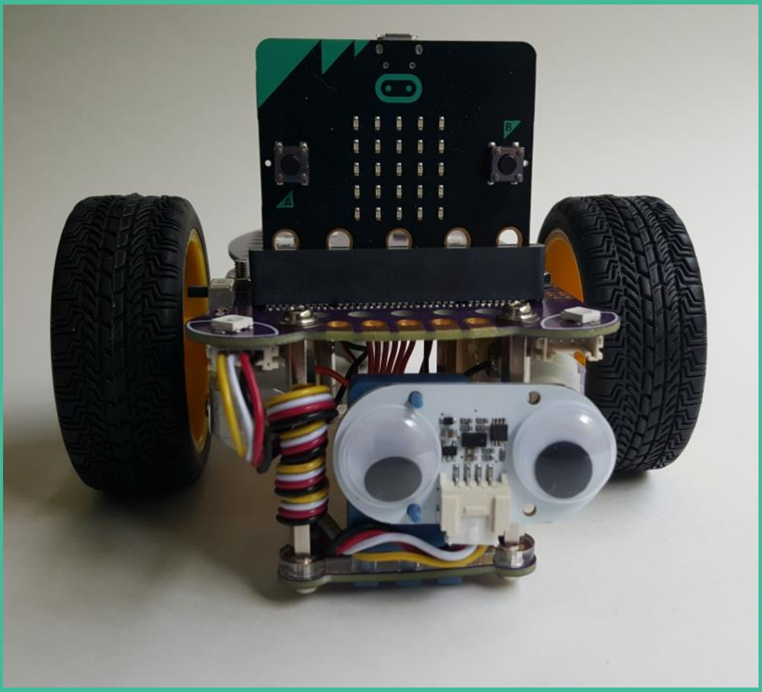
SCAREDY CAT

MISSION 4



GiggleBot

TABLE OF CONTENTS

• SCAREDY CAT 1
Your Role: Zoologist 1
Your Task: Investigate a New Creature 1
Considerations 1
Materials 1

• LEARN: CONNECT THE DISTANCE SENSOR 2

• LEARN: WHILE LOOP 3

• LEARN: USE THE DISTANCE SENSOR 4

• LEARN: WHAT IF 7

• PLAN IT OUT 9
Build Your Creature 9
Build Your Program 9

• ARE YOU STUCK?? 10

• TRY IT OUT 12
Iterate 12
Extension 13

> SCAREDY CAT

YOUR ROLE: ZOOLOGIST

ⓘ A zoologist is a scientist who studies animal behaviors.

YOUR TASK: INVESTIGATE A NEW CREATURE

You have just discovered a new species of animal and are investigating its behaviors. This creature reacts to getting close to objects.

CONSIDERATIONS

- Will the animal be afraid of obstacles and run away from them? Or, will the animal look for obstacles and then “attack” them?
- What will the animal look like?
- Will the animal mimic behaviors of other animals you know about?

