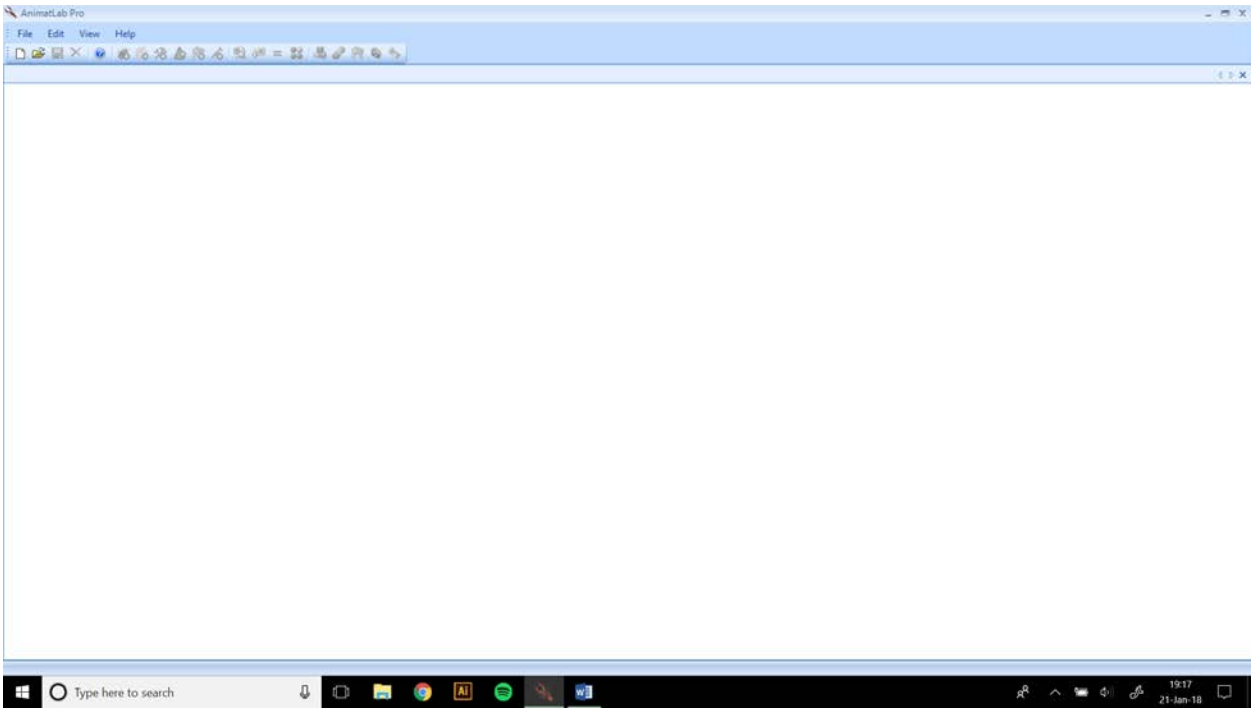
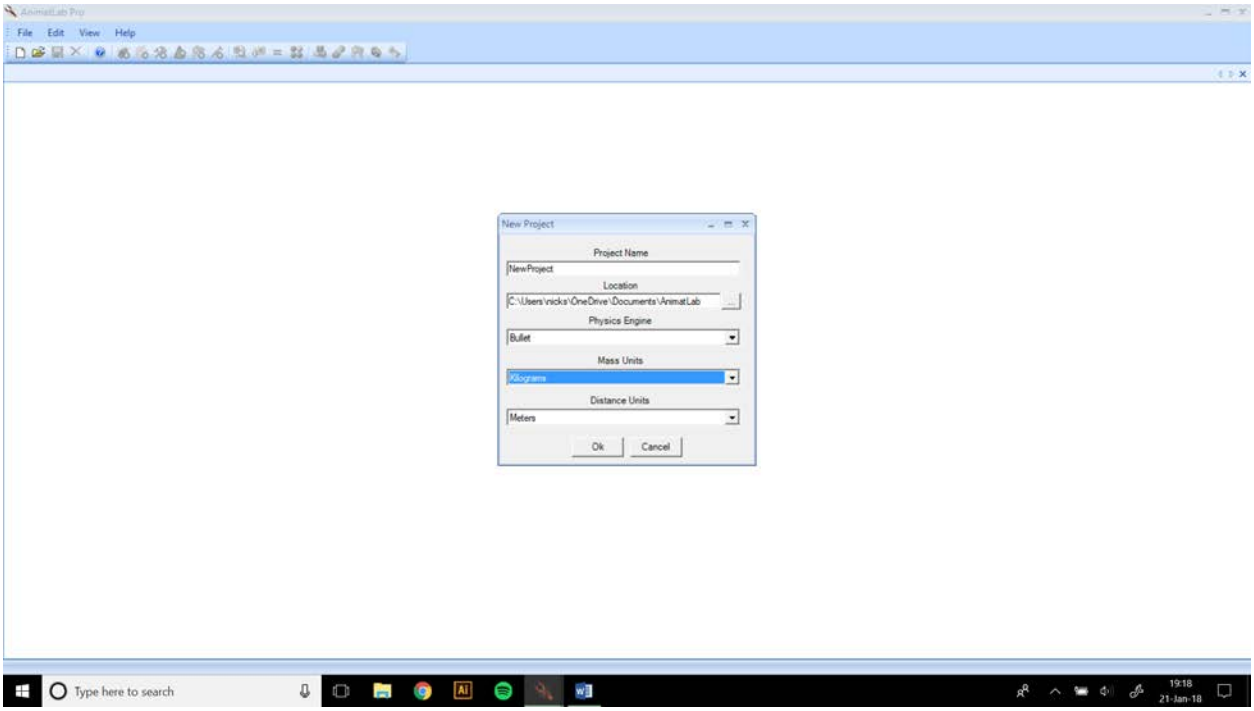
## The very basics.

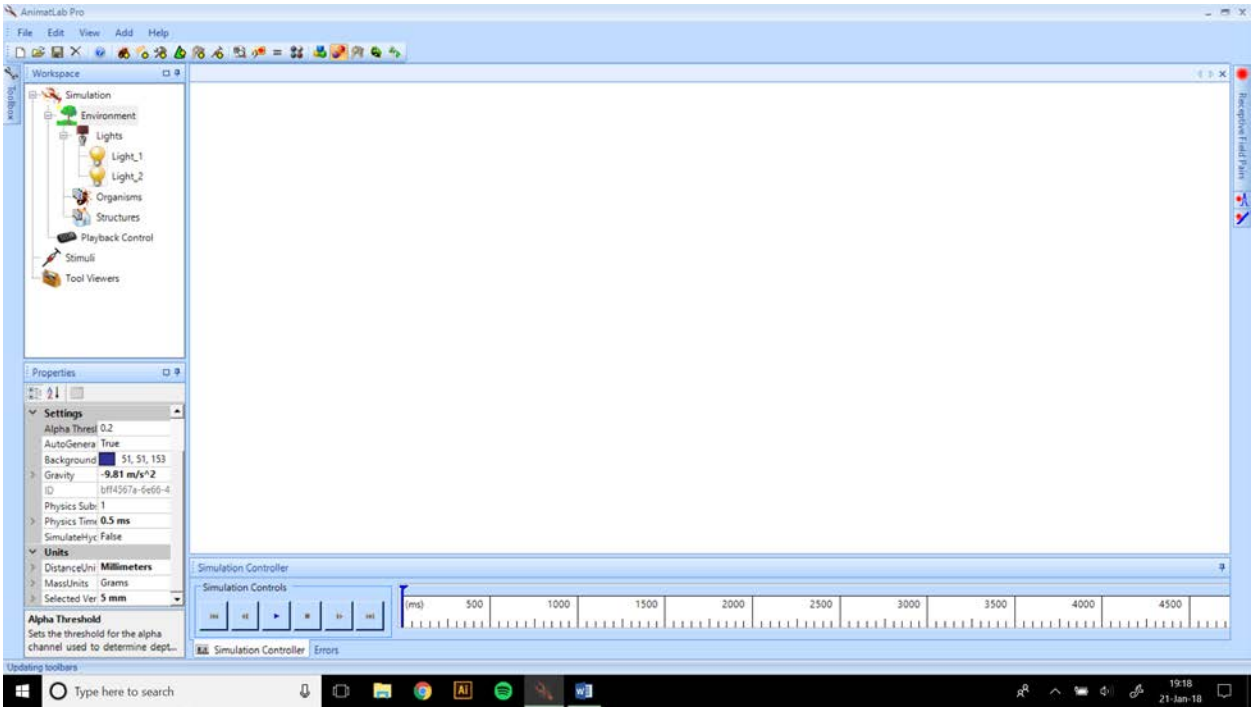
Here is what you see when you open AnimatLab 2:



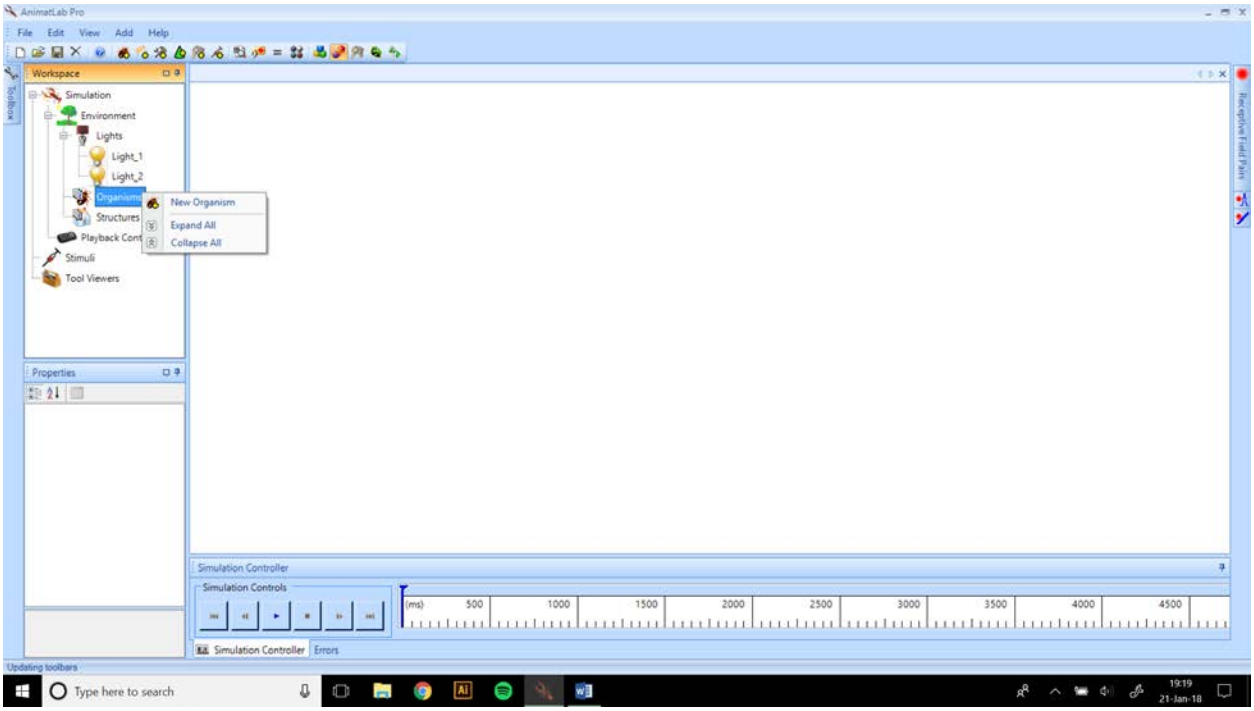

To get started, click File>New, and create a new project. Name it what you want, and point it to the folder where you plan to keep your projects. Keep the physics engine as Bullet. The last two parameters are important, the mass units and distance units. These determine how the physics engine is scaled. If you are modeling an insect, then grams and millimeters is a good choice. If you are modeling a robot, then kilograms and centimeters may be better. Click OK.



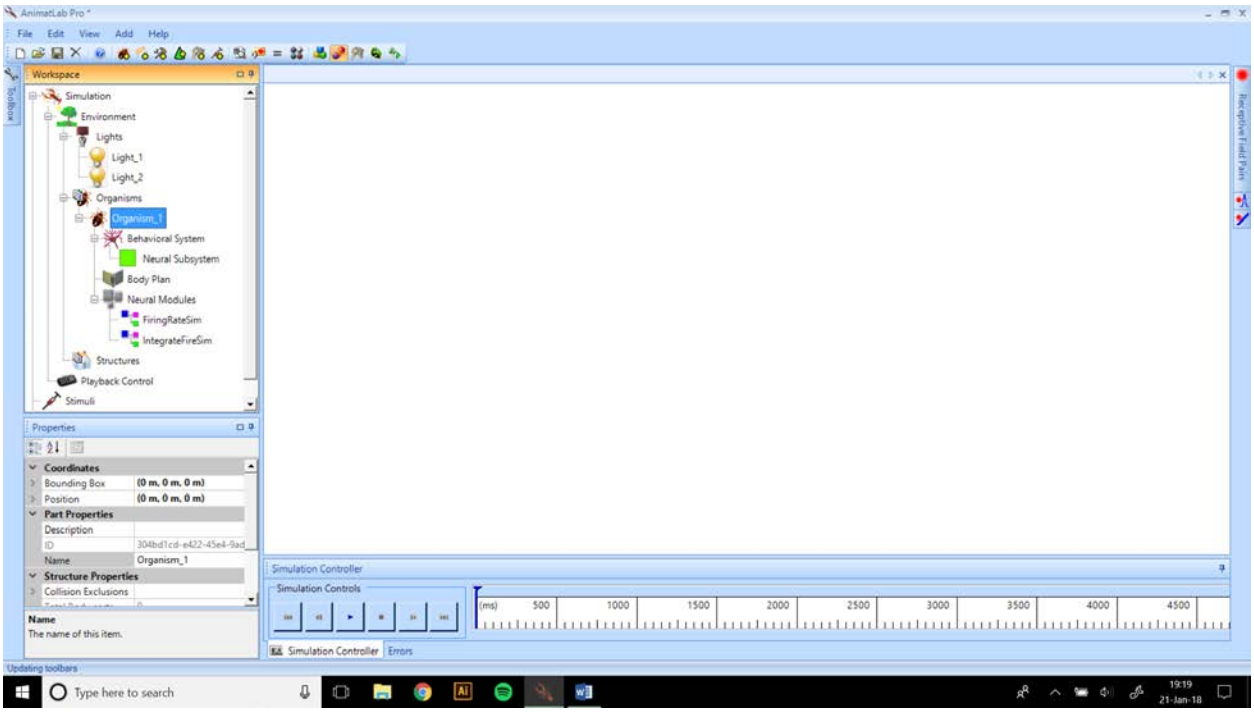

Menus and windows will pop up for your new project. On the left you'll see the Workspace. This window is a tree of all of the objects in the simulation. I may refer to it as "the tree" for short. Below the Workspace is the Properties window. It displays the properties of whatever is selected in the Tree. For example, you can see the "Environment" is selected in the tree, and for instance, "Gravity" is a property of the environment.



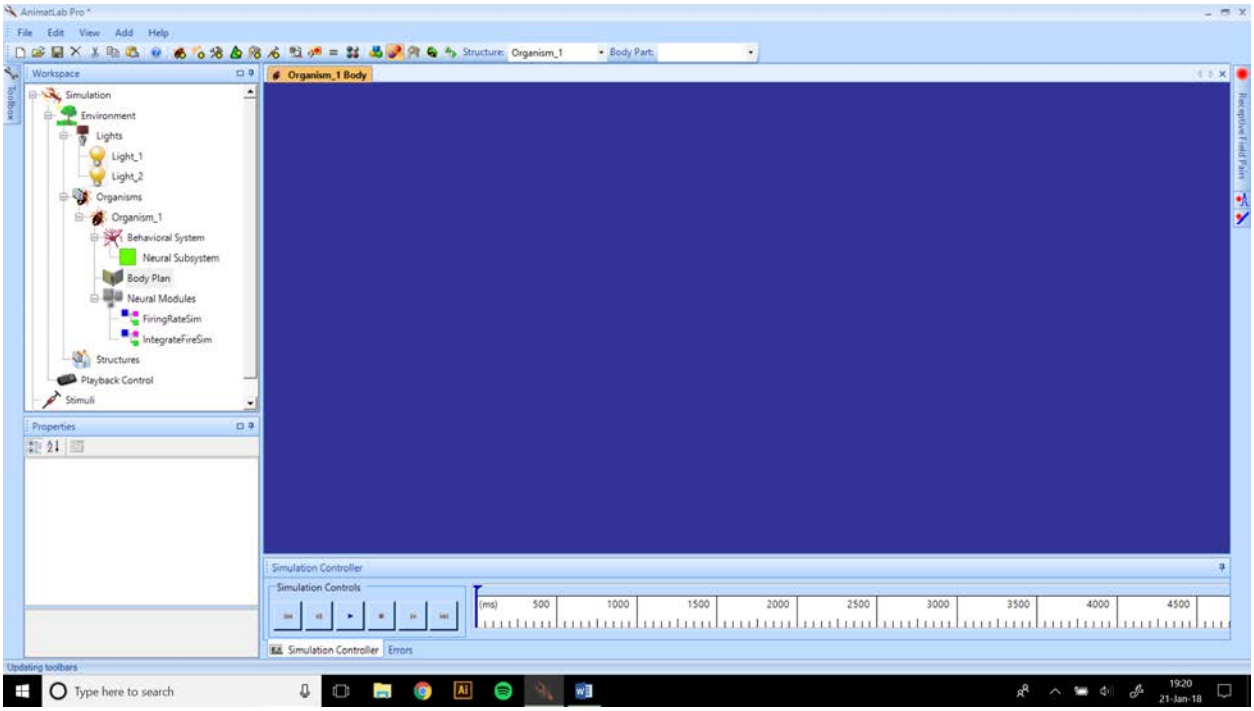

The Tree is boring until you add an Organism, so right-click on Organisms and click "New Organism".



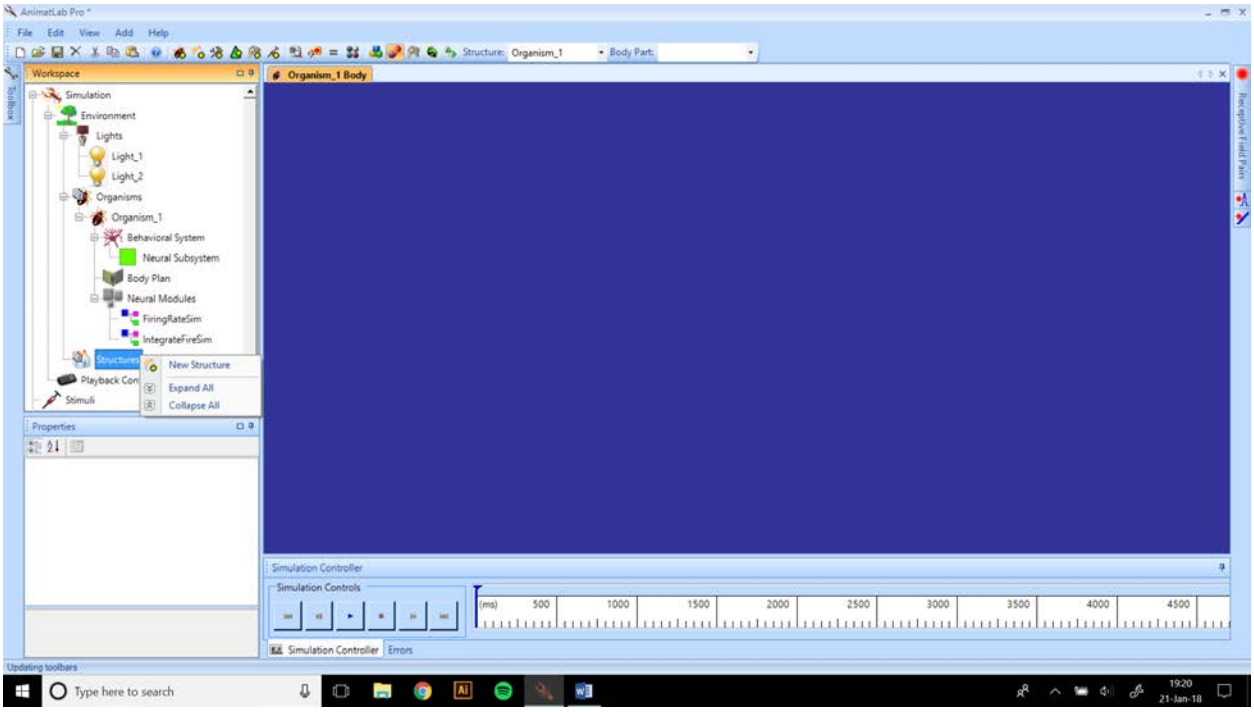

You can see that the Tree now has new branches related to Organism_1. These are the Behavioral System, Body Plan, and Neural Modules. We will discuss each of these in time.

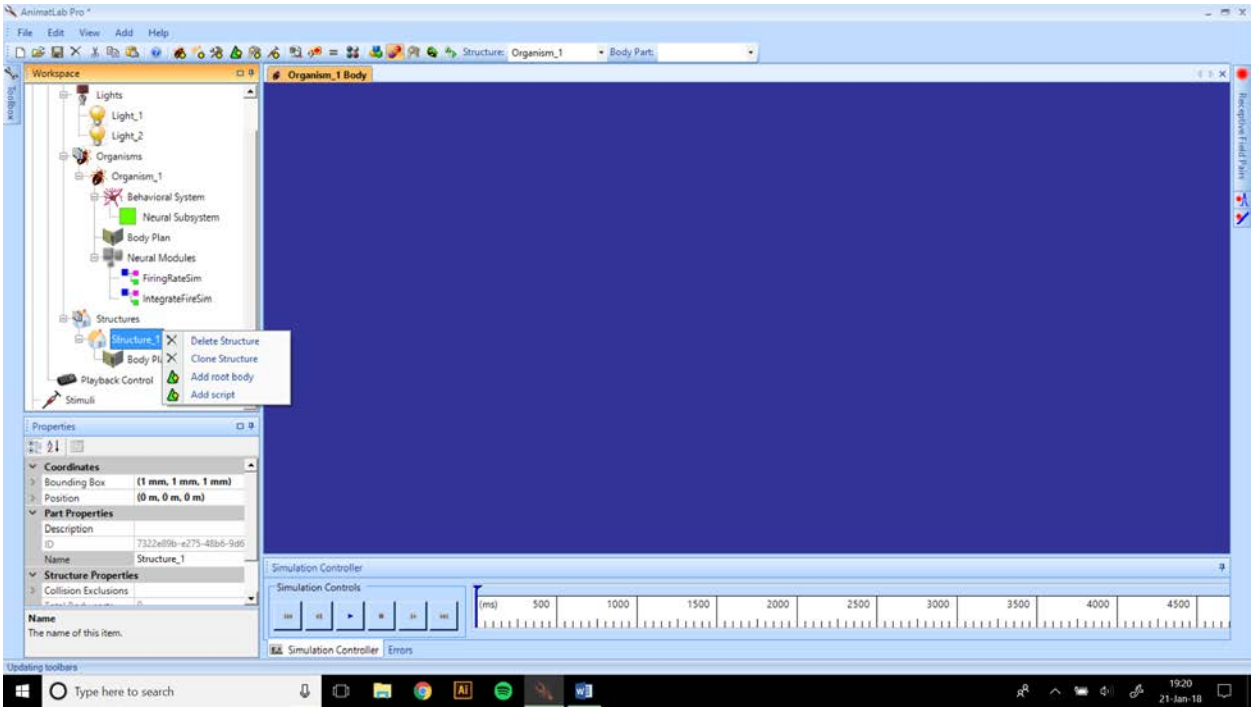



If we double-click on the Body Plan, then the main window shows us an empty universe. That's where the physics will happen. Let's add some things to the universe.

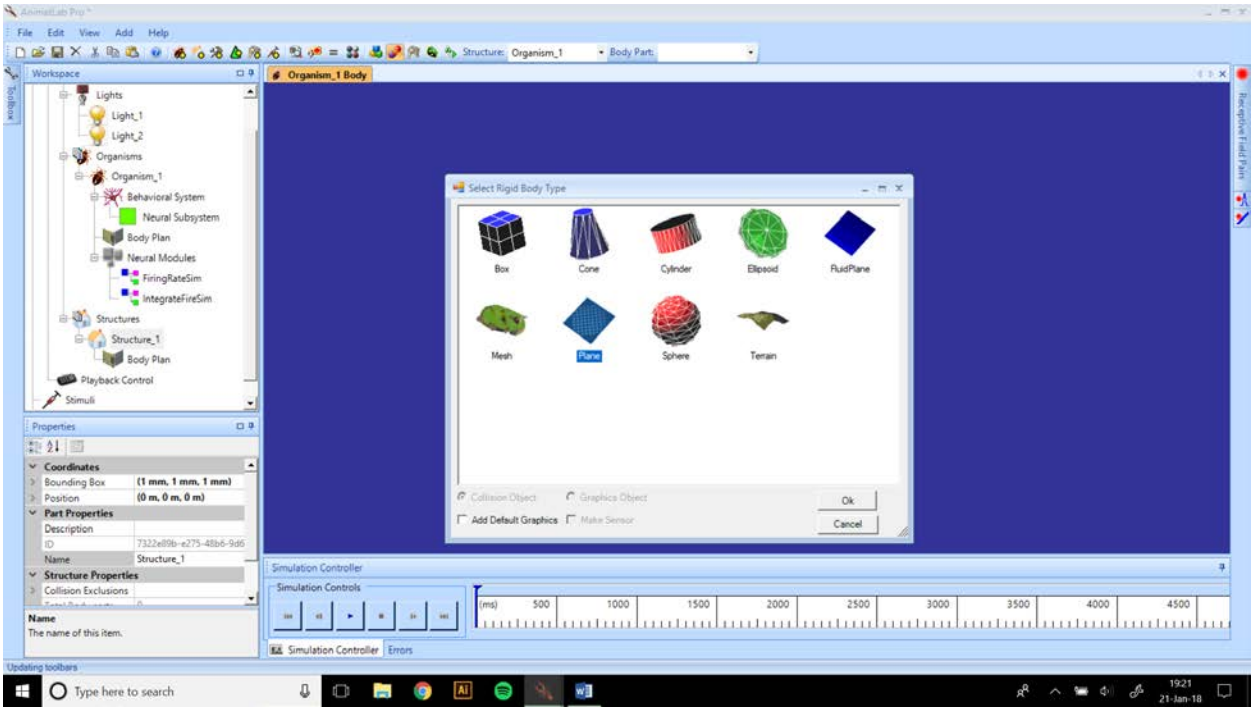



To give our organism something to stand on, right click on Structures in the Tree and add a New Structure.

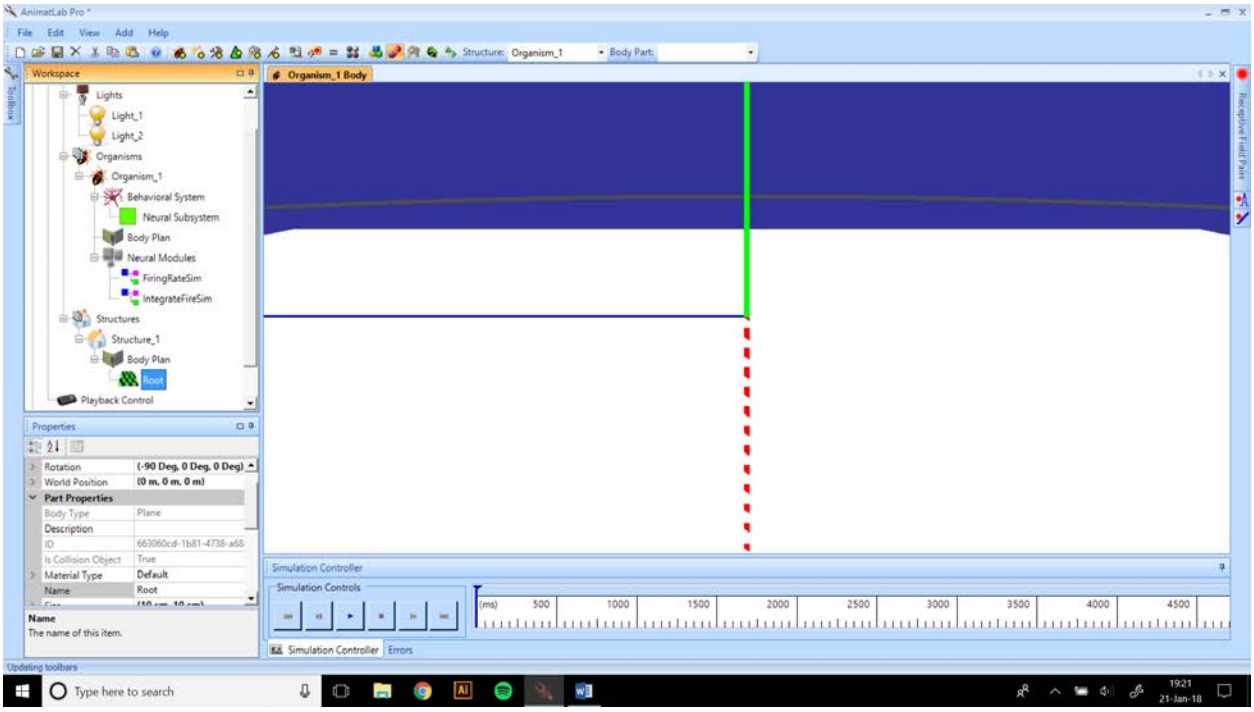




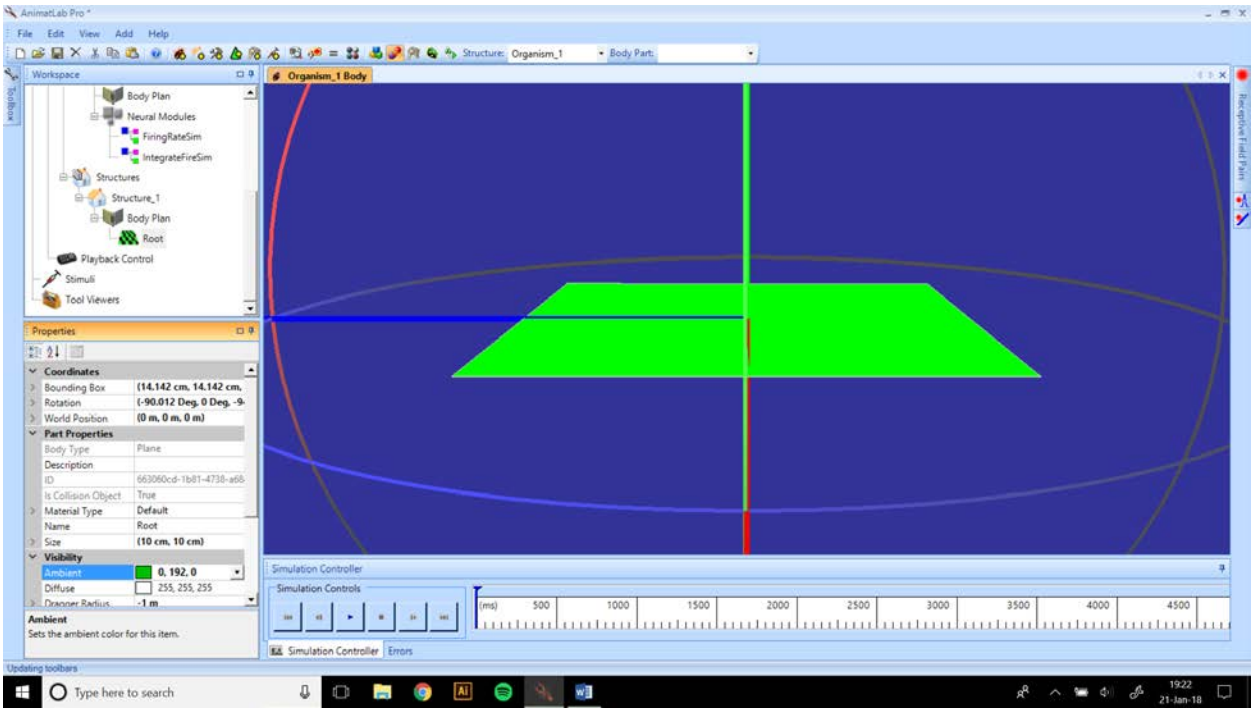

This will bring up a window to Select Rigid Body Type. This window comes up a lot when building your organism's animat and environment. Select Plane and click OK.



