Motivations:

* Create a scenario where the child encounters a monster outside their bedroom. Initially they are afraid but if they have their teddy bear then they become more courageous and the monster won’t frighten then.

Objectives:

* Create a scenario where the child is initially afraid of the monster but becomes courageous if the teddy bear is out.
* The monster’s participation in the network will cause both weights to decrease.
* When the teddy bear is out it will cause both weights to increase the more the monster participates.
* When the bear leaves the child’s courage will increase the more the monster participates.
* Add some effects to scenario.

Software components:

* This scenario contains one world class, six actor classes, and two imported class.
* Actor: Button
  + This actor provides common methods and property for all buttons.
  + It contains a constructor that sets the button image. There is a click property that is used to indicate whether the image has been clicked once or twice. The class contains a getter and setter method. The setter sets the click value based on number of times image was clicked. The getter is used to check the value of the click property.

* Actor: Lightning
  + This actor controls how the lightning gif is displayed and when it’s played in the world.
  + Once the actor is created the gif, which is composed of several lightning images, is scaled down and a scale black background is placed in the world until the monster button is pressed. The class contains a setter method that allows the gif to play in the world. There is a method that only allows the gif to play once if it does play in the world. Uses methods from imported GifImage class.
* Actor: Monster
  + The monster that the child encounters. This monster can influence how afraid the child becomes when it appears. This is determine by one of its properties: toleranceFactor. The range of this factor is between 0 and 1. The higher the factor the less influence the monster has over the child and vice versa. This class contains methods that increase and decrease this factor based on whether the teddy bear is present in the world. Also contains a getter method to retrieve the factor.
* Actor: Teddy
  + Child’s teddy bear. Like the monster the bear can influence the child’s behavior. In this case it will make the child more courageous. The bear has a factor (braveFactor) that ranges from 0 to 1. The higher the factor the more influence the bear has over the child and vice versa. This class contains methods that increase and decrease this factor based on whether the teddy bear is present in the world. Also contains a getter method to retrieve the factor.
* Actor: Window
  + A prop image that does nothing meaningful.
* Actor: Counter
  + Imported class from Greenfoot library. In this scenario it will display how courageous (positive number) and afraid (negative number) the child is during each iteration.
* World: BedRoom
  + This class adds all necessary objects to the world and instantiates the neuro-network and its copy (2d arrays), input layer (1d array), and threshold. The neuro-network copy is here to prevent any changes from occurring in the original until weight adjustment. This neuro-network contains one neuron that accepts two inputs. An output of 1 means the child has gain some courage and both weights should increase. Output of 0 indicates the child has become more frighten so both weights should decrease.
  + Under act() the class will check if a button has been pressed. When the monster button is pressed the lightning gif will play and the monster and teddy bear factors will increase (if teddy bear button is pressed) or decrease (if teddy bear button is not pressed). These updated factors will then be stored in the input array. Next it pass the neuro-network and its copy, input array, and threshold to neuronOutput(). The code also checks if the effect button is clicked.
  + The method neuronOutput() is where the Hebbin algorithm is applied using the step function as the activation function with output y. If y is 1 then the counter (called braveMeter) will increase by 2. Else it will decrease by 2.

How to run the program:

* The network is trained while the program is running. Pressing the monster button will cause the counter on the top left of the screen to update. It should become more negative after several button presses. Next press the teddy bear button and a bear image should appear. Press monster button serval more times and the counter should become more positive. To perform a test click on teddy button and the image will be removed. Now click the monster button several more times. It should become more positive. Eventually the counter will start to decrease. The effect button allows you to turn effects on or off.

Results:

* The counter decreases when the monster appears and increases when the teddy bear is with the monster. When the bear leaves the monster’s appearance will cause the counter to increase for a while, then it will decrease after several iterations. The numerical distance of the weights becomes smaller with each iteration, so sometimes the counter increases when it’s not supposed to.

Discussions:

* Strength
  + Fast learner.
* Weakness
  + Numerical distance of the weights becomes smaller with each

iteration (but they never equal each other). Will infrequently cause

incorrect results to occur. For example the braveMeter may increase

when its suppose to decrease.

* + Very forgetful (quickly forgets what it learned).