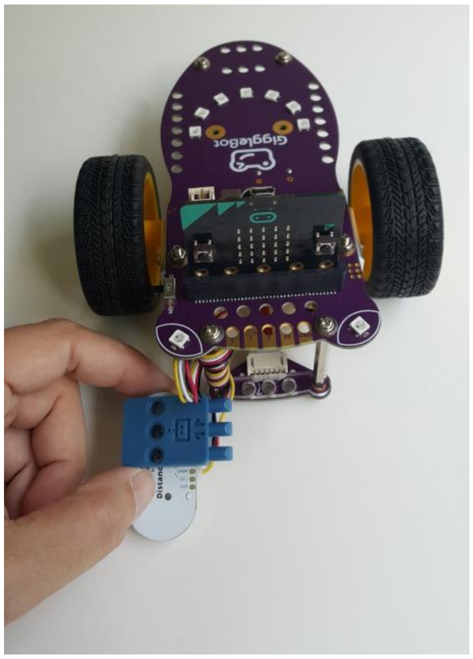
MATERIALS

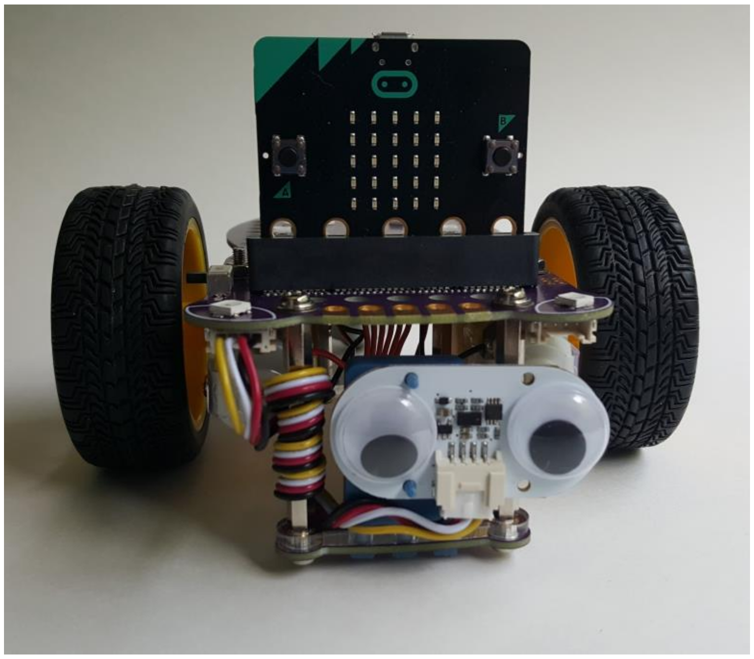
- GiggleBot and good batteries
- micro:bit and provided cable
- Laptop / computer
- Distance sensor and sensor mount
- Optional: craft supplies to transform the GiggleBot into an animal such as construction paper, pipe cleaners, or cardboard.

> LEARN: CONNECT THE DISTANCE SENSOR

Connect the distance sensor to either one of the ports on the front of the GiggleBot. The ports are located right underneath the LED eyes.

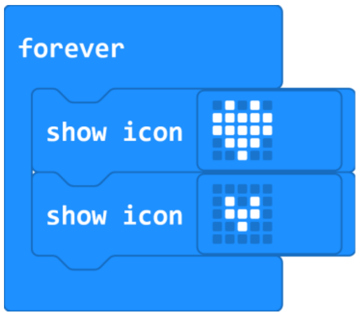
Attach the distance sensor to the GiggleBot using the blue sensor mount.





The distance sensor works by sending out a pulse of light in front of it. It then measures the time it takes for the light to reflect off of something and return to the sensor to measure how far it is from an object. Using this sensor your GiggleBot can detect obstacles and react to them.

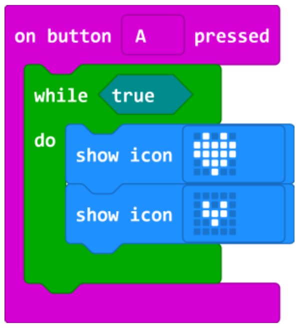
> LEARN: WHILE LOOP



Before coding our project, let's take a small detour and have a look at how to do code that repeats itself. We'll be coding a beating heart as our example. All these blocks are found under **Basic.**

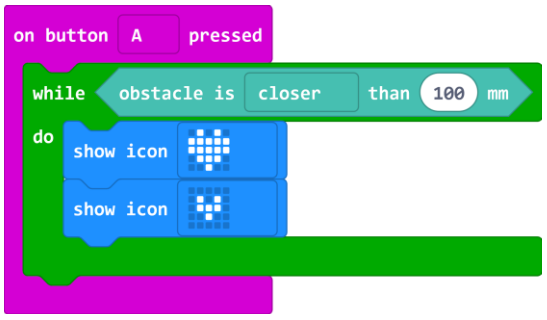
This code will display a beating heart on the micro:bit screen. It will start as soon as the micro:bit is turned on, and will beat until the micro:bit has no power. Transfer this code to your GiggleBot to see it in action.

WHAT IF WE WANT TO CONTROL WHEN THE HEARTBEAT STARTS?