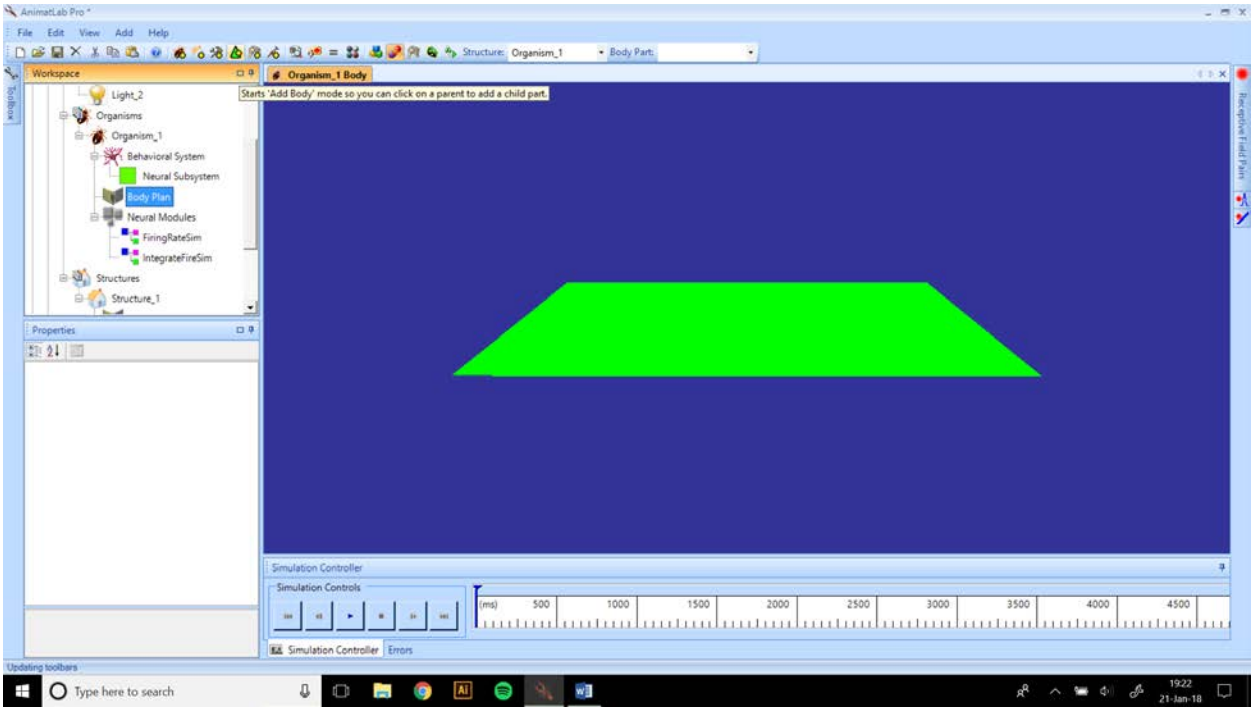
Now, your universe has a floor. We are zoomed in very close, but we can zoom out with the mouse. Click and hold right mouse, or use the scroll wheel to zoom; left mouse to rotate; and middle mouse to translate.



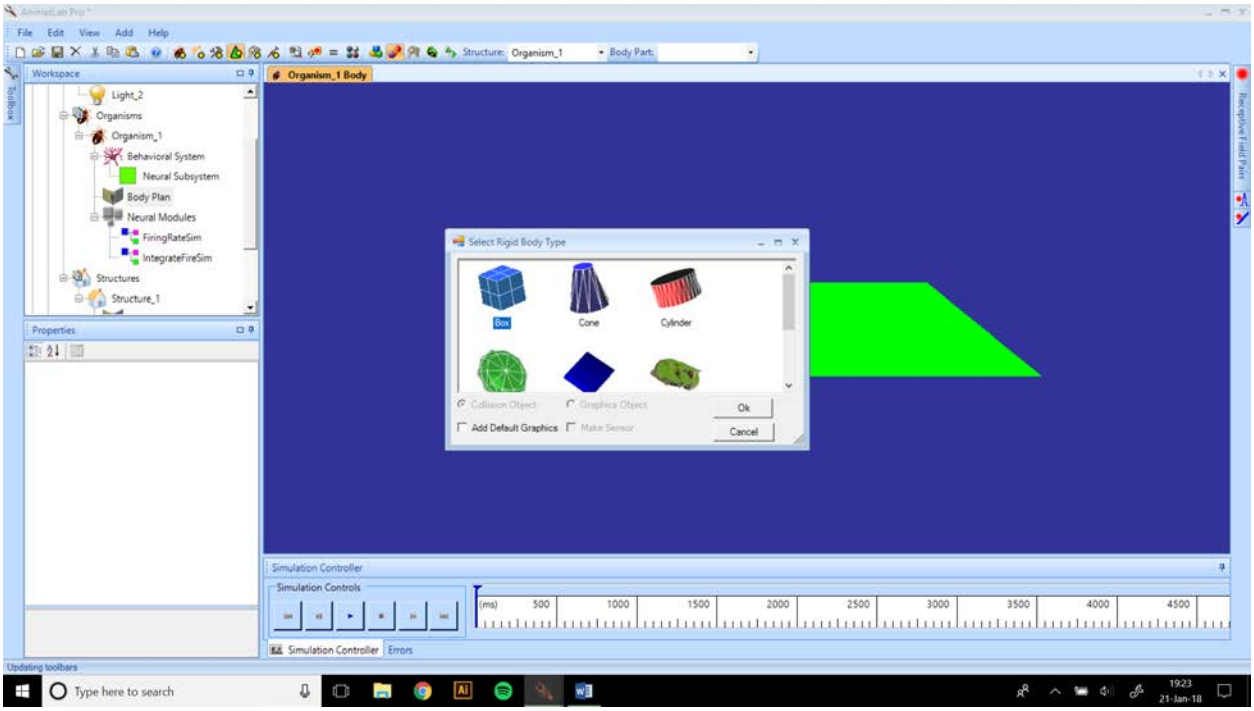

I'm going to change the color of the floor to a vibrant grassy green. Click on the Root body in the Tree, then look for Ambient in the Properties window. Change it to whatever color you want. This is the same for all mechanical bodies. Note that you can also change its transparency and visibility.



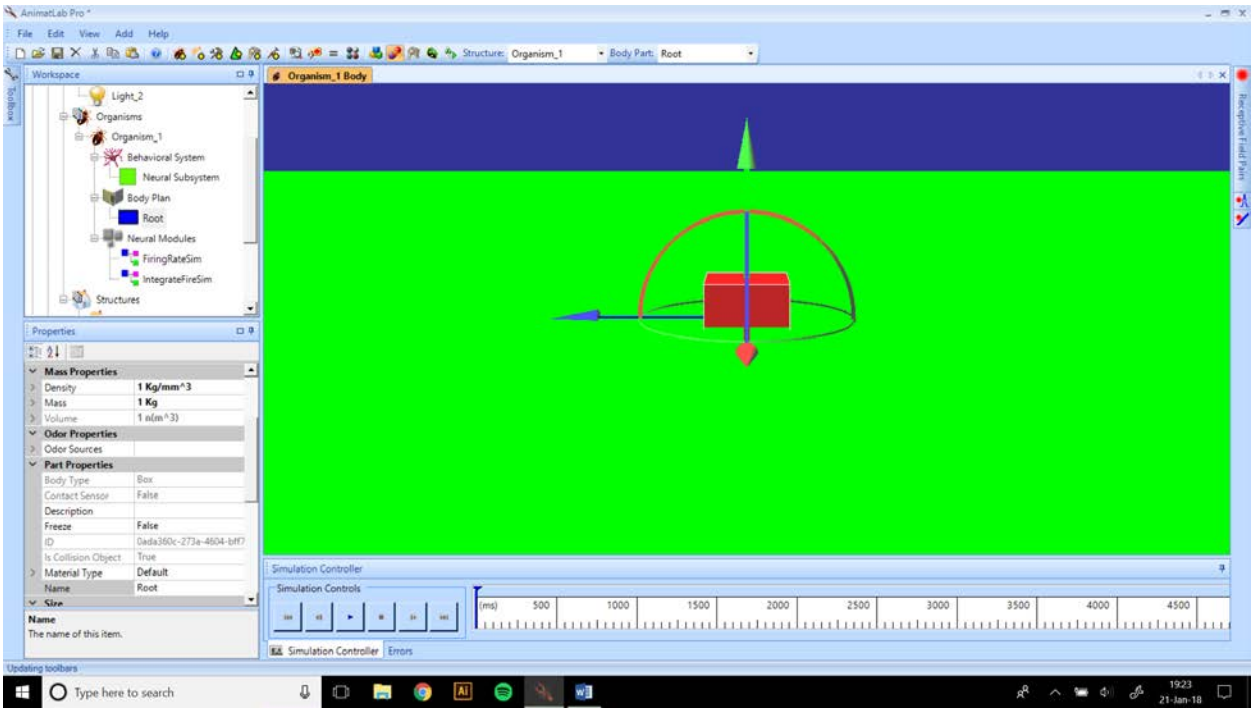
Now, let's add a body for our organism. Click on the Body Plan, and then click the Add Body button on the top toolbar (the icon is a green cone with a yellow plus).

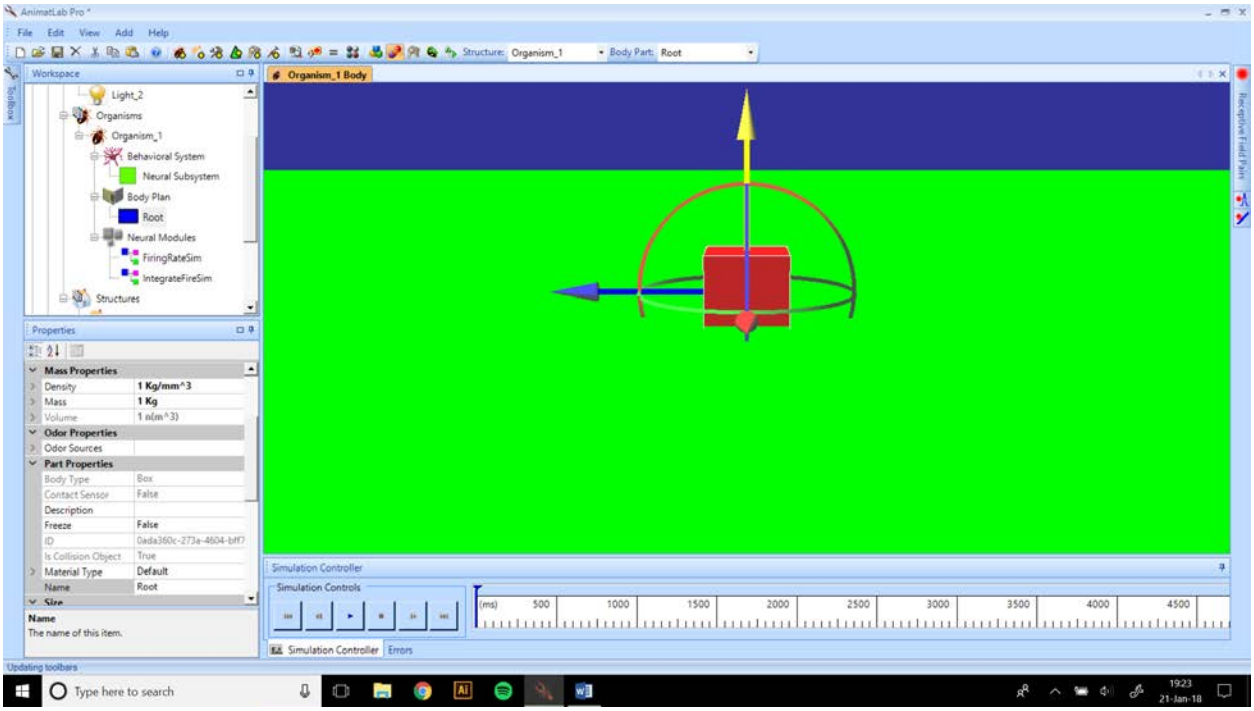



This again brings up the Select Rigid Body Type window. Select Box. There is also a checkbox below for "Add Default Graphics". I prefer to leave this unchecked, because otherwise it will create a second body that goes on top of the original body, and then I accidentally change the properties on that object instead of the object I want to change. When you uncheck Add Default Graphics, you will need to select Box again. I admit that it is silly. Click OK.

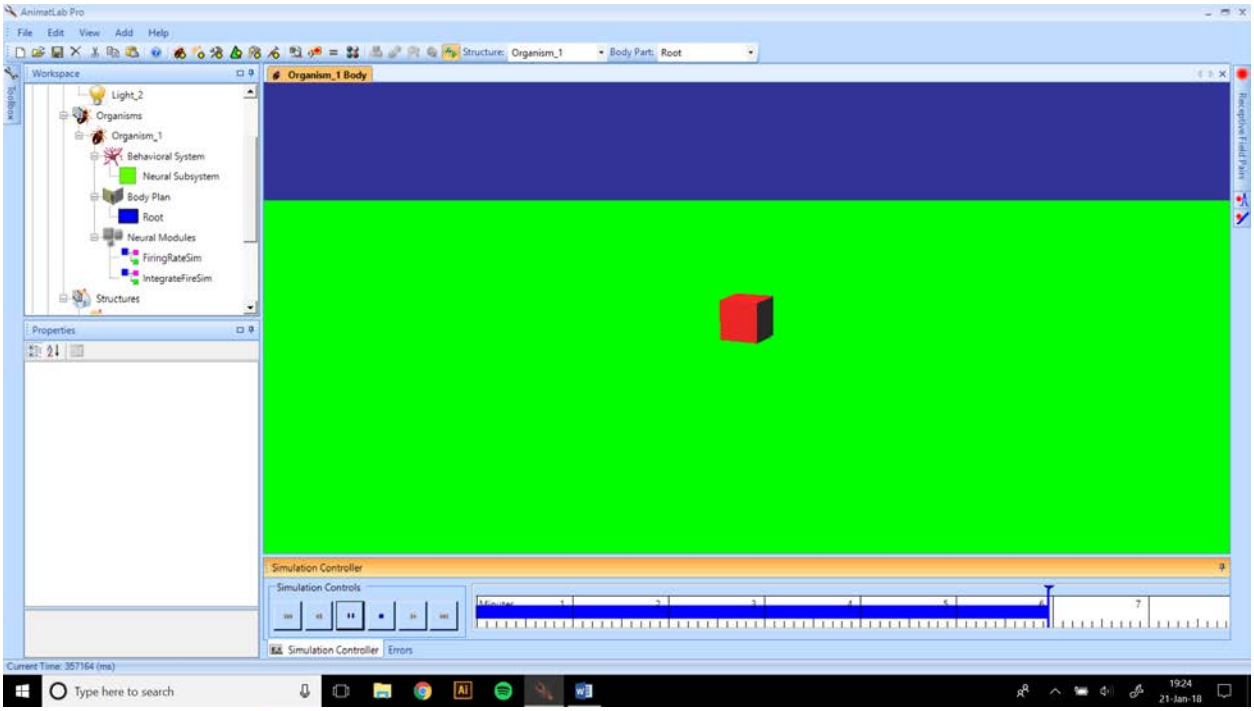



Your organism now has a body. It is initialized halfway through the floor, so do it a favor and lift it out of the floor by clicking on the green arrow. You can translate or rotate any body by clicking and dragging on these axes. For more precise placement, you can enter these values directly into the body's Properties window.

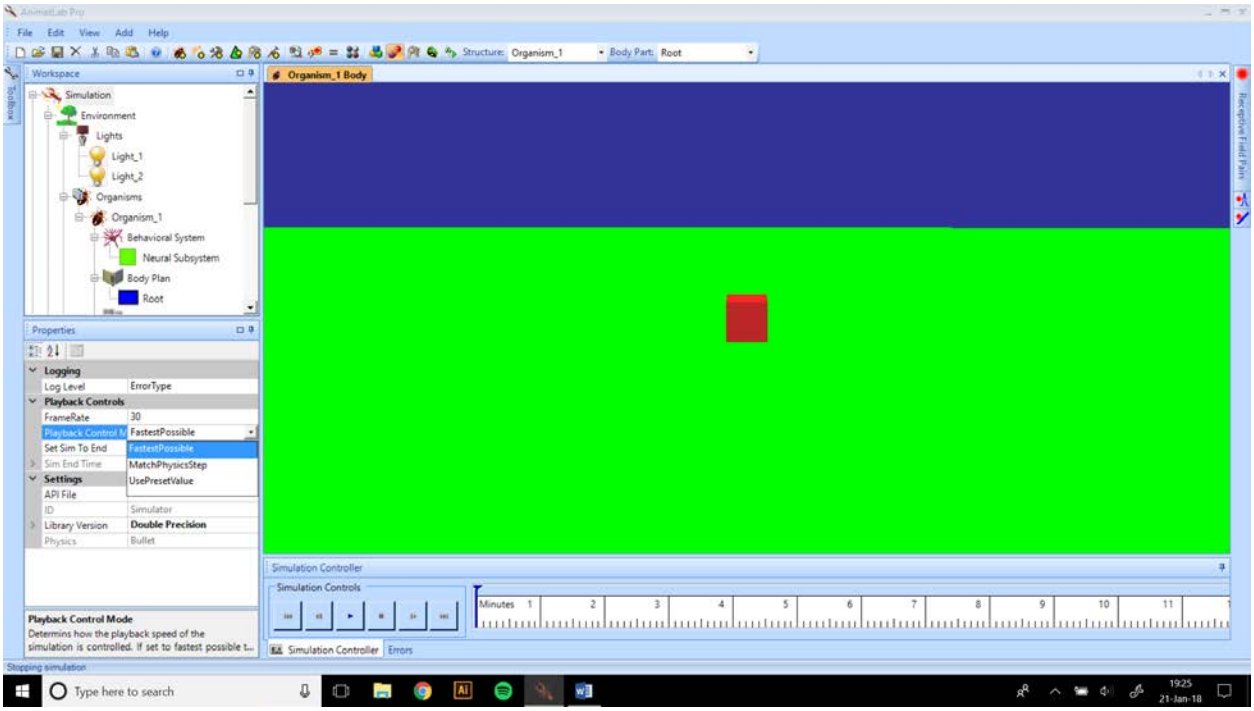




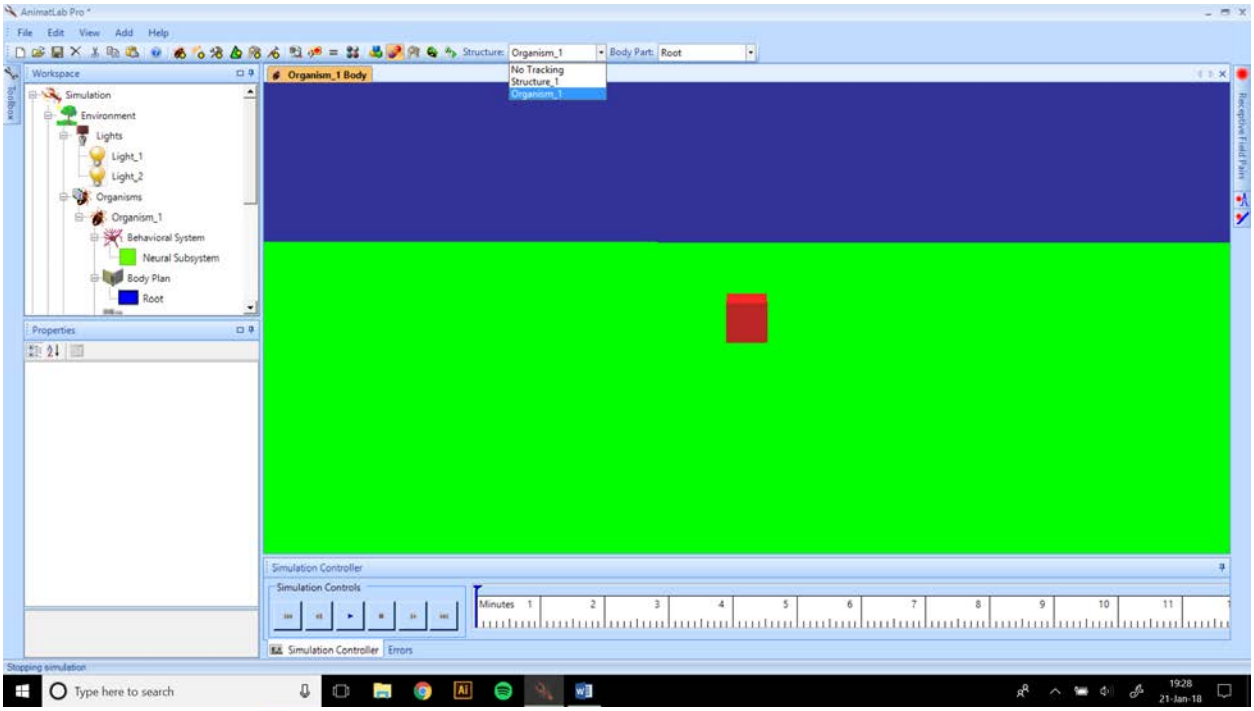

Now your organism has a body floating in the air. You can run the physics simulation by clicking the play (triangle) button in the Simulation Controller window. When you do this, some weird things will happen. The box may not appear to go anywhere. That is because of two reasons.



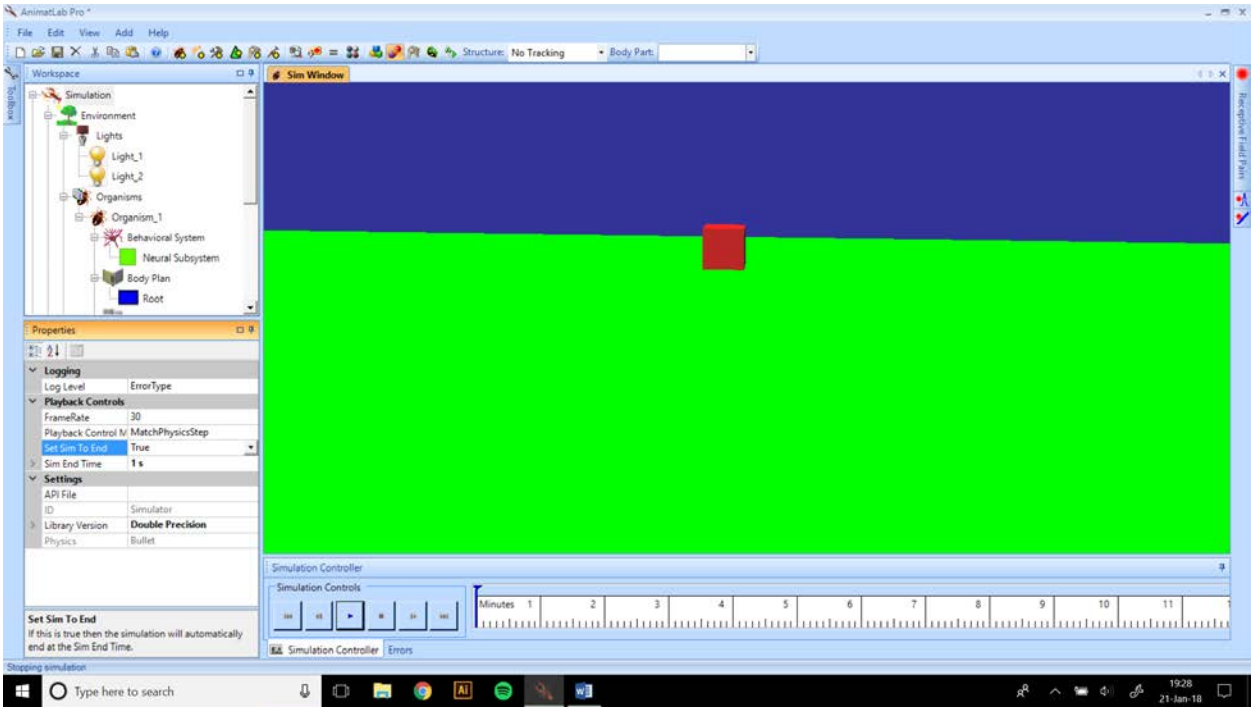

First, you need to slow down the simulation. Go to the Simulation's properties and change the playback control from "Fastest Possible" to "Match Physics Step". Now, if it runs faster than real time, it will slow itself down.



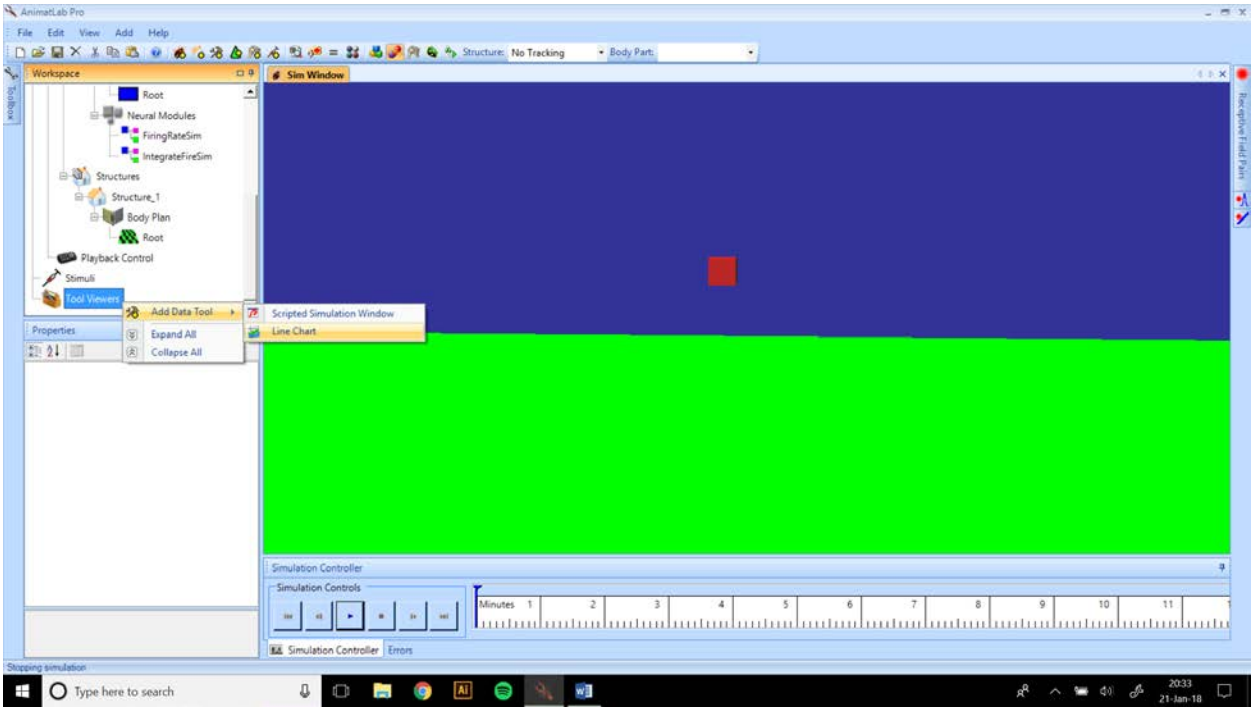

Second, you need to turn off body tracking. The simulator can track any body you like during the simulation. That can be helpful later, but for now, click the menu on top and change Structure from “Organism_1” to “No Tracking”.



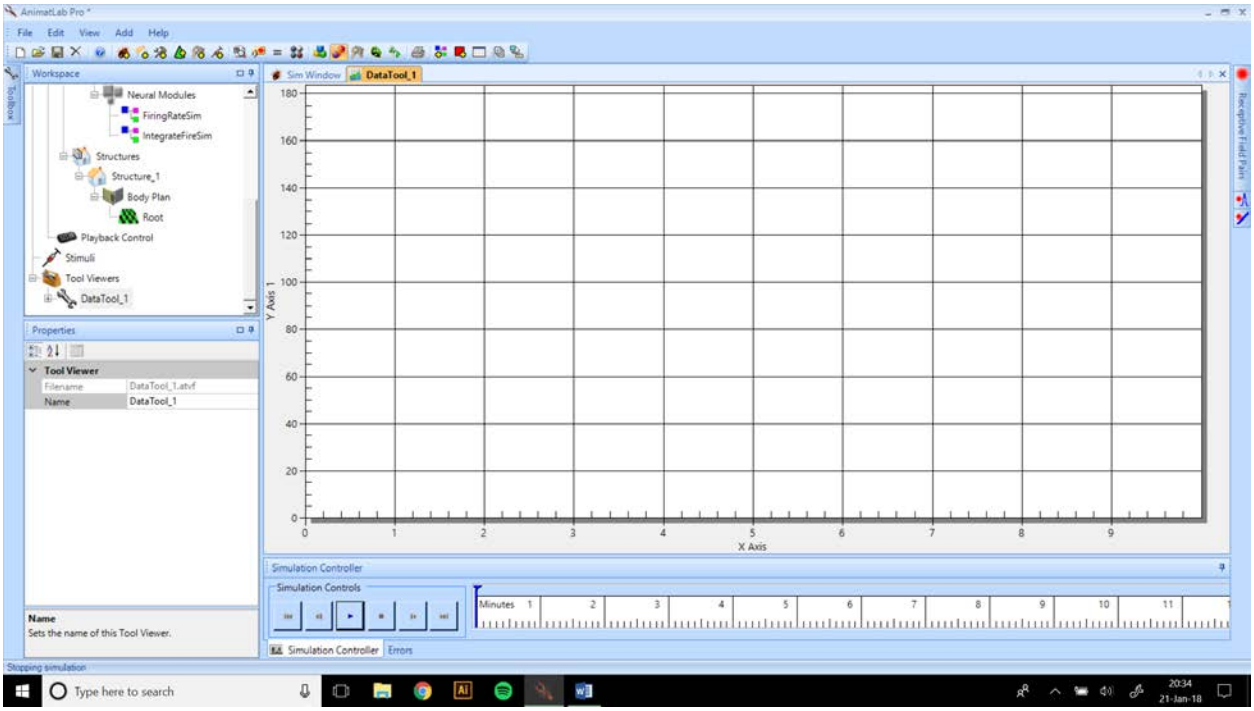

Now when you run the simulation, you should see the block fall to the floor. However, you will also notice that the simulation goes on and on and on. So, click on Simulation again, and “Set Sim To End” to true. Pro tip: just double click “Set Sim to End” to quickly toggle Boolean values.



