Overview
In this activity, you will work with coding concepts that utilize loops. Three basic loops \( \text{loop}(), \text{for}(), \) and \( \text{while}() \) will be used to control various outputs on your robot, Bee-Dee. In this scenario, Bee-Dee is a robotic prototype of a smart car that is still under development and has encountered system failures during a shakedown, or the initial test run of a product under development.

One of the fundamental types of control structures for C-based text programming is a loop. Loops are meant to do precisely what the name implies. They provide a method of repeating behaviors defined by lines of code set between braces. How the different types of loops control the repeating behavior is the focus of this activity.

Real-World Connection
Code is needed to blink Bee-Dee’s lights, specifically the red light on the PRIZM board, which will be our first example of outputs controlled by PRIZM. This blinking will take the form of a SOS call for help. SOS has long been the standard communication that indicates help is needed. It originated with Morse code (an early precursor to telephones that involved dots, dashes, and spaces) and was then used for communication from point to point using flashing lights (such as on a ship). In popular usage, SOS stands for “Save Our Souls” or “Save Our Ship.”

The blinking of the SOS signal can be done in a variety of ways using coding. We are going to look at three different loop structures: the main \( \text{loop}() \), \text{for}() loop, and \text{while}() loop.

As we work through these loops (and the remainder of the activities within this guide), we are going to write pseudocode to help us translate our thoughts and words into computer code.

Coding Essentials Covered
- Pseudocode
- \text{Main loop}()
- \text{for}() loop
- \text{while}() loop

Materials Needed
- Built Bee-Dee robot
- Test-rig platform
- Computer with Arduino Software (IDE) and PRIZM library installed
- USB cable
- Student logbook (spiral notebook or equivalent)
- Digital camera or smartphone with a camera (optional)

Fun Fact: The SOS signal technically does not consist of three individual letters, but is a continuous string of three dots, three dashes, and three dots (no spaces are included between letters).

Note: The shakedown is a critical part of the development process for the hardware and software portion of a product. As the hardware is tested to see what happens under certain conditions, we can determine if the parts work, how the parts work together, and how the software interacts with the hardware. Any unexpected issues can be addressed before the product development goes any further, and benchmarks can be set to provide a sense of what is expected with further prototypes.
Plan the Work

Writing pseudocode is much like writing a paper or a story in words. For efficient and effective writing, an outline of the story or paper is a good first step. It is the first step in planning your code.

An outline breaks down your story into significant, logical parts. The sequence of these parts might be critical to the reader understanding the story. If you are writing a mystery, putting the identity of the villain in the front of the story will not provide much suspense. Logically, the revelation of the villain's identity would be one of the last parts of the story.

In coding, the pseudocode outline is simply putting the solution to the problem in word form.

Pseudocode is commonly referred to as “fake code.” This is because if you typed the pseudocode directly into the computer within the Arduino Software (IDE) environment, it would tell you that there are errors because it cannot understand the pseudocode language. Pseudocode is language that you understand, not the computer. As you begin to write your sketch, you will translate what you have written in words into commands that the computer understands.

Our problem to solve with Bee-Dee involves flashing the red LED (light) on the PRIZM controller. This is closely related to the Hello World! activity that was done in the PRIZM Programming Guide. To flash the red LED on and then off three times, we could outline this process in words such as:

- Red LED on
- Red LED off
- First blink
- Red LED on
- Red LED off
- Second blink
- Red LED on
- Red LED off
- Third blink

The pseudocode above should cause the robot to flash its red LED three times. If we wanted to have the robot flash the red LED five times, we could add more to the outline.

You will notice that most of this code is very repetitious. Essentially, four lines are executed over and over – each time turning the LED on for a second and off for a second. Writing out pseudocode will sometimes alert you to the fact that there might be other ways to solve the problem – that your code could be condensed somehow.

Tip: The red LED is located at D6 on the PRIZM board.
In the PRIZM Programming Guide, you were introduced to the `loop()` control structure, sometimes called a main `loop()` to help distinguish between other types of loops. Loops within sketches are known as structure statements, indicating how the code that follows will be executed. If we changed our pseudocode to take advantage of a main `loop()` structure, here is what that pseudocode might look like to keep the red LED flashing:

Repeat the following steps until the code is aborted:

- Red LED on
- Red LED off

Notice that parts of this pseudocode are indented – just like portions of outlines are indented. This helps keep the steps organized, which will help you write the code in the language that PRIZM will understand. The main `loop()` (indicated by the first statement of pseudocode) sets up the structure of the code that follows – it will keep repeating everything until the code is aborted.