Code this sequence and transfer it to your micro:bit. Now the beating heart starts when button A is pressed. The "**while true do**" loop (found under **Loops**) is almost like a forever loop, but we get to control its starting point. Does the heart stop beating when you let go of button **A**? No, it keeps on beating and it will beat forever or until the micro:bit loses power.

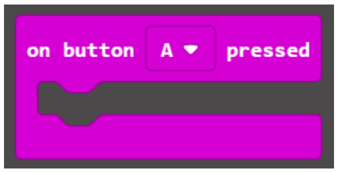
WHAT IF WE WANT TO CONTROL WHEN THE HEARTBEAT STOPS?



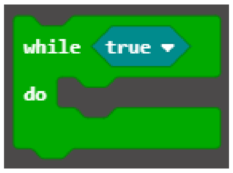
We can change the **true** part of the **while** loop to any condition we want. If that condition ever turns out to be false, the **while** loop will exit! The condition "**obstacle is closer than 100 mm**" is found in the **GiggleBot** category, near the bottom. You may have to scroll down.

- Transfer this sequence to your micro:bit and try it.
- Put your hand really really close to the distance sensor, almost covering it, and press button A. The beating heart will start when you press the button.
- Remove your hand and the beating heart will stop.
- If you want to restart the beating heart, it's tempting to cover the distance sensor again but that's not the proper starting condition. Can you figure out what will start the heart again? What happens if you press button **A** but do not have your hand in front of the distance sensor? (the heart never beats because the condition is **false**)

> LEARN: USE THE DISTANCE SENSOR

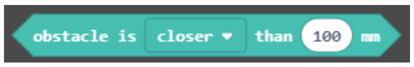


Let's begin by moving an block **on Button A pressed** (found under **Input**) to the workspace. We used this block in Mission 2: Light Show to initiate all or part of a program to control the lights on the GiggleBot. This time we will use it to start a different type of program.

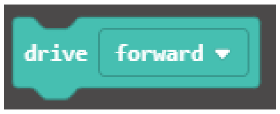


Place a block **while true do** (found under **Loops**) inside of the **on Button A pressed** block .

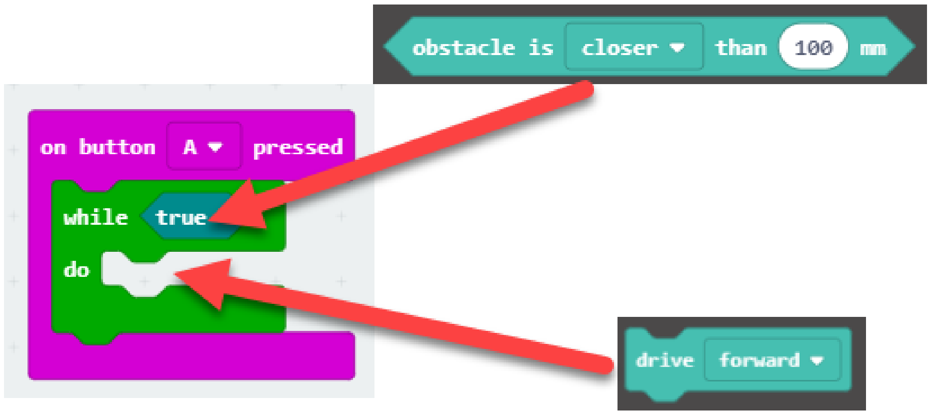
Instead of waiting until the GiggleBot is close to something to react, we are going to tell it to drive forward *while* it is a safe distance away from an obstacle.



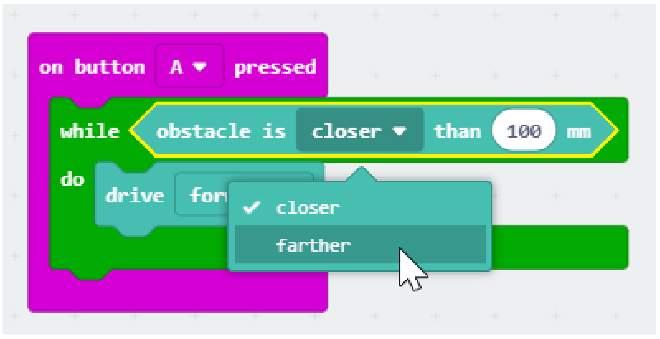
Place an **obstacle is closer than 100 mm** block (found under **GiggleBot**) in the **while... do...** block where it says true. That means that the GiggleBot will *do* whatever is connected to the **do** portion of the block **while** the condition is **true**.



