

Writing GO: A Programmer's Notebook

By Thomas J. Clancy, Jr.

Monday, January 15, 2001 8:32 PM

This is the tale of a mediocre GO player who is attempting to write a computer game called GO, one of the hardest games to write and, indeed, it has stumped the best minds who have tried to create a game worthy of even a mere amateur ranked player.

So, I've taken on the extremely ambitious and perhaps somewhat foolhardy journey into the realm of writing GO. It all began with a simple idea, really. I was bored and wanted something to do, and having only my books and some PC-based GO games with which to study and play, I began to think not about playing the game so much as about creating my own computer GO game. The idea kept eating away at my brain, burrowing, in fact, into the depths of my subconscious, and I began to even dream about a computer GO game. And unfortunately when I get my mind wrapped around something like this, I can't stop obsessing over it, and so one day, recently, I decided to take action.

I decided that I didn't necessarily want to jump right in and take on the creation of an entire GO-playing computer program, but rather I thought that I should, over time, write the tools in C++ necessary to begin the creation of a GO-playing program. Basically, I began writing a foundation, a framework, if you will, for designing a GO game. And I plan, over time, to learn more about the AI necessary in creating a computer GO player, what I call the GO-engine. Actually, I've come up with a slightly snazzier name than simply "GO-engine." I thought the "Revelation Engine" had a nice ring to it, what with its connotations of divine truth and biblical rapture.

In fact, the Merriam Webster Collegiate Dictionary, 10th edition, defines revelation as "an act of revealing or communicating divine truth," or "an enlightening or astonishing disclosure." Perhaps the name, being biblical and apocalyptic, isn't necessarily appropriate, but I like it. Then again, perhaps the "Wrath of God" connotations it seems to evoke will inspire me to delve deeply into creating from nothing this monstrous beast, a Dan-level-strong game, and thus aid me in procuring that million or so reward that has thus far eluded many an attempt at writing such a thing.

Now you might think that like any disciplined researcher I would have spent many days, nay weeks! pouring through the literature and theory of games, especially the game of GO. Perhaps you might also have thought that I would have downloaded whatever source code that I could find (gnugo anyone?), studying it for hours in order to understand the nature of a GO computer game. But no, not me. I admit that I did download the source to gnugo and I do, every once in a while, even browse through it. And I also admit having, from time to time, read some bits and pieces of GO computer game theory, but for the most part, and like the lazy fool that I am, I decided instead to go about doing my own work, my own thinking, making my own mistakes.

While I plan to read about neural networks and other potentially useful ideas for creating my Revelation engine, I've decided that I should not pollute my mind with the thoughts of others. At least not now. I'd like, rather, to crash and burn on my own than to take the easy way out. And besides, it gives me the intellectual

stimulation I need away from work, the stimulation that I don't get anywhere else, the stimulation that only comes through self-discovery and meditation.

The Game...

Thus far I don't have a game, but rather I have the notion of a GO board, playing pieces (go stones), points in space on the GO board, the notion of chains ('Friend' stones that are linked on the GO board), and a rudimentary display abstraction for displaying the GO board (currently it shows a simple text-based board similar to that of gnugo's text board; and no, I did not look at the source code, I merely copied the output).

So far you can place a stone on the board, and some rudimentary rules will check to make sure that you didn't place it on a point out of bounds (e.g. x(20), y(32) on a 19x19 Board) or on a point already occupied by a stone.

The board keeps track of a map of stones keyed by normalization of the points. By normalization I mean $(X*100+Y)$. Thus, a point on the board at x(1), y(2) hashes to a value of 102. The second piece of data of the map pair is a smart pointer to a stone. I try to not waste memory by keeping smart pointers to each piece used, thus a stone on the board is the same as the same stone managed in a chain or in the captured pile of a player--a single instance of a stone is the same stone throughout its lifetime during the game.

Chains are merely another way of looking at the stones, and are useful in determining the strength or weakness of a group of connected stones on the board, or so I imagine. That is to say, while the board keeps track of all stones on the playing field, chains keep track of those 'like' or 'friend' stones (i.e. stones of the same color) on the field that are connected. These are sometimes called strings, or perhaps they're always called strings. I like the notion of chains. Chains to me are symbols of strength. Maybe I can take the metaphor further and speak of walls and barricades. And I haven't even begun to think about eyes. Whilst my toolkit can managed and recognize chains, and soon be able to count the liberties of a chain, it has no notion of eyes... Eyes will be an interesting challenge. And truthfully I have no idea if I'm even going about this in the right manner, but I suppose that any manner in which one chooses to go is right for him, as long as it gets the job done to his satisfaction, or until he becomes completely bored of the entire thing and decides to go do something else.