Let’s do a side-by-side comparison of the pseudocode and affiliated C-based code:

<table>
<thead>
<tr>
<th>Pseudocode</th>
<th>C-Based Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeat the following steps until the code is aborted:</td>
<td>void loop(){</td>
<td>Brace () indicates the start of the loop</td>
</tr>
<tr>
<td>Red LED on</td>
<td>prizm.setRedLED(HIGH); delay(1000);</td>
<td>Turns on the red LED and pauses for a second (1,000 ms) with the red LED on</td>
</tr>
<tr>
<td>Red LED off</td>
<td>prizm.setRedLED(LOW); delay(1000);</td>
<td>Turns the red LED off and pauses for a second with the red LED off</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
<td>Brace () indicates the end of the loop and begins executing the four lines of code over and over until the code is aborted (the Stop/Reset button is pressed)</td>
</tr>
</tbody>
</table>

Samples of pseudocode will be provided throughout these activities. After you have done a few of these, you will get into a rhythm in writing them, and it will become second nature to you.

**Work the Plan and Execute the Code**

Now that you've created your plan or outline of your code, you're ready to see how the plan stands up when you create the actual code for your first sketch! Welcome to the world of coding!
The Main loop()

The loop() structure is unique in that while it is the simplest and most basic of loop structures, it can also be considered the most important. It is so important and fundamental to C-based, text-driven programming that it is commonly described as a main loop(). Almost all sketches or programs will contain at least one of these structures. It is simple because we can count on everything defined in the braces to repeat over and over until we either abort the program or include other means of stopping the sketch. Let’s see exactly how it works.

1. Start the Arduino Software (IDE). If a new sketch window does not automatically load, go to the pull-down menu File > New.

2. Connect Bee-Dee with a USB cable from the computer to the PRIZM controller mounted on Bee-Dee. Turn on Bee-Dee with the on/off switch by moving the switch to the on position. You will be able to confirm you have power by observing the blue LED glowing on the PRIZM.

3. After Bee-Dee is connected and powered up, access the pull-down menu Tools to check two different things that will confirm the computer recognizes Bee-Dee and will be able to communicate (upload) to her. Look at Tools > Board and confirm you see Arduino/Genuino Uno as the option shown. If not, please choose Arduino/Genuino Uno as the board type. Under the Tools > Port option, you should see either com # if you’re on a Windows computer or something that looks like /dev/cu.usbserial-AI02W2RB if you are on a Mac computer.

4. After we have confirmed Bee-Dee is connected to the computer and can communicate, we are ready to begin setting up a new sketch. A new sketch will automatically include the two most basic sketch structures: the void setup() and the void loop(). We need to add two important statements of every PRIZM-oriented sketch: the include statement for the PRIZM software library and the object declaration. In the sketch, above the void setup(), we can add the include statement by using the pull-down menu Sketch > Include Library > TETRIX_PRIZM. The object declaration can be added by typing “PRIZM prizm;” — when you’re done, it should look something like this:

```
sketch_apr17a

#include "PRIZM.h"

PRIZM prizm;

void setup() {
}

void loop() {
}
```

Tip: The include statement is used to add the functionality of the PRIZM library. The object declaration is used to help manage programs as they grow in size and complexity. For more of a refresher, refer to the PRIZM Programming Guide, Activity 3.
5. Initialize the PRIZM controller in the sketch by adding `prizm.PrizmBegin();` to the `void setup()` function.

```cpp
#include <PRIZM.h>
PRIZM prizm;

void setup() {
    prizm.PrizmBegin();
}
```

6. Translate the steps from your pseudocode into C-based statements inside the `void loop()` structure function, remembering our syntax rules so that we end each statement with a semicolon to indicate that it is the end of that line of code. This is not the complete code, but the beginning should look something like this:

```cpp
void loop() {

    prizm.setRedLED(HIGH);
    delay(500);
    prizm.setRedLED(LOW);
    delay(500);

    prizm.setRedLED(HIGH);
    delay(500);
    prizm.setRedLED(LOW);
    delay(500);
}
```

7. Verify your code to make sure you have no errors by using the **Verify** button.

8. Click the **Upload** button in the sketch menu to upload the code to Bee-Dee.

9. Press the green Start button to start the code. The red LED should flash on and off in one-second intervals until the red Stop/Reset button is pressed.
Now, let’s return to our problem to solve: How do we get Bee-Dee to signal an SOS signal with her red LED?

