a i - j u n k i e

Putting it all together

I'm not going into a detailed description of the `CMineSweeper` and `CController` classes because they should be easily understood from the comments within the code. I *will* include them in the tutorial however if enough people pester me. I will describe the main loop though, just so you know exactly what's going on in there. I have omitted some lines that are included in the actual source for clarity. The missing code just deals with cosmetic stuff like updating the graph display and other stats.

```
bool CController::Update()

{

  //run the sweepers through CParams::iNumTicks amount of cycles. During

  //this loop each sweeper's NN is constantly updated with the
appropriate

  //information from its surroundings. The output from the NN is obtained

  //and the sweeper is moved. If it encounters a mine its fitness is

  //updated appropriately,

  if (m_iTicks++ < CParams::iNumTicks)

  {

    for (int i=0; i<m_NumSweepers; ++i)

    {



      //update the NN and position

      if (!m_vecSweepers[i].Update(m_vecMines))

      {

        //error in processing the neural net

        MessageBox(m_hwndMain, "Wrong amount of NN inputs!", "Error",
MB_OK);

        return false;
```

```
      }



      //see if it's found a mine

      int GrabHit = m_vecSweepers[i].CheckForMine(m_vecMines,
CParams::dMineScale);



      if (GrabHit >= 0)

      {

        //we have discovered a mine so increase fitness

        m_vecSweepers[i].IncrementFitness();



       //mine found so replace the mine with another at a random

       //position

       m_vecMines[GrabHit] = SVector2D(RandFloat() * cxClient,
RandFloat() * cyClient);

      }



      //update the fitness score

      m_vecThePopulation[i].dFitness = m_vecSweepers[i].Fitness();

    }

  }
```

This first part of the if statement runs all the minesweepers through one generation (one generation consists of `CParams::iNumTicks` amount of computer cycles) updating their neural nets and their positions accordingly. If a land-mine is found it is removed and that minesweeper's fitness score is increased by 1. The land-mine is then replaced by another at a randomly generated position.

```
  //Another generation has been completed.

  //Time to run the GA and update the sweepers with their new NNs

  else

  {
```

```
    //increment the generation counter
    ++m_iGenerations;


    //reset cycles
    m_iTicks = 0;


    //run the GA to create a new population
    m_vecThePopulation = m_pGA->Epoch(m_vecThePopulation);


    //insert the new (hopefully)improved brains back into the sweepers
    //and reset their positions etc
    for (int i=0; i<m_NumSweepers; ++i)
    {
      m_vecSweepers[i].PutWeights(m_vecThePopulation[i].vecWeights);
      m_vecSweepers[i].Reset();
    }
  }
  return true;
}
```

The else statement kicks in at the end of every generation. It's this chunk of code which collates all the minesweepers chromosomes and fitness scores and sends the information to the genetic algorithm. The GA does its stuff, passes the new weights back which then get put into a new generation of minesweepers brains. Everything is reset and a new cycle is run as per the previous paragraph.

This `Update` function loops endlessly until you decide the minesweepers have evolved interesting enough behaviour. This usually takes around fifty generations.

Hitting the 'F' key when the program is running will put the program into accelerated time mode and you'll see a simple graph of the population's progress.

Stuff to Try

Evolve minesweepers that avoid the mines.

Evolve minesweepers that pick up the mines but avoid another type of object. (not as easy as you think)

When you've played around a little with the code the more observant of you will notice that the simple crossover operator used here is not very effective. Can you think why? Can you design a more effective crossover operator?

It's possible to design the neural networks in a way that uses far fewer inputs and hidden neurons. How small can you make a network and yet still evolve effective behavior?

And that's all folks!

I thought I was never going to get to the end of this but here we are at last! If any of you do anything interesting with neural nets after you have read this tutorial I would love to be informed. Please feel free to use my code in your own projects but I'd appreciate it if you give me credit where due.

And most of all, have fun!

If you have enjoyed (or not!) the tutorial please take a moment to comment or ask questions on the message board. Getting feedback from you makes the effort seem all the more worthwhile.