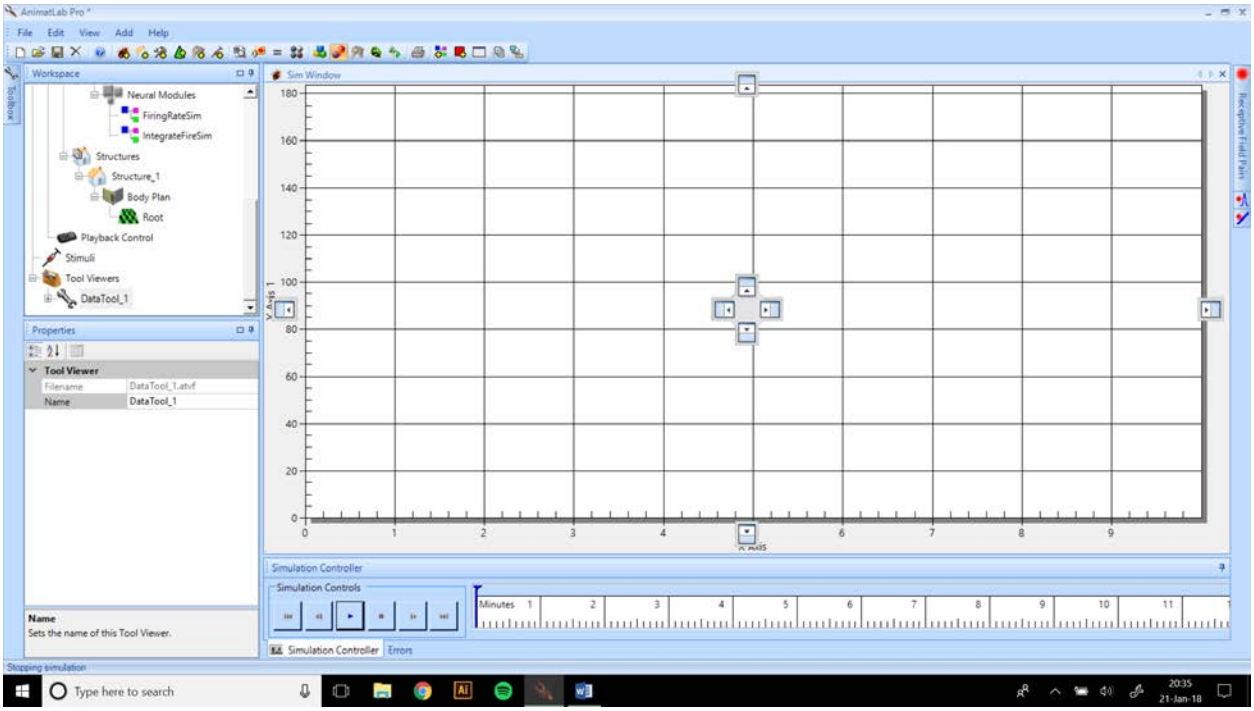

Ok, so now the block falls to the floor and we can watch it. But visual inspection is not sufficient for data collection. So, scroll to the bottom of the Tree, right click on Tool Viewers, then click on Add Data Tool>Line Chart.



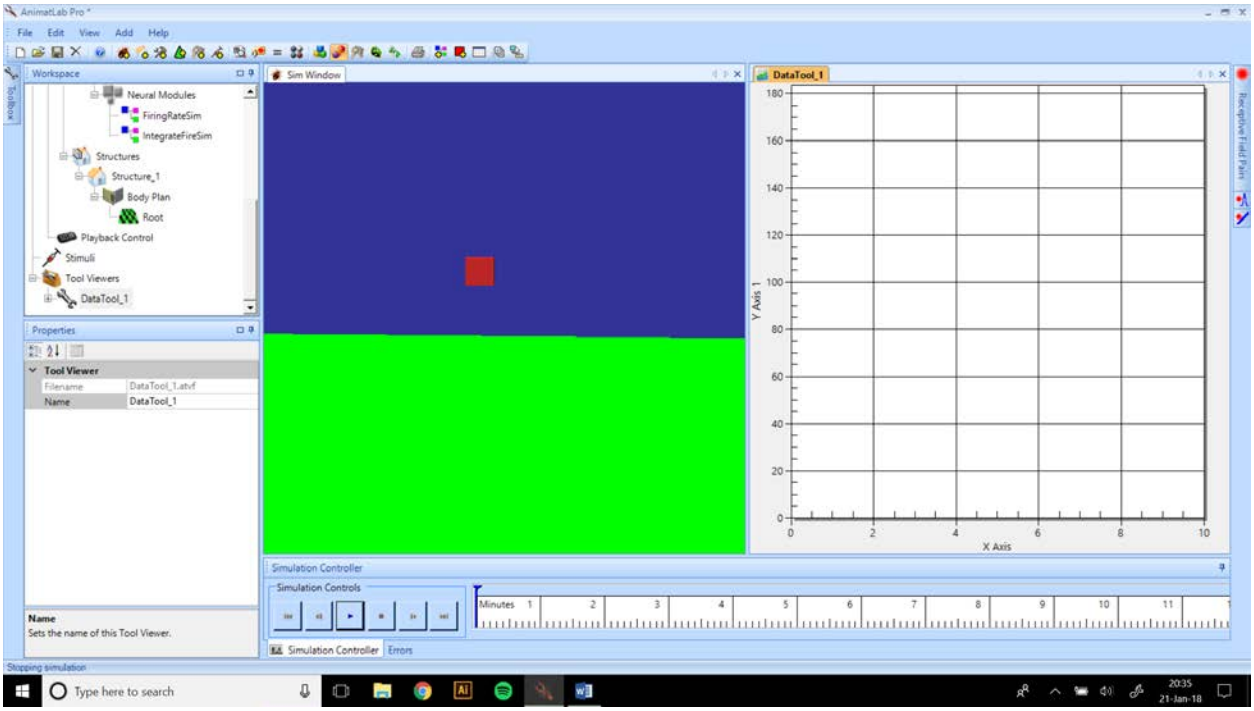

Now, the screen should be filled by a graph. Let's dock it beside the Sim Window. Click and hold on the tab on top:



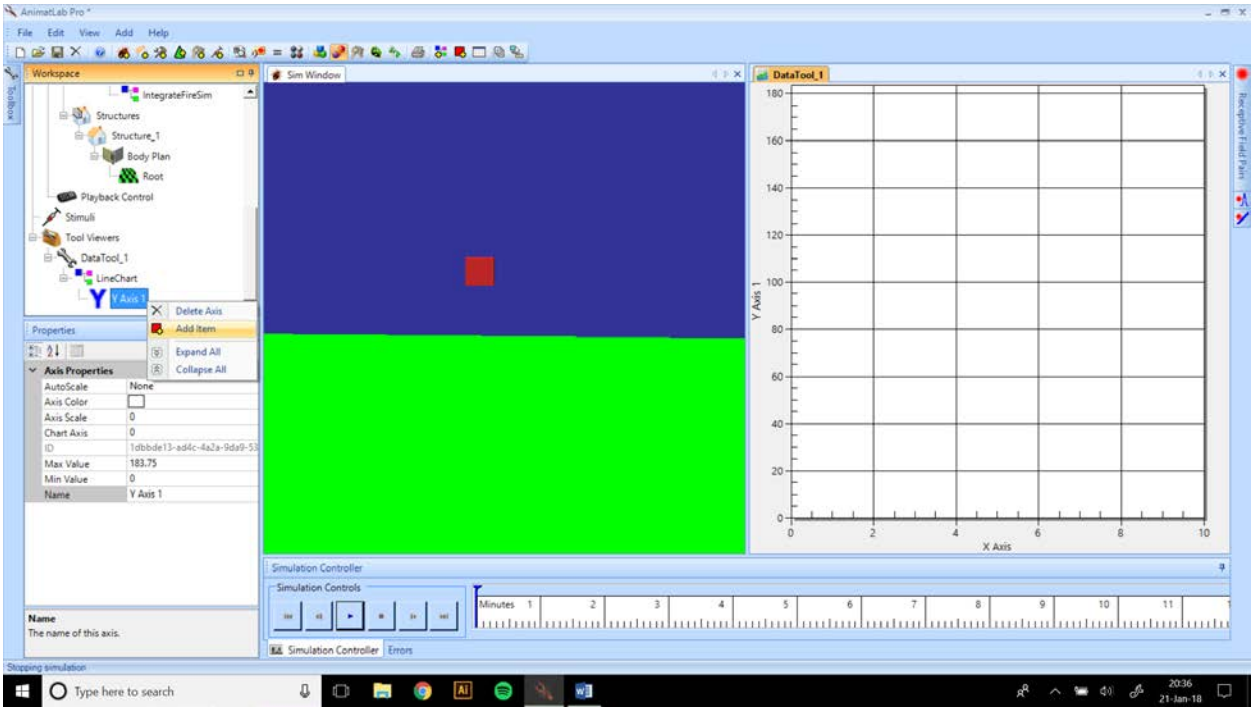

Then, drag the tab elsewhere on the screen. Options will appear:



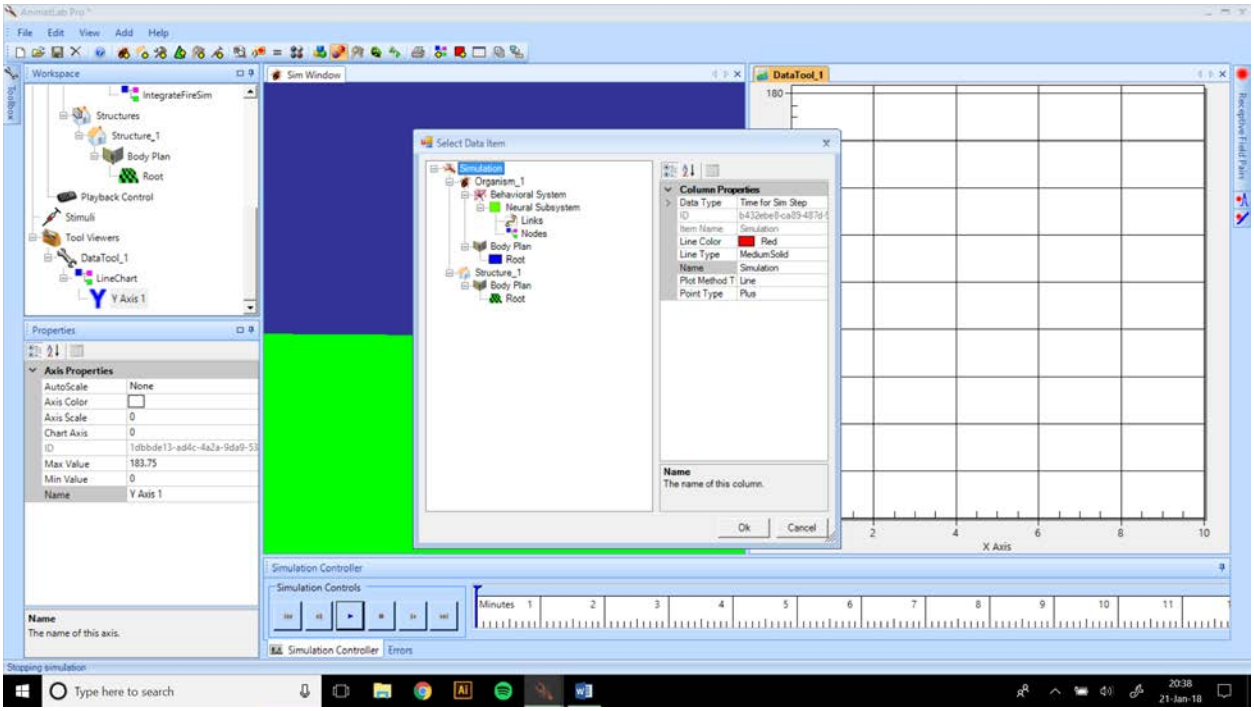

I'm going to drop it on the right. Hover over the rightmost icon, and release to dock the window. You can do this with any window in the interface, so feel free to customize as you like.



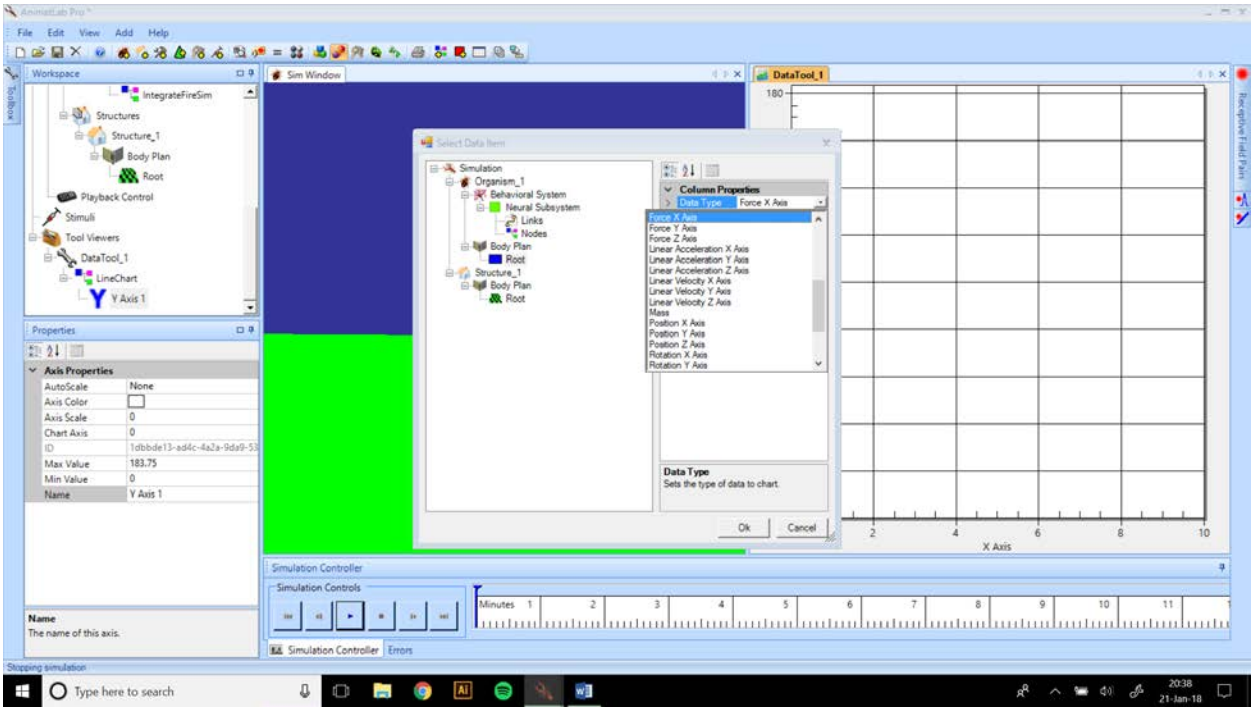

Now, expand the DataTool_1 Tree, and right-click on Axis 1, then select “Add Item”.



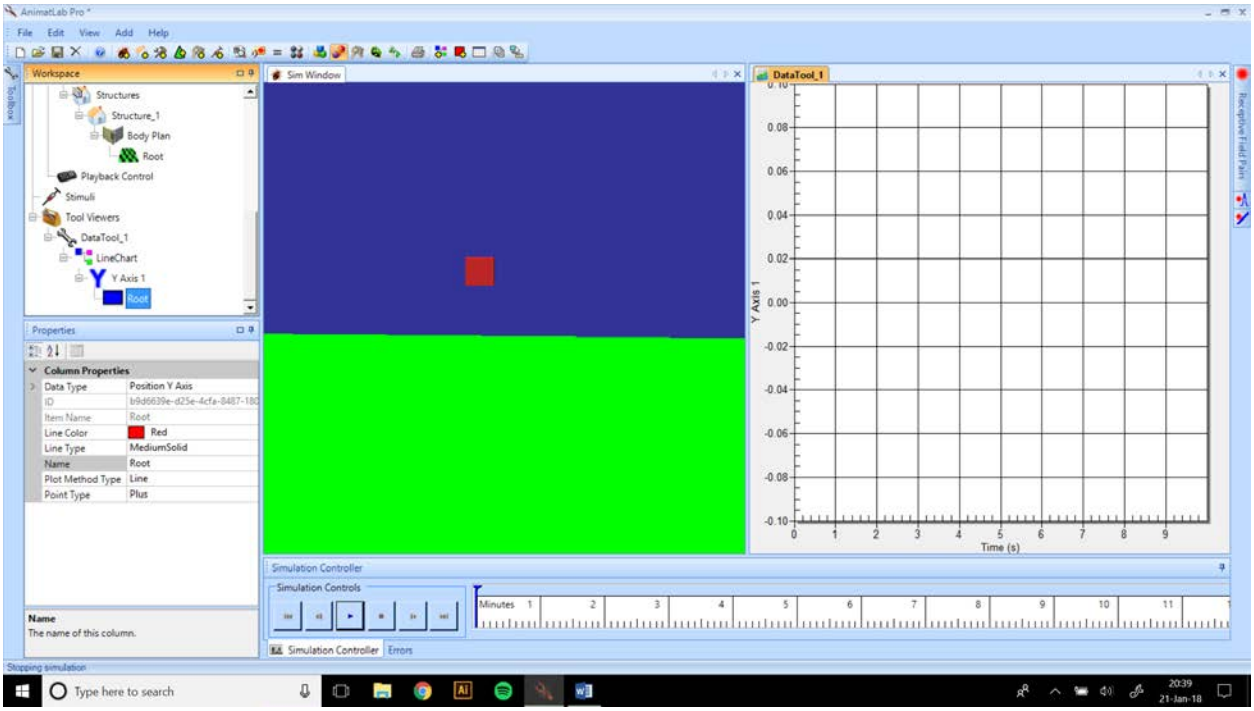

This will bring up another window, which holds a copy of the Tree.



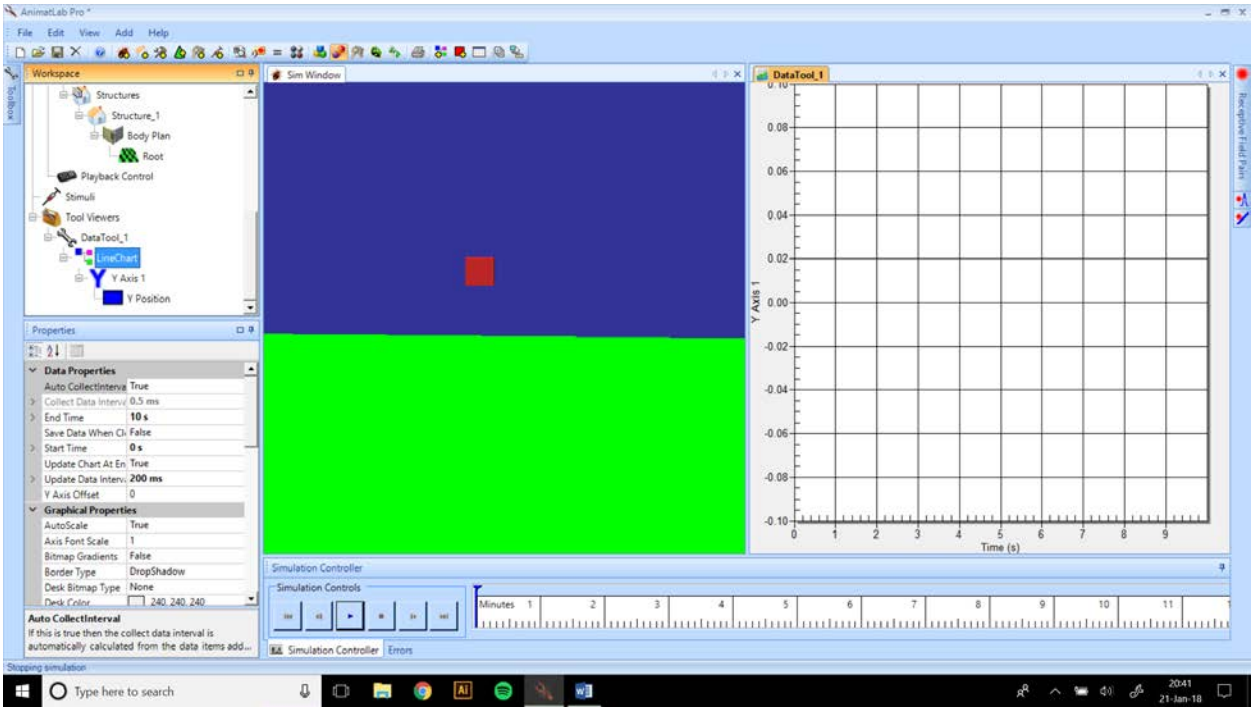
Now, you can select any value from the simulation to plot. Let's select Organism_1>Body Plan>Root, and then Position Y Axis on the "Data Type" drop down:



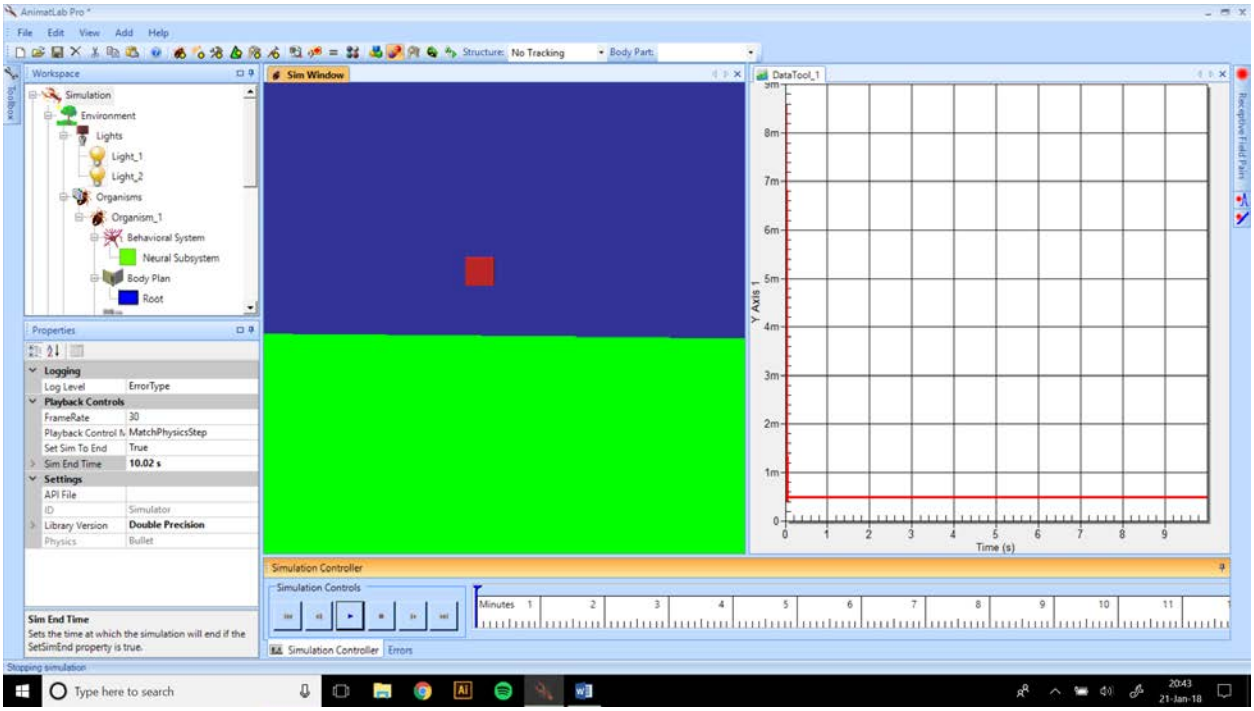
The Tree now has a new data column under Axis 1:



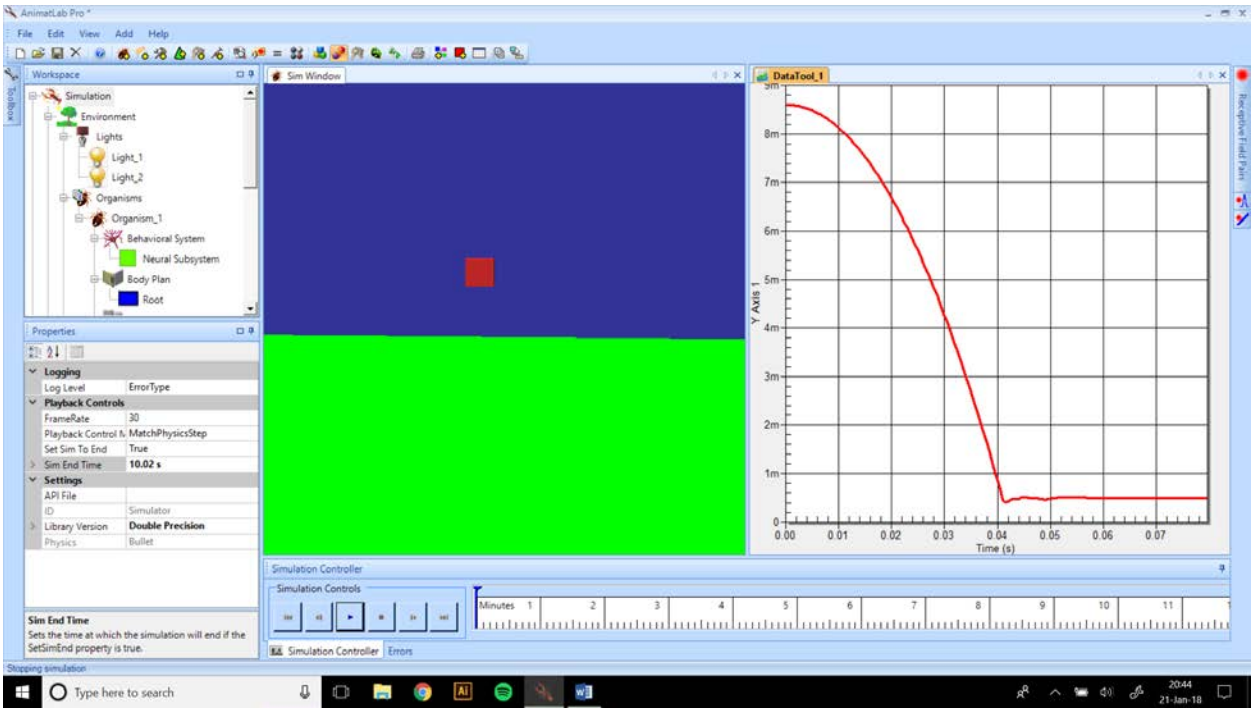

Let's give it a descriptive name. Change the Name from "Root" to "Y position". Now, run the simulation again. Unfortunately, our plot is still blank. To see why, click on DataTool_1>LineChart in the Tree. The End Time is 10 s:



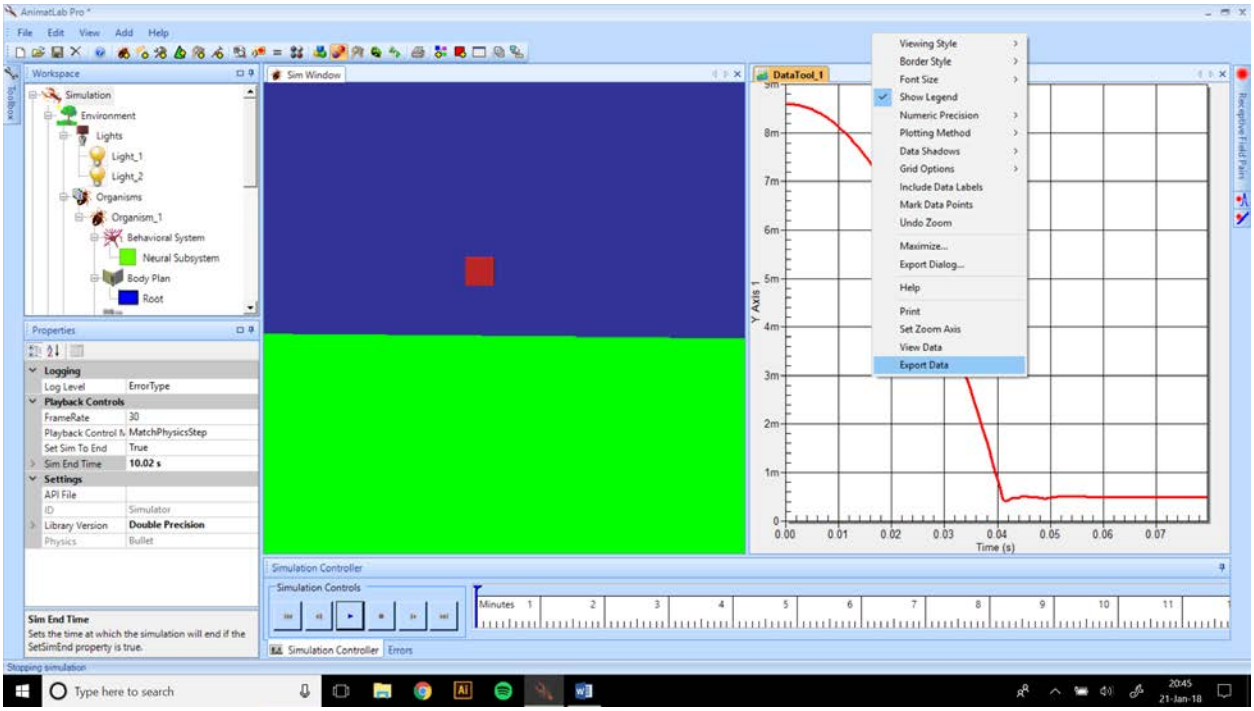

Since we set the simulation to only run for 1 s, the chart doesn't plot. So go back to Simulation on the Tree and set the End Time to anything *strictly greater* than 10 s. I chose 10.02 s. Now, when I run the simulation, data is drawn in the plot:



