A major task accomplished by real and artificial agents is sensori-motor control, that is, responding appropriately to a dynamically changing environment. Indeed, there is a whole area of engineering and math devoted to this called "**control theory".**

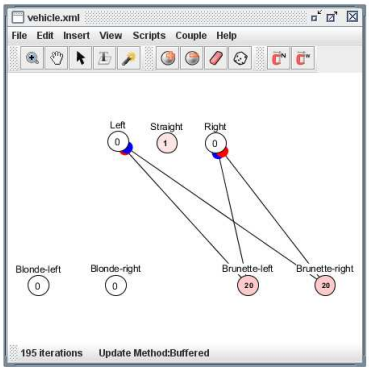
Our human neural networks are clearly very good at this, and it's all the more impressive given the massive amount of information they are bombarded with. In reaching for a loaf of bread, catching a ball, or combing my hair, I have to react to millions of signals at my sensory receptors to produce the correct millions of signals in return. How would this be modeled in a neural network? On a first pass, by a vector-valued-function, a map, which takes a long list of input values, a sensory vector, and associates it with a long list of output values, a motor vector. How does this function work?

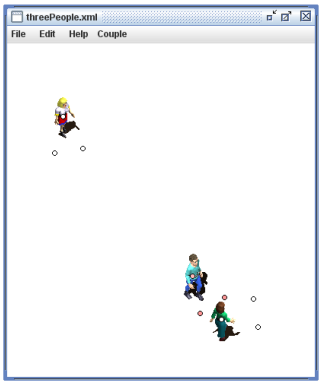
The topic is a big one both in industrial applications of neural networks and in cognitive modeling. Artificial neural networks turn out to be quite good at controlling complex processes, which is not surprising since our biological neural networks do it so well. We can begin, however, with a very simple simulation, which shows how even very small systems with only a handful of neurons can manage themselves nicely in a complex and changing environment.

Chasing Amy

Begin by clearing the workspace and opening your starting neural network and world:

- Clear the workspace by using **File** --> **Clear Workspace** located in the workspace window.
- Open a workspace using **File** --> **Open Workspace**.
- Select **ChasingAmy.zip** in the **"labs"** folder.





You can now see a network and a world window. The world is populated by **John Doe**, **Susy Blonde** and **Amy Brunette**. John's brain has only eight neurons. Five if you don't count the three that are basically muscles, **Left** that when active makes him turn left, **Straight** that impels him forward, which he does constantly, and **Right** that when active makes him turn right. This brain has only two neurons if you don't count the three that presently aren't connected to anything. And it makes sense not to count the unconnected ones, since most of the secrets of the brain's abilities are in the connections among neurons.

John Doe has begun a flirtation with Amy Brunette, but he's not coy – he continually charges ahead! Amy Brunette wears cheap perfume, causing John's left neuron to be most active when Ms. Brunette is on his left, and his right neuron to be more active when she's on his right.

Notice how the network is connected. The John-Left neuron excites the Left muscle motor neurons via an excitatory synapse (depicted by red) and inhibits the Right motor neuron thanks to an inhibitory synapse (depicted by blue). The John-Right odor-detecting neuron has the opposite connections. As in the moment the world snapshot was taken, this causes Joe M. to have more activity in the Right neuron when Amy is on his right. So he turns right, all the while moving forward as well.

Play with **John Doe**:

- Press the **play** button (a triangle) that's uppermost on the screen (NOT the play button for the network alone).
- Drag John around within the world.
- Note how he turns, depending on whether he is to the left or right of Amy Brunette, thanks to more or less activation on the left or right odor-detecting node triggering activity in the left and right motor neurons.

Play with **Amy Brunette**:

- While the simulation is still playing, drag Amy's image around and note the network reaction.
- Press the **play** button that's uppermost on the screen.
- Move Amy around in the world as indicated above and watch John react.
- Stop the network using the **stop** button.

Chasing Amy and Susy

Let's suppose John has also taken a liking to Susy Blonde. How can we rewire his brain so that he will go after both Amy and Susy?