An SOS signal is a series of three dots (dots have a short duration, about 0.5 seconds) immediately followed by three dashes (dashes are three times as long as dots, so they are about 1.5 seconds) immediately followed by three more dots. The length of time in between SOS signals can vary, but for our purposes, let’s have an interval of five seconds between each SOS.

Consider the pseudocode and C-based code we just input that blinks the red LED. We are going to develop code that will use the main loop() to signal SOS until the code is aborted.

1. In your logbook, write pseudocode that would output the three dots, followed by pseudocode that would output the three dashes, followed by pseudocode that would output three more dots.
2. Follow the pseudocode from Step 1 with pseudocode that would insert a five-second delay before starting over.
3. When your pseudocode is complete, start a new sketch with all the necessary setup steps and then begin translating your pseudocode into C-based instructions. Try to solve this challenge on your own or with your partner if you are in a team. If necessary, ask your teacher for assistance if you get stuck in this process. He or she might provide you with a hint of how to proceed or share examples of pseudocode and C-based code to help you.
4. After your pseudocode is translated and typed into the sketch, upload the code to Bee-Dee and see how it works.
5. After the code runs successfully, save the file as directed by your teacher.
6. In your logbook, jot down notes about the success of the code and if there were any parts of the code that had to be revised so that the code worked correctly.
7. If you have access to a camera, take a quick video of Bee-Dee flashing the SOS signal. Your teacher might ask to see it at some point.
8. Compare your sketch to a sample sketch that solves this problem. Your teacher will provide this sketch in print or digital format for you. Record in your logbook how your code and the sample code differ.

Tip: Keep in mind that there are often multiple strategies in coding that result in a similar or identical outcome. This is something to remember through the rest of your coding journey. It’s often a very cool experience to learn a new and different way to create a familiar outcome. It can keep coding fun and fresh no matter how long you’ve been a coder.

Tip: The process of finding and solving coding issues is known as troubleshooting. Troubleshooting is an essential skill for programming. It requires patience and a keen eye. One of the best ways to troubleshoot is to compare the pseudocode to the C-based code and see if your translation makes logical sense.
The for() Loop

Now, let’s look at another way to send SOS signals for Bee-Dee – using the for() loop. The for() statement repeats a set of statements enclosed in braces for a specified number of times. A counter is usually used to count the number of times the set of statements has been executed, and when the counter is finished, the loop is terminated.

The for() loop looks like this:
```
for (initialization; condition; increment) {
  // block of commands
}
```

The initialization indicates where the counting starts for the loop. Typically, it starts at zero and uses integers to count upward. So, in our initialization portion of this statement, we will declare that our variable x is going to be an integer “int” and that its value will begin at zero: x=0.

The condition is usually a test. For our code to solve this problem, we want the value of x to be less than or equal to three. (If we start with zero, counting up four times will put us at three.) So, we write the condition to test where our variable is within the counting process: x<=3. Each time through the loop, the value of x is tested – is it less than three or is it equal to three? If the answer is yes, then we go through the loop again. If not (when x becomes greater than three), then we exit the loop and move on to the next line of code.

The increment of the loop determines how much the variable within the loop (x in this case) increases or decreases each time through the loop. We want x to increment by one each time through the loop. This is written as x++. This means that the value of x is incremented by one.

In this case, we will use the for() loop to have SOS signaled four times before stopping. Notice that the for() loop sets the conditions at the beginning of the sketch. It checks to see how many times the loop has executed before it starts another loop.

When the for() loop is complete, we need to include a prizm.PrizmEnd(); statement to stop the code from continuing to execute.

Tip: An integer is a positive or negative counting number, such as one, two, negative three, zero, negative 19, and so on. Integers have no fractional or decimal components.
Let’s write our pseudocode to use with the for() loop.

1. In your logbook, look at the pseudocode that you developed for the main loop(). Some of that code can be reused to configure a for() loop. What part of the code do you think is usable?