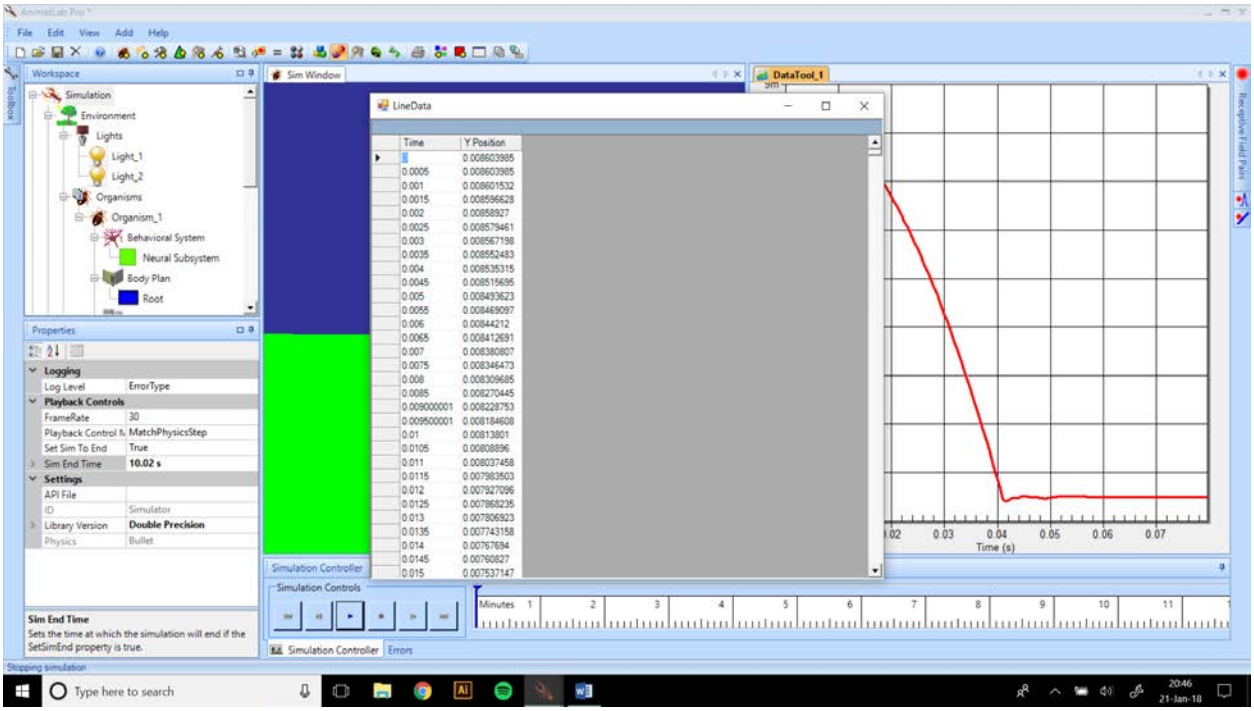

However, the X axis scale isn't very useful. We can zoom along the x axis by clicking at one edge of our desired window, dragging to the other edge, and then releasing. The result is a more useful plot:



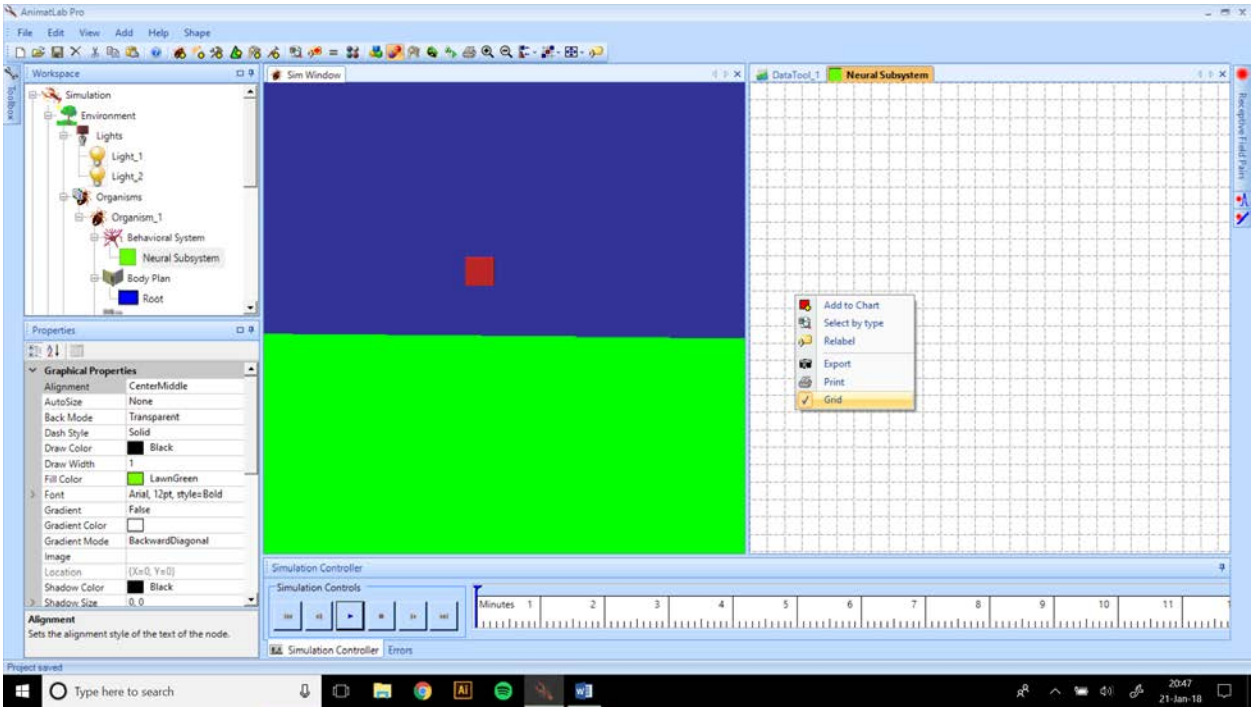

If we want to use this data for another purpose, we can right click on the plot, and then select "Export Data":



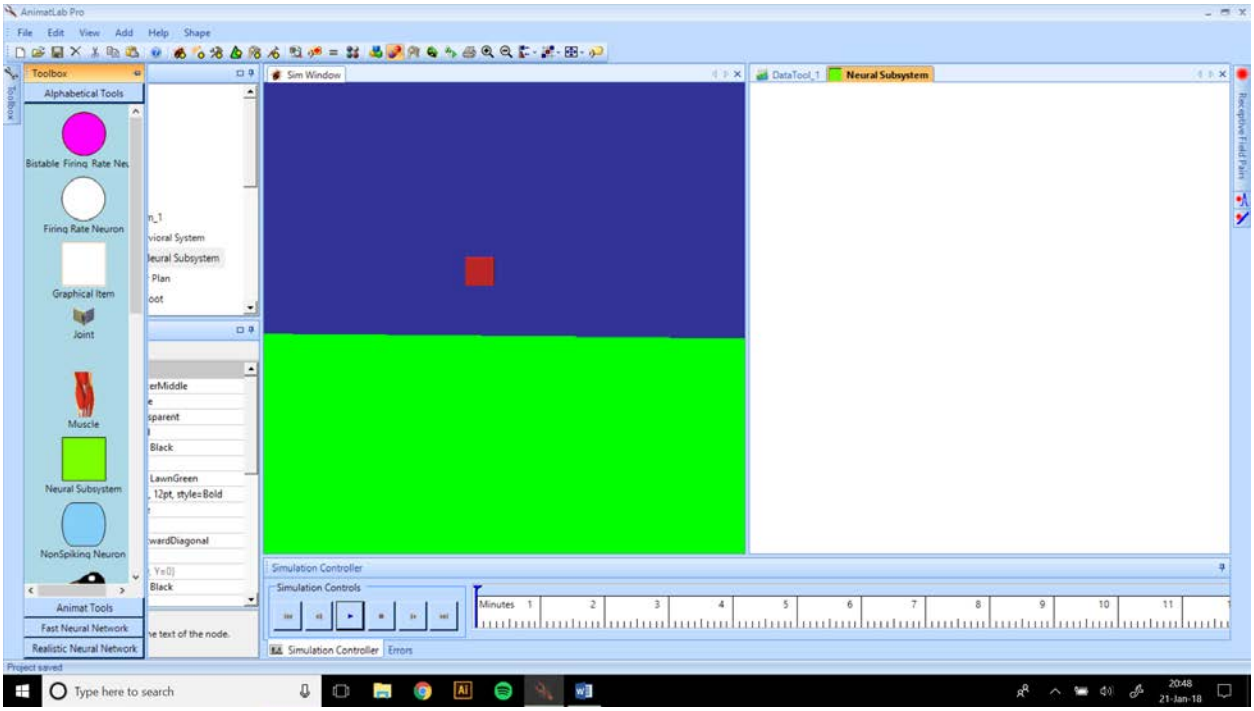

This will save a tab delineated file named DataTool_1.txt in the folder in which this project resides. If you don't want to export the data, but you want to take a closer look at specific values, you can click "View Data", which enables you to look at the data as a matrix:

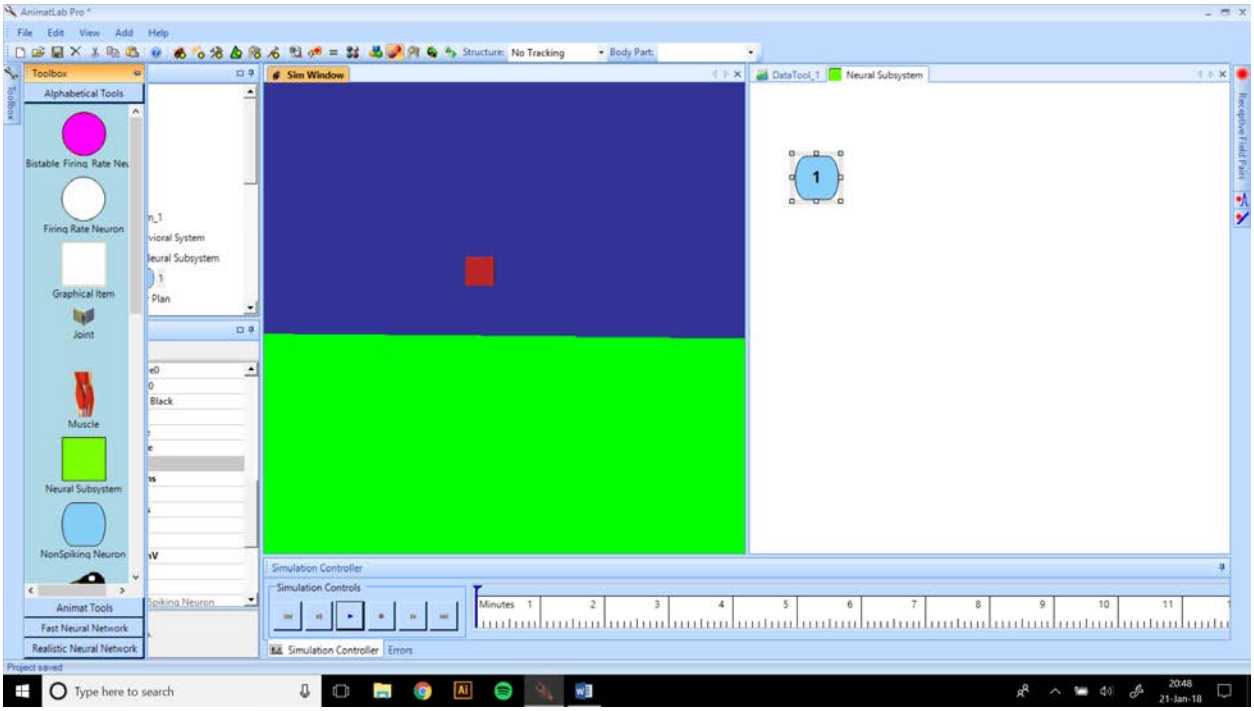



The point of AnimatLab is to enable neural and mechanical systems to interact with one another. Double click the Neural Subsystem in the Tree to open the window. Turn off the grid by right clicking on it, and unchecking the Grid option.

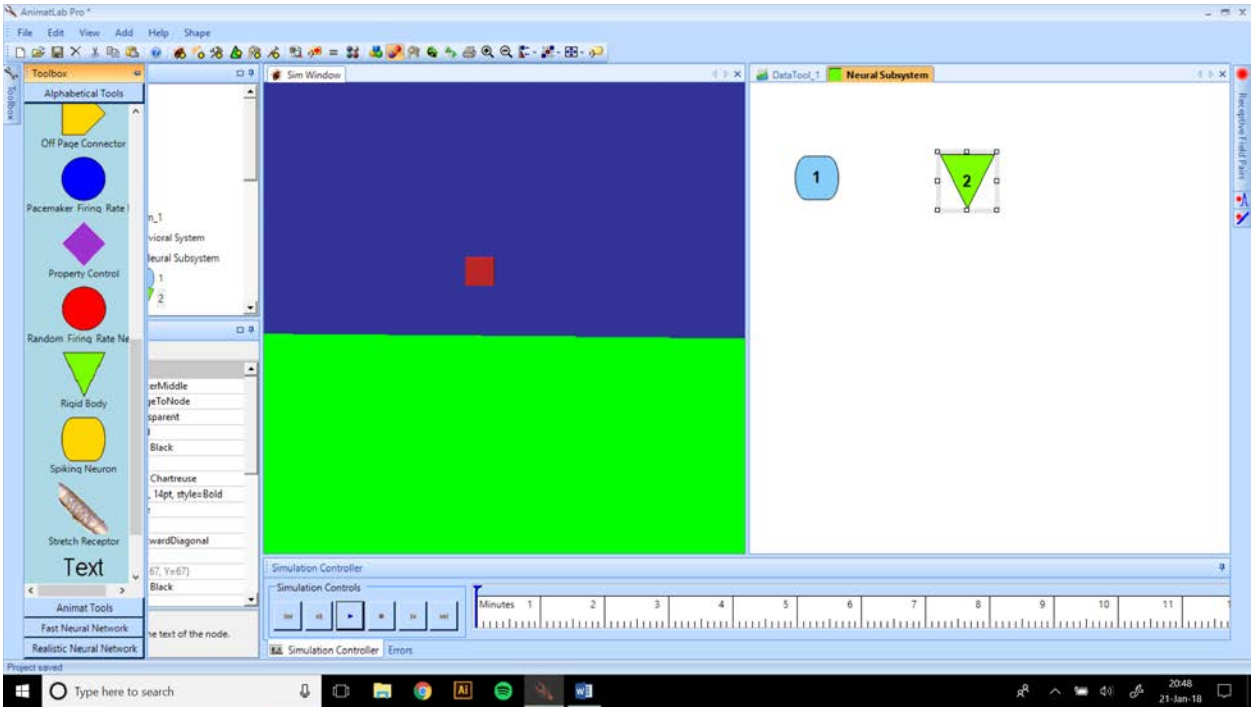



Next, click on the Toolbox, which is docked on the left. This will reveal a number of components. Click on the blue NonSpiking Neuron and drag it into the Neural Subsystem. That will create a new neuron. It will number them as you add them.

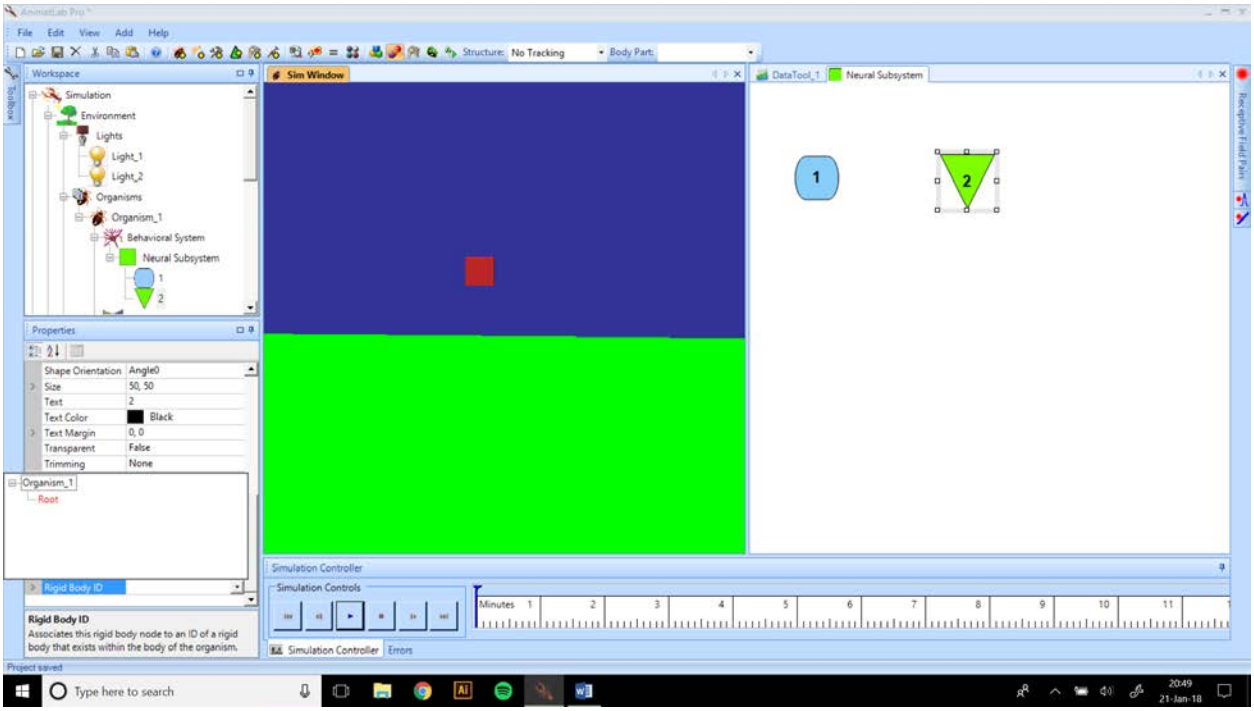




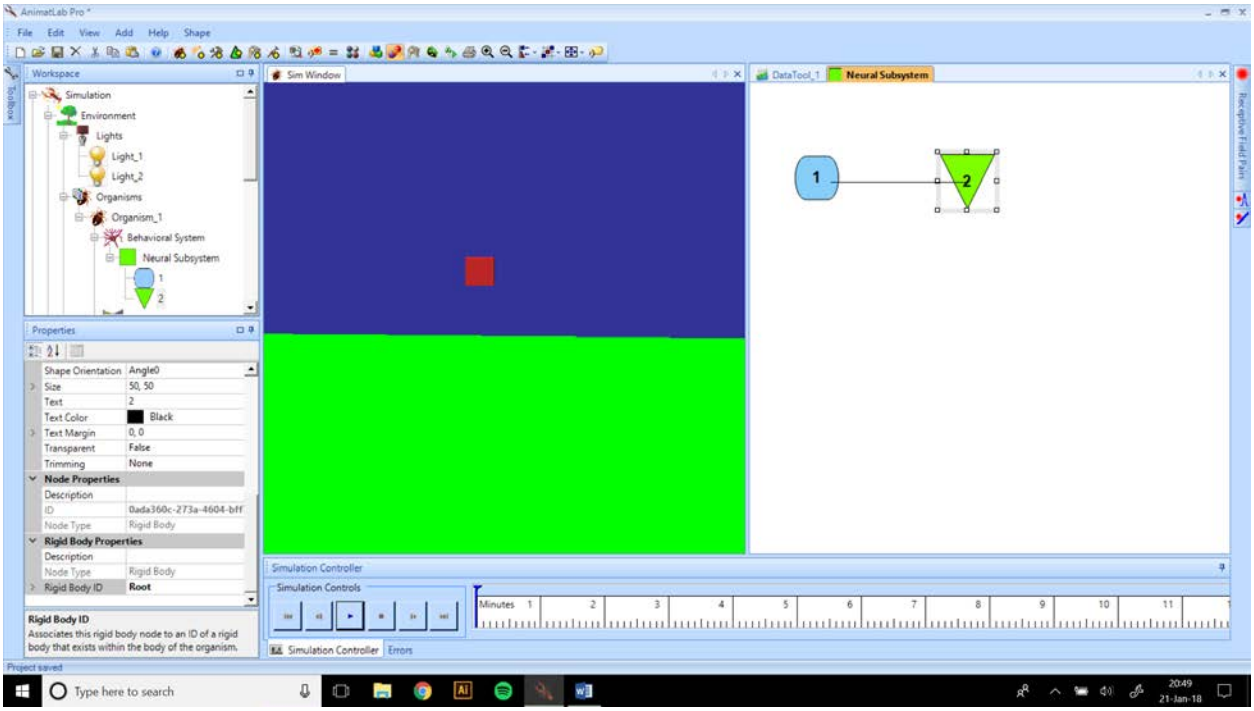

Now, let's make this neuron's activity correlate to the acceleration of the block, like an accelerometer. First, we need to add the block to the neural subsystem. Find the Rigid Body object, and drag that into the Neural Subsystem.



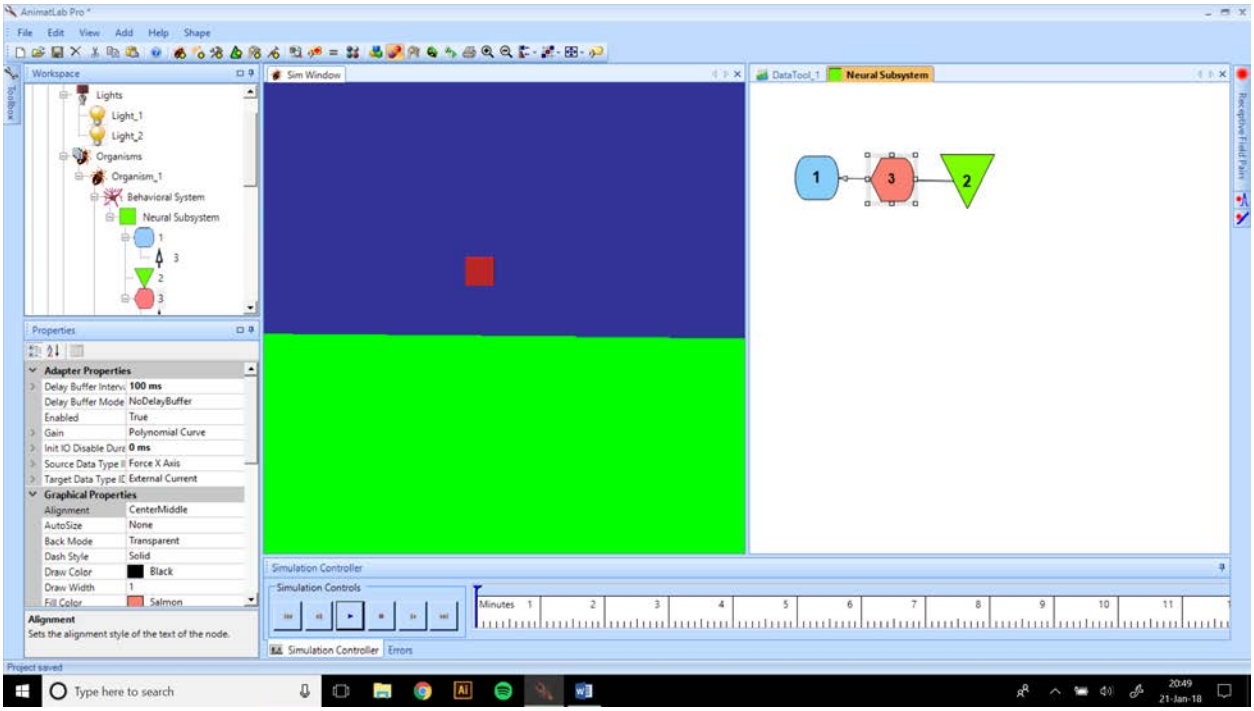

Next, go to its Properties and change the Rigid Body ID to Organism_1>Root.

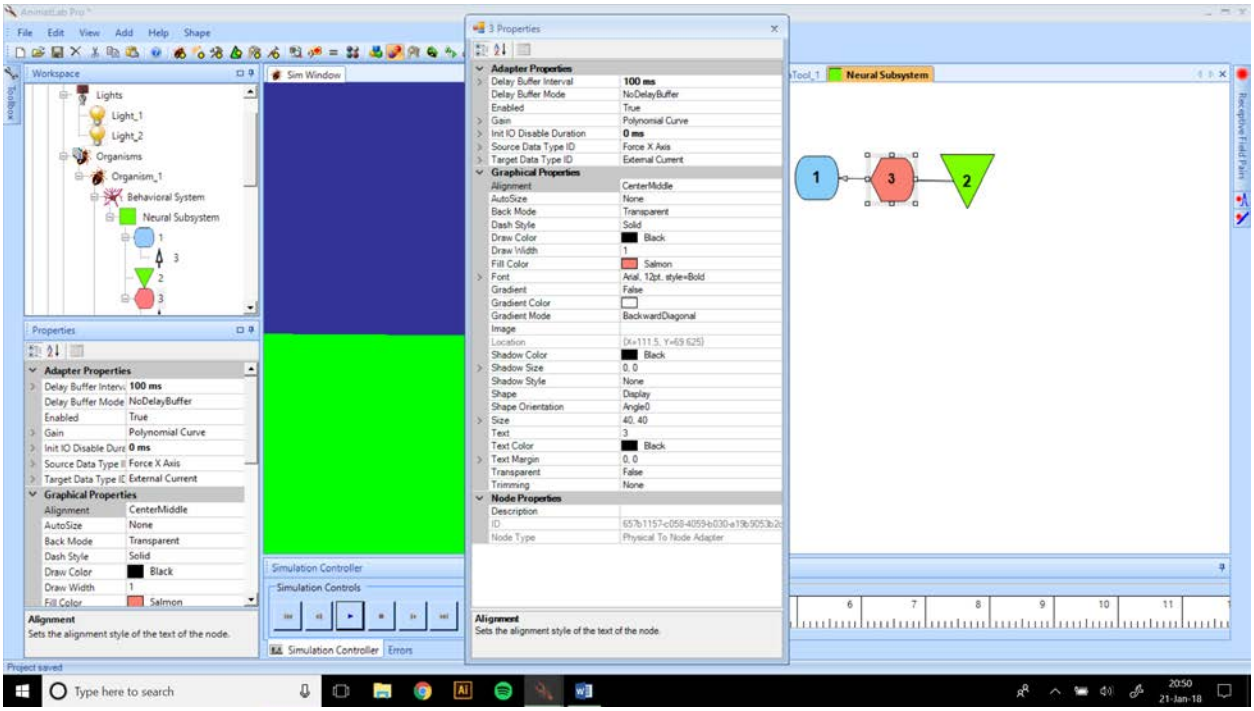



Click and drag a connection from 2 to 1.

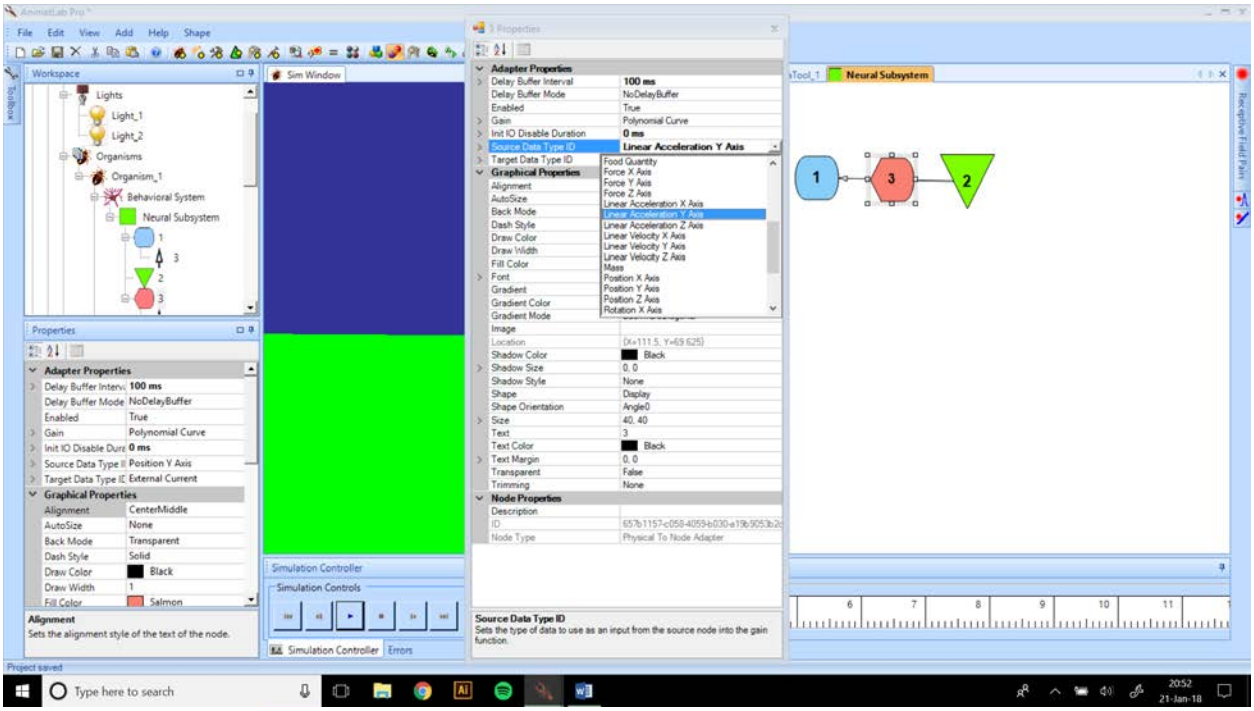



When you release, a pink adapter node will appear. Double click on the node to set its properties.