Connect a block **drive forward** to the **do** portion of the **while... do...** block. This is the action that the GiggleBot will do inside the **while** block, over and over and over, and over.



If we ran this program, then the GiggleBot would drive forward when the distance sensor reads a value less than 100 mm (10 cm). This is the *opposite* of what we want it to do!!!



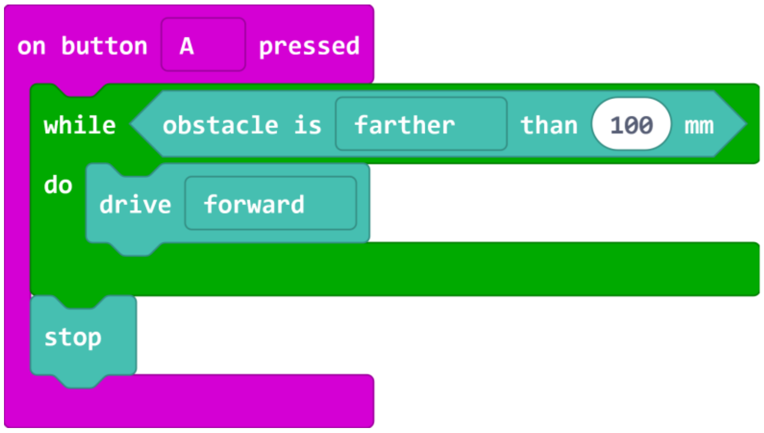
Use the drop-down menu to change **closer** to **farther**.

Last, we need to tell the GiggleBot what to do once the condition is no longer met. In this example we are going to tell the robot to *stop* so that it does not run into the object. Connect a **stop** block to the outside of the **while-do** block. We do not want the GiggleBot to stop if an object is a safe distance away, only if it gets too close.

HOW CLOSE IS TOO CLOSE?

How close do you think the GiggleBot should get to an object before stopping? Change the **100** to your desired value.

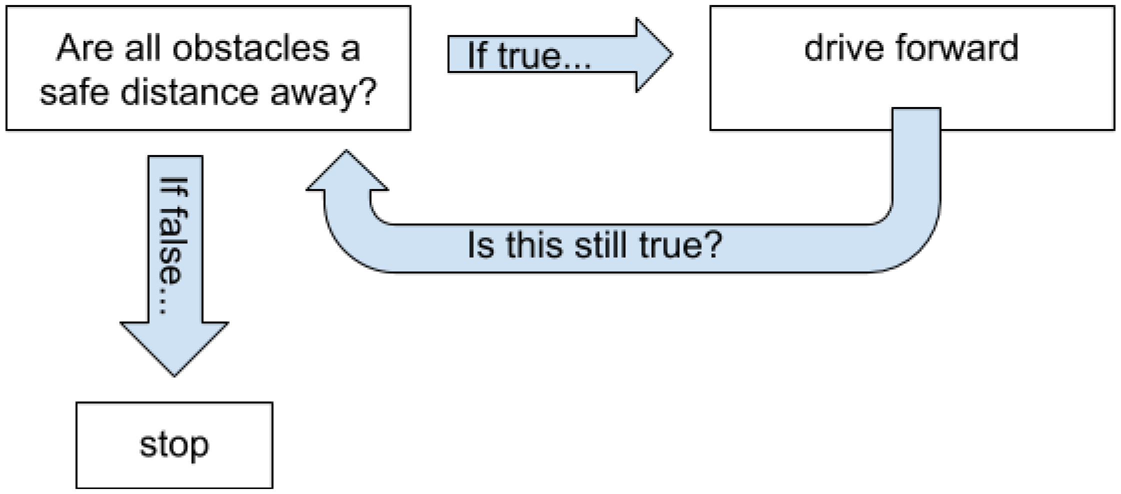
ⓘ ** This measurement is in millimeters (mm). 10 millimeters = 1 centimeter **



Create a name for your program and save it. Then, transfer the file to your GiggleBot and test it out.