A little background... I began one night last week just putzing around. I created a simple console application called gotools, and then began creating each of the C++ classes in the same file. My next step, I think, is to reorganize the source code, putting each class into its own header file and source file, thereby making things a little easier to read. Not that Microsoft's Visual Studio doesn't do a good job what with it's class browser, but I don't plan to keep it specifically a Visual C++ application. I am designing it purely in standard C++ so that it can be built on any platform that has a standard C++ compiler. Of course specific platform code will reside in subclasses of the IDisplay interface allowing it to be compiled conditionally, or perhaps through project and make file configurations. I don't know yet. Right now my main concern is building the framework so that I

can begin to design the Revelation engine. A text-based interface, for the moment, will suffice.

Here in fact, in Figure 1, is a printout of what my go tools look like. Well, the text-based display, anyway.

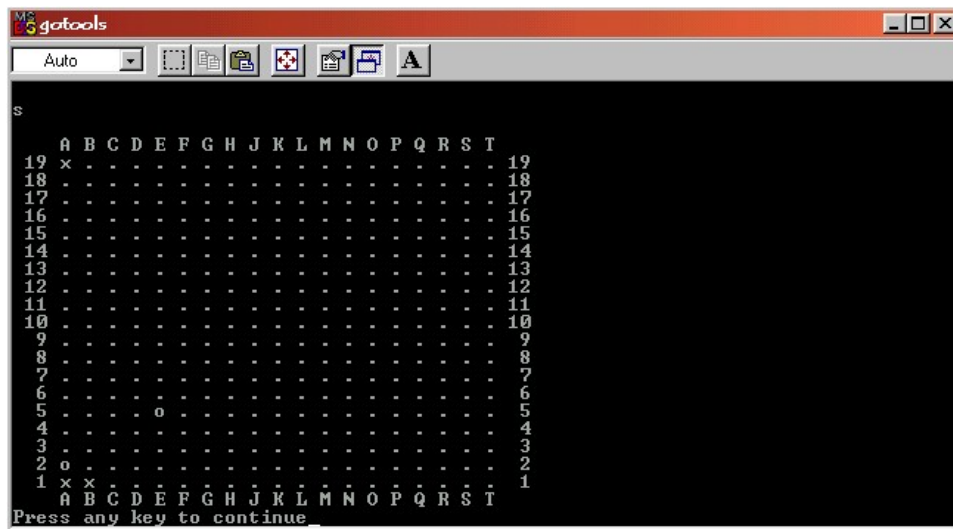


Figure 1. Text-based GO board display.

I had in fact created a version previous to this one, but after I'd noticed that the text board that comes with gnugo was better presented, I decided to borrow it's design. Note that the 'I' is missing from the coordinates across the X axis. I don't know why this is except that in a number of books and in a number of GUI-based interfaces for IGS (Internet GO server), the 'I' is left out on purpose. I'll have more on this later. My first thought is that it's left out so that it doesn't clash with the number '1'. I9 seems a little odd, but I wouldn't think that it would be too difficult to interpret by eye. Still, there is a reason and I shall uncover it.

The thing to note here, too, is that the letter 'x' stands for black pieces while 'o' stands for white pieces. Using 'b' and 'w' seemed awkward. The x and o are symmetrical enough that they appear to be round stones, assuming you can stretch your imagination just a bit.

11:49 PM

Rudimentary chain management works. I can merge chains if a stone connects two or more chains together. For example assume I have the following scenario (forgive the crudeness of these sketches):

```

  A B C D E
5 . . . . .
4 . x . . .
3 . x . x .
2 . . . . .
1 . . . . .

```

Initially, after these black pieces have been placed, the board's chain manager recognizes two chains of black stones, [B4, B3] and [D3] (I'll use the simple set notation here to denote a chain of stones.) If I were to then place a black stone at C3, the chain manager would merge the second chain [D3] with the first chain [B4, B3] and then add C3 to the merged chain, thus creating one chain from two.

```

      A B C D E
5    . . . . .
4    . x . . .
3    . x x x .
2    . . . . .
1    . . . . .

```

I now have a single chain [B4, B3, C3, D3].