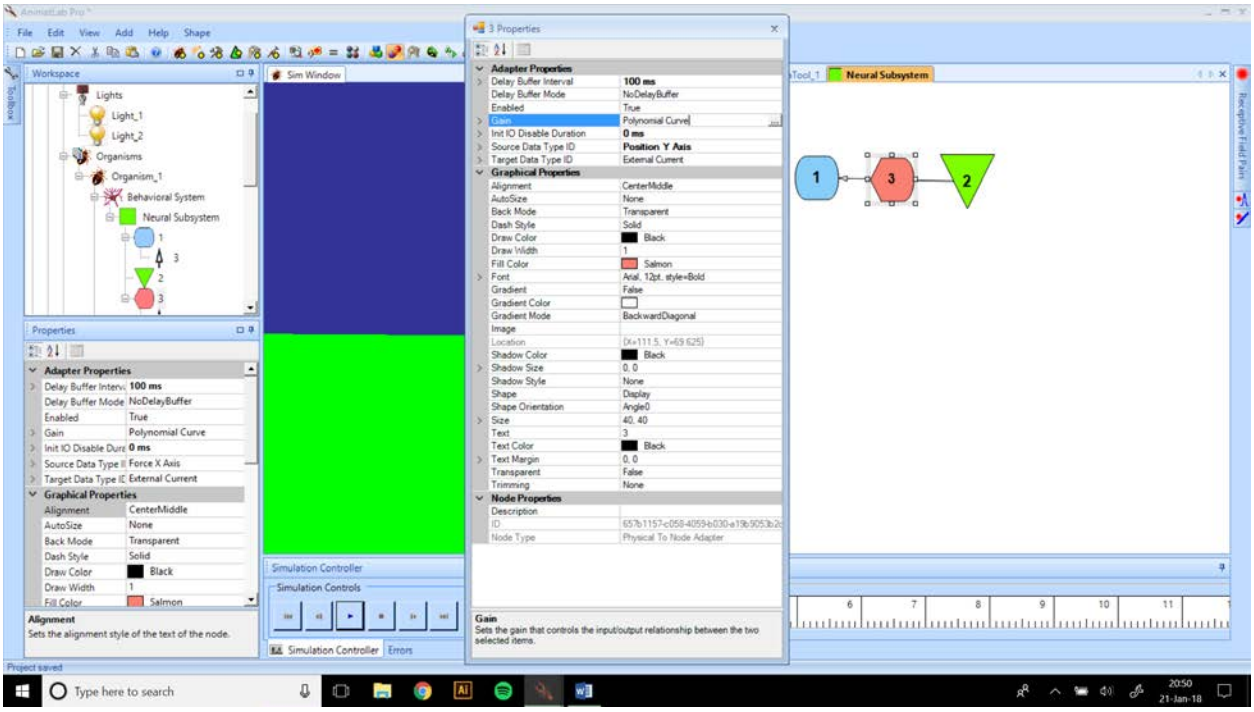




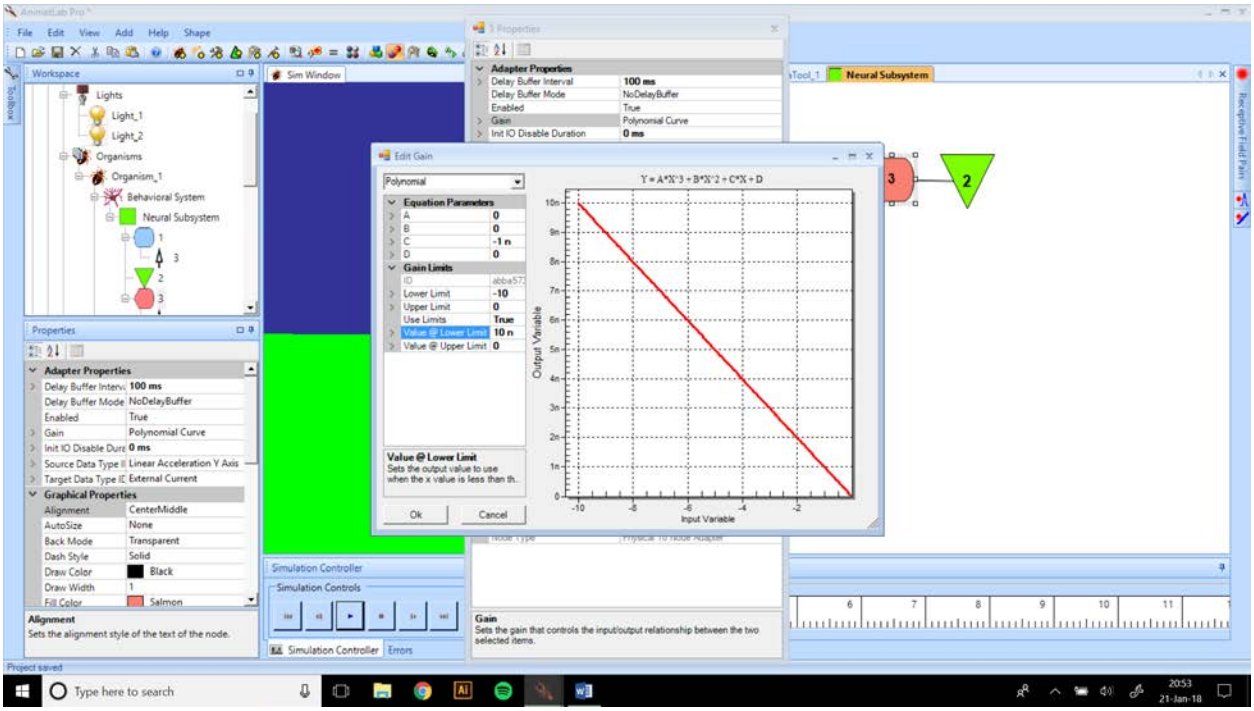
First, select a Source Data Type. You can see that many options are available. Not all of them work, and I don’t know why, but be advised. Select “Linear Acceleration Y Axis”.



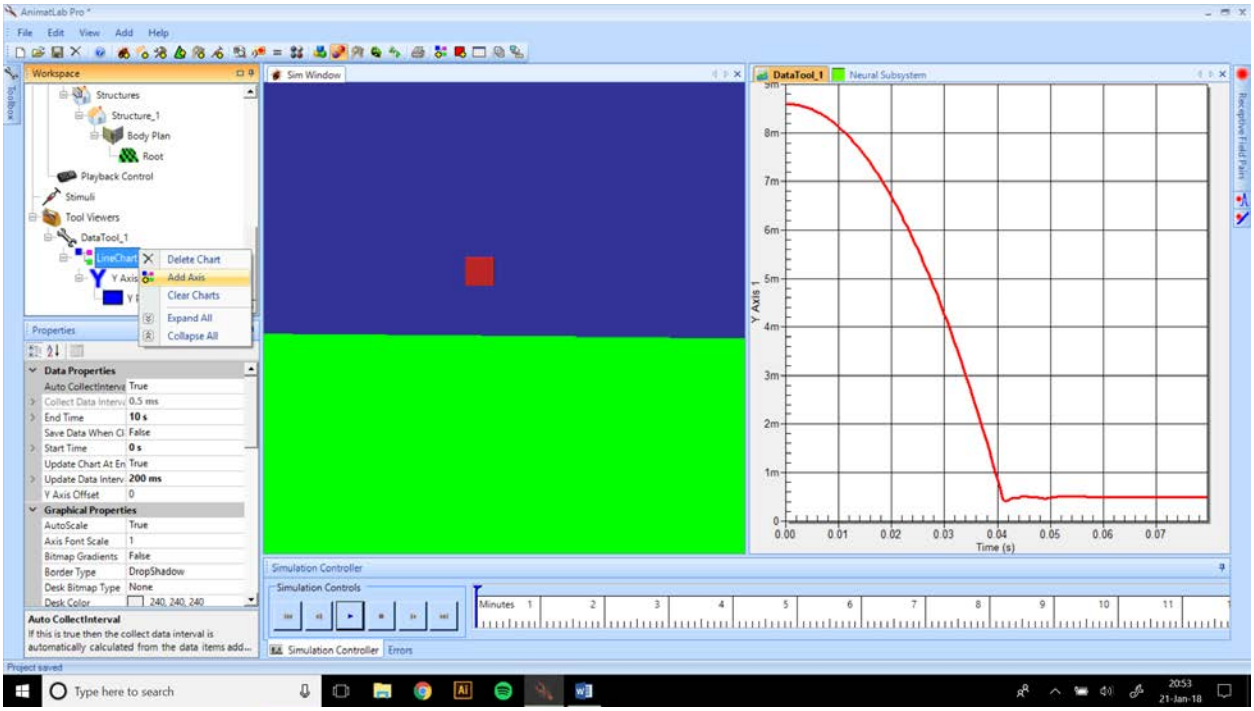
Next, we need to map the acceleration of the block to the current that neuron 1 receives. Open the Gain menu by clicking on the ellipses where it says “Polynomial Curve”.



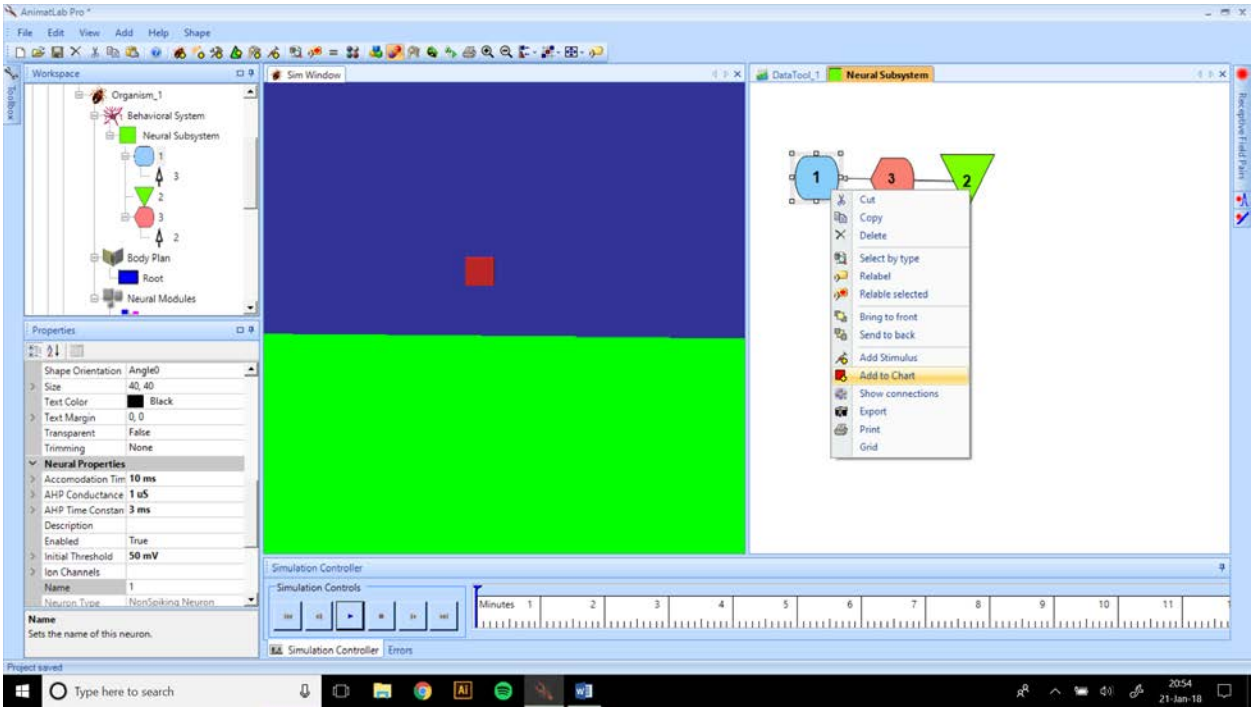
A new window will appear. The default mapping is polynomial, but you can also make Gaussian or Sigmoidal mappings with the drop down menu on top. I set the adapter to be a linear mapping with a slope of -1x10^(-9), or -1 n. Change the limits in the lower box to match my settings. This ensures that the mapping saturates at the extremes. Click OK.



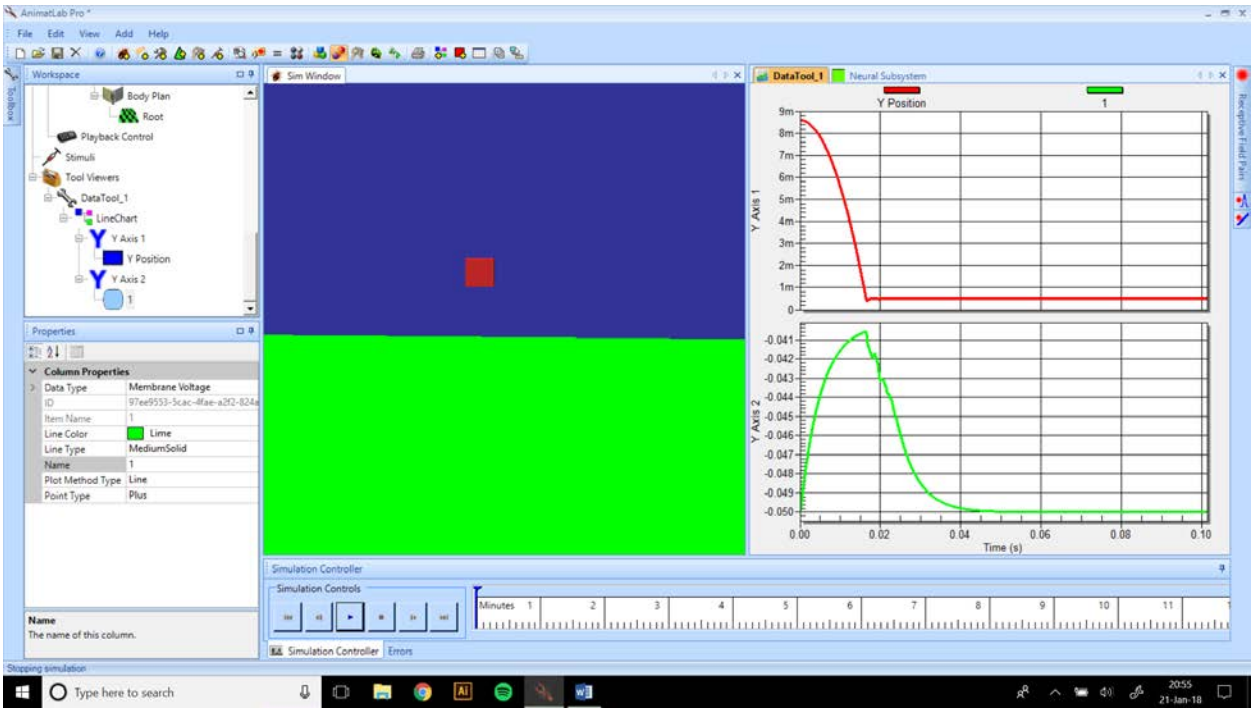
Now, add a new axis to DataTool_1:



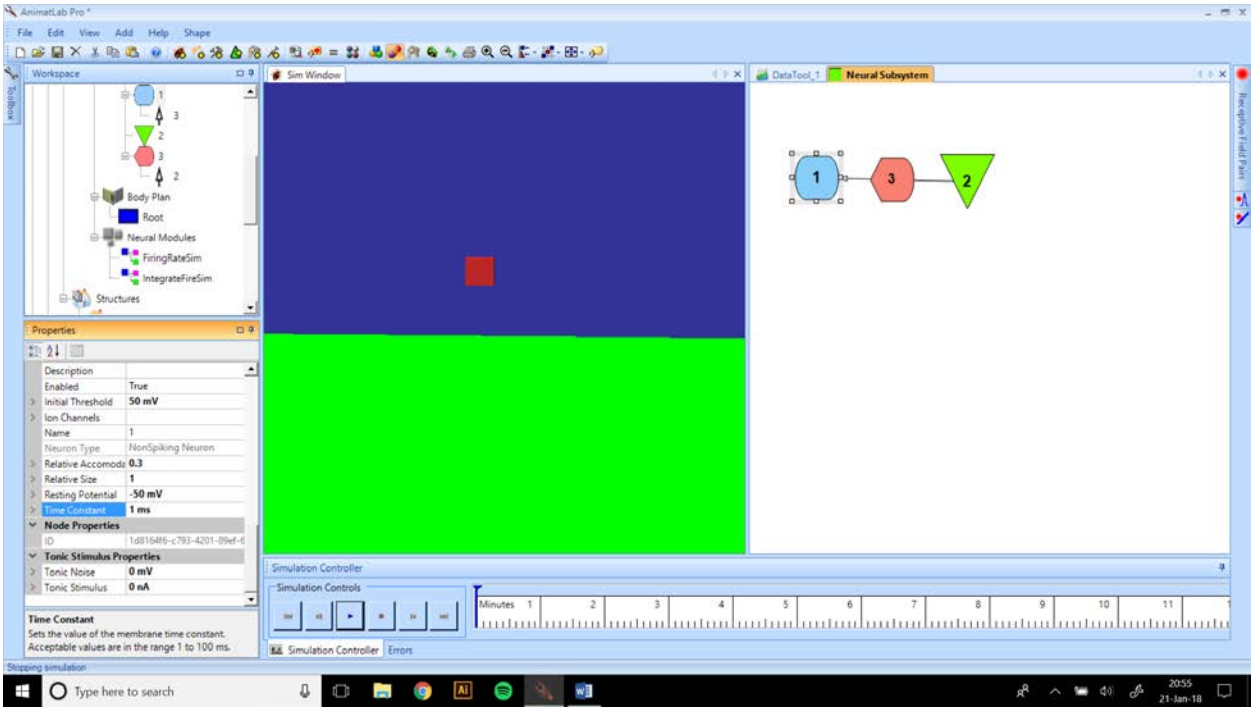
And add neuron 1 to the graph. Another way to add columns to the chart is to right click on items, and then select “Add to Chart”. It will add that item to the last axis you have selected.



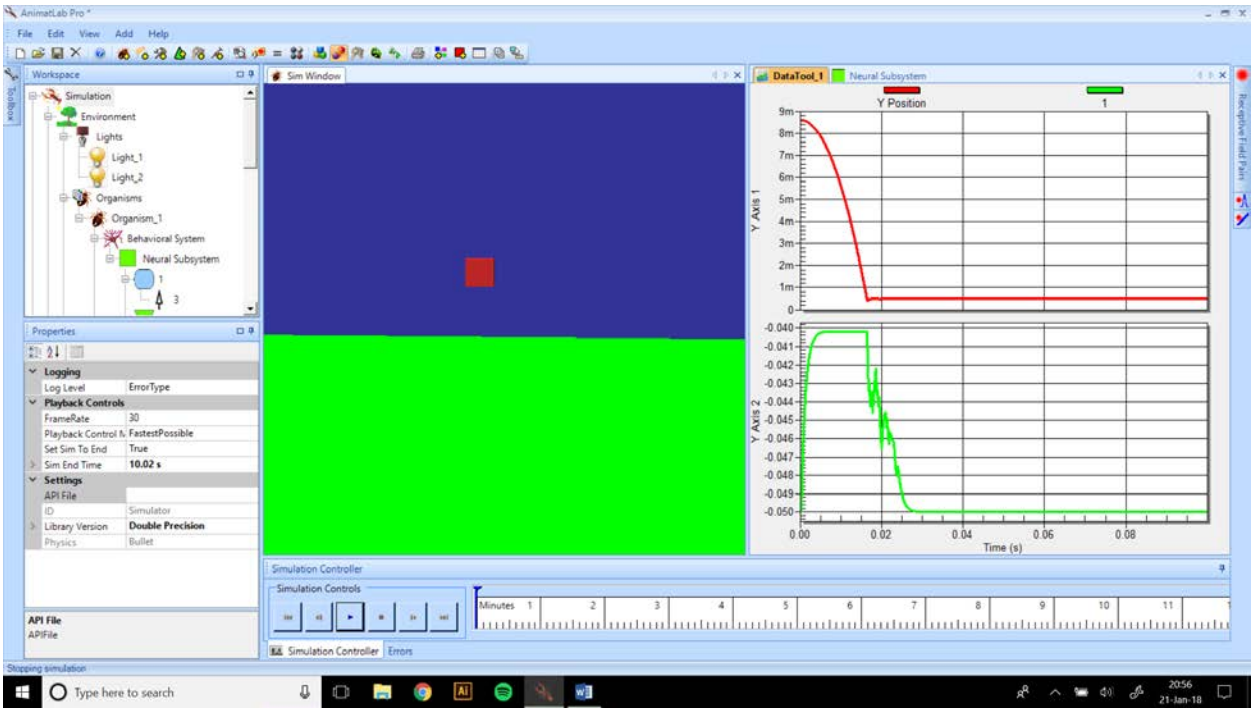
Now when we run the simulation, we get two plots. The true acceleration of the block should look like a step-down function. Our neural activity looks similar, although instead of a sharp step function, it is rounded. That is because the neurons are low-pass filters, so they naturally smooth incoming signals. We can improve the performance by making the time constant shorter.



Click on neuron 1, and change its Time Constant to 1 ms.

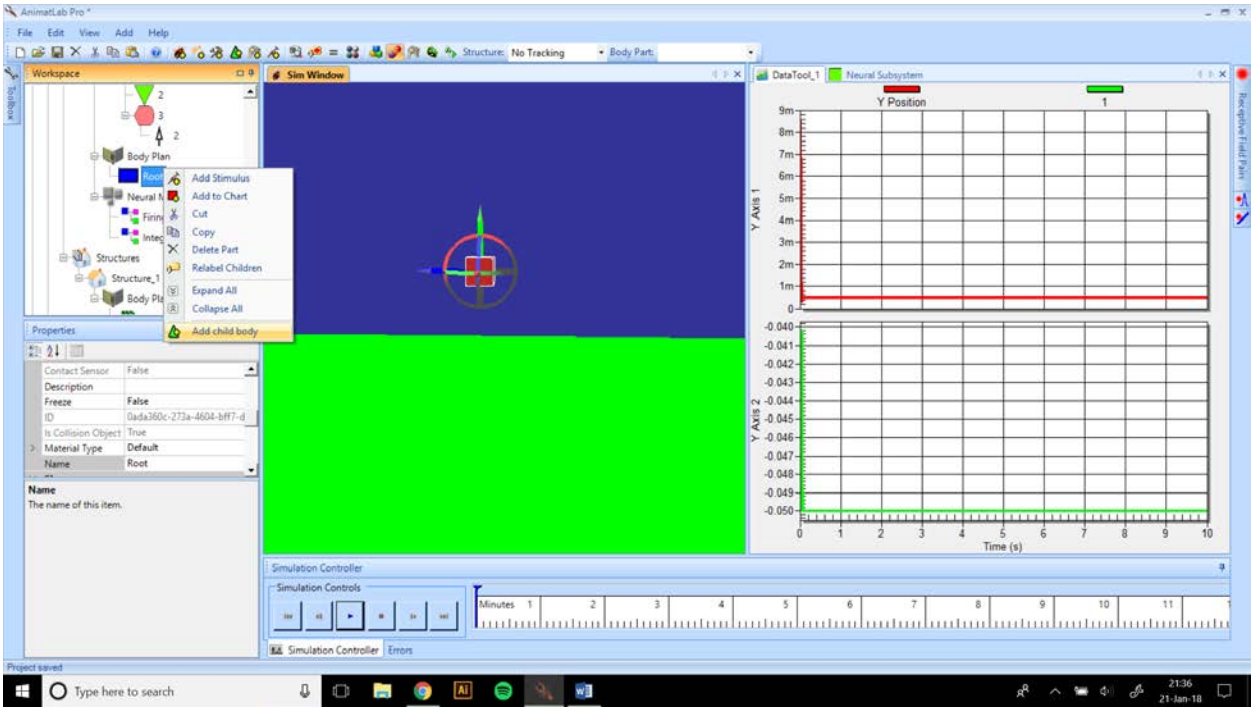



When we run the simulation again, we get something closer to what we expect: a plateau that encodes the free-fall acceleration. Of course, a real system, whether an animal or a robot, might deliberately have longer time constants to filter out high-frequency noise. In our simulation, however, that isn't a problem.

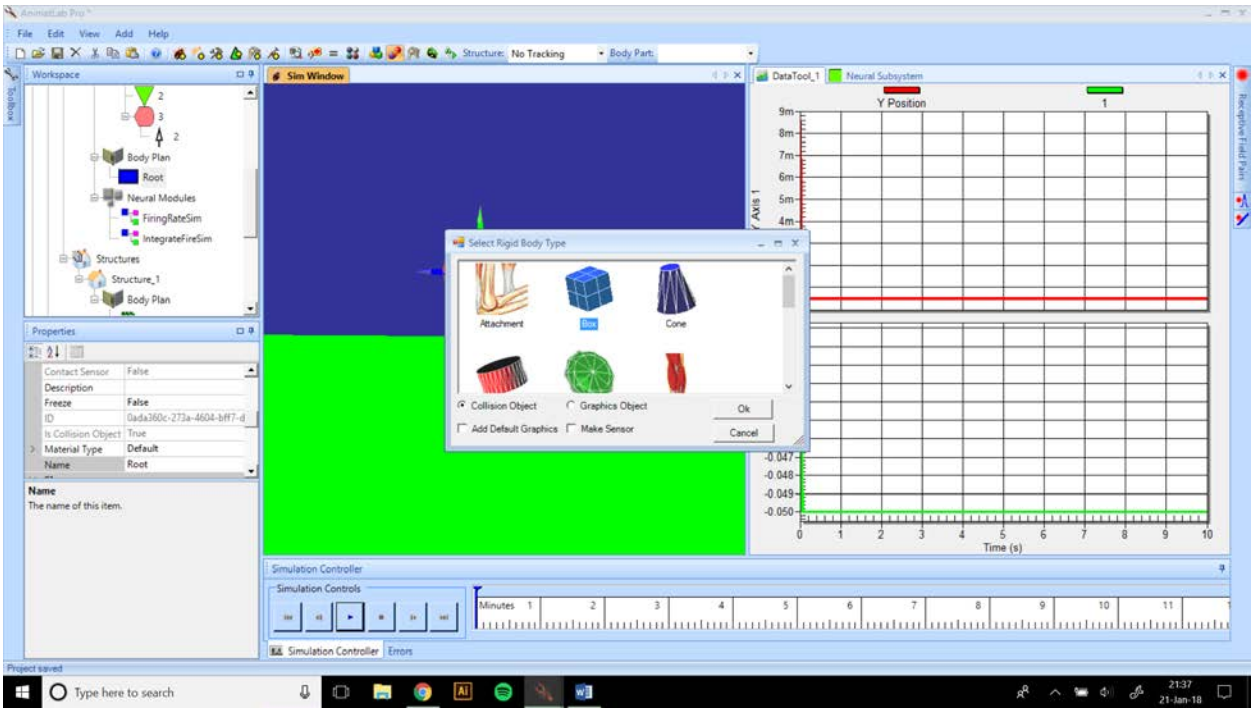


## The slightly less basic.

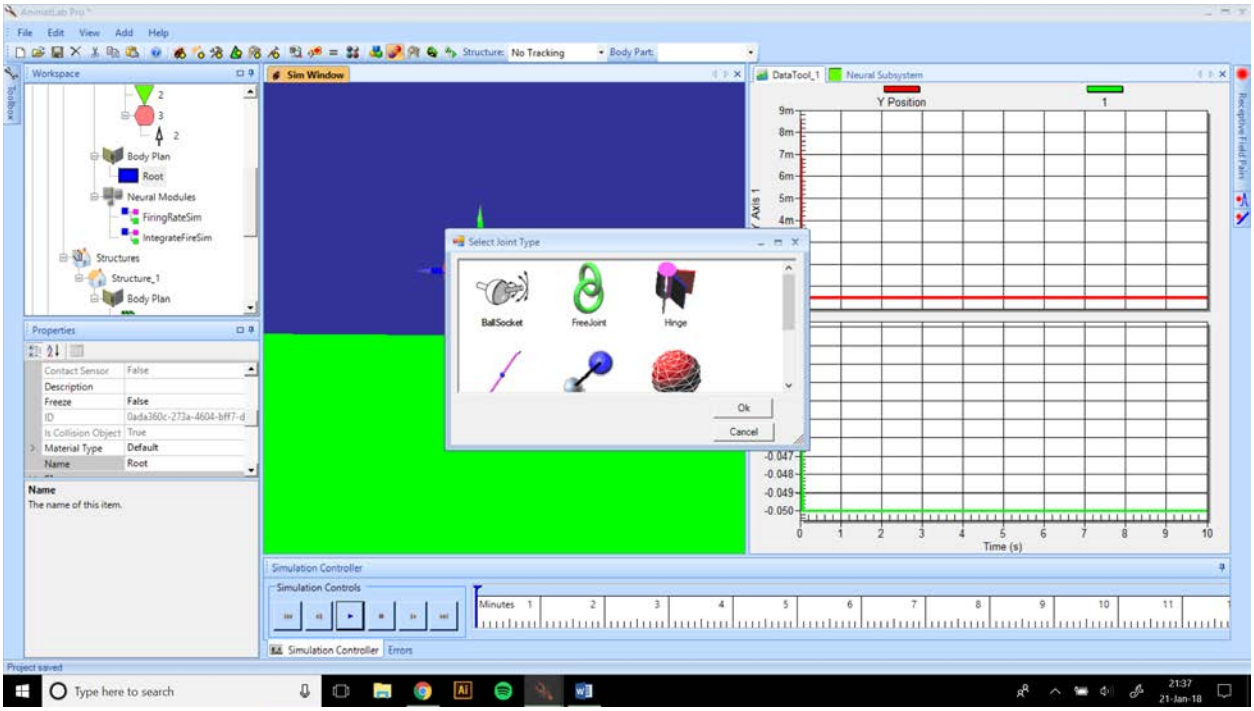
Now, the tutorial will move a bit faster. We will add another segment to our organism's body, add an actuator, and design a feedback look to control the position of the second segment.



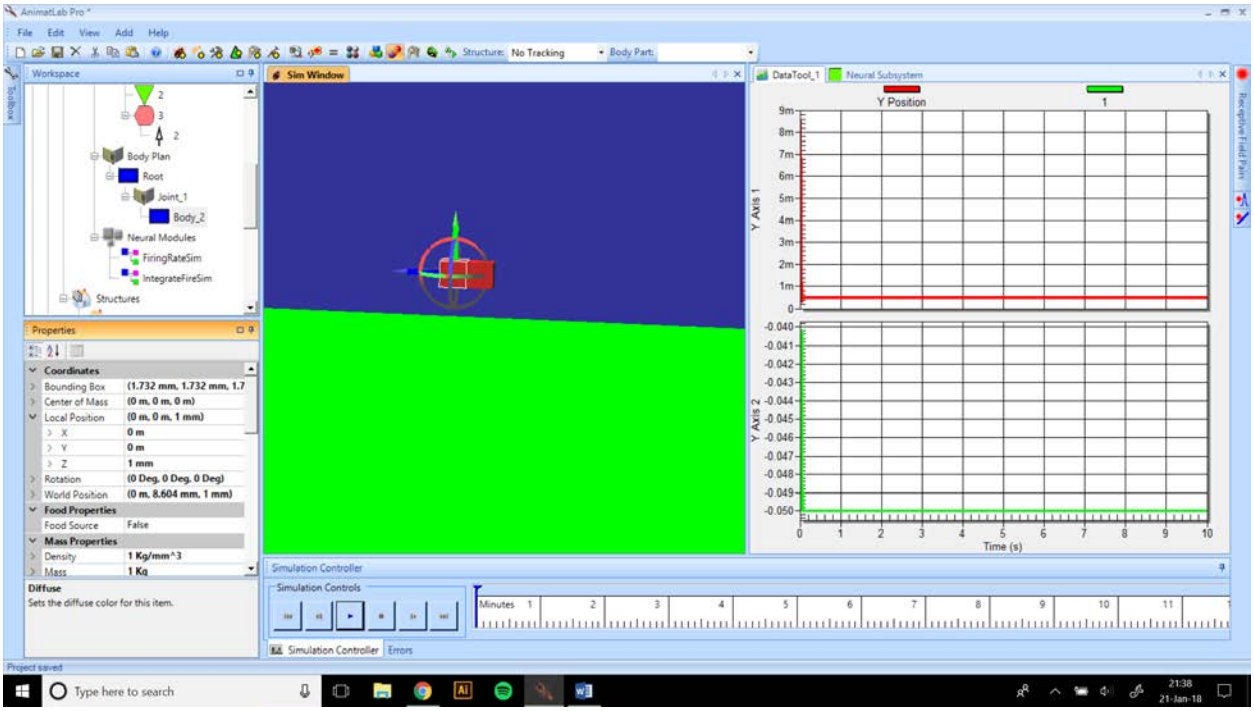

Select the body type "Box", and deselect the "Add Default Graphics" option.