The Blonde-Left neuron responds when Susy is on John's left, and the Blonde-Right neuron when she is on his right. Connect Blonde-Left and Blonde-Right to the appropriate motor neurons to make John chase Susy.

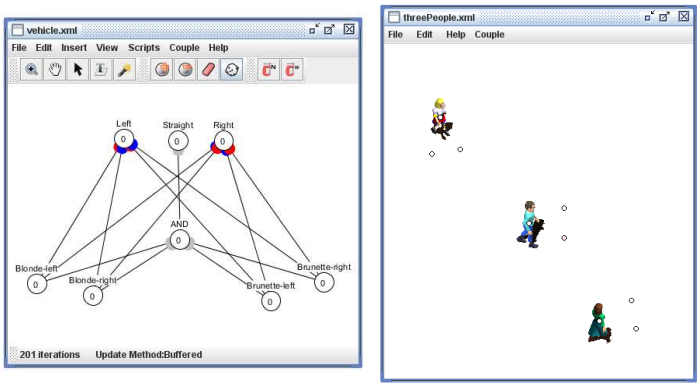
- Click once on the neuron above the Blonde-Left label and press **1** to make it a source
- Now select the Left and Right motor neuron and press **2** to connect the source to these target.
- Do the same now for the Blonde-Right and the Left and Right motor neurons.
- These connections you created in the last step are currently excitatory. We want to make the contralateral ones inhibitory. To do so, double-click on the little half-circle representing the synapse (if it is hard to isolate with the mouse among all the other synapses, you may need to drag the whole neuron around until you can double-click). In the Synapse Dialog box that pops up, change the strength to **-1**.

John Doe should now run after both Amy and Susy – drag Amy or Susy around and click **Play** at the very top to watch John go.

- Drag Amy Brunette back such that she is in the same place (overlapping) with Susy Blonde.

Chasing Amy or Susy not both

John needs to get smarter. He should realise that if Amy and Susy are both in the same location, he should not go there, because they may not feel very special if they see John going after both of them. The network needs to solve this, in formal logic terms, exclusive or (**XOR**) problem— if both A (Amy) and B (Susy) are true (present), A XOR B = false (don't pursue), whereas if one is true (present) and the other false (absent), A XOR B = true (do pursue).



First, we'll start with the network that chases Amy and Susy indiscriminately:

- Clear the workspace by using **File** --> **Clear Workspace** located in the workspace window.
- Open workspace using **File**-- > **Open Workspace**, and
- Select **ChasingAmySusy.zip** in the **"labs"** folder.

To prevent John from hitting on both Susy and Amy simultaneously (i.e. to solve the XOR problem), we will have a neuron compute whether both Susy and Amy are nearby (**AND** operation in logic terms). If indeed they are both nearby, this neuron will then prevent John from moving forward, saving his game.

- Select the Brunette-Left and Right neuron and press **1** to make them a source.
- Now select the lonely unconnected neuron, which we'll call the **intermediate neuron** and press **2** to connect these.
- Do the same with the Blonde Left and Right nodes and the intermediate neuron.

Now we want to make this intermediate neuron fire if and only if its activity is so large that John Doe must be near both Amy and Susy.

- Double-click on this neuron
- Set the Neuron type to **Binary**
- Set the Threshold to **38**

The intermediate neuron will only fire when the sum of all the activity exceeds **38**, which only occcurs when John is near both Amy and Susy. Now we must have this neuron inhibit the Forward neuron.

- Select the intermediate neuron and connect it to the **Straight** neuron
- Double-click on the synapse and set Strength to **-1**

Ensure that Susy Blonde and Amy Brunette are in the same location and then click the topmost **Play** to watch John stop himself before he reaches the party! Do you understand the network?

Although the example we used is cheesy, the network illustrates show how simple networks can embody important computational operations. These include the addition operation that the Left neuron computes over its odor-detecting neuron inputs, and the exclusive or (XOR) operation that the intermediate neuron computes over its inputs.

When more neurons are added, the resulting behavior can be extremely sophisticated. Consider that all the software of the desktop computer you run can be implemented by stringing together the simple logical operations of addition and exclusive OR, plus a readable and writable memory.

This **concludes** the tutorial. We hope you **enjoyed** it!