The next task will be to calculate a chain's liberties. A liberty is a free adjacent point of a stone. Thus

```

      A B C D E
5    . . . . .
4    . . . . .
3    . . x . .
2    . . . . .
1    . . . . .

```

[C3] has four liberties, (C4), (D3), (C2) and (B3) (note here I used parenthesis around points on the board that are empty. This is, of course, my own notation and, apart from the standard coordinate system of numbers on the Y-axis and letters on the X-axis, adheres to nothing else that I know of.

For future reference, I will have to remember to make my toolkit compatible with SGF, which I believe stands for "Standard Game Format." It is a standard file format for storing moves made in two-player games such as GO, Backgammon, Chess (I think), etc. I looked at a sample of an SGF file and it seems quite cryptic. But I think I've found some BNF grammars for the SGF language and thus I can probably write a parser and generator to read and write SGF files, respectively. I might probably create, for the moment, my own persistent storage mechanism based on XML for storing GO moves. And I might, at first, keep it simply for GO and not worry about keeping it abstract enough for any two-player game, although that, in itself, shouldn't be too difficult a task.

TODO:

1. Create static handicap Points for the different board sizes and then allow a handicap to be specified when generating the board.
2. Allow the create of a stone's coordinates (point) to be specified as "<letter><#>". For example, "C5", "T19", "D7", etc. Add this to the StoneFactory.
3. Design the liberty counting function for chains.
4. Pray for divine inspiration.

Tuesday, January 16, 2001 11:21 PM

As usual, my initial design has come up from behind to bite me in the ass. I guess it's just lack of foresight on my part. In order to correctly calculate the number of liberties around a string, I need to get a list of the unique empty points surrounding the string.

```
      A B C D E
5    . . . . .
4    . x . . .
3    . x x x .
2    . . . . .
1    . . . . .
```

For example, in the above scenario, the number of empty points surrounding the string is 10, so you might think that this is the number of liberties. But in fact, if you do, then you've counted (C4) twice, because it is a liberty to both [B4] and [C3]. So, in order to correctly count the liberties you need to create a unique set of points, thus discarding duplicates, and count them.

My current design keeps track of only the stones on the playing field and not the empty points. Thus, naively, I decided to call a method of the Board class that returns the adjacent stones surrounding each piece in the chain. From there I extrapolated the liberties by taking the number of total potential liberties for a stone (edge stones have 3, corner stones have 2 and all the rest have 4 total potential liberties) and subtract from that the number of adjacent stones. This, of course, presented the duplicate counting problem because, while I now know what stones are adjacent to me, I have no idea which empty points are adjacent and thus cannot create a unique set of these points. I'm sure there's probably a simpler way, but this is my first stab at it.

In order to solve this problem as I see it, I will need to create a method in the Board class that will return to me the set of empty points surrounding a stone. From there I can place the hash of each point into a set and then count the number of elements in the set. This could get expensive in computation when you're dealing with a large chain, but I think the impact will be minimal—or so I would hope. I'll have to run some timing tests once I create my good old Stopwatch class.

Wednesday, January 17, 2001 11:12 PM

I made the mistake the other day of telling several people at work that I'd decided to name my AI GO-playing engine "Revelation." I got a quizzical look from Sean and Corby asked me, "Why?" Oh well. I suppose that I ought to keep my grand visions to myself.

Thursday, January 18, 2001 10:43 PM

This evening I decided to modularize the code, so I broke up all the classes into logical source files. Not that it makes any difference in the class browser window, but it's sure nice to be able to easily page through a header for a specific thing than to wade through all those lines in one file.

I have a strange dependency and I'm not sure how to refactor it. Specifically, the Board class creates an instance of a chain manager, but the chain manager takes a pointer to the board because it uses methods within the Board to do some calculations. I was thinking that I ought to nest the Chain and ChainManager classes inside the Board class since, really, the Board class is the only thing that uses the Chains. But then I would need access to the ChainManager for creating the game engine.