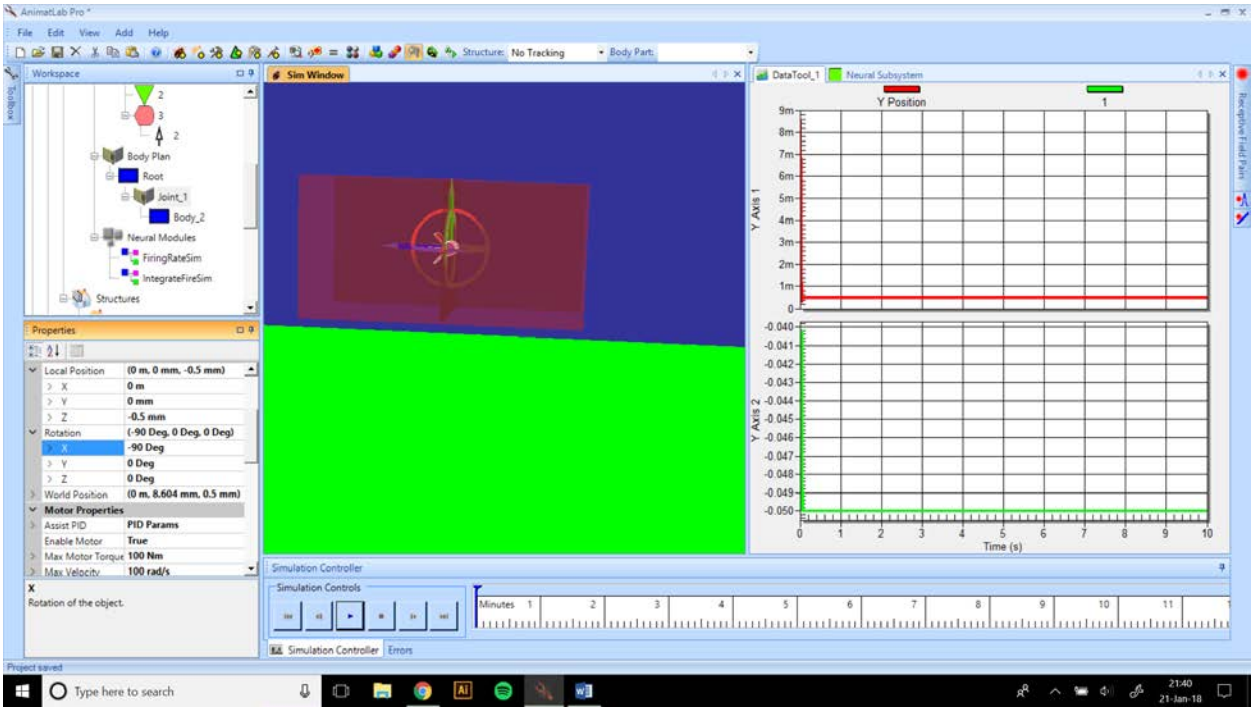

Next, you will need to Select Joint Type. Select "Hinge".



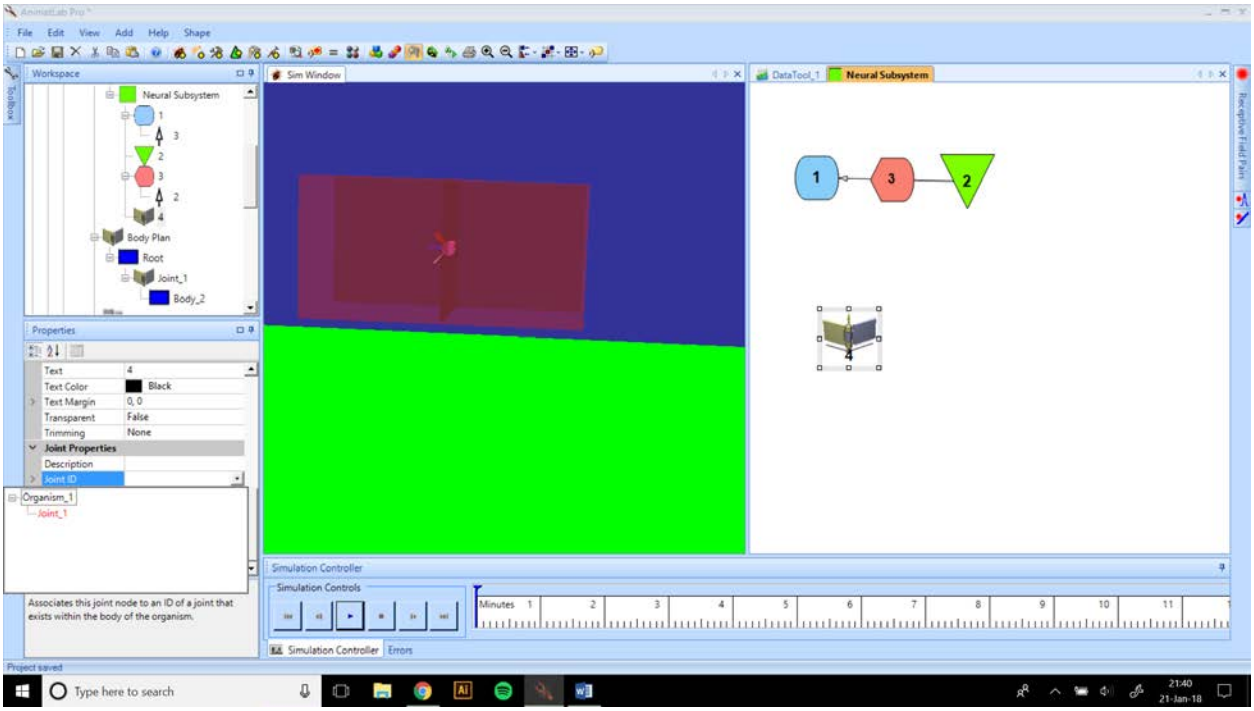

Now we need to position the bodies properly. Place Body_2 next to Root. In my simulation, that means that the local position is [0,0,1] mm. Note that coordinates can be specified in either the global frame, or local frame (with respect to the proximal segment).



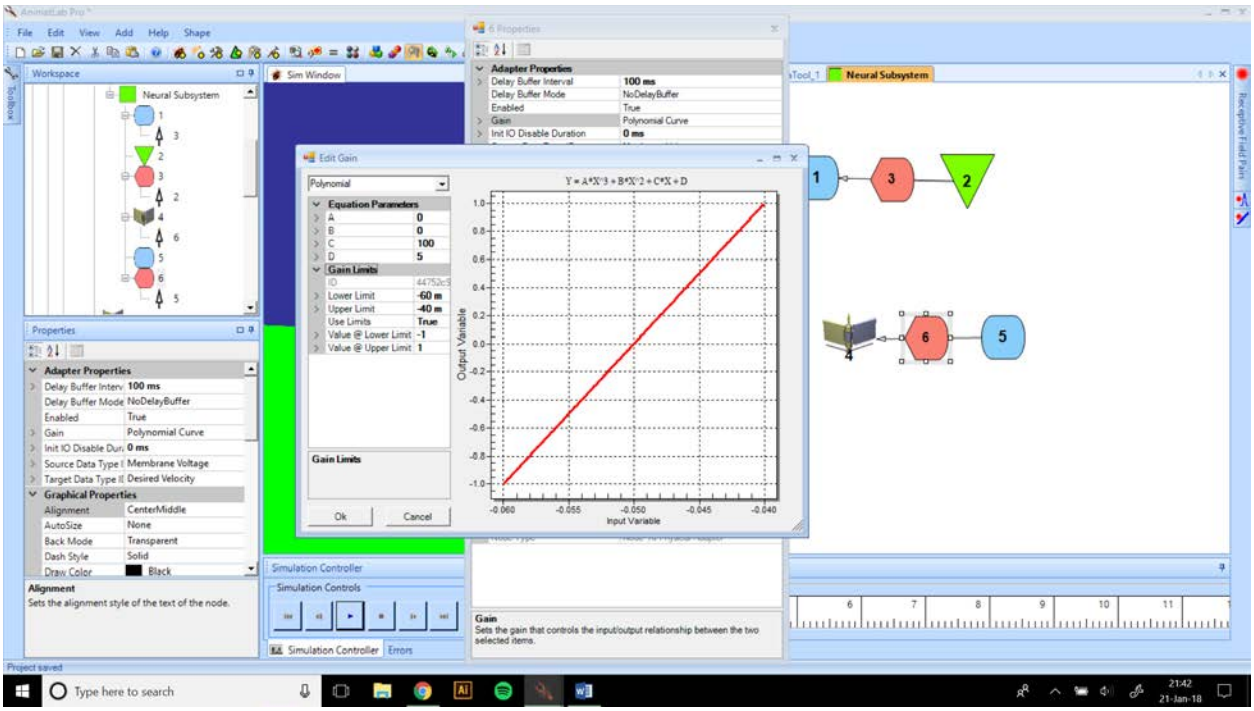

Next we need to position Joint_1. Set the local position to [0,0,-0.5] mm. Also, set the x rotation to -90 degrees, and set “Enable Motor” to true. This will allow us to control the joint like a DC motor, in which the voltage across the terminals sets the steady-state velocity of the rotor.



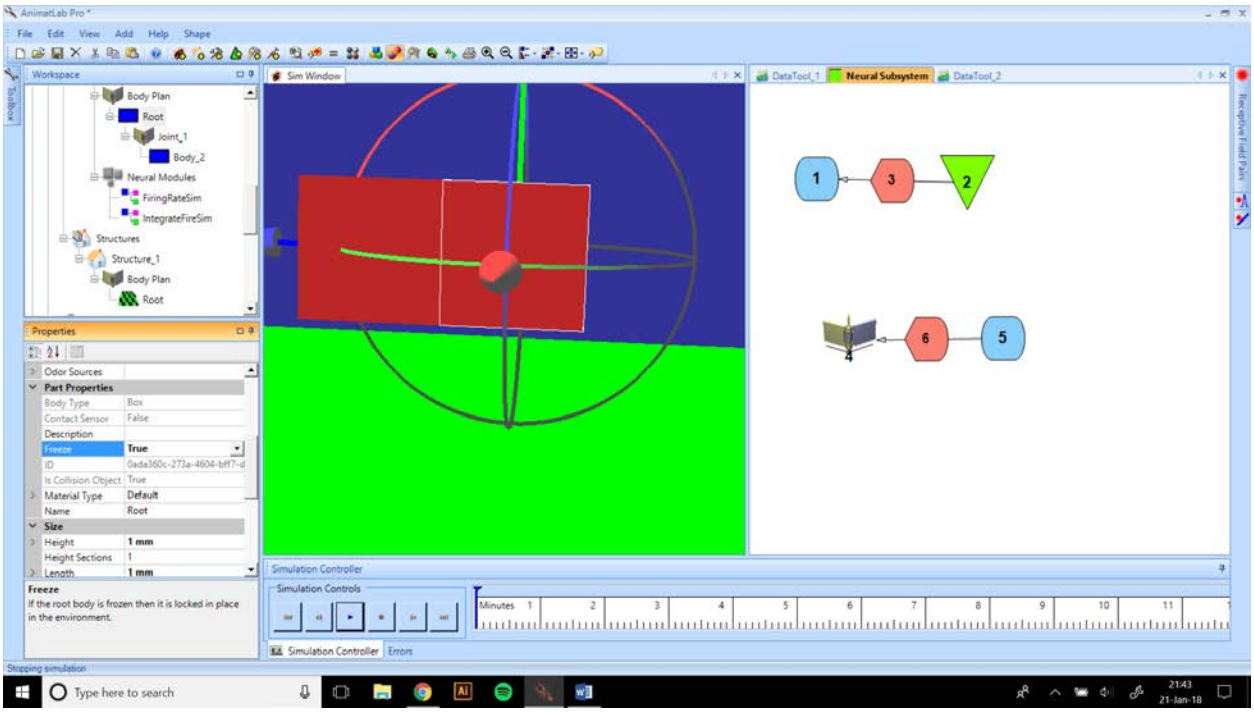

Next, we need to add components for a control network. Click on the Toolbox and drag a Joint object into the Neural Subsystem. Set its Joint ID to Joint_1. When the joint's motor is enabled, this node can be both an input and an output.



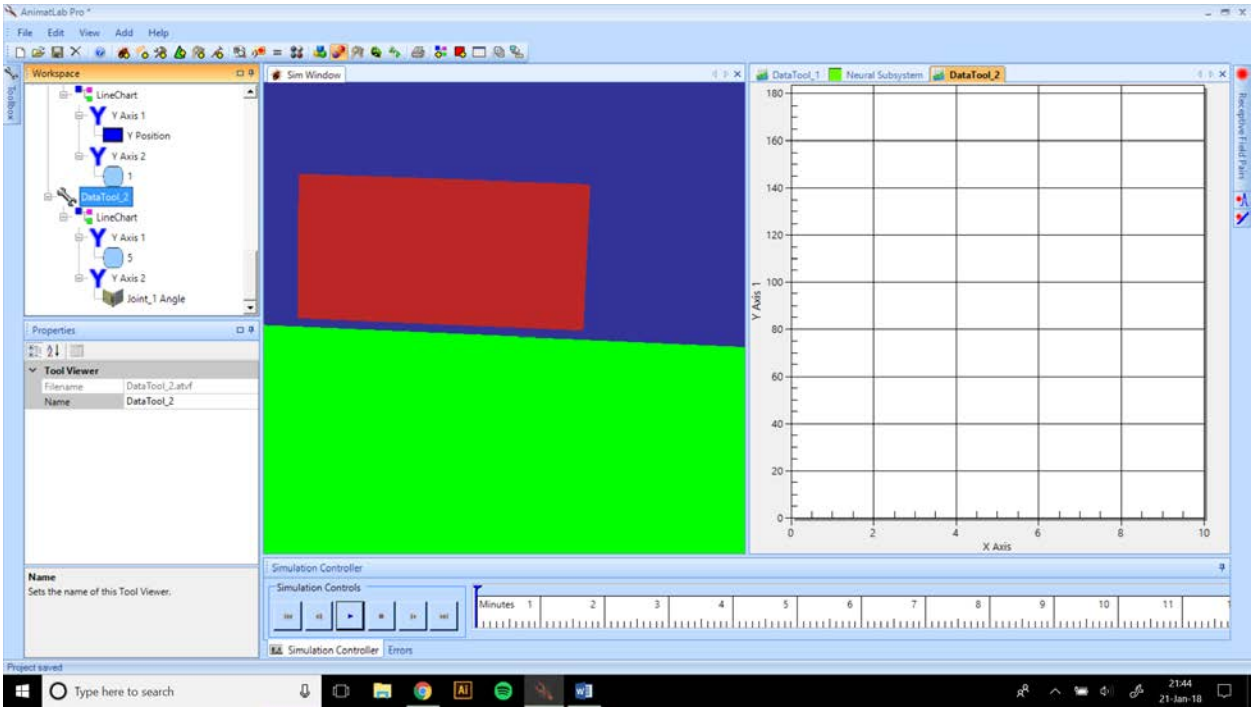
Drop and drag another nonspiking neuron (neuron 5), and then drag an adapter from the neuron to the joint. This will set the map between neuron 5's voltage, and the commanded velocity of the motor. You can see that I've set my controller to command 1 rad/s when neuron 5 is excited to -40 mV, and -1 rad/s when neuron 5 is inhibited to -60 mV.



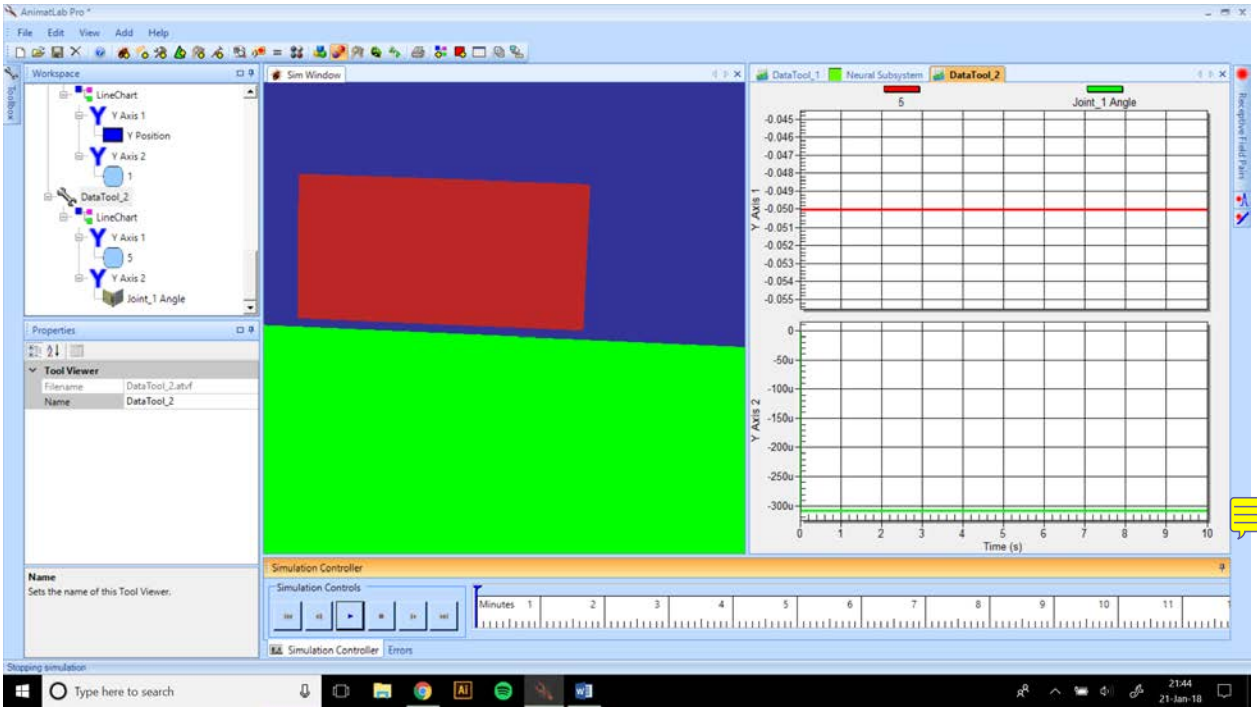

Next, I'm going to freeze body Root, so it no longer falls to the floor when the simulation runs. This will enable us to observe Joint_1's rotation.



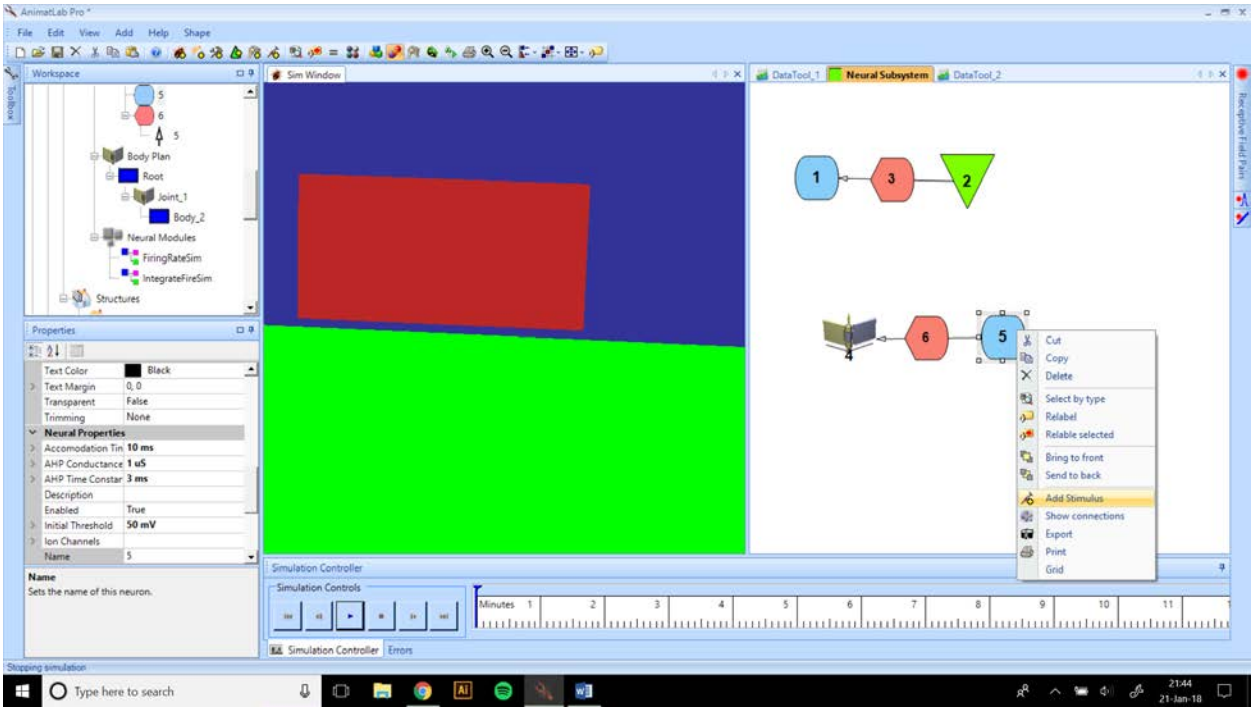
Create a new line chart, DataTool_2, and add an axis for neuron 5, and an axis for the Joint_1 angle.



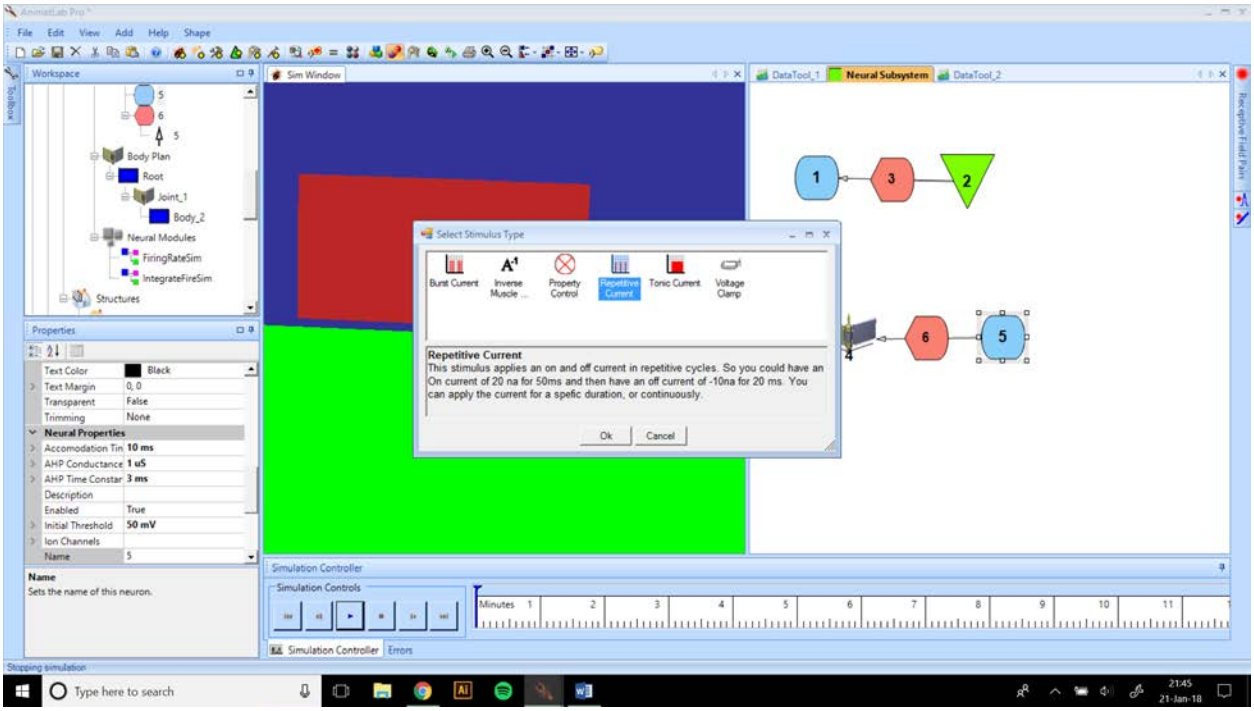
Run the simulation. Neuron 5 remains at its rest voltage of -50 mV, and after sagging under gravity, the limb also maintains a constant rotation.



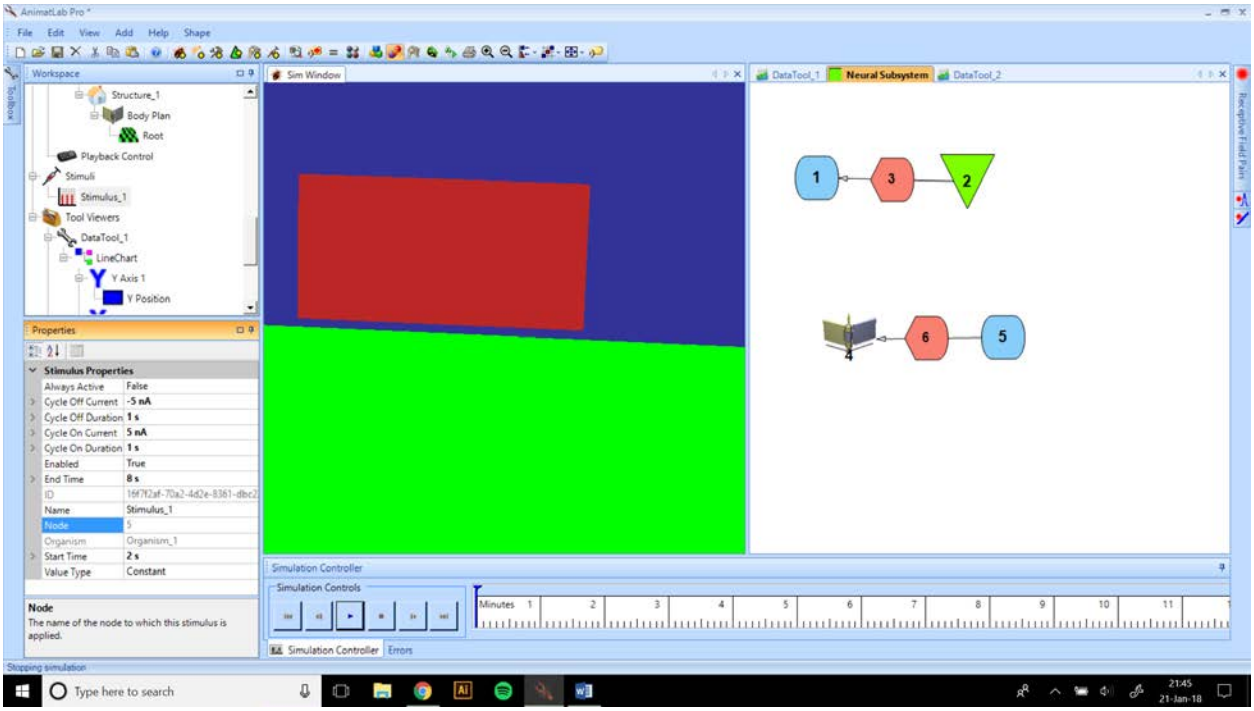
Let’s try something a little more dynamic. We’re going to add a square wave stimulus to neuron 5 and observe the limb’s response. Right click on neuron 5 and click “Add Stimulus”.



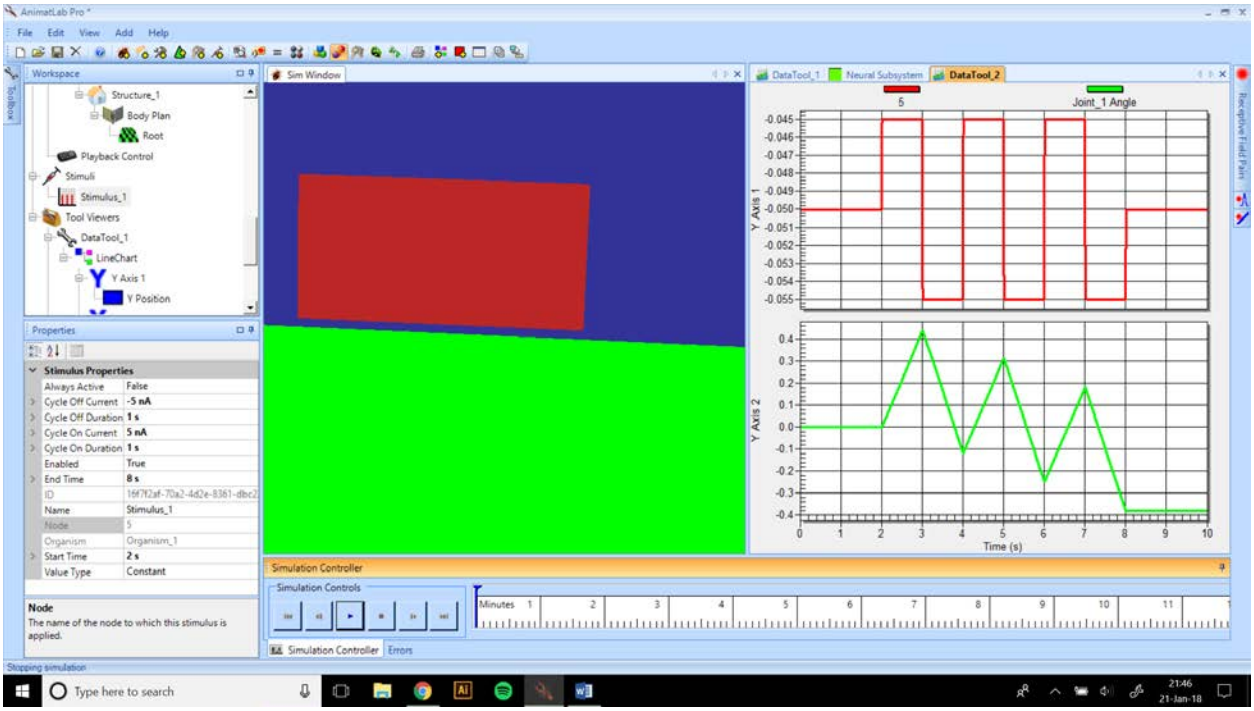

When prompted, select “Repetitive Current” and click OK.