2. Begin writing your for() loop pseudocode in your logbook. For the for() loop, we might start our pseudocode like this:

   Repeat the following steps four times and then stop.

3. What is it that we want to repeat over and over for four times? It’s the part of the code that creates the SOS, so our pseudocode might go something like this:

   Repeat the following steps four times and then stop:
   - Flash the red LED three times at a rate of 1/2 second per flash.
   - Flash the red LED three times at a rate of 1-1/2 seconds per flash.
   - Flash the red LED three times at a rate of 1/2 second per flash.

4. Write your version of the pseudocode in your logbook.

5. Open a new sketch with all the necessary setup steps and then begin translating your pseudocode into C-based instructions. The for() loop needs to begin with how many times the instructions within the loop will repeat. This statement will look like this:

   void loop() {
   for(int x=0; x<=3; x++){
    }
   prizm.PrizmEnd();
   }

6. From that point, translate your pseudocode for the statements that should be within the loop. Again, seek help if you get stuck.

7. After your pseudocode is translated and typed into the sketch, upload the code to Bee-Dee and see how it works.

8. After the code runs successfully, save the file as directed by your teacher.

9. In your logbook, jot down notes about the success of the code and if there were any parts of the code that had to be revised so that the code worked correctly.

10. If you have access to a camera, take a quick video of Bee-Dee flashing the SOS signal 10 times. Your teacher might ask to see it at some point.

11. Compare your sketch to a sample sketch that solves this problem. Your teacher will provide this sketch in print or digital format for you. Record in your logbook how your code and the sample code differ.

Tip: Remember our troubleshooting skills. One of the best ways to troubleshoot is to compare the pseudocode to the C-based code and see if your translation made logical sense.
The while() Loop

The last loop we are going to utilize is the while() loop. In this loop, a set of instructions keeps looping while a certain condition within the parentheses of the statement is met. That condition, you should notice, is checked before the set of instructions has been executed.

The while() loop is often used with sensors, executing the loop while the sensor values are within a certain range. We will learn more about using while() loops with sensors in a later activity.

The while() loop looks like this:

```java
while(expression){
}
```

The expression within the while() loop can be a mathematical expression (such as \(x==5\), \(2>x>5\), or \(t>8\)), or it can be an expression that reads the status of a button on PRIZM. If reading the status of the green Start button, then the while() loop statement would look like this:

```java
void loop() {
    while(prizm.readStartButton()==1){
        ...
    }
}
```

To begin running the sketch, you press the green Start button. That will start the while() loop, which waits for the green button to be pressed again. When the green button is pressed the second time, it returns a value of one, and the loop begins to execute, signaling SOS. If the green button is not pressed, then a value of zero is returned, and the loop waits until it sees a value of one – meaning the button was pressed a second time.

Let's write some pseudocode to perform the while() loop.

1. The first statement says what the while() loop is going to do:
   - Wait to do the following until the green button is pressed.

2. From that point, the pseudocode would be very similar to, if not the same as, pseudocode that you wrote with the for() loop. Our complete pseudocode for flashing SOS would look something like this:
   - Wait to do the following until the green button is pressed.

```plaintext
Flash the red LED 3 times at a rate of 1/2 seconds per flash.
Flash the red LED 3 times at a rate of 1-1/2 seconds per flash.
Flash the red LED 3 times at a rate of 1/2 seconds per flash.
```

Each of these lines could have several lines of code involved.
3. Write your version of the `while()` loop pseudocode in your logbook. Seek help if you get stuck.

4. Open a new sketch and write C-based code for the `while()` loop, referring to your pseudocode as needed.

5. After your pseudocode is translated and typed into the sketch, upload the code to Bee-Dee and see how it works.

6. After the code runs successfully, save the file as directed by your teacher.

7. In your logbook, jot down notes about the success of the code and if there were any parts of the code that had to be revised so that the code worked correctly.

8. If you have access to a camera, take a quick video of Bee-Dee flashing the SOS signal until the Start button is pressed. Your teacher might ask to see it at some point.

9. Compare your sketch to a sample sketch that solves this problem. Your teacher will provide this sketch in print or digital format for you. Record in your logbook how your code and the sample code differ.

