Project 4 IT3708 Torgeir

A)

The genotype is represented as a byte array with length corresponding to the total number of weights, time constants, biases and gains needed in the network. The phenotype is a NeuralNetwork and effectively consists of three arrays of Neural nodes, each array corresponding to a layer in the network. In creation of a network, the byte array from the genotype gets distributed to each node so that each node receives the correct amount of bytes. The node also receives arrays of nodes whose values are used for activation. When an individual node is created, the byte values are converted and scaled to fit the given intervals for the different parameters.

When using the network, one sets the values for the input nodes based on the sensor data given from the problem logic. Then the network calls the activation function of the nodes in the hidden layer, then the nodes in the output layer. The result is the output values in the output layer nodes. The action to be taken depends on these values. In the pull scenario, the move action is taken if the new output neuron has a value higher than a set value, like 0.7. Otherwise the action is taken based on the values of the other output nodes. If the first node has a value below 0.5, then the movement will be to the left, otherwise it will be to the right. The magnitude is given by the value of the second node. I use a hard coded array to set the limit between each magnitude. At the moment, the array is {0.2, 0.45, 0.5, 0.7}. The magnitude is simply the index of the first of these number that is greater than the value, or 4 if all of them are smaller than the value. The resulting move in encoded as an integer, simply the movement in the x-direction, which is negative when going to the right. The pull move has the value 42.

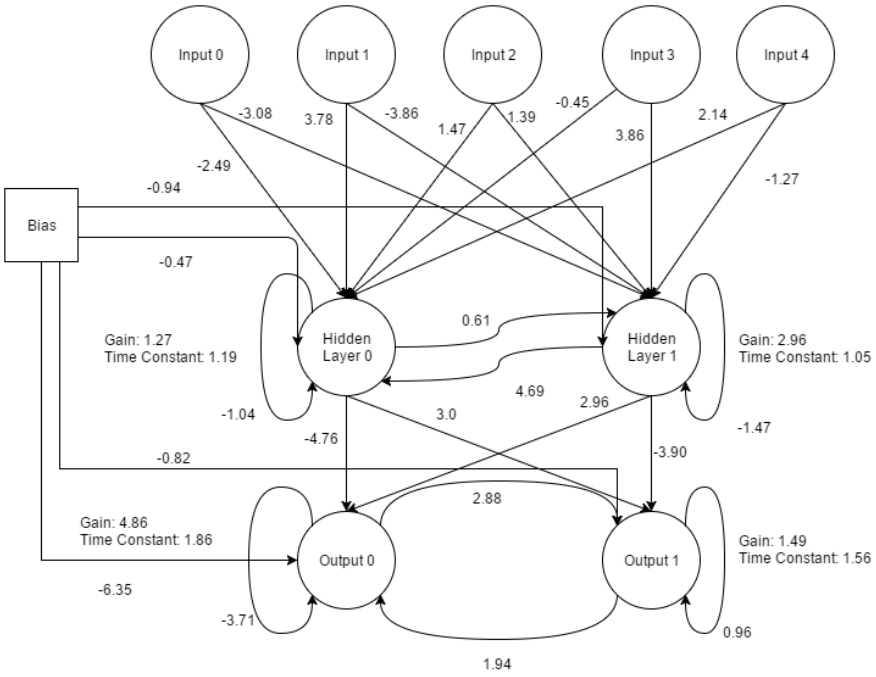
B)

The fitness function is simply put an addition of values at each timestep dependant on what happens.

```
if(          output == Impact.VOID)          value += 0.0;
else if(     output == Impact.AVOIDBIG)      value += 20.0;
else if(     output == Impact.AVOIDSMALL)    value -= 100.0;
else if(     output == Impact.CATCH          value += 25.0;
else if(     output == Impact.HIT)           value -= 100.0;
else if(     output == Impact.SMALLPART)     value += 5.0;
else if(     output == Impact.BIGPART)       value -= 30.0;
```

Where void is used for the timesteps when the object has not yet fallen to the bottom. The result is a hit if the entire tracker is hit by an object, regardless of it having size five or six, even though the latter would technically be the same as a big part.
The standard scenario, my EA usually evolves agents that consistently move to one directions, usually to the left. They move until they find an object, then move slowly, and stop if it is small and move along if they are big. If the next object spawns closely to the left, then the agent passes under it and catches it on the next pass.
For the pull scenario, the behaviour is mostly the same. The agent will pull instead of waiting. If the next object spawns closely to the left, then the agent might pull even though the new object is big.
For the no wrap scenario, the agent makes more mistakes, but is usually able to move the other way after hitting a wall.

C)



One evolved network has weights like given in this diagram. Looking at the first step with this configuration, all the nodes apart from the input layer start with a value of 0.0. The inputs are 0, 0, 0, 0 and 1, because we were lucky and got the leftmost part of an object above the tracker. In this case, the hidden layer nodes got values of 1.40 and -1.62 respectively. So they got outputs of 0.85 and 0.01 respectively. In the output layer, the nodes got values of -5.58 and 1.09. In the end, their output values were 0.00 and 0.84. Since the first of these values were less than 0.5, and the second was greater than 0.7, the result was a left move of magnitude four.

At timestep 86, the sensor data revealed that there was an object above the rightmost three units of the tracker. The object is six units long. Before calculating the move, the internal state values were 1.46, -2.15, -7.80 and 1.82 respectively. After computing the move, the values change to 1.74, 3.33, -6.76 and -0.06. The end outputs resulted in moving two units to the left. Later, on timestep 280, the sensor data is the same again, this time because an object of size three is above the rightmost three units of the tracker. This time the internal state has values of 3.19, 0.82, -6.50 and -0.50 before the move. After the move, these values change to 5.53, 2.23, -7.05 and -1.08. This resulted in no movement, because the output value of the last node ended up as 0.17, which is small enough for there to be no movement. Previous move was of magnitude one, so the change in internal state did not have to be big. The state in all the nodes changed in the same direction in both of these cases, but not by the same amount.

An interesting note about the weights is that on the edges from the input nodes, the sign on the weights change from input node to input node. This must be to make the agent stop if the object is short enough to only cover the middle three sensor or fewer. Otherwise, the agent needs to assume that the object has parts it cannot sense.