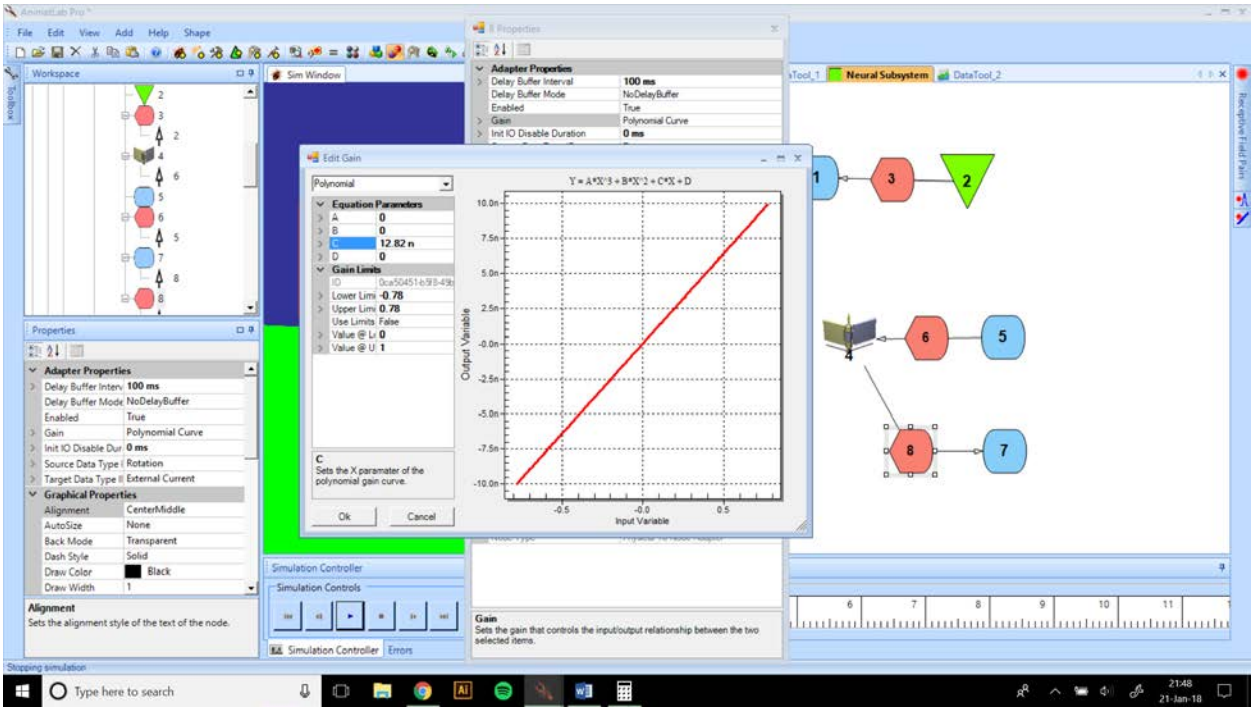

Set up the properties of the stimulus. Set the “Cycle On Current” to 5 nA, set the “Cycle Off Current” to -5 nA, set the “Cycle On Duration” and “Cycle Off Duration” each to 1 s. Set the “Start Time” to 2 s and the “End Time” to 8 s.



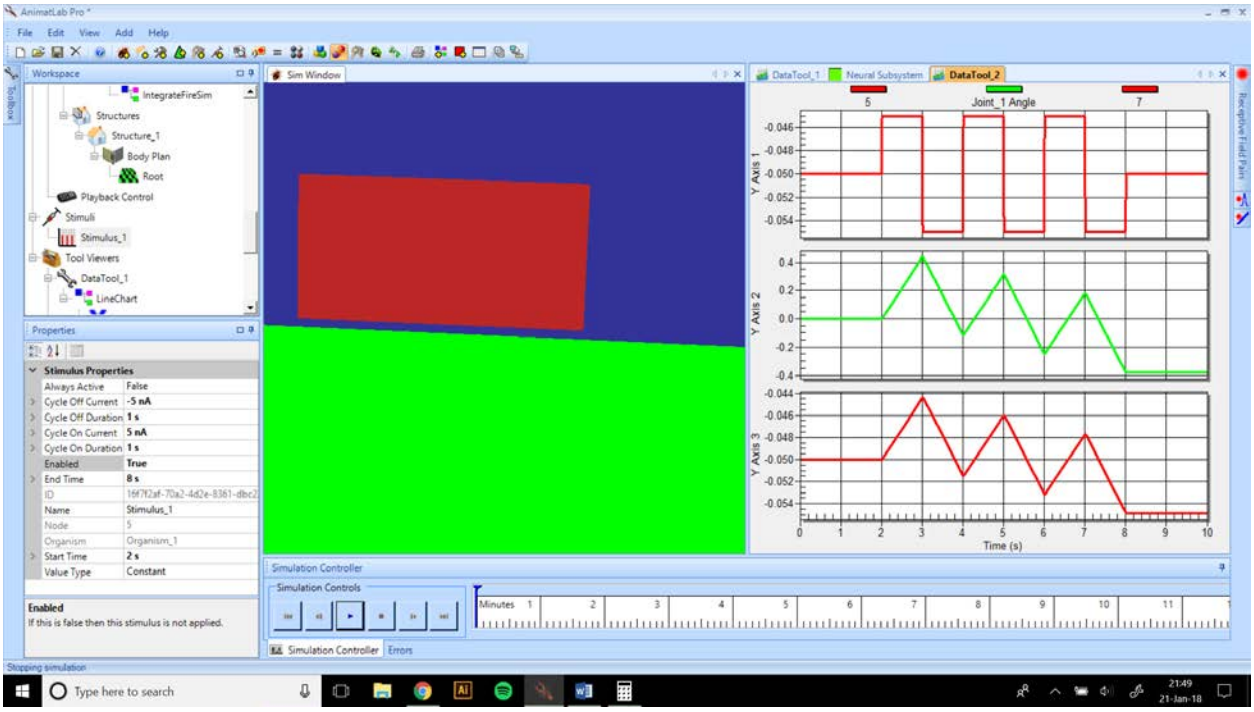
Run the simulation. Observe that neuron 5 shows activity that corresponds to the stimulus. This indeed sets the velocity of the joint, although the joint continuously sags under gravity. Let's add position feedback to prevent the sagging.



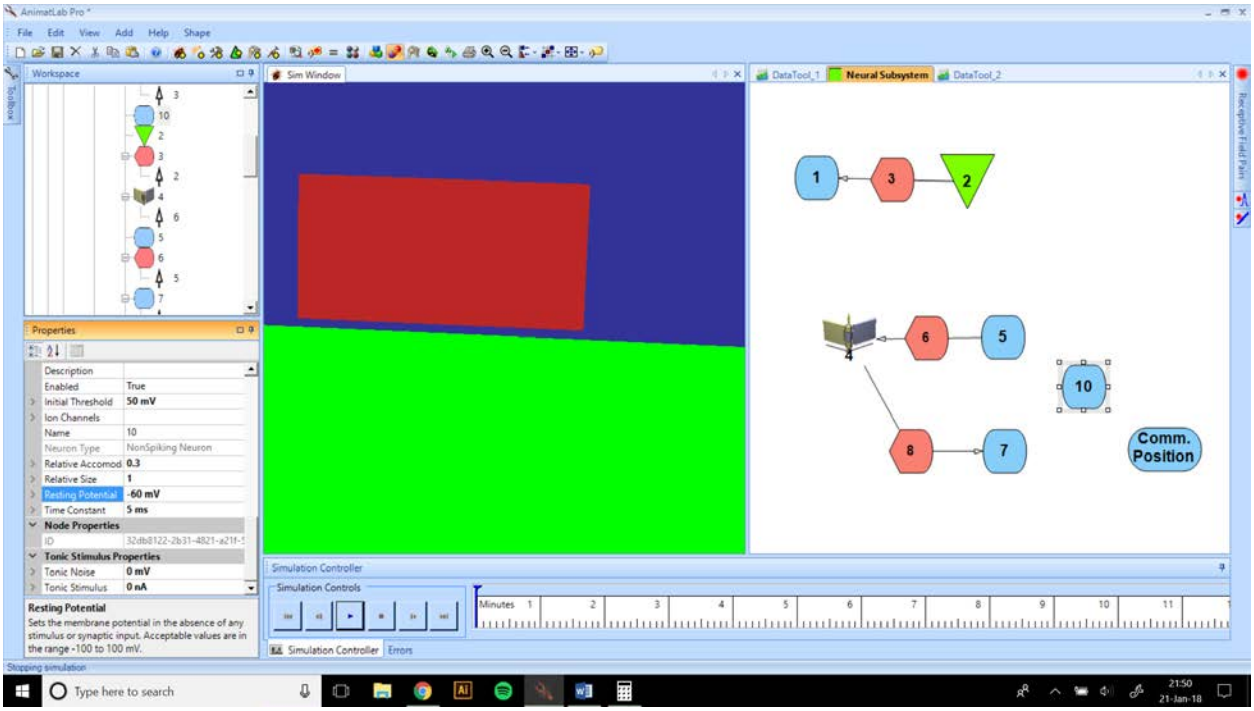
Drag and drop a new nonspiking neuron (neuron 7), and then drag an adapter from the joint to the new neuron. We want to map a 90 degree range of motion to a 20 nA interval, so we set the parameter values as shown in the popup window:



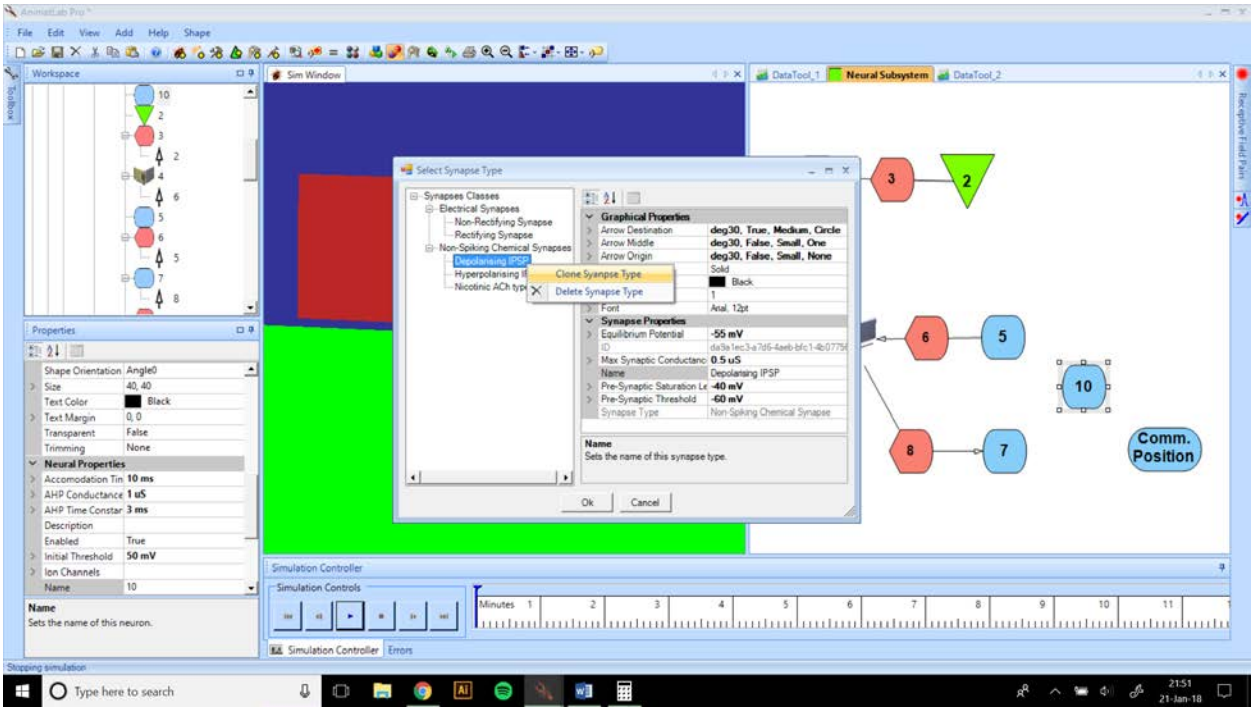
Add neuron 7 to DataTool_2, and run the simulation. Observe that neuron 7's voltage mirrors Joint_1's rotation. Zooming in shows that neuron 7 smooths this signal, as we expect.



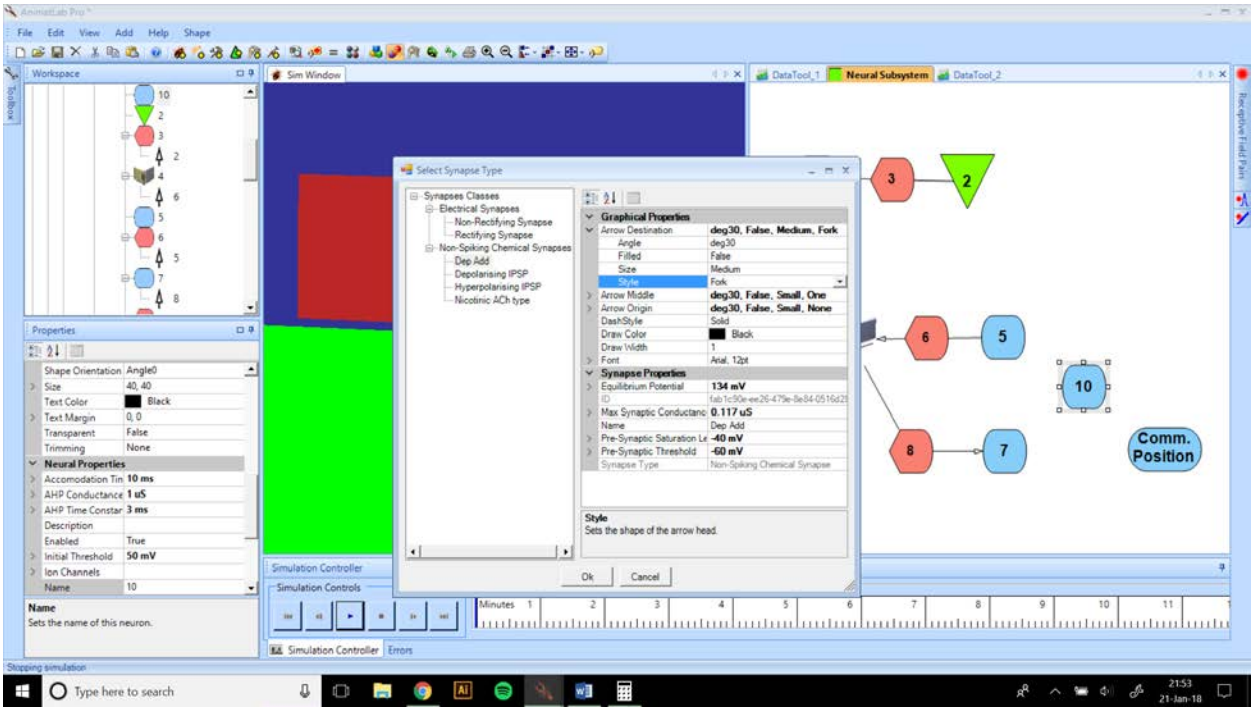
Next, we want to use this feedback to control the velocity of the motor. Let's add two more neurons. Name one "Comm. Position", short for commanded position. For the second one, set the resting potential to -60 mV.



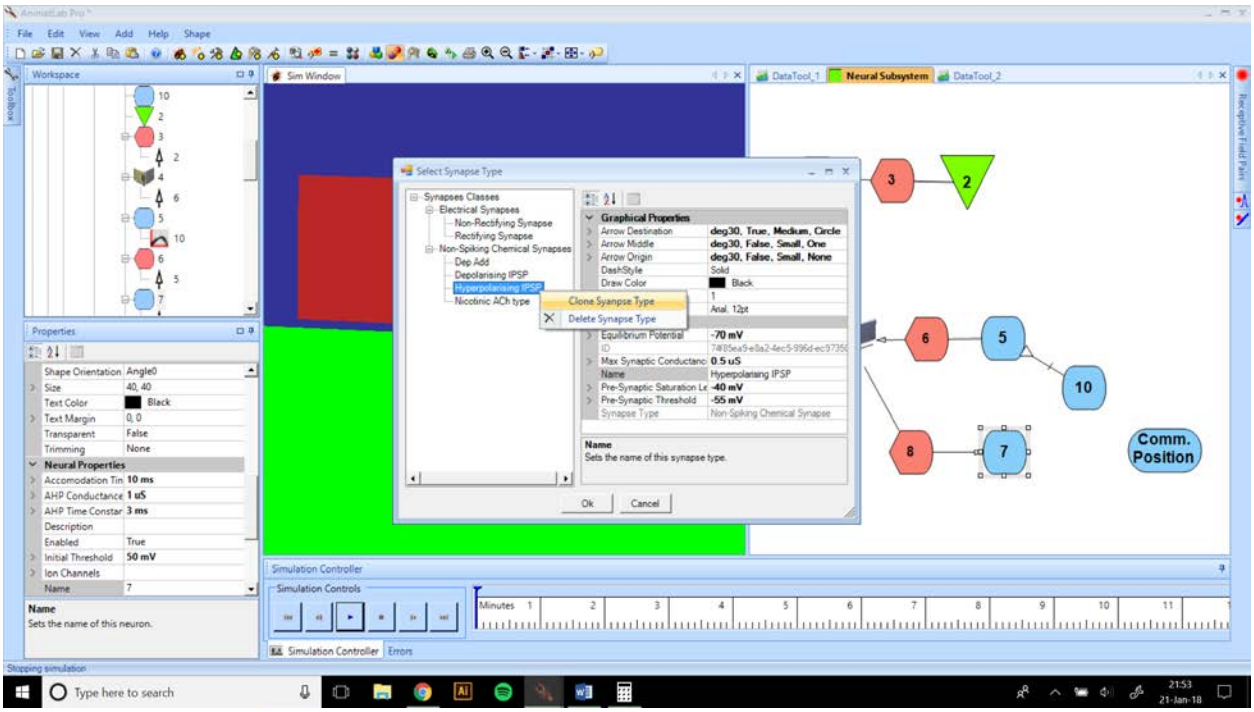

Now, we will connect the neurons with Non-Spiking Chemical Synapses. Click and drag from neuron 10 onto neuron 5. That will cause this window to pop up. Right click on Depolarising IPSP and select “Clone Synapse Type”.



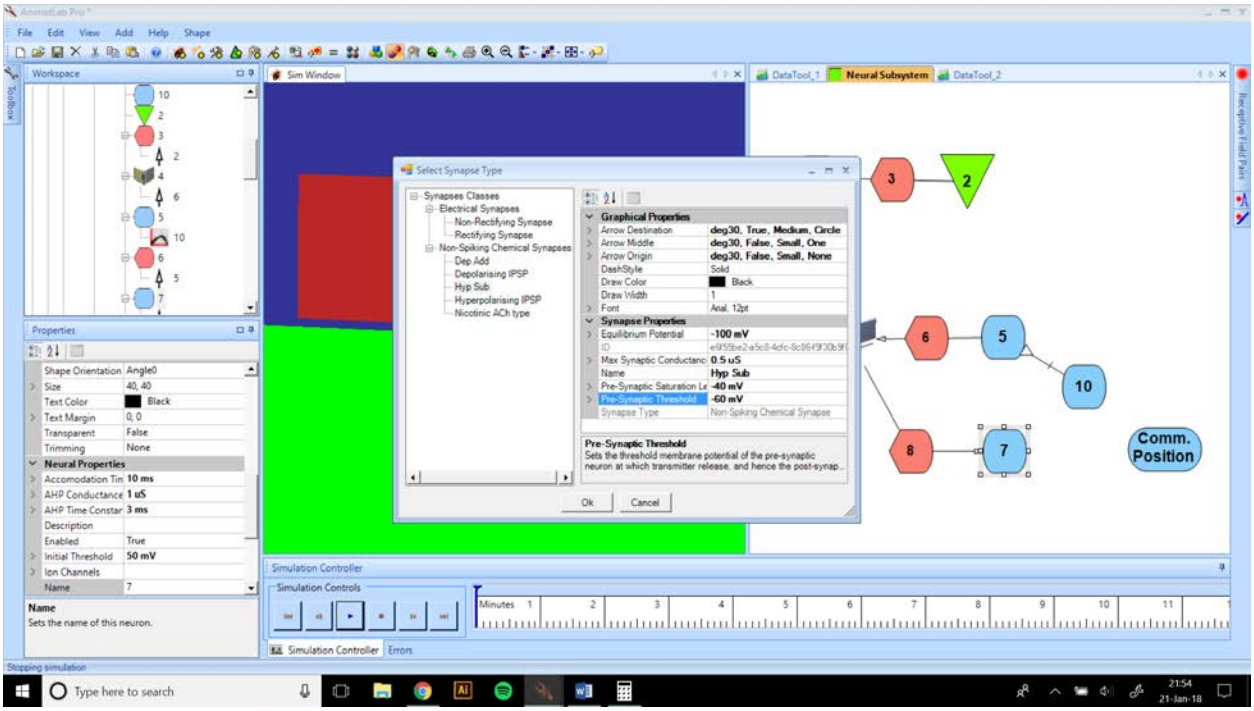

Now, rename the synapse type “Dep Add”, and set the other parameters to match. This synapse will cause neuron 5’s voltage to closely follow neuron 10’s voltage. Click OK.



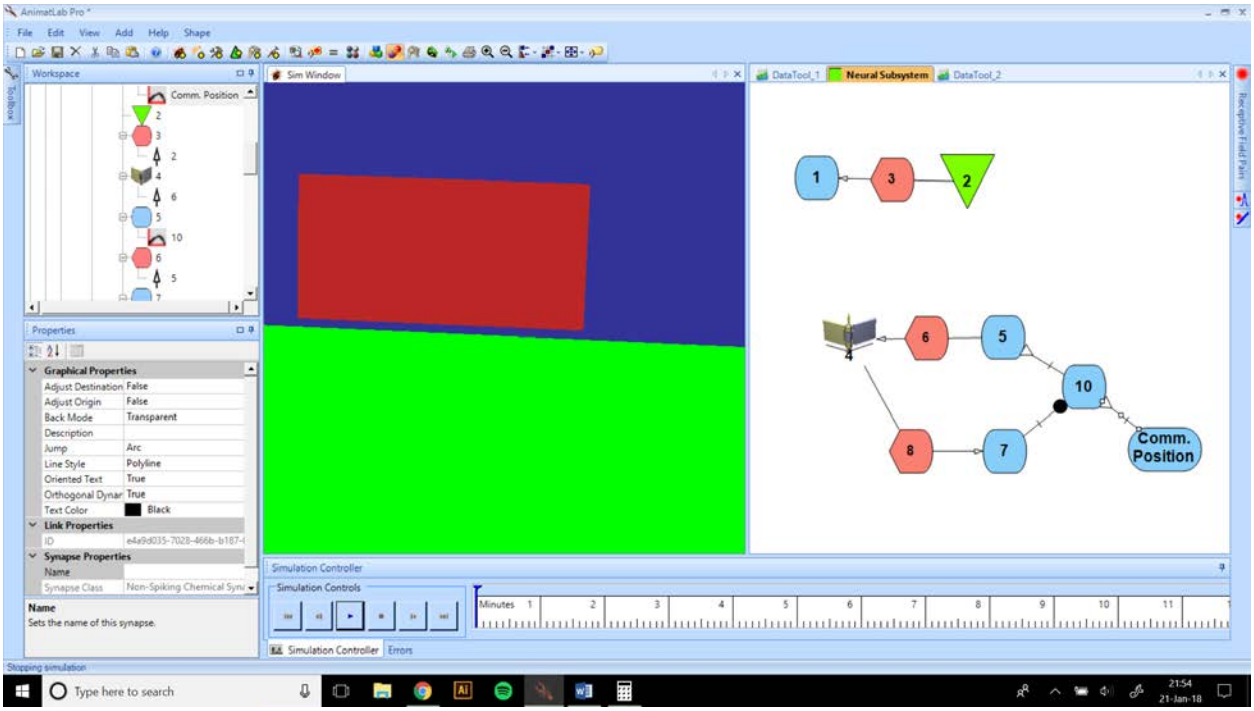

Now, click and drag a synapse from neuron 7 to neuron 10. This time, clone the Hyperpolarising IPSP synapse.



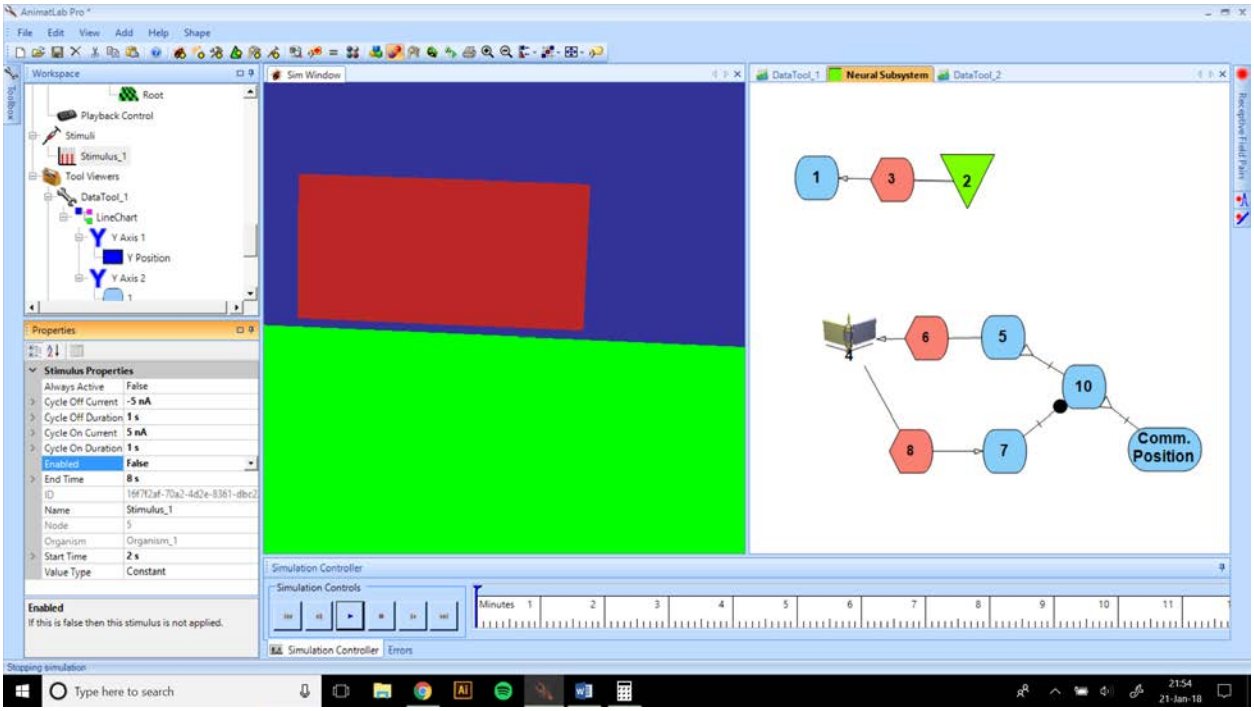

Name it “Hyp Sub”, and set the parameters to the value shown below.



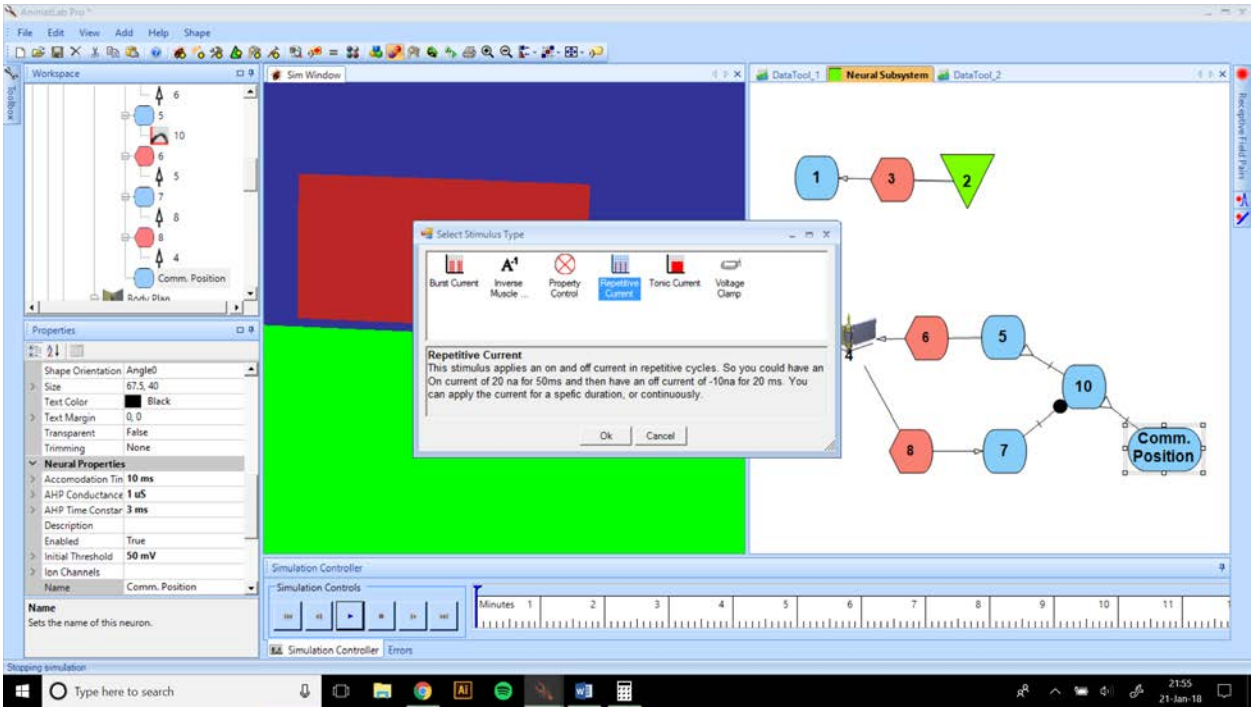

Click and drag one more synapse from Comm. Position to neuron 10, and select “Dep Add” as the type. Now, we might expect the voltage of the Comm. Position neuron to control the velocity of the motor to drive it toward the commanded position.