Wave That Arm

Up to this point, we have focused on using loops with blinking lights. Truthfully, with only two lights available (even though one is red and one is green), there is only so much fun you can have with blinking lights. They are great as indicators, and we will use them as such in further programs, but now we get to move on to motion.

As you learned in the activities in the PRIZM Programming Guide, PRIZM can control the movement of servos through C-based code. If you don't remember much about this, you might want to review Activity 3 of that guide: Moving Your Servo Motors. What is important to note is that with servos, there are two basic functions involved with controlling a servo: servo speed and servo position.

It is also important to note that there are two types of TETRIX servos available: standard and continuous rotation. In this activity, we will be working with the standard servo that has been mounted on Bee-Dee.

The function `prizm.setServoSpeed()` has two parameters: the servo number and the percent of power used to rotate the servo. During Bee-Dee’s construction, the standard servo wire was connected to Servo Port 1, so that is the number we will use for the first parameter. Any number from 1 to 100 may be used for the power parameter. A function statement for Servo 1 moving at 35% power would look like this:

```c
void setup() {
    prizm.PrizmBegin();
    prizm.setServoSpeed(1, 35);
}
```
The second servo function, `prizm.setServoPosition()`, controls the position of the servo. The two parameters for the function are the servo number and the servo position. We are still using Servo 1, so that will be the servo number, and the servo position can range from 0° to 180°. A function statement for Servo 1 moving to a position of 160° would look like this:

```c
void loop() {
    prizm.setServoPosition(1, 160);
}
```

Remember that the movement of the servo to a certain position is not instantaneous. Depending on the power provided to the servo, it might take several seconds to move the servo to the given position. Many times, a delay will follow a servo position function to give the servo time to move before executing another statement:

```c
void loop() {
    prizm.setServoPosition(1, 160);
    delay(3000);
}
```

Notice that on Bee-Dee, there is a flat 64 mm TETRIX MAX element attached to the servo. This serves as Bee-Dee's arm. In this instance, we are going to have Bee-Dee wave her arm to signal that she is a damsel (or in this case, a smart car) in distress.

1. In your logbook, write pseudocode that uses a main `loop()` to have Bee-Dee move her arm up and down until the code is aborted.
2. Write C-based code to perform the action in the previous step.
3. Upload your sketch to Bee-Dee and run it. Troubleshoot the code and make any necessary changes so that the sketch works properly.
4. If you have access to a camera, take a short video of Bee-Dee as the sketch runs.
5. Repeat Steps 1-4, but use a `for()` loop as the code structure for the sketch.
6. Repeat Steps 1-4 one more time, using the `while()` loop code structure for the sketch.
Get Your Motors Running

It’s time to get Bee-Dee moving. As such, it is important to remember to place Bee-Dee on her test-rig before uploading a sketch that involves her DC motors. The platform will keep her wheels off the tabletop but will allow you to watch the motors run and adjust the code without fear of Bee-Dee running off the table. When Bee-Dee and the code are behaving properly, Bee-Dee can be disconnected from the USB cable, be placed on the floor, and execute the code.

DC motors most often drive wheels or gears to perform various tasks. You might have wondered, “What does DC mean?” Well, it could be a distant cousin of Bee-Dee – but it’s not. DC stands for direct current, a form of electricity in which the current (electrons) is always moving in one direction.

With Bee-Dee, the DC motors drive the wheels directly – for every revolution of the motor, the attached wheel revolves once as well. Mathematical computations can be done to determine how far Bee-Dee will go for each wheel rotation, as well as the length of time it takes to go that far. Of course, the time required will vary with the amount of power provided to the motor through the `prizm.setMotorPower()` function.

As you learned through the activities in the PRIZM Programming Guide, the DC motor control function `prizm.setMotorPower()` has two parameters: the motor channel number and the power percentage. The power percentage can provide several varieties of motor control:

- A positive number for the power percentage turns the motor clockwise (as you have the motor shaft pointed at yourself).
- A negative number for the power percentage turns the motor counterclockwise (as you have the motor shaft pointed at yourself).
- As the power percentage increases from one to 100, the speed of the motor increases.
- If the power percentage is zero, then the motor is in coasting mode, meaning it does not stop abruptly.
- If the power percentage is 125, then the motor is in hard braking mode to stop abruptly.