LOOP CODE

This is what happens when that sequence of code gets run. The condition **obstacle is closer than 100 mm** gets tested over and over again. As the command **drive forward** is quick to execute, the loop is done extremely fast.



ⓘ CAUTION: If the body of the **while** loop takes a long time to execute (for example, cycling a rainbow for many seconds) then the condition gets tested only after it finished executing. It does NOT get tested continuously. The robot can miss obstacles that way.

ANIMAL REACTION

> What else could you add to the program?

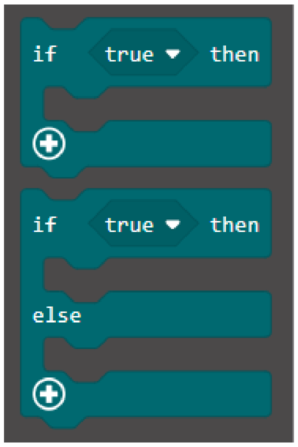
> Do you want to turn on LEDs when an object gets close?

> Do you want it to turn around?

> Create a couple more versions of this program by adding in more movements and/or LEDs.

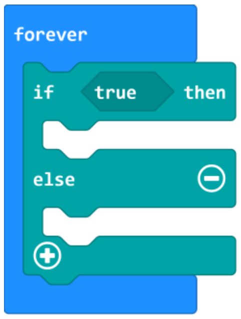
> LEARN: WHAT IF

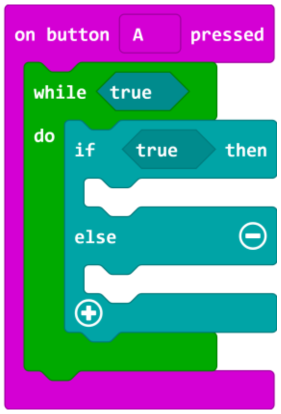
Now, we are going to use a new type of block to program the GiggleBot to do more than just stop if it comes close to an object. It will react in some way. Maybe it will turn and run away. Maybe it will turn on different color LEDs. Maybe it will rapidly spin in a circle. It is up to you to decide how your GiggleBot will react to an object.



> Find the "**if ... then ...**" blocks under **Logic**. These blocks tells the robot to do an action if a value is *true*. Notice there is an "**if ... then ...**" block and an "**if ... then ... else ...**" block.

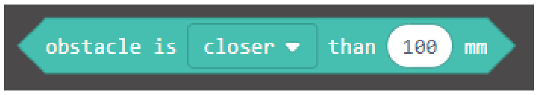
> If you click on the **+** sign you can modify these blocks to add either an **else ...** or an **else if ... then** statement.

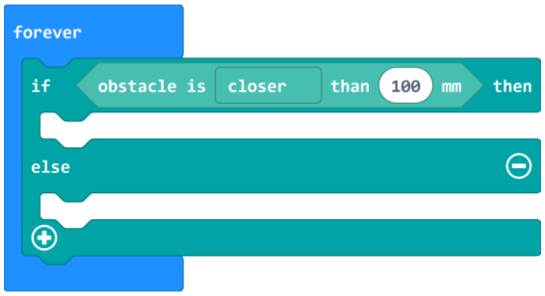




Choose one of the two following options:

1. Move a forever block into your workspace and place an **if then ... else ...** block inside
2. Use a **on button A pressed** with a **while true** combination instead if you prefer to control when the program starts.