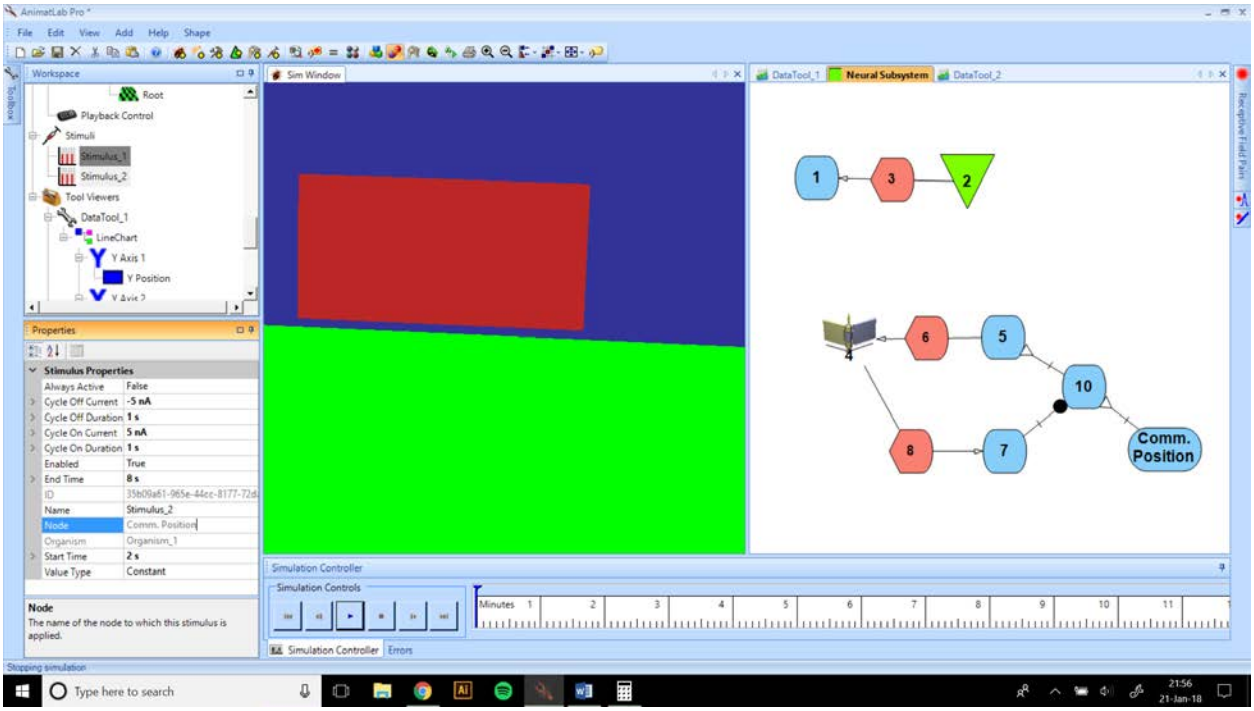

To test this, first disable Stimulus_1, which stimulates neuron 5. Stimuli, and most other objects, have an “Enabled” flag, which when set to false, will disable it.



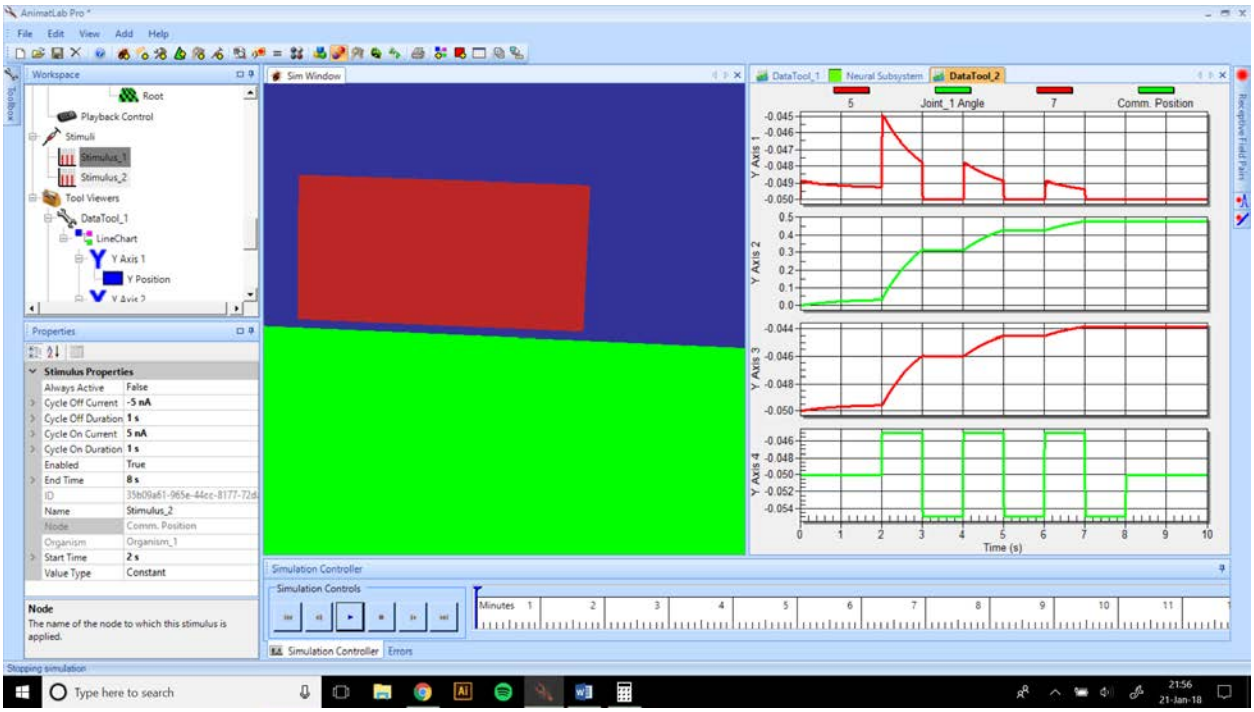

Next, add a new stimulus to the “Comm. Position” neuron. Select “Repetitive Current” and click OK.



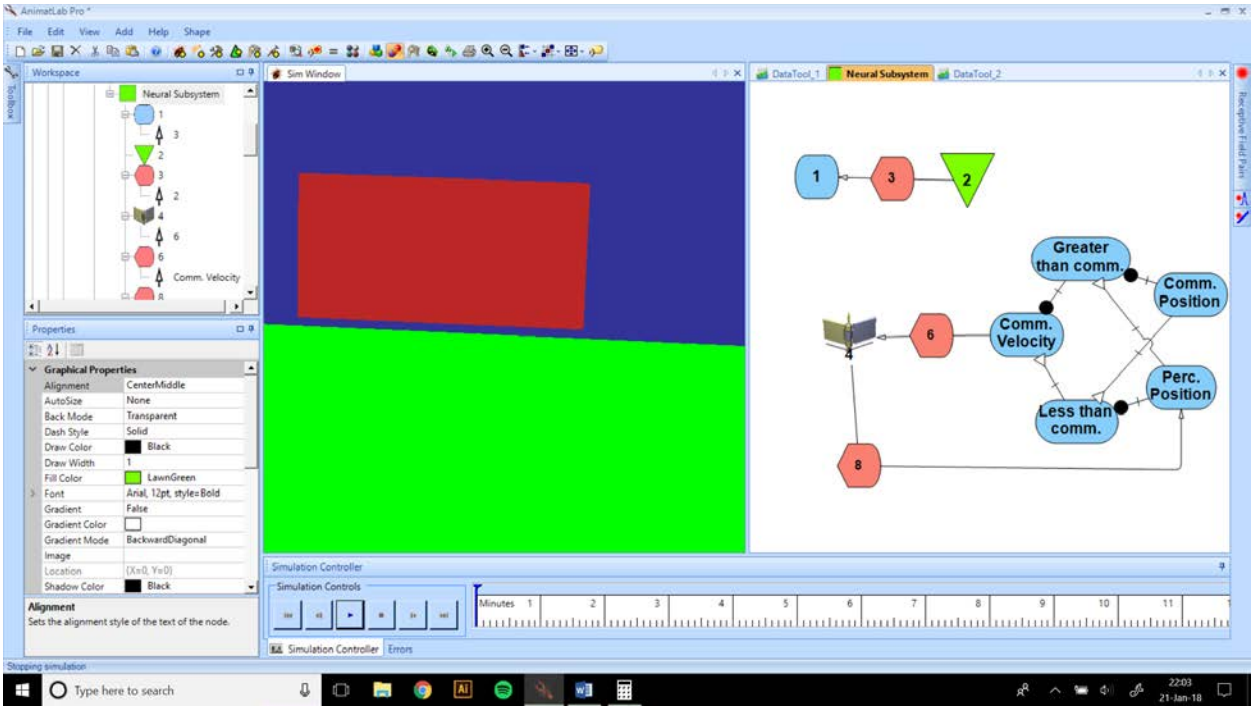

Set the parameter values to be the same as our previous stimulus: 5 nA for 1 s, then -5 nA for 1 s, between t=2 s and t=8 s.



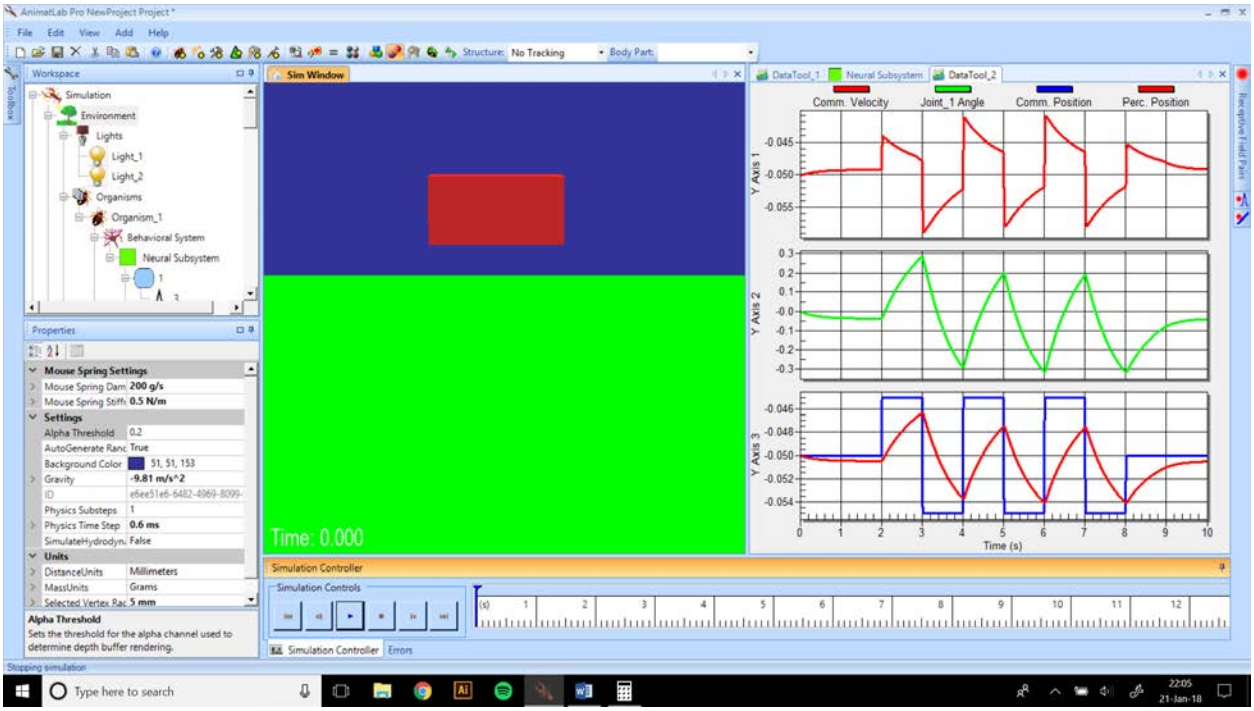

Run the simulation. Unfortunately, we do not get our intended outcome. Instead of the joint going up and down, it can only go up, because there is only one, excitatory input to neuron 5, which commands the motor's velocity. We can expand this controller by adding a mirror structure. The structure above will increase the velocity if the position is too low; we can add another structure to decrease the velocity if the position is too high.



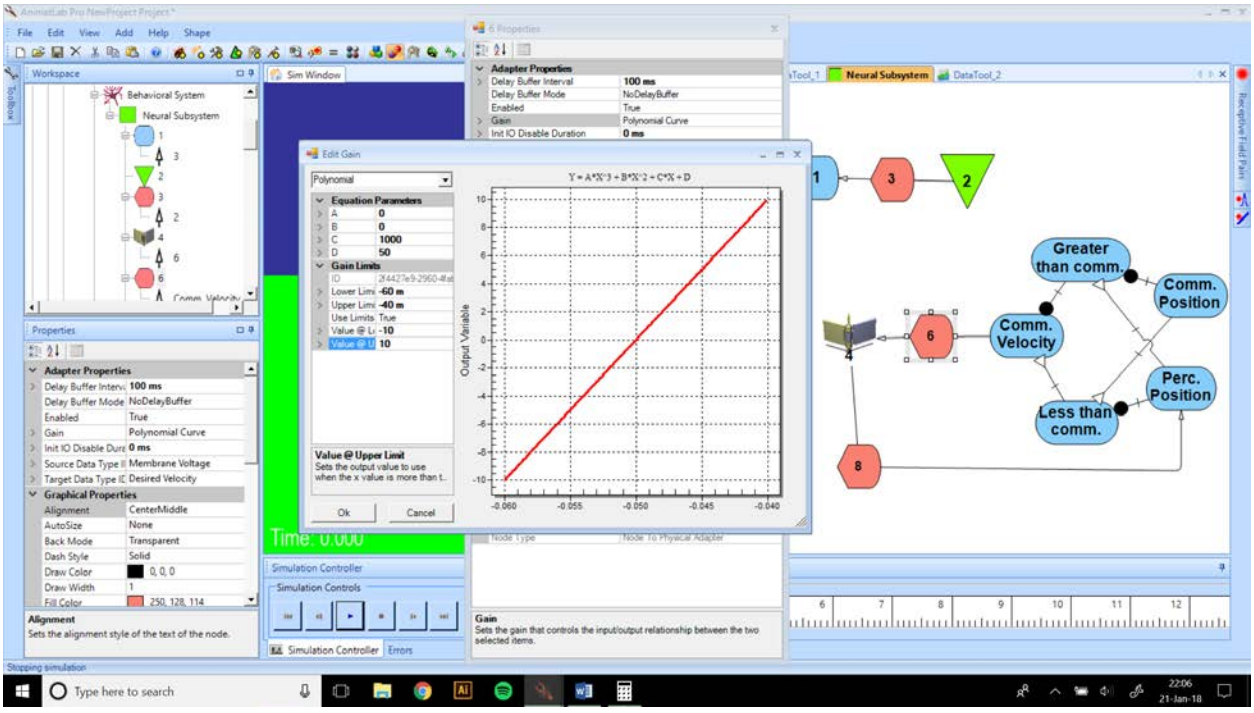
I have added such a structure, and renamed the neurons to have meaningful names. Now we have a “Comm. Position” and “Perc. Position” (perceived position). Subtracting them one way excites the “Greater than comm.” neuron, which reduces the velocity, while subtracting them the other way excites the “Less than comm.” neuron, which increases the velocity.



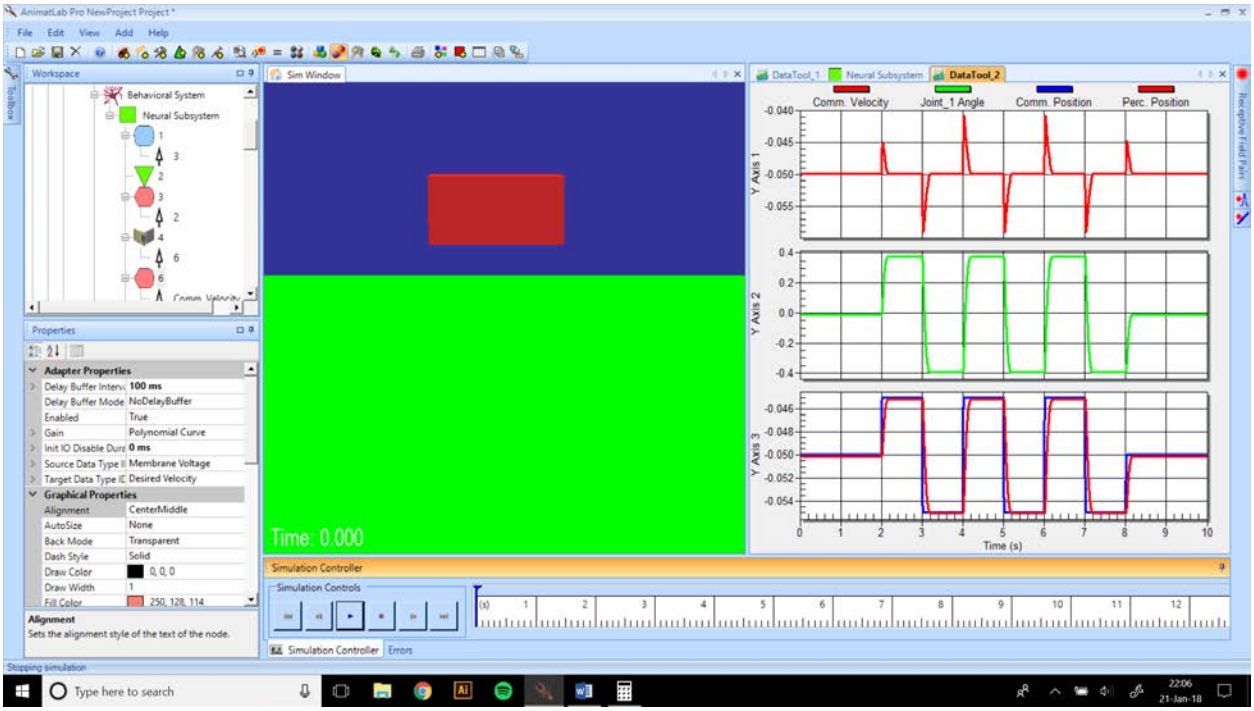

Running the simulation shows that this generally works: The “Perc. Position” trace follows the “Comm. Position” trace, albeit with an impractically long time constant. We can fix this by increasing the gain of the controller.



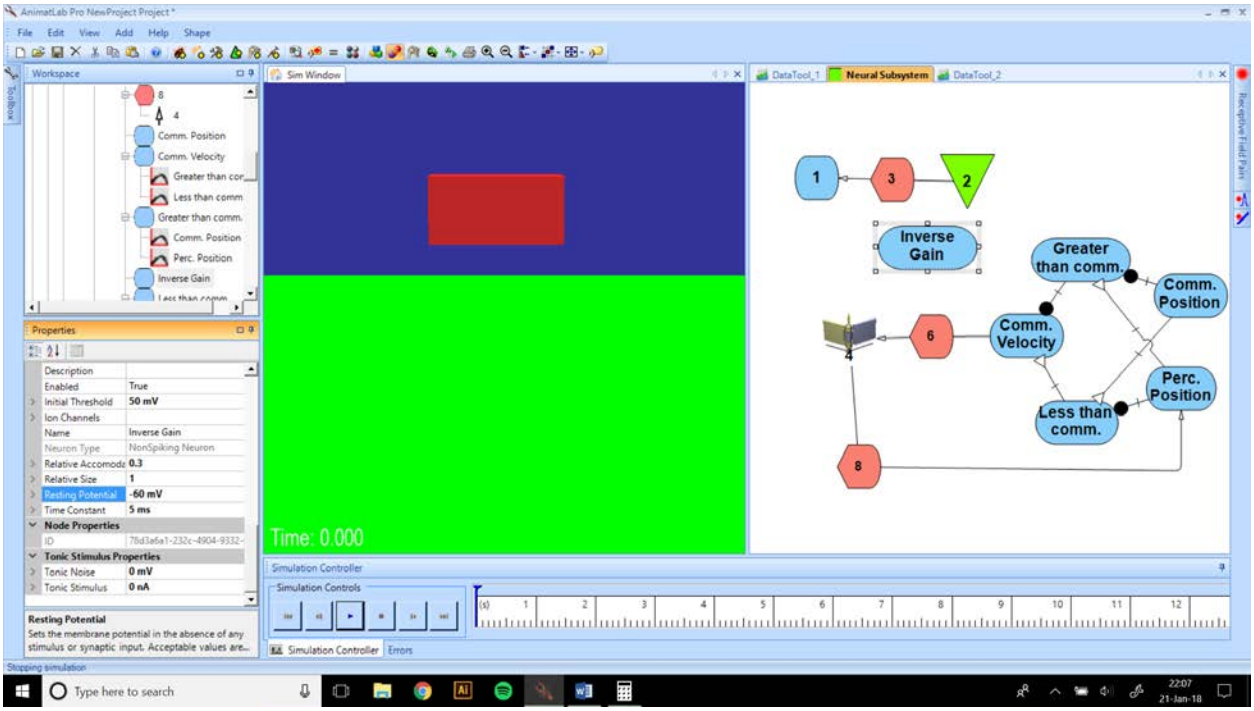

Double-click adapter node 6, which maps the “Comm. Velocity” neuron voltage to the velocity of the motor. Increase the slope by a factor of 10. Change the D parameter, and limit parameters to match the screenshot below.



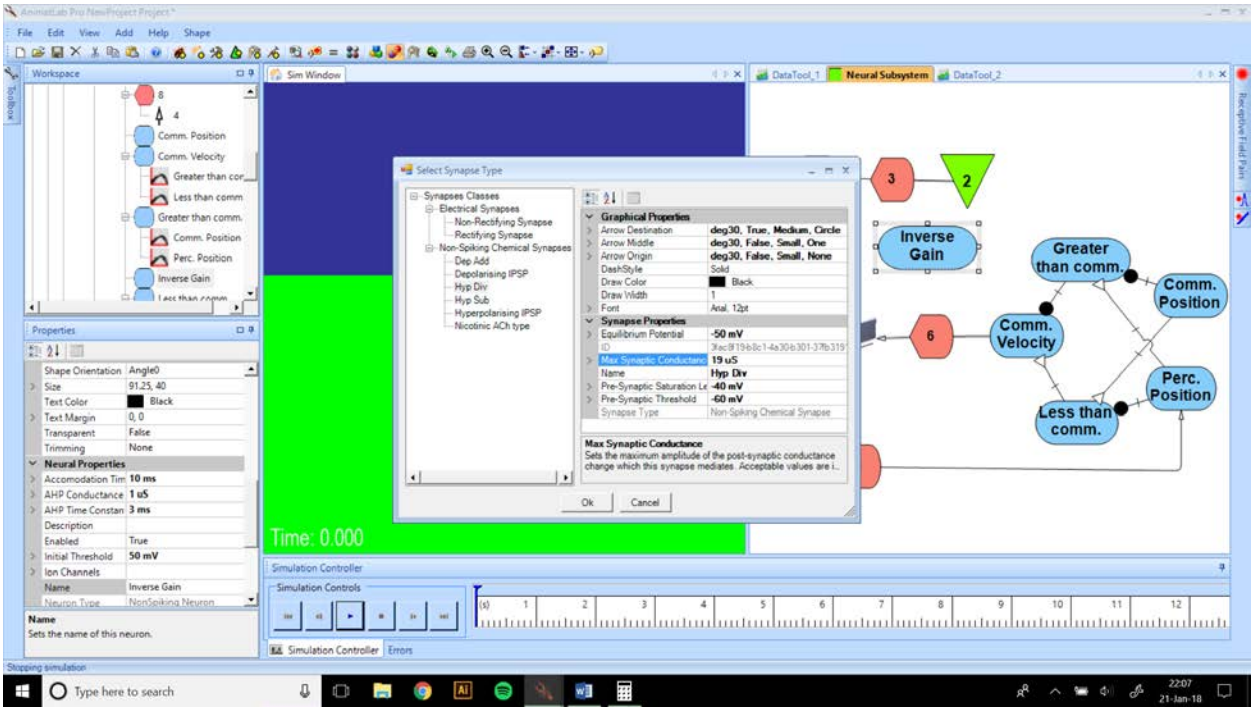

Run the simulation. Now, the controller performs very well, with the “Perc. Position” trace almost overlaid on the “Comm. Position”. But animals adjust the stiffness of their limbs all the time. Can we somehow achieve the same effect?



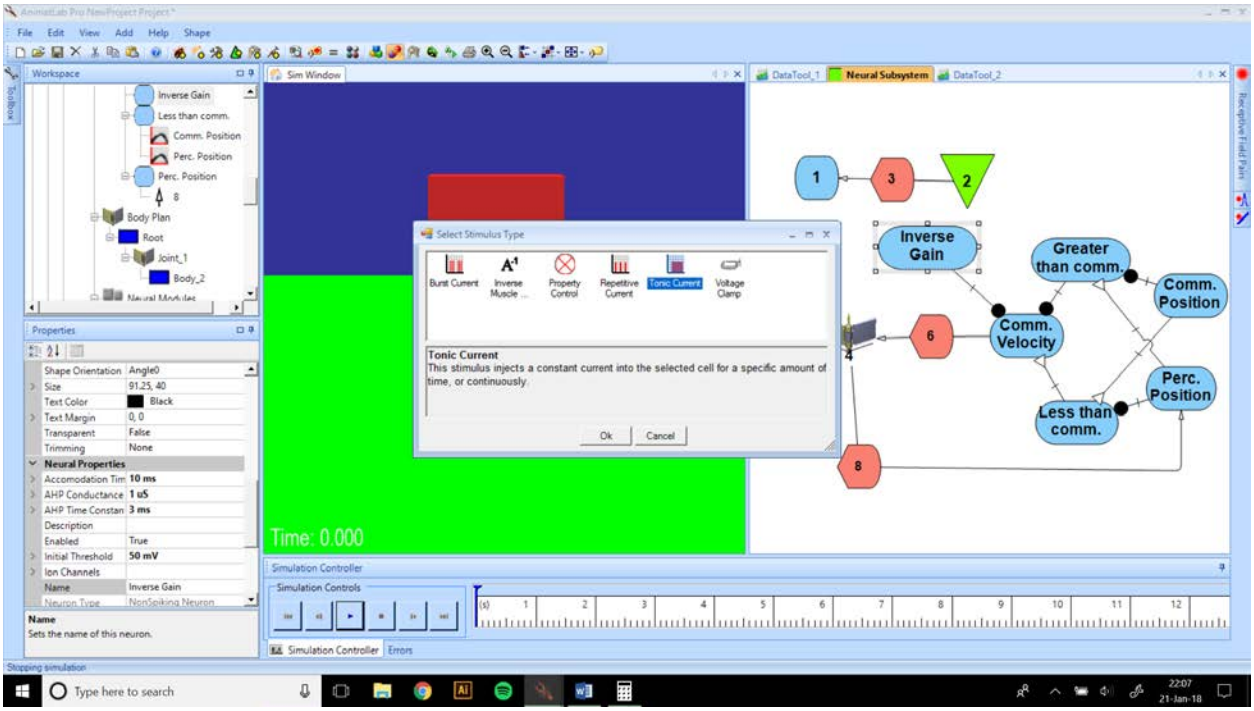

Add a new neuron, rename it “Inverse Gain”, and set its resting potential to -60 mV.



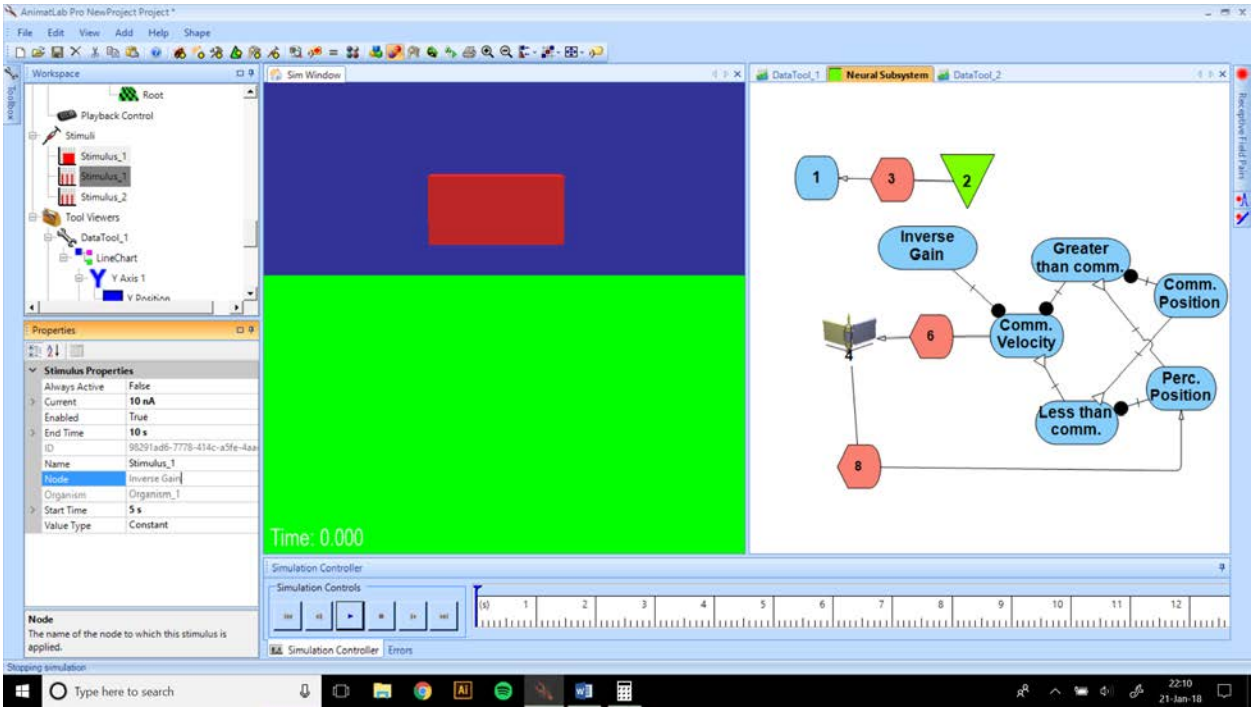

Click and drag a synapse from “Inverse Gain” to “Comm. Velocity”. When the Synapse Type window appears, clone the “Hyp Sub” type, and rename it “Hyp Div”. Change the parameters to match the screenshot.




Next, add a stimulus to “Inverse Gain”. This time, select “Tonic Current”, and click OK.




Set the “Start Time” to 5 s, and the “End Time” to 10 s. This will reduce the gain of our network by a factor of 10 when t is between 5 s and 10 s.




Run the simulation. The effect of the “Inverse Gain” neuron is clear: when that neuron is active, the controller is much less stiff, and looks very much like our first functional position controller, above.