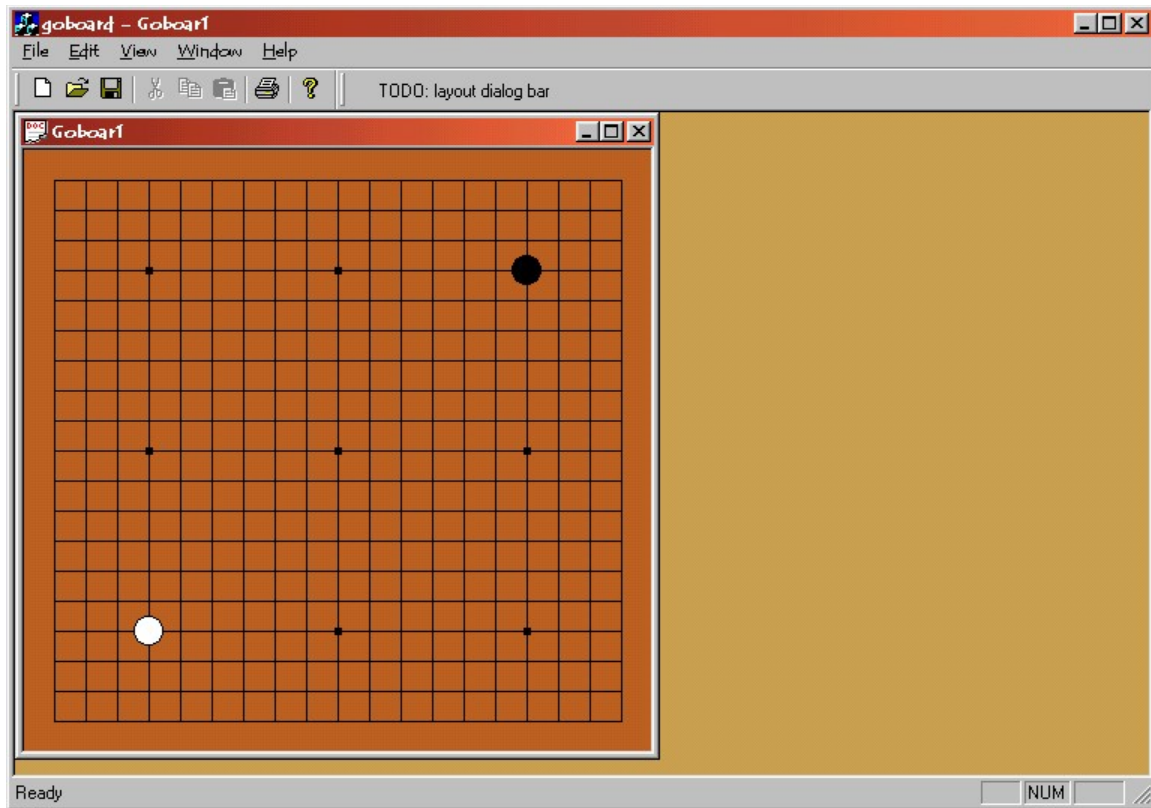
11:36 PM

I finally got the liberty calculator working. It's a little weird, but it works. I'd forgotten that the Board class contained several Point functions for determining an out-of-bounds condition and whether the point is occupied. The out-of-bounds condition takes care of corner and edge pieces and the occupied check takes care of any point that already contains a stone. I'm beginning to think that there is a more efficient way to handle all of this, but I'm hoping that as it evolves and as I refactor various pieces over time, the game toolkit will mature.

Oh yeah, it's now called the GO ToolKit. In fact, I named the C++ namespace GOTOolKit for lack of a better and more original name. I had been considering the name Pup-a-Go-Go, which brings back memories of this little hotdog stand that used to reside in the South Hills Village Mall when I was a child. But, I don't know. It sounded too obscure. Not at all serious, and although I don't want to be too damned serious, I figure that I should at least be somewhat serious.

Sunday, January 21, 2001 11:39 PM

I decided to take a break from coding the actual GO Toolkit and instead concentrate on brushing up on my Windows programming skills by creating an MDI (multiple document interface) application that draws GO boards. I managed to get the grid proportions correct. It merely draws simple black and white ellipses to denote the stone, but what the hell. It actually looks quite similar to the default Turbo GO board, but the background color of mine is slightly pale in comparison. Anyway, here is a snapshot of my work.



You can resize the GO board and everything will resize in proportion. It was quite simple, after I'd played for a while, to get all the proportions correct. Of course, if you make the Y axis shorter, everything looks squinty as does stretching the X axis. Still, it's pretty cool.

Unlike Turbo GO, I don't yet draw the coordinates around the border, which, I suppose, would come in handy. But for now this should be fine. Tomorrow, perhaps, I will attempt to draw the coordinates. Fortunately, having a pointer to a CDC (device context) makes things much easier. And I'm learning a shitload about MFC and Windows programming.