Remember, too, when motors are placed on different sides of a vehicle, they need to spin in opposite directions to make the vehicle move forward or backward. For a robot like Bee-Dee, spinning the two motors in opposite directions at the same power settings will cause her to move in a straight line. Spinning the motors at different power settings will cause Bee-Dee to turn. The greater the difference in the power settings, the tighter the turn will be.
The function `prizm.setMotorInvert()` enables you to invert the rotational direction of a motor. When two motors are mounted on opposite sides, this function enables you to give a single direction command to both motors and have them work together. This makes coding easier. There are two parameters to the function. The first parameter designates the motor channel, and the second parameter designates no invert (0) or invert (1).

Another function is available for DC motor control within sketches: `prizm.setMotorPowers()`. This function enables you to set the power level of Motor 1 and Motor 2 at the same time. The two parameters set the speed for each motor.

But for now, Bee-Dee is out on the road with a major malfunction. We have sent an SOS signal and waved her arm, but we need to get her off to the side of the road and out of traffic.

To do this, we need to have Bee-Dee turn slightly right and then left while moving forward. This moves her off the road and out of harm’s way. An overhead shot of her actions would look like this (symbolically):

1. In your logbook, write pseudocode that uses at least one `for()` loop to have Bee-Dee move off the highway.
2. Write C-based code that reflects the pseudocode and uses the `for()` loop within it to solve the problem.
3. Upload your sketch to Bee-Dee and run it. Troubleshoot the code and make any necessary changes needed so that the sketch works properly.
4. If you have access to a camera, take a short video of Bee-Dee as the sketch runs.
5. Repeat Steps 1-4, using the `while()` loop code structure within the sketch.
Hack the Code Challenge 2 Brief

Within this challenge, your team should choreograph a demonstration that would highlight as many safety features of Bee-Dee as possible, utilizing flashing LEDs, arm signals, and movements (forward, reverse, and turns). Think outside the box to come up with features that would dazzle a group of investors. Create a script that could be read while Bee-Dee is going through her demonstration, explaining Bee-Dee’s actions and how they make her a safe, roadworthy vehicle.

Materials Needed

• Built Bee-Dee robot
• Computer with Arduino Software (IDE) and PRIZM library installed
• USB cable
• Test-rig platform
• Digital camera or smartphone with a camera (optional)
• Student logbook (spiral notebook or equivalent)

Challenge Parameters

• Your demonstration should be about 30 seconds long.
• The code must include two different loop structures.
• The code must incorporate all outputs of Bee-Dee (lights, servos, and DC motors).

Jump Start

In your logbook, use pseudocode to help plan the flow of behaviors and necessary C-based code for the demonstration. When finished, take a video of your demonstration. Your teacher might have you run your demo (or show your video) for the entire class.

1. Refer to the Challenge Brief above and create a pseudocode plan to successfully demonstrate Bee-Dee’s outstanding safety features.
2. When you have created the plan for Bee-Dee’s demonstration, go through each step of the pseudocode. Make sure that you have included every action that will need to be taken.
3. Within the Arduino Software (IDE), open a new sketch, include the TETRIX PRIZM library <PRIZM.h>, and make the object declaration PRIZM prizm;
4. Initialize the PRIZM controller in the sketch by adding prizm.PrizmBegin(); to the void setup() structure function.
5. Save your sketch.
6. Translate your pseudocode into C-based statements.
7. Save the code and use Bee-Dee on the test-rig to actively test and tweak your program.
8. Did the code execute like you thought it would? If you must troubleshoot any portions of the code to make it work, note the changes you had to make in your logbook. As a portion of troubleshooting the code, you might want to ask your teacher for an example sketch (printed or digital) for one solution to this problem. You can compare your code to the example code and possibly determine how to modify your code to work. Remember, there are many ways to write code that produces the same output.

9. If you have access to a camera, take a video of Bee-Dee executing your code. Your teacher might have you run your demo (or show your video) for the entire class.

10. Within your logbook, provide answers to the following self-assessment questions:
   A. Did the code work correctly?
   B. Did your smart car (Bee-Dee) demonstrate the safety features effectively?

Ideas for Consideration

- Create a mixture of outputs in different orders as the demonstration moves forward.
- Demonstrate repeatability by placing Bee-Dee in the same starting position and then running the demo and showing how close Bee-Dee is to the starting position.