Activity 2.3.2 Robot Behaviors and Writing Pseudocode

Introduction

A behavior is anything your robot does: turning on a single motor is a behavior, moving forward is a behavior, tracking a line is a behavior, navigating a maze is a behavior. There are three main types of behaviors that we are concerned with: basic behaviors, simple behaviors, and complex behaviors.

Basic Behaviors

*Example: Turn on Motor Port 3 at half power*

At the most basic level, everything in a program must be broken down into tiny behaviors that your robot can understand and perform directly. In ROBOTC, these are behaviors the size of single statements, like turning on a single motor.

Simple Behaviors

*Example: Move forward for 2 seconds*

Simple behaviors are small, bite-size behaviors that allow your robot to perform a simple, yet significant task, like moving forward for a certain amount of time. These are the most useful behaviors because they are big enough that you can describe useful actions with them, but small enough that you can program them easily from basic ROBOTC commands.

Complex Behaviors

*Example: Follow a defined path through an entire maze*

These are behaviors at the highest levels, such as navigating an entire maze. Though they may seem complicated, one nice property of complex behaviors is that they are always composed of smaller behaviors.

```
task main()
{
  startMotor(RightMotor, 63);
  startMotor(LeftMotor, 63);
  wait(2);

  startMotor(RightMotor, -63);
  startMotor(LeftMotor, 63);
  wait(.5)

  startMotor(RightMotor, 63);
  startMotor(LeftMotor, 63);
  wait(2)
}
```

Basic Behavior – This code turns the right motor on at half power.
Simple Behavior – This code makes the robot turn on for .5 second.
Complex Behavior – This code makes the robot move around a corner.
The most important idea in behaviors is that they can be built up or broken down into other behaviors. Complex behaviors, like going through a maze, can always be broken down into smaller, simpler behaviors. These in turn can be broken down further and further until you reach simple or basic behaviors that you recognize and can program.

<table>
<thead>
<tr>
<th>Complex behavior</th>
<th>Simple behaviors</th>
<th>ROBOTC – ready (Basic) behaviors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Follow the path to reach the goal</td>
<td>Go forward for 3 seconds</td>
<td>1. Turn on left motor</td>
</tr>
<tr>
<td>Go forward 3 seconds</td>
<td>Turn left 90°</td>
<td>2. Turn on right motor</td>
</tr>
<tr>
<td>Turn left 90°</td>
<td>Go forward for 3 seconds</td>
<td>3. Wait 3 seconds</td>
</tr>
<tr>
<td>Go forward 5 seconds</td>
<td>Turn on left motor</td>
<td>4. Turn off left motor</td>
</tr>
<tr>
<td>Turn right 90°</td>
<td>Turn on right motor</td>
<td>5. Turn off right motor</td>
</tr>
<tr>
<td>Go forward 2 seconds</td>
<td>Wait 3 seconds</td>
<td>6. Reverse left motor</td>
</tr>
<tr>
<td>Turn right 90°</td>
<td>Turn off left motor</td>
<td>7. Turn on right motor</td>
</tr>
<tr>
<td>Turn off right motor</td>
<td>Turn off right motor</td>
<td>8. Wait 0.8 seconds</td>
</tr>
<tr>
<td>Turn off right motor</td>
<td>Turn on left motor</td>
<td>9. Turn off left motor</td>
</tr>
<tr>
<td>Go forward 2 seconds</td>
<td>Wait 5 seconds</td>
<td>10. Turn off right motor</td>
</tr>
<tr>
<td>Turn left 90°</td>
<td>Turn on right motor</td>
<td>11. Turn on left motor</td>
</tr>
<tr>
<td>Reverse left motor</td>
<td>Wait 0.8 seconds</td>
<td>12. Turn on right motor</td>
</tr>
<tr>
<td>Turn off left motor</td>
<td>Turn off right motor</td>
<td>13. Wait 5 seconds</td>
</tr>
<tr>
<td>Turn off left motor</td>
<td>Wait 5 seconds</td>
<td>...</td>
</tr>
</tbody>
</table>

Step by step
1. Start with a large-scale behavior that solves the problem.
2. Break it down into smaller pieces. Then break the smaller pieces down as well.
3. Repeat until you have behaviors that are small enough for ROBOTC to understand.

Sometimes it can be hard to tell whether a behavior is “simple” or “complex”. Some programs are so complex they need multiple layers of simple behaviors before they reach the basic ones! “Basic,” “Simple,” and “Complex” are categories of behaviors which are meant to help you think about the structure of programs. They are points of reference in the world of behaviors. Use these distinctions to help you, but don’t worry if your “complex” behavior suddenly becomes a “simple” part of your program.

Pseudocode is a compact and informal description of a computer program. It is a hybrid language which combines the features of the programming language with the native language of the person writing the program. Emphasis is placed on expressing the behavior or outcome of each portion of code rather than on strictly correct syntax (it does still need to be reasonable, though). In general, pseudocode is used to outline a program before translating it into proper syntax. This helps in the initial planning of a program, by creating the logical framework and sequence of the
code. It captures the logic and flow of a solution without the bulk of strict syntax rules.

Below is some pseudocode written for a program which moves as long as a touch sensor is not pressed, but stops and turns to the right if its sonar detects an object less than 20 in. away. This pseudocode example includes elements of both programming language, and the English language. Curly braces are used as a visual aid for where portions of code need to be placed when they are finally written out in full and proper syntax.

```
/*
task main
{
  while
  {
    if chair pushbutton pressed
    {
      if bottom floor limit switch is pressed
      {
        Chair moves up
        Top limit switch pressed
        Chair stops
      }
    }
    else (either the chair is at the top or midway)
    {
      Chair moves down
      Bottom limit switch pressed
      Chair stops
    }
  }
}
*/
```

Indicates multi-line comment

Some intact syntax – the use of a while loop in the pseudocode is fitting because the way we read a while loop is very similar to the manner in which it is used in the program.

Description – There are no actual motor commands in this section of the code, but the pseudocode suggests where the commands belong and what they need to accomplish.

Indicates end of multi-line comment

In this activity you will practice writing behaviors and pseudocode for projects you will build in future classes.

**Equipment**

- Engineering notebook
- Pencil
**Procedure**

Complete the chart below by writing the basic, simple and complex behaviors for this mobile robot: when a pushbutton is pressed, a mobile robot will go as fast as possible for 20 feet, and then stop.

<table>
<thead>
<tr>
<th>Complex Behaviors</th>
<th>Simple Behaviors</th>
<th>Basic Behaviors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Now fill in the template below to write the pseudocode for this mobile robot: when a pushbutton is pressed, a mobile robot will go as fast as possible for 20 feet, and then stop.

Pseudocode:

```*/
  task main()
  {
  *
  */
  ```
Conclusion

1. List at least five smaller behaviors you could break the complex behavior “brushing my teeth” into.

2. Why is it important to think of a computer program as a set of basic, simple and complex behaviors that a robot needs to follow?

3. What is the purpose of a set of curly braces { } in a ROBOTC computer program?

4. What is the role of a programmer?