoeksverslag zoeken, sturen en bewegen* (2013).