Here is the actual code to draw the grid properly within the child window (note that each variable beginning with an "m_" means that it is a member variable and is defined in the class header.)

And I must be getting rusty at this sort of thing because it took me a number of hours to figure this out. You should have seen the original attempts. Lines every which way. Man, what a mess. Now it all seems to work fine. I'm sure I could be more efficient somehow, but for now, this is fine.

```
void CBoardDisplay::Draw(CDC *pDC)
{
    CRect rect;
    pDC->GetWindow()->GetClientRect(&rect);

    m_pDC = pDC;

    m_InnerRect =
        CRect(CPoint(20,20),
```

```

        CPoint(rect.BottomRight().x-20, rect.BottomRight().y-20));

int nWidth = m_InnerRect.Width();
m_nWSpacing = (int)nWidth/(m_nCols-1);
int nWSpace2 = m_InnerRect.TopLeft().x+((m_nCols-2)*m_nWSpacing);

while(m_InnerRect.BottomRight().x - nWSpace2 != m_nWSpacing)
{
    m_InnerRect.BottomRight().x--;
    nWidth = m_InnerRect.Width();
    m_nWSpacing = (int)nWidth/(m_nCols-1);
    nWSpace2 = m_InnerRect.TopLeft().x+((m_nCols-2)*m_nWSpacing);
}

int nHeight = m_InnerRect.Height();
m_nHSpacing = (int)nHeight/(m_nRows-1);
int nHSpace2 = m_InnerRect.TopLeft().y+((m_nRows-2)*m_nHSpacing);

while(m_InnerRect.BottomRight().y - nHSpace2 != m_nHSpacing)
{
    m_InnerRect.BottomRight().y--;
    nHeight = m_InnerRect.Height();
    m_nHSpacing = (int)nHeight/(m_nRows-1);
    nHSpace2 = m_InnerRect.TopLeft().y+((m_nRows-2)*m_nHSpacing);
}

CBrush brush;
brush.CreateSolidBrush(m_BoardColor);
pDC->FillRect(&rect, &brush);

CPoint points[5] = {
    CPoint(m_InnerRect.TopLeft().x, m_InnerRect.TopLeft().y),
    CPoint(m_InnerRect.BottomRight().x, m_InnerRect.TopLeft().y),
    CPoint(m_InnerRect.BottomRight().x, m_InnerRect.BottomRight().y),
    CPoint(m_InnerRect.TopLeft().x, m_InnerRect.BottomRight().y),
    CPoint(m_InnerRect.TopLeft().x, m_InnerRect.TopLeft().y)};
pDC->Polyline(points, 5);

for (int i=1; i < m_nCols-1; i++)
{
    CPoint points[2] = {
        CPoint(m_InnerRect.TopLeft().x+(i*m_nWSpacing), m_InnerRect.TopLeft().y),
        CPoint(m_InnerRect.TopLeft().x+(i*m_nWSpacing),
            m_InnerRect.BottomRight().y)};
    pDC->Polyline(points, 2);
}

for (i=1; i < m_nRows-1; i++)
{
    CPoint points[2] = {
        CPoint(m_InnerRect.TopLeft().x, m_InnerRect.TopLeft().y+(i*m_nHSpacing)),
        CPoint(m_InnerRect.BottomRight().x,
            m_InnerRect.TopLeft().y+(i*m_nHSpacing))};
    pDC->Polyline(points, 2);
}

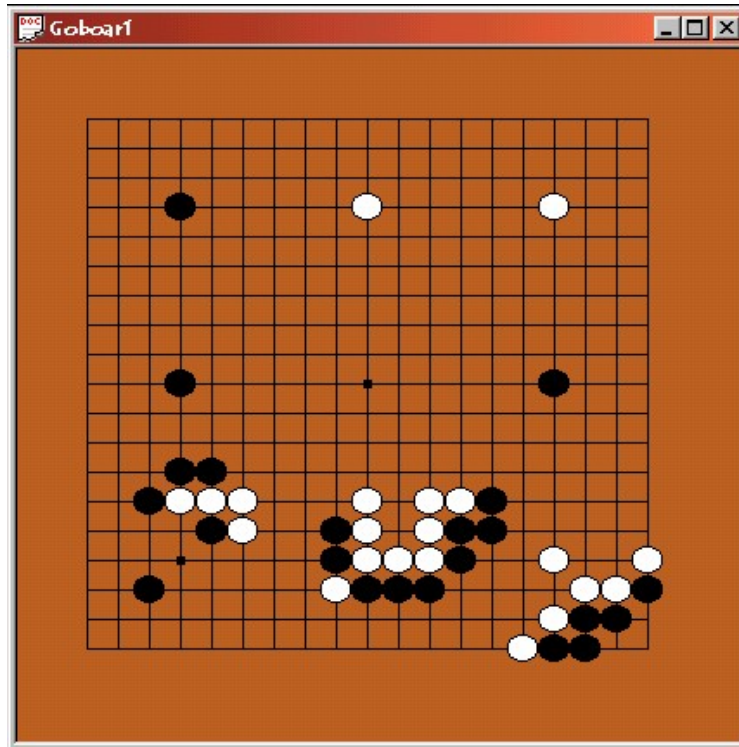
DrawHandicapPoints(pDC);
}

