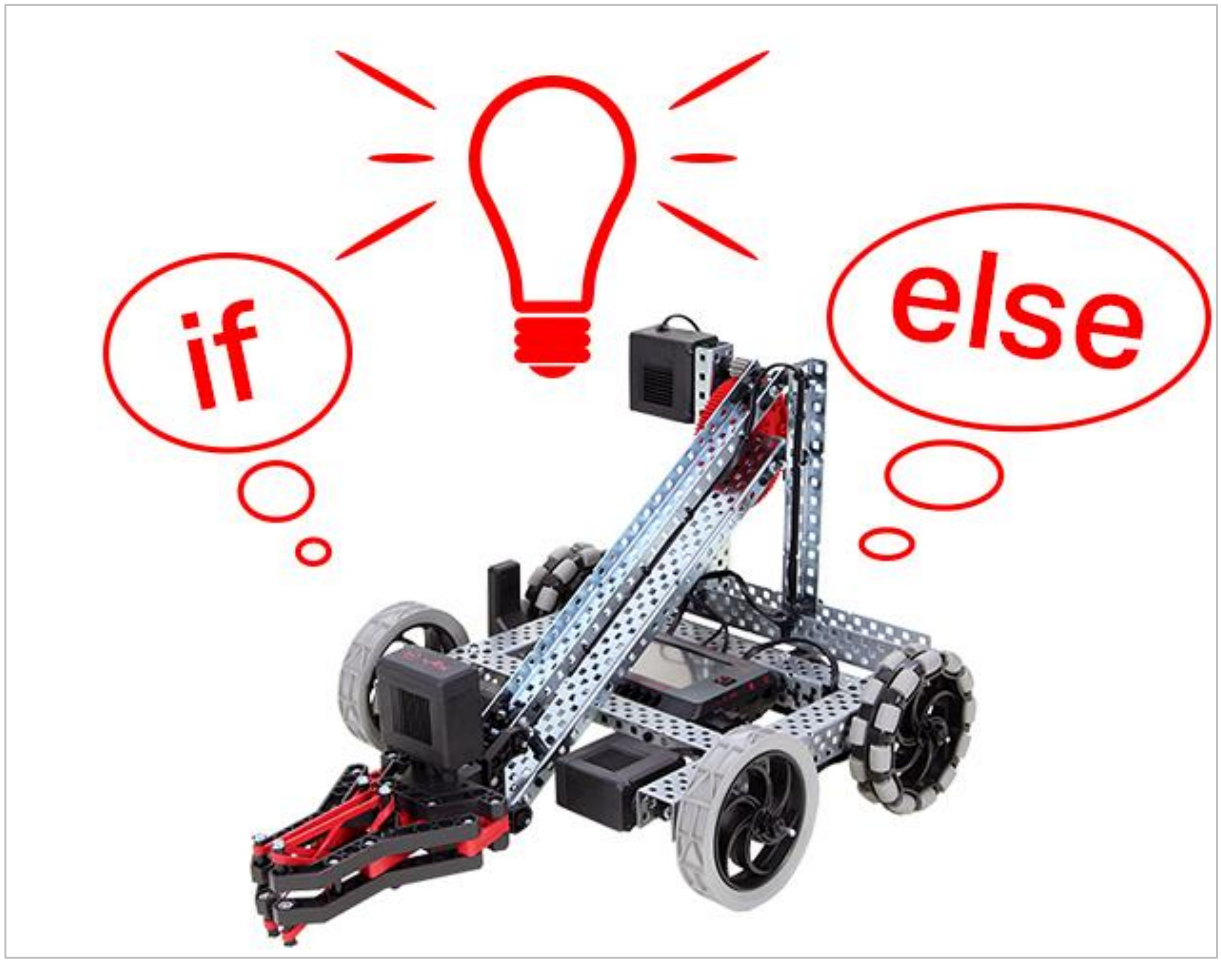
