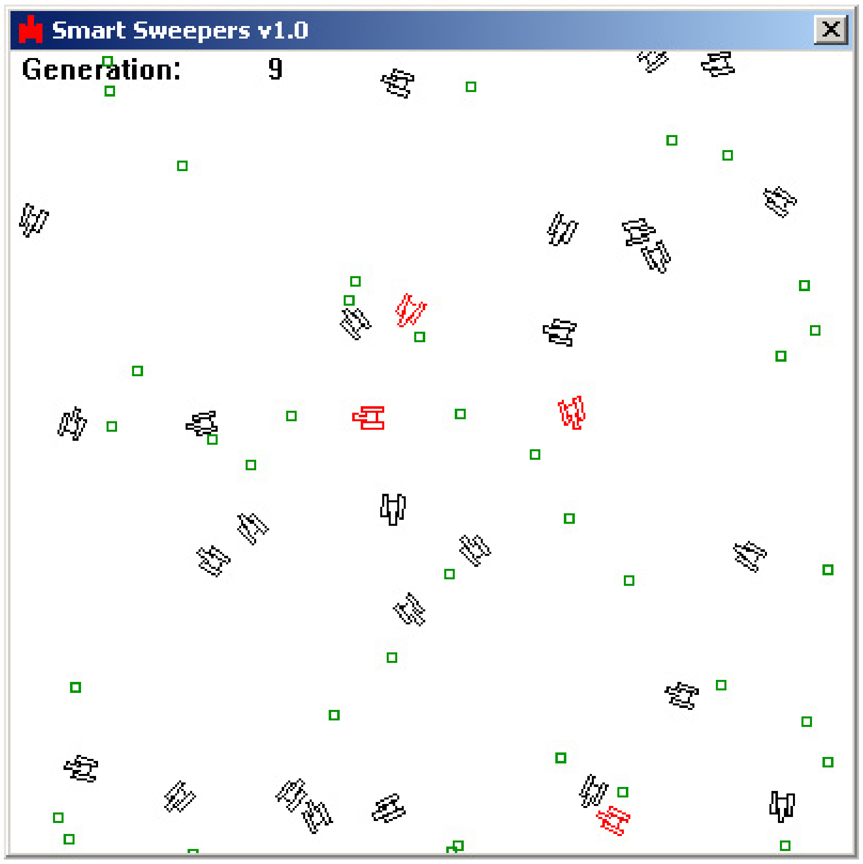
a i - j u n k i e

So, what's our project going to be fup?

We are going to evolve virtual minesweepers to find and collect land-mines scattered about a very simple 2D world. (In the original version of this tutorial the program evolved ants that collected food but I fancied a change. ;0))

This is a screenshot of the application:



As you can see it's a very simple display. The minesweepers are the things that look like tanks and the land-mines are represented by the green dots. Whenever a minesweeper finds a mine it is removed and another mine is randomly positioned somewhere else in the world, thereby ensuring there is always a constant amount of land-mines on display. The minesweepers drawn in red are the best performing minesweepers the program has evolved so far.

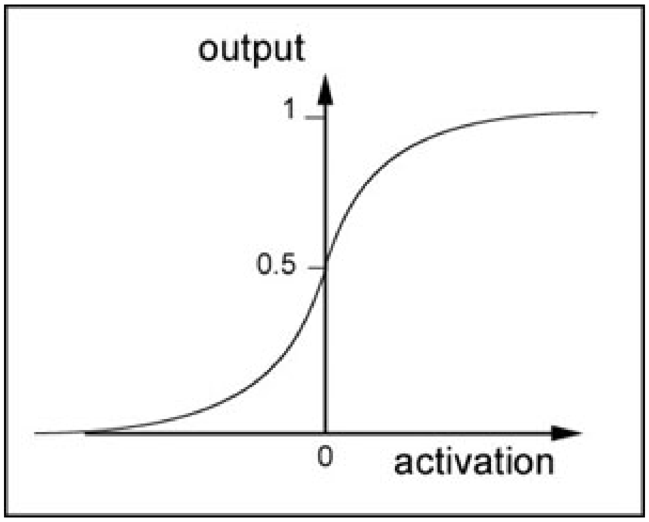
How is a neural net going to control the movement of a minesweeper? Well, just like the control of a real tank, the minesweepers are controlled by adjusting the speed of a left track and a right track. By applying various forces to the left and right side of a minesweeper we can give it a full range of movement. So the network requires two outputs, one to designate the speed of the left track, and the other to designate the speed of the right track.

The more thoughtful of you may be wondering how on earth we can apply varying forces when all we've discussed so far are binary networks outputting 1's and 0's. The secret to this is that instead of using a simple step (threshold) activation function we use one which softens the output of each neuron to produce a symmetrical curve. There are several functions which will do this and we are going to use one called the *sigmoid* function. (sigmoid, or sigmoidal is just a posh way of saying something is S shaped)

$$output = \frac{1}{1+e^{-a/p}}$$

This equation may look intimidating to some of you but it's very simple really. The *e* is a mathematical constant which approximates to 2.7183, the a is the activation into the neuron and p is a number which controls the shape of the curve. p is usually set to 1.0.

This function is terrific and comes in handy for all sorts of different uses because it produces an output like this:



The lower the value of p the more the curve begins to look like a step function. Also please note this curve is always centred around 0.5. Negative activation values produce a result less than 0.5, positive activation values produce a result greater than 0.5.

Therefore, to obtain a continuously graded output between 0 and 1 from our neurons we just have to put the sum of all the inputs x weights through the sigmoid function and Bob's your uncle! So that's our outputs dealt with, what about the inputs?

I have chosen to have four inputs. Two of them represent a vector pointing to the closest land-mine and the other two represent the direction the minesweeper is pointing. I call this vector, the minesweepers look-at vector. These four inputs give the minesweeper's brain - its neural network - everything it needs to know to figure out how to orient itself towards the mines.

This is not the time and place to start explaining what vectors are so if you do not understand them I suggest you break off and learn about them before you continue with this tutorial.

Now we have defined our inputs and our outputs what about the hidden layer/s? How do we decide how many layers we should have and how many neurons we should have in each layer? Well, this is a matter of guesswork and something you will develop a 'feel' for. There is no known rule of thumb although plenty of researchers have tried to come up with one. By default the simulation uses one hidden layer that contains six neurons although please spend some time experimenting with different numbers to see what effect they may have. I'd like to emphasise here that the more you play around with all the parameters the better the 'feel' you are going to develop and the better your neural networks will be.

Time to look at some code now. Here's a quick breakdown of the important classes.

`CNeuralNet` is the neural net class (surprise surprise).

`CGenAlg` is the genetic algorithm class.

`CMineSweeper` is a data and controller class for each minesweeper.

`CController` is the controller class which ties all the other classes together.

`CParams` is a class which loads in all the parameters for the application. They can be found in the file 'params.ini'. I strongly suggest you play around with the settings in this file when you start to play around with the code.

---

1 2 3 4 6 7 8 **Next** **Home**