FourInaRow 3.1 (95.01.02)

<std.disclaimer>

Contents

Preface

This program is based on code written by me (Wizo) and Mikael Fröberg in a course in algorithmics we took this spring (93). One of the lab tasks was to write a simple program which played four in a row, based on some algorithms we studied. So we did, and then I took the code home to my Amiga (500+) and compiled it. It worked fine except that it was dead slow. (It was still fast enough on the SUN Sparcstations at school). However at the end of the lab I had figured out a good way to improve the speed, so one day I started working on it at home. Soon I had a version (1.0) which was about 30 times faster than the original. Most code is rewritten, only the basic idea remained the same. And then of course I had to make a version (2.0) with a simple GUI to make it more playable. Now a year later I have made a new version (3.0) with improved GUI, and more more features.
I also sort of needed to learn a little about programming in Windows, so I thought it would be a nice idea to make a Windows version of FourInaRow. It was actually pretty easy to port the code, and the result was a FourInaRow almost identical to the Amiga version. I apologize if you find the graphics ugly. That might have something to do with that this is developed on an old PC with only 16 colors.
Version 3.1 contains some bug fixes, most important is font sensitivity/resolution independence (I hope it works now). Also the graphic drawing is now done more like the way it's supposed to be done in Windows (before it was done closer to how graphic drawing is supposed to be done (which is not possible in Windows)).

About the algorithm
Instructions

Contents

Instructions

The goal of FourInaRow is to get four bricks in a row horizontally, vertically or diagonally. There are two players (each player can be played by either a human or the computer), one is yellow and the other is red. You can see who is yellow and who is red in the upper right corner of the window. Each player alternate drop a brick in one of the columns of the board. The brick will appear in the bottom of the column. The game continues until one player has four in a row or the board is full. You can see whose turn it is to the right.
If it's your move just click your left mousebutton in the column where you want to drop a brick. You can also enter the figure associated with the column on the keyboard. A brick will appear and it's the other player's turn.
If it's the computer's turn you will see it's thinking progress and then it will make it's move.
You can see various other messages to the right:
Eval is a value the computer uses - the higher it is the better it think it's position is.
Level and search depth indicates how strong the computer plays, more about that later.
You can also see where the last brick was played.
At the bottom you can see how much time each player has used.

The menus

Contents

THE MENUS

Notice that many of the menu choices has keyboard shortcuts

Game

New	- Starts a new game.
Undo moves	- Will let you undo any number of moves.
About	- About this game.
Help	- What you are reading now.
Quit	- Exit this game.

Players

Changeside	- Whoever was yellow will be red and vice versa.
Yellow is human	- Yellow will be played by a human player.
Yellow is computer	- Yellow will be played by the computer.
Red is human	- Red will be played by a human player.
Red is computer	- Red will be played by the computer.

Eval (You can ignore this whole menu.)

Normal	\ These are the two eval methods to choose from.
Differential	/ Always choose differential - it's faster.
Endgame inc searchdepth	- See level menu below.

Level

Here you can set the level from 1 to 10. The level controls the search depth. The greater the level the better the computer plays, but the more time it uses. Increasing the level one step will increase the time for each move with about 5 times. At start level is the same as search depth but if you've set *Endgame inc searchdepth*, the searchdepth will increase as the computer needs less time for each move. This will give closer to using constant time for each move, and will make the computer play better at little cost. You can see both level and searchdepth to the right.

You can select most menu items also when the computer is thinking and get immidiate action and the computer will cancel it's thinking.
Changing level while the computer is thinking will not stop the computer thinking and it might take a little while before the search depth changes.

Instructions

Contents

About the algorithm

The program uses a simple minimax routine with alpha-beta priming. For you who don't know what that is here's an explanation:
The program tries all moves for a couple of moves ahead and then examines the board trying to estimate the position with an eval value. Then it goes back one move and tries another last move and evaluates the board again etc etc. Every second move it tries to make a move which maximies the eval value and the other moves it tries to minimize the eval value. Actually it doesn't examine all moves because the alpha-beta priming means that it sorts out certain moves (ie branches in the search tree) that can't possibly affect the result. Evaluating the board is done by looking up a table row for row, column for column and diagonal for diagonal.
Four in a row is worth 10000, three (not necessarily in a row but with possibilites for getting four) is worth 100 (one possible way to get four) or 200 (two possible ways to get four) and two is woth 10-20. The value is positive for the computer and negative for the human. After all these moves are tried it makes the best one.
The differential eval method works a little bit different. Instead of making a couple of moves (depending on the search depth) and then evaluating the whole board it evaluates the differens for each move that is made. That is - it only needs to check one row, one column and two diagonals for each move made. It isn't hard to realize that this means a heck lot less calculations. Well this is a very brief description, but it's actually very simple so anyone with a little knowledge about programming, minimax- and alpha-beta routines could easily do this. Then of course my implementation with a precalculated table makes it quite fast. But if you know of any better/faster methods I would like to know. About the only improvements I can think of now is making the computer calculating while the human player is thinking. With the differential eval method I have used that would be easy to implement, maybe I will do that sometime. And then there is a couple of things to make it a little more intelligent. For instance having two (or more) possibilities to get for in a row if they both need playing in one particular place, isn't any better that just one possibility (see below).

```
 .  .  .  .  .  .  .
 X  O  O  .  .  .  .
 X  X  X  .  .  .  . ←
 O  O  X  .  .  .  .
 O  X  O  O  O  X  .
 X  O  O  X  O  X  X
          ↑
```

X will get two four in a row if he gets to play here
but that doesn't increase his possibility to win.

Things like this is currently not taken care of by the program. The problem is that taking care of cases like this takes a lot of time. As long as one only look at small local parts (like one row) at one time it's possible to make a fast routine like those I have made. But the above case takes comparing of several rows, columns and diagonals. The way I see it this "improvement" of the program will make it so much slower that it will play worse than it is now given a certain allowed average time per move. Again if you have any ideas, which might improve the program without making it that much slower I would like to hear them. (I haven't thought about it that much myself because I think it's impossible to find any good solutions.)

This program is (of course) written in C. Apart from this Windows version (made using Visual C++ 1.00), I also have an Amiga version (made using Lattice/SAS C 5.10) and a version without the GUI which will compile with about any (ANSI) C compiler in most environments.

How to contact the author

Contents

Contacting the author

If you like this program, I would like you to send some of your own programs to me. If you do that you will most likely receive more of my programs in return (but probably only if you have an Amiga since this is my first Window program). If you don't write own programs, a small amount of money, a couple of disks or anything else is always welcome (but not absolutely necessary).
If you are a student or if you don't have a job this program is for free.
If you have opinions, ideas, bug reports or whatever, don't hesitate to contact me:

email: d1wizo@dtek.chalmers.se

snail: Ola Lundkvist
Ekv. 8
S-360 44 INGELSTAD
SWEDEN

Contents