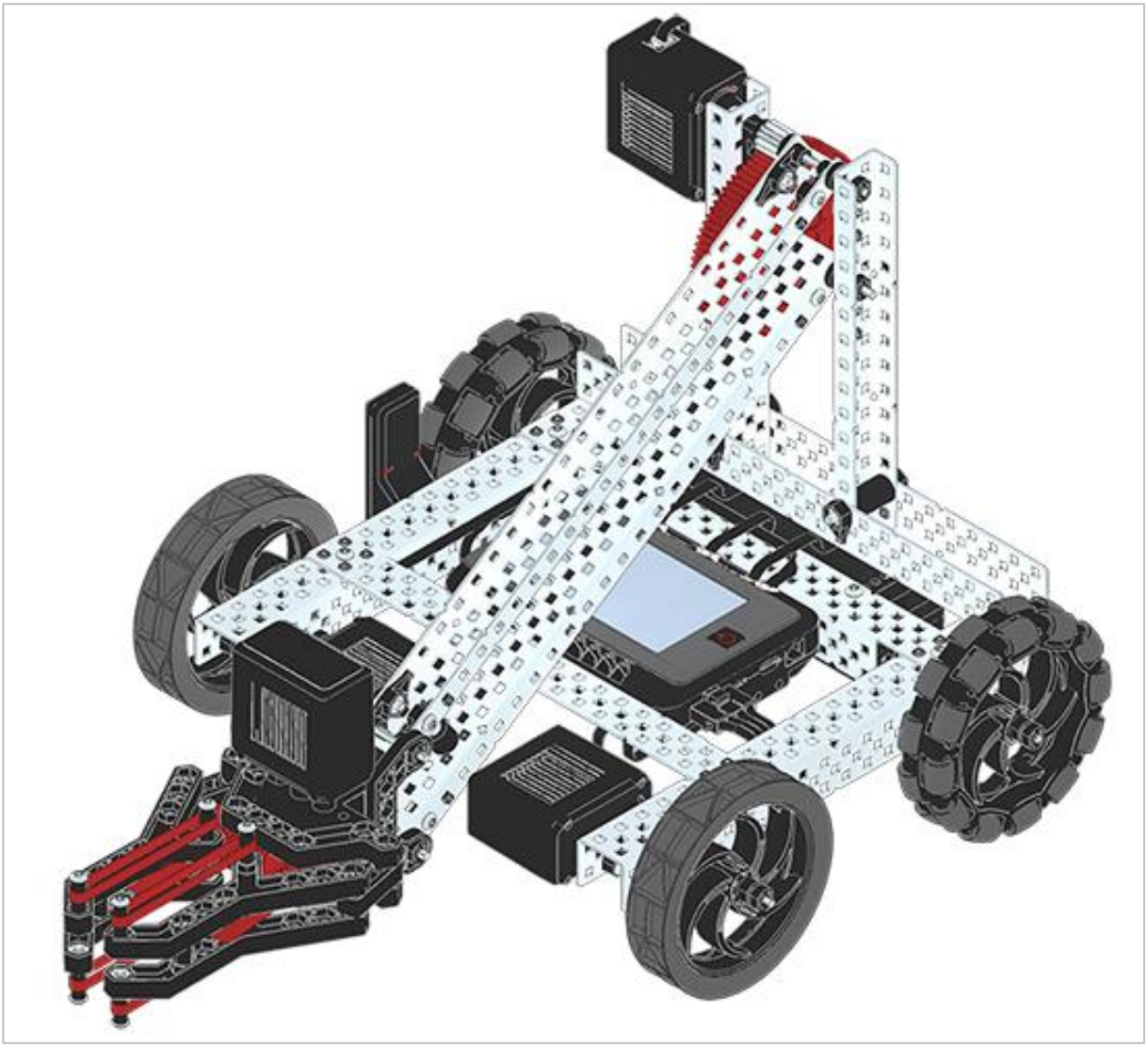
To Do, or Not to Do