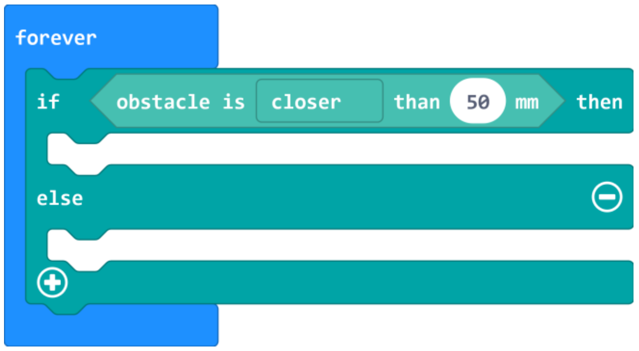


Next, we need to determine what values we want the GiggleBot to be looking for. Since we are using the distance sensor, think about how close you want the robot to get to an object before reacting.

Do you want it to stop when it is 100 mm away? 50 mm away? One way to do this is to use a block **"obstacle is closer than __ mm"** again. Connect this block to the "**if then …**" portion of the "**if then … else …**" block where it says **true**.

You can use the drop down menu on the block to change "**closer**" to "**farther**" depending on what you want the GiggleBot to do. Change the "**100**" value to tell the GiggleBot when to react.

ⓘ ** The distance is measured in millimeters (mm). 10 mm = 1 centimeter. **

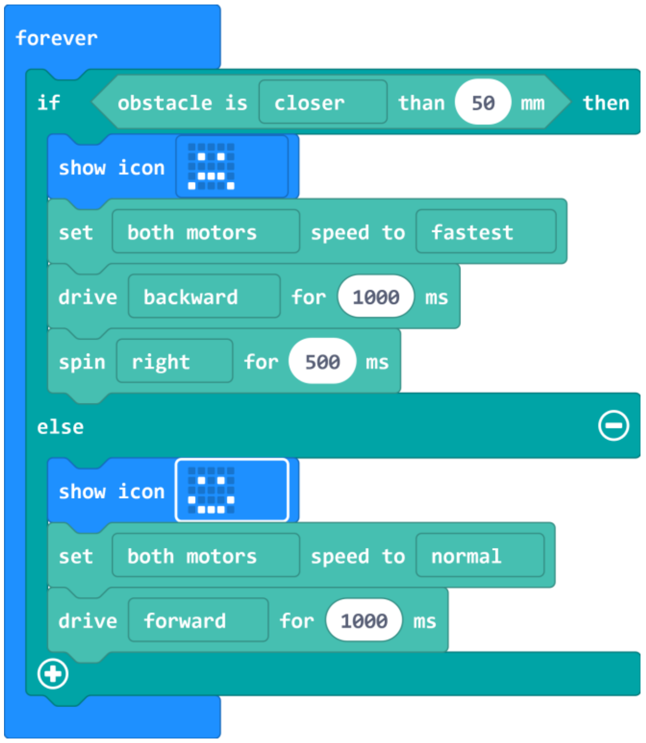


```
forever
    if  obstacle is  closer  than  50  mm  then

    else

```

> In this example, the value was changed to 50 mm. This means the GiggleBot will do an action when the distance sensor reads a value less than 50 mm, in other words the GiggleBot will react if an obstacle is within 50 mm of the distance sensor.



```
forever
    if  obstacle is  closer  than  50  mm  then
        show icon
        set  both motors  speed to  fastest
        drive  backward  for  1000  ms
        spin  right  for  500  ms
    else
        show icon
        set  both motors  speed to  normal
        drive  forward  for  1000  ms
```

> Inside the **if then…** block opening, add blocks to tell the GiggleBot what to do if it gets closer than _____ mm to an obstacle. In this example, the GiggleBot will display a sad face, drive backwards quickly, and then turn around when an obstacle is detected that is closer than 50 mm.

> Inside the **else …** block opening, add blocks to tell the GiggleBot what to do if there isn't an obstacle. In this example, it will drive forwards with a happy face if there is no obstacle.

When you are finished writing your program, name your program and save it. You can transfer it to your GiggleBot to try it out.

> PLAN IT OUT

Consider the following questions as you plan out your creature. Jot down your thoughts on a piece of paper and sketch out what your creature will look like.

- What is your creature going to look like?
- Will it use the LEDs on the micro:bit to show emotions?
- Will it use the smile LEDs on the back to change colors like a chameleon?
- Does your creature like to get close to objects?
- Does it fear getting close to things? How will it react?