```

Monday, January 22, 2001 10:47 PM

This evening I spent some time marrying the two projects, my GO Toolkit and my Windows GO Board interface, and am now able to alternately place stones by pressing either the left mouse button to place black stones or the right mouse button to place white stones. The `GOToolKit::Board` interface is now part of the visual interface, so whenever I place a stone on the visual board, a virtual stone gets placed on the virtual board and the chains get updated, etc.

Here is a picture of the go board after randomly blacing stones.



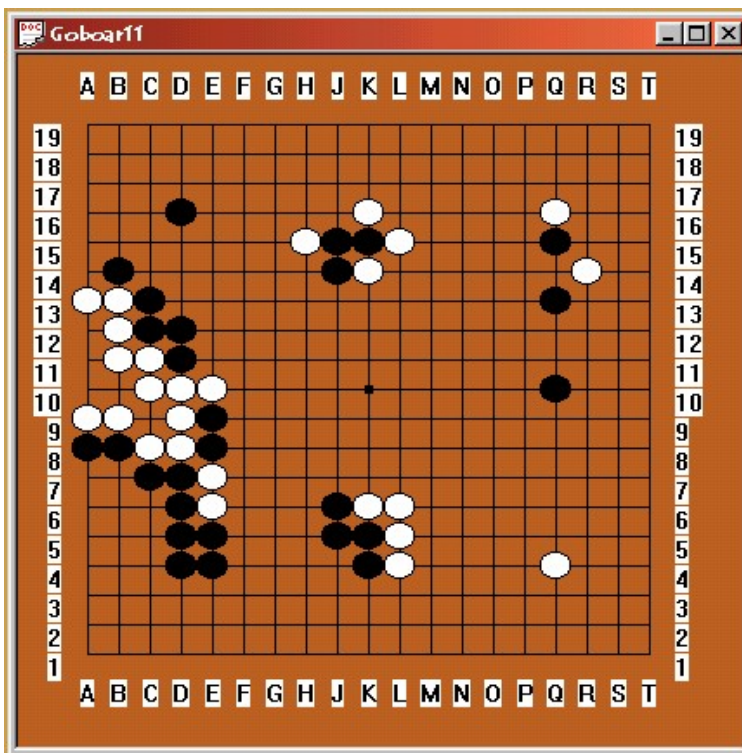
Well, this really wasn't a game or anything. I was trying to make moves and such. But still, you get the point.

Of course, there is no notion of gameplay yet. That is, I cannot press the left mouse button and have it do black first, then white on the next click. I will need the beginnings of a game interface and a player. The player, a human player, will somehow need to be tied into the Windows event model so that a click, for example, is the players event to make a move.

Tomorrow, perhaps, I will attempt to put the coordinates around the board. I don't suppose it will be too difficult. There seems to be some text tools as part of the device context that gets passed into the `OnDraw()` method of the view. The view is the inside of the window above. It's what I am painting brown and placing lines and circles on.

Wednesday, January 24, 2001 12:28 AM

I managed to get some rudimentary coordinated drawn, although I don't like them much. Here is a snapshot:



Note the white borders around the characters. I wish I could figure out how to make the letters draw transparently or how to make their background color the same as the board. So far I've found little to help me. Still, I can now see the coordinates and in fact it kinda looks a little like the text version of my board.

Wednesday, January 24, 2001 11:01 PM

Today at work with the help of sean who told me about setting the background mode in the device context to opaque, I managed to make the text's background the same color as the board.