Explore the logic of if then else!

Discover new hands-on builds and programming opportunities to further your understanding of a subject matter.

The Completed Look of the Build



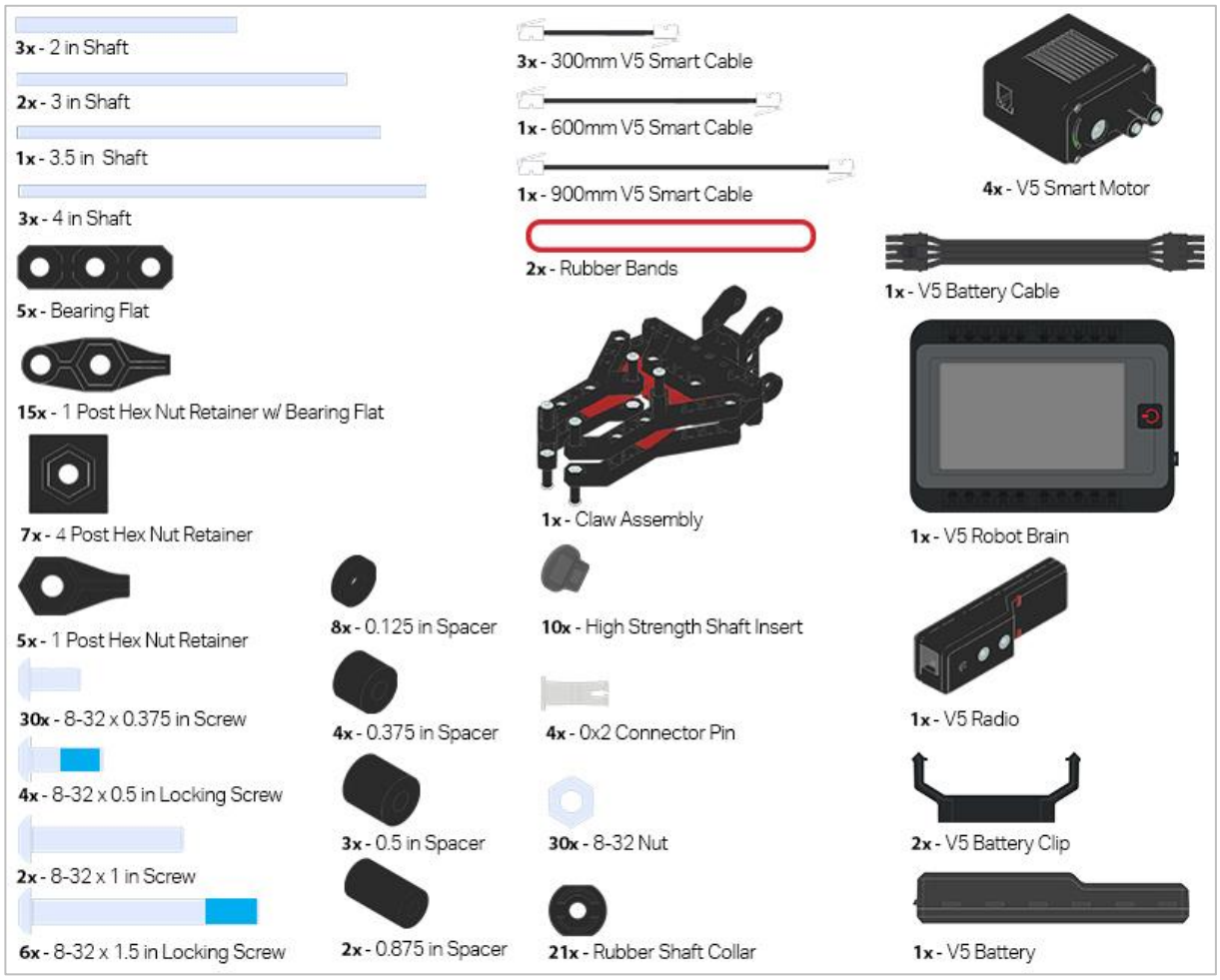
Completed VEX V5 Clawbot

The VEX V5 Clawbot is an extension of the VEX V5 Speedbot that can be programmed to move around and interact with objects.