BUILD YOUR CREATURE

Use craft materials and recyclables to make your GiggleBot look like an animal. You could add things such as a tail, arms, legs, ears, and/or a nose. Ensure that the added materials do not cover the distance sensor or impede the movement of the wheels.

BUILD YOUR PROGRAM

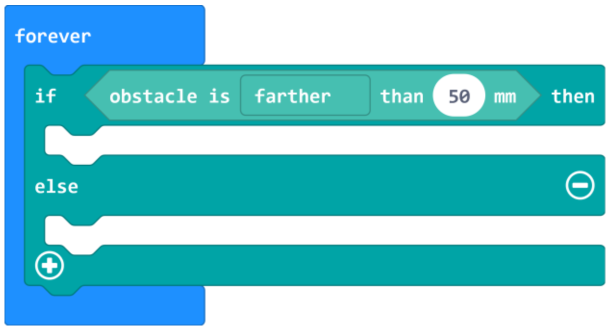
Use what you learned in the Learn section to program your GiggleBot. You can remix the programs you created earlier or write new ones.

> ARE YOU STUCK??

Let's build a program together!

Since we already programmed the GiggleBot to run away from obstacles in the **Learn** section, now we are going to program the GiggleBot animal to "attack" obstacles.

- > Move a "**forever**" block and "**if then … else …**" block onto the workspace.
- > Next, put an "**obstacle is closer than __ mm**" block in to the space that says **true**.
- > Change **closer** to **farther.**

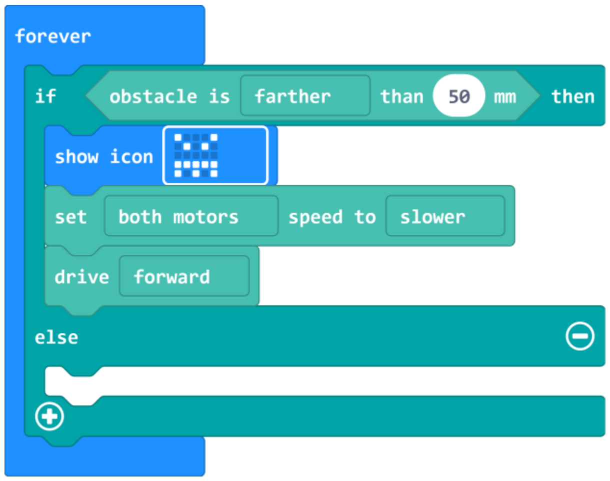


```
forever
  if obstacle is farther than 50 mm then
  else
```

Now we need to decide what the GiggleBot animal will do if an obstacle is farther than the distance you decided on, or if there's no obstacle at all.

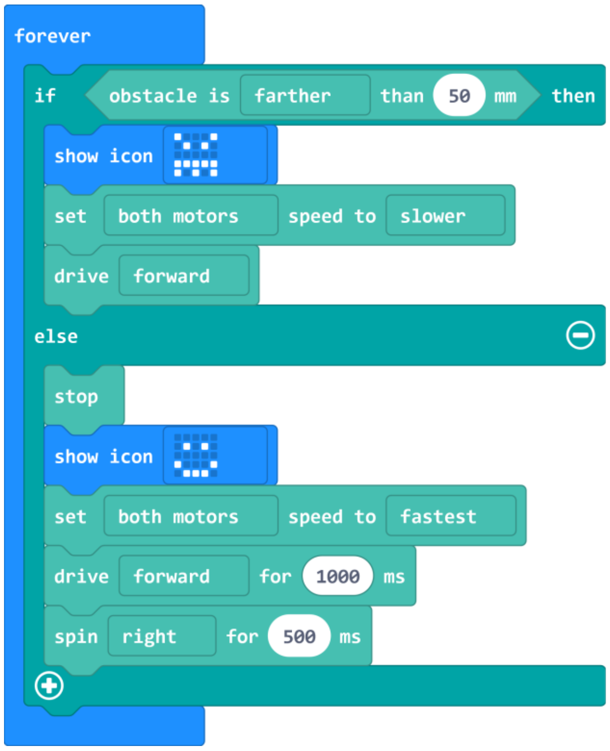
- > Do you want it to drive around slowly? Quickly?
- > Display a smile? Display a grimace?

Place the appropriate blocks for your choices after then. Or look at the example for inspiration.



```
forever
  if obstacle is farther than 50 mm then
    show icon
    set both motors speed to slower
    drive forward
  else
```

Next, we will choose what the GiggleBot will do when an object is detected less than 50 mm away. Instead of having the GiggleBot fear obstacles, this one will "attack" them gleefully. What blocks could you place after **else** to tell the GiggleBot to "attack" the object? An example is shown here.



```
forever
    if obstacle is farther than 50 mm then
        show icon
        set both motors speed to slower
        drive forward
    else
        stop
        show icon
        set both motors speed to fastest
        drive forward for 1000 ms
        spin right for 500 ms
```

- Name your project
- Download your file
- Transfer it to the GiggleBot.

ⓘ Congratulations! You just created a creature that seeks out obstacles and then attacks them!

> TRY IT OUT

Run your program to see if your GiggleBot avoids or attacks obstacles. If not, you may need to adjust the speed at which the GiggleBot moves and/or the distance(s) at which it reacts.

ITERATE

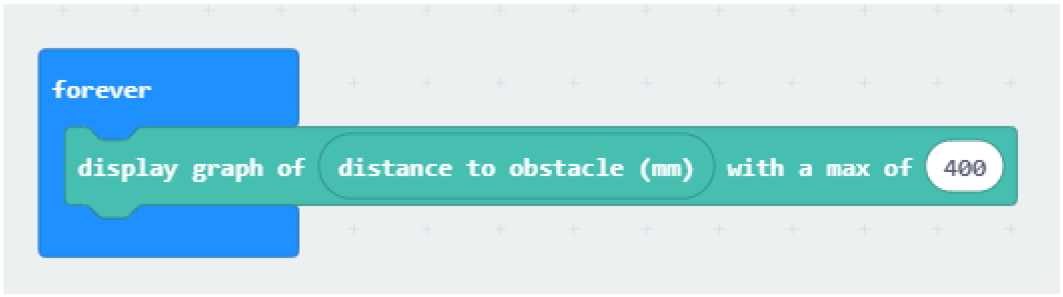
- Did your GiggleBot avoid (or attack) obstacles as you anticipated?
- Do you see any areas that you would like to modify or improve upon? In engineering, we try a solution, think about what needs to change, and we iterate.
- As you modify and revise your program, save each new program with a version number – you never know when you might want to take another look at an older version (for example: Distance_V1 where "V" stands for version).

EXTENSION

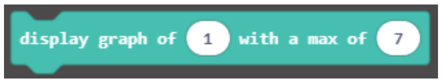
Think about what you learned in the previous three missions. Find a way to add in use (or more use) of the onboard LEDs to your program. Or, utilize the pins on the GiggleBot to further control the robotic animal.

GIVE ME A SIGN

Build the program below and transfer it to your GiggleBot.

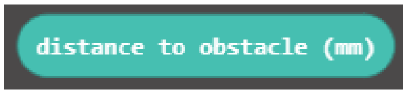


```
forever
    display graph of (distance to obstacle (mm)) with a max of (400)
```



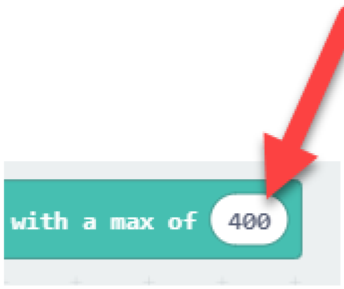
```
display graph of (1) with a max of (7)
```

- **display graph of __ with a max of __** is found under **Lights**



```
distance to obstacle (mm)
```

- **distance to obstacle** is found under **GiggleBot → more**.



```
with a max of (400)
```

- Change the max value

- Place your hand in front of the distance sensor and then move it away. What happens?
- Vary the distance between your hand and the distance sensor
- What happens if you change the maximum value?
- What could this program be used for?
- Can you think of ways to integrate this into your existing code?