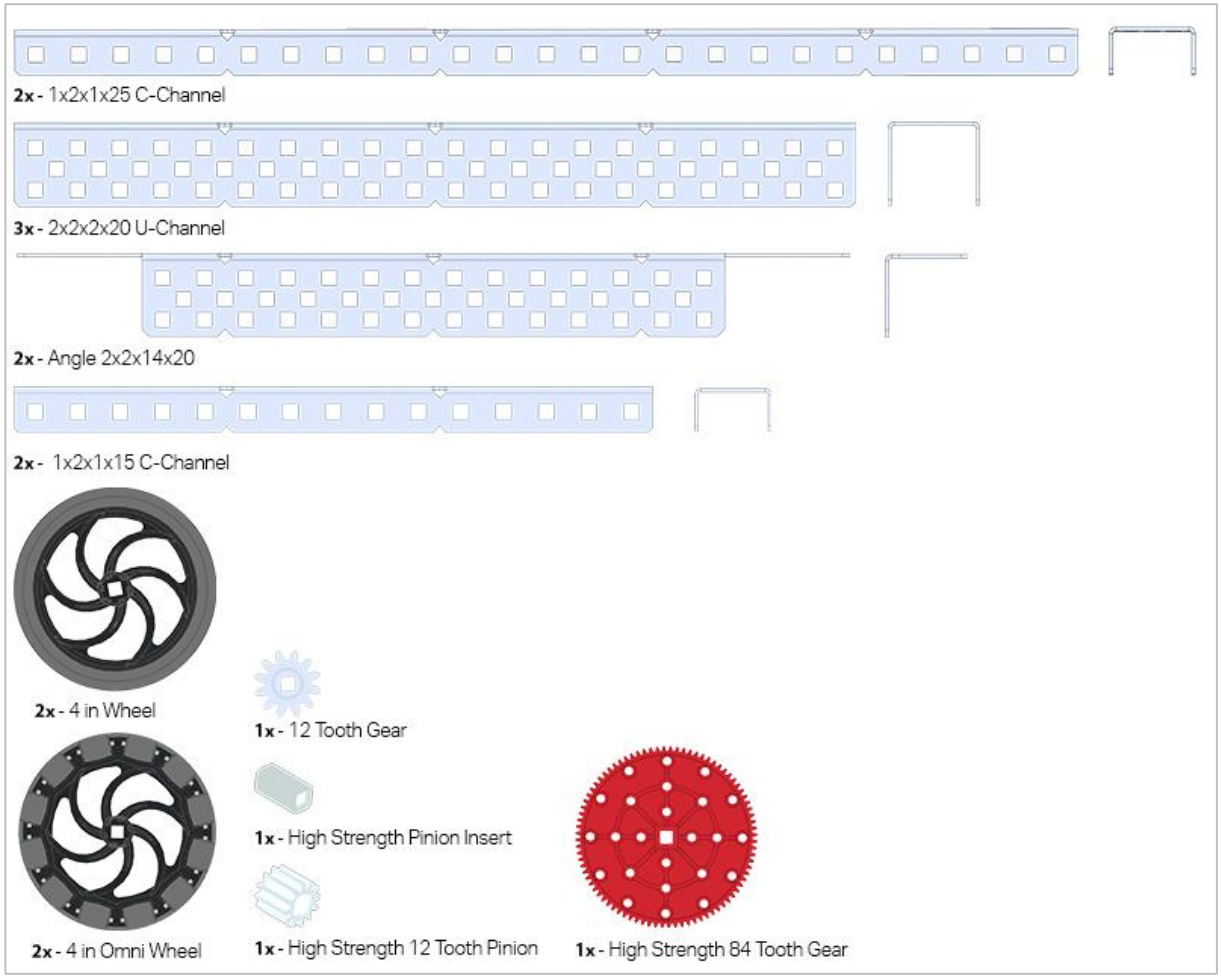
Parts Needed: Part 1

Can be built with:

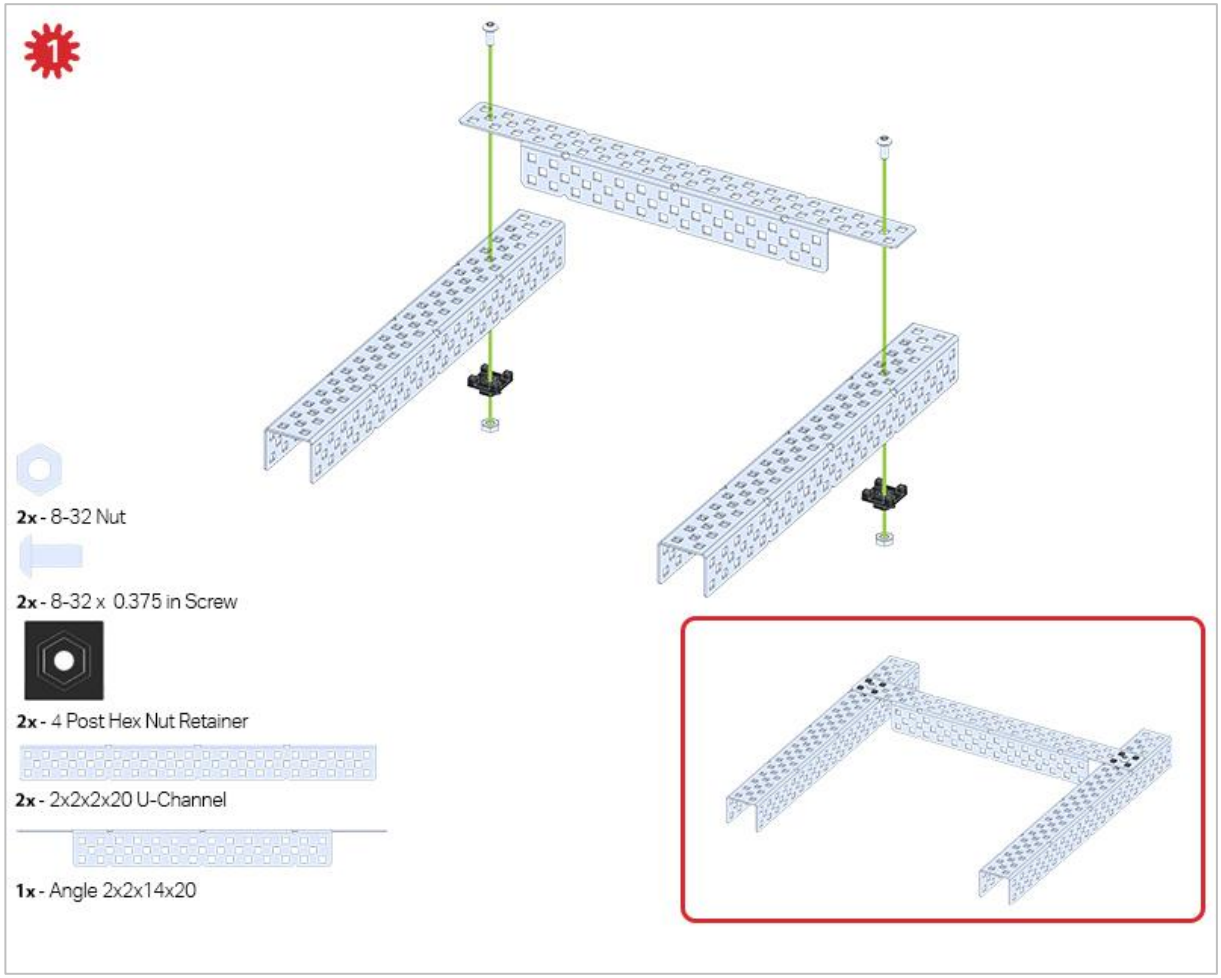
- VEX V5 Classroom Starter Kit



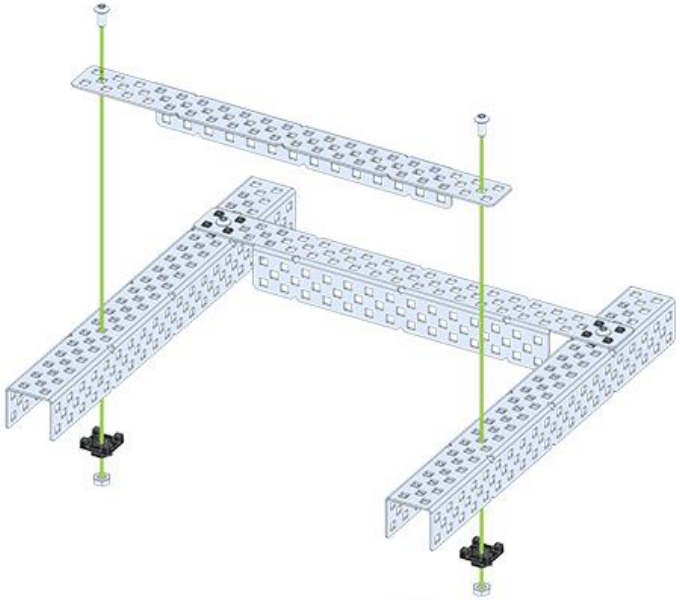
Parts Needed: Part 2



Build Instructions



2

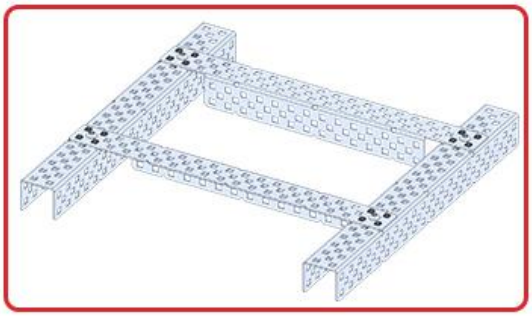


2x - 8-32 Nut

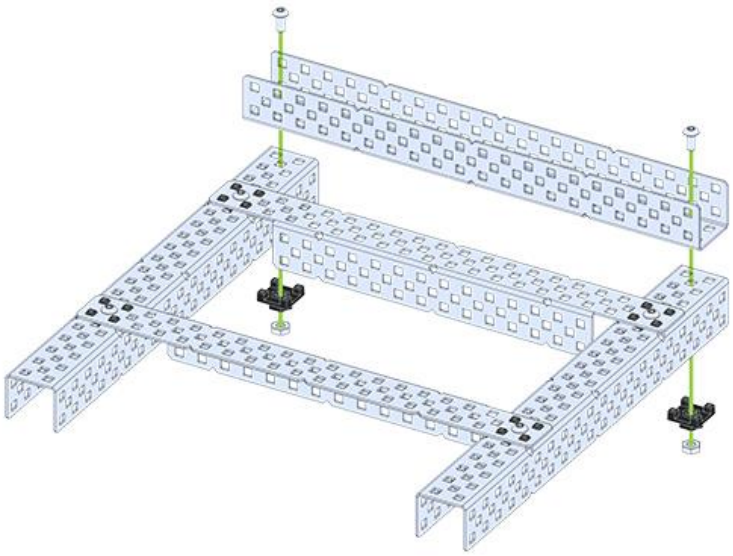
2x - 8-32 x 0.375 in Screw

2x - 4 Post Hex Nut Retainer

1x - Angle 2x2x14x20



3

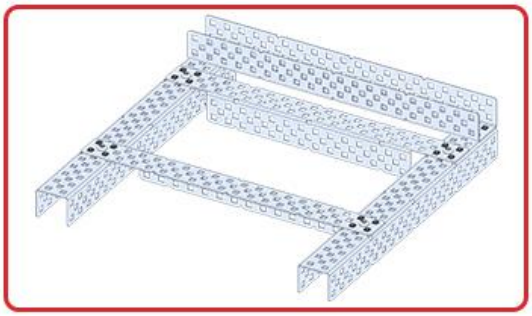


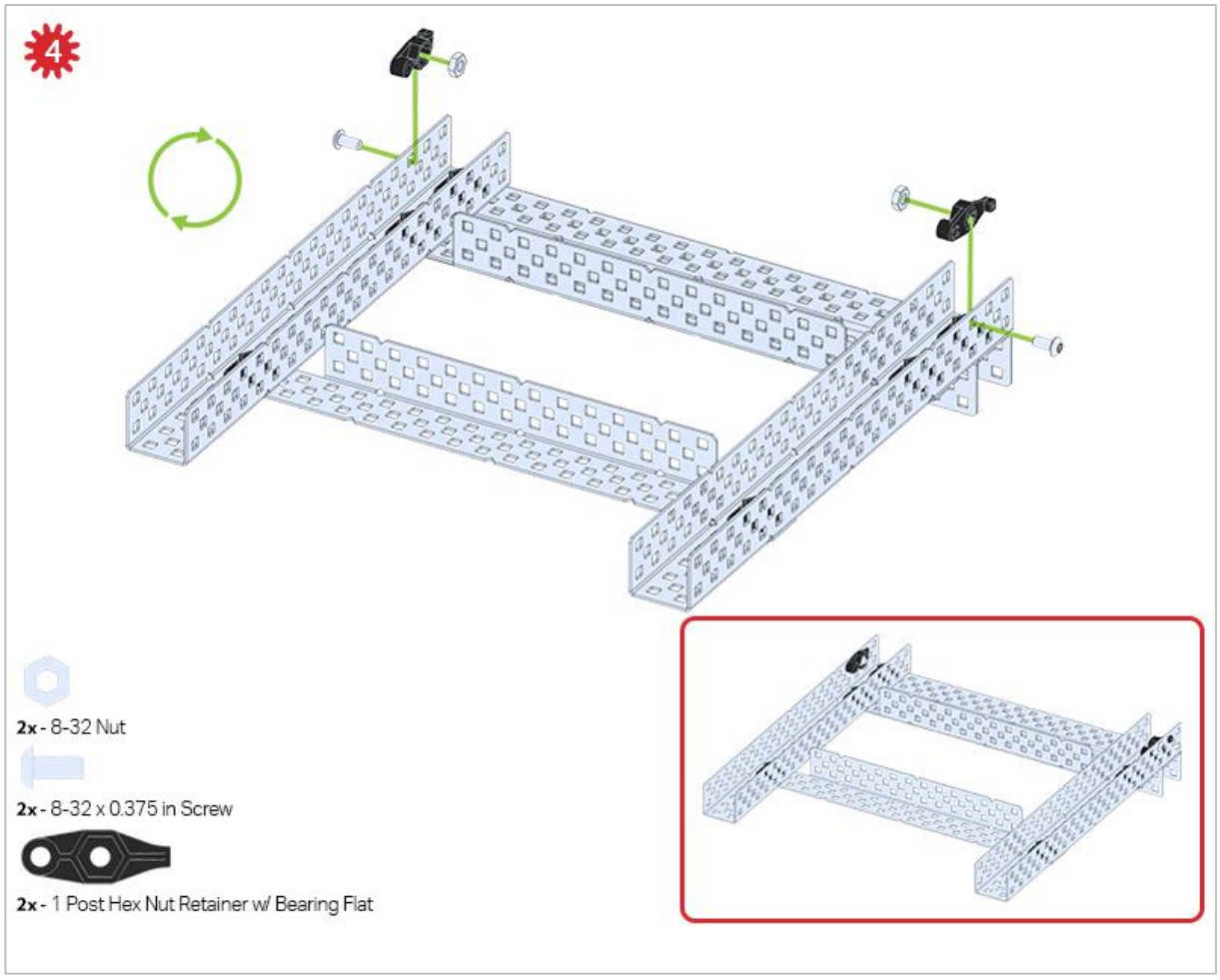
2x - 8-32 Nut

2x - 8-32 x 0.375 in Screw

2x - 4 Post Hex Nut Retainer

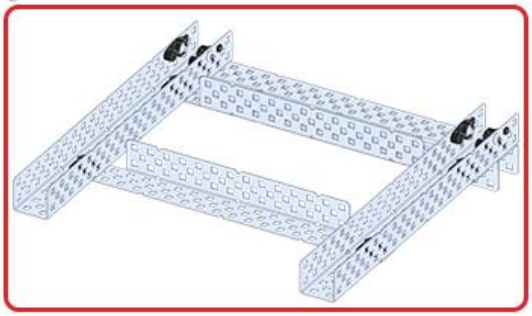
1x - 2x2x2x20 U-Channel





The green icon indicates that the build needs to be flipped over (upside down).

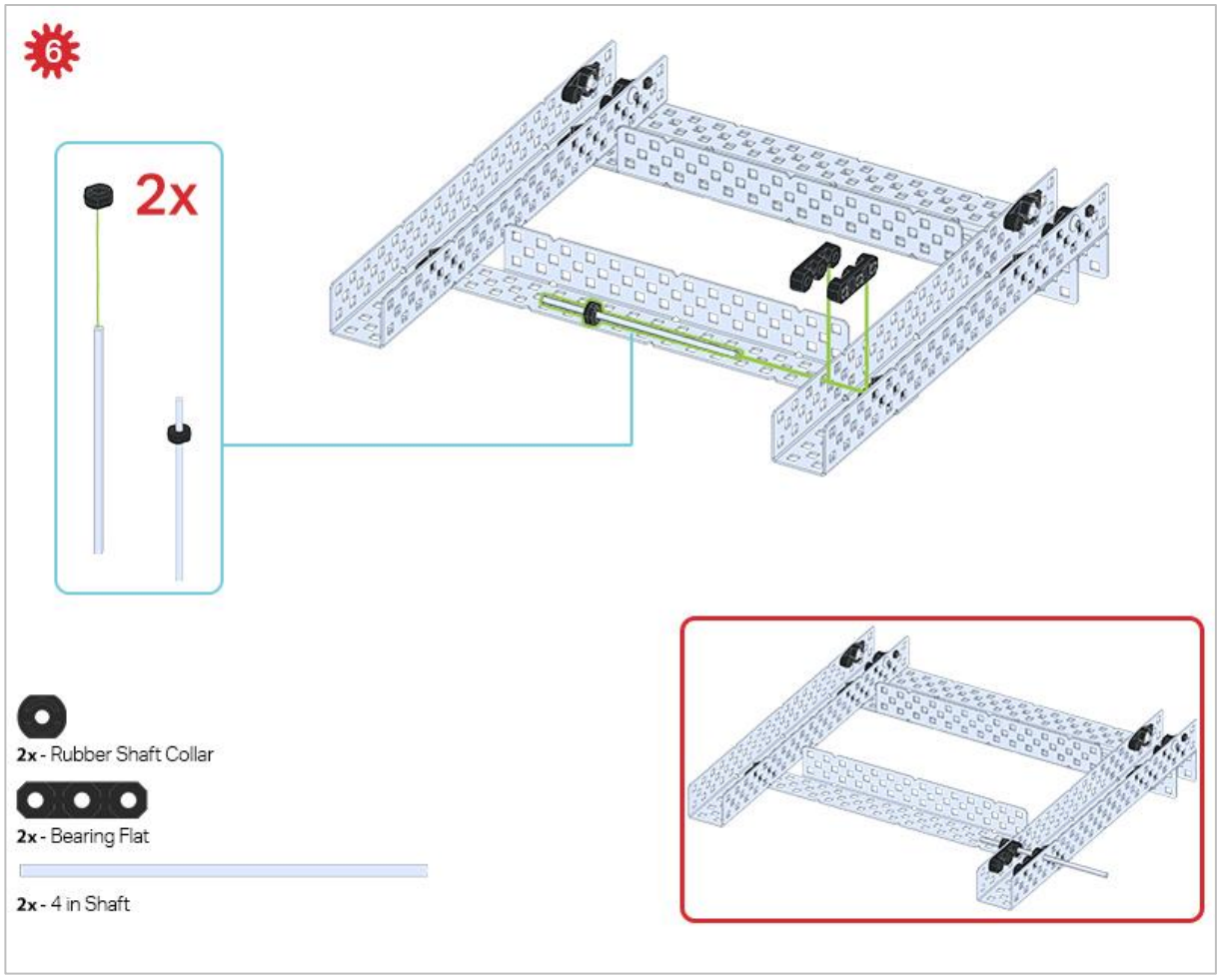
5



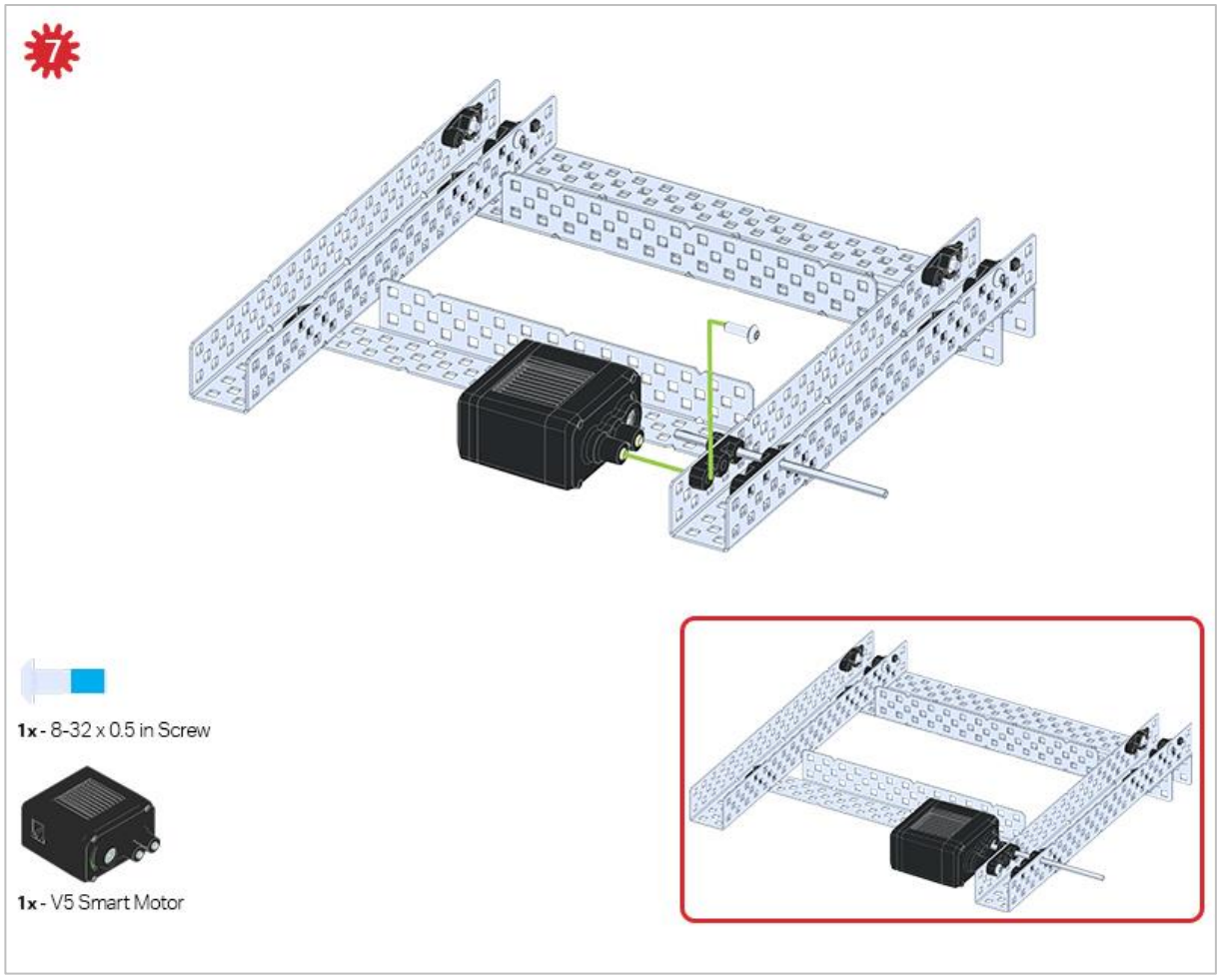
2x - 8-32 Nut

2x - 8-32 x 0.375 in Screw

2x - 1 Post Hex Nut Retainer w/ Bearing Flat

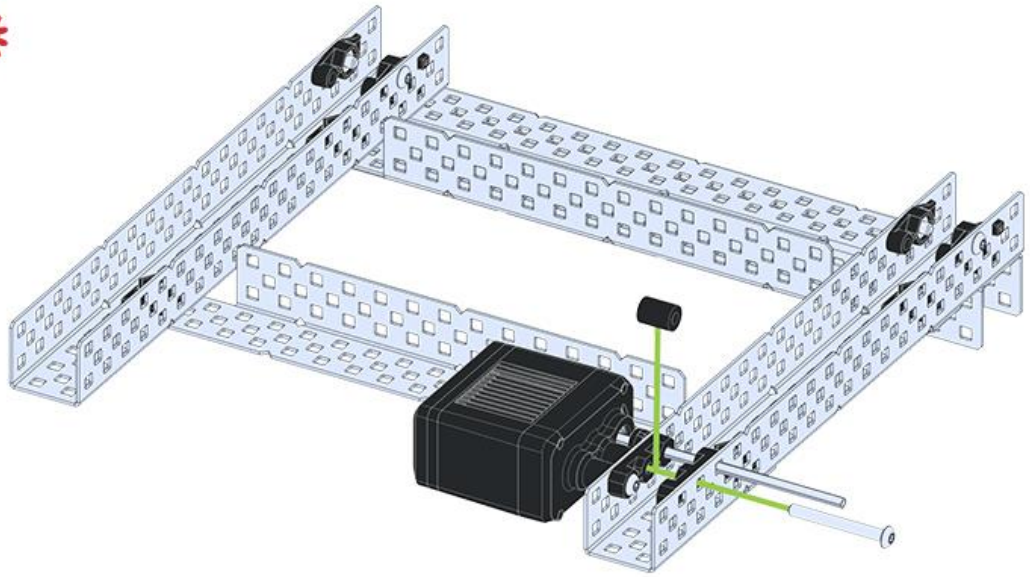


Only one of the two sub-assemblies made in this step is used right now. The other will be used later in step 9.



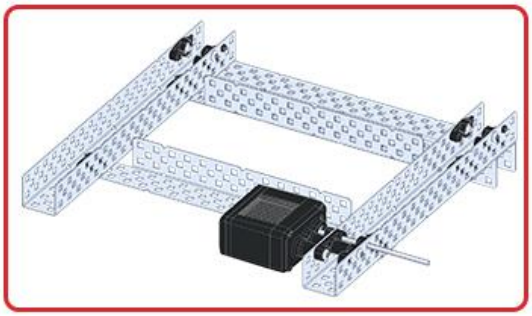
Make sure your Smart Motors are oriented in the correct direction (screw holes facing the outside of the build and the shaft hole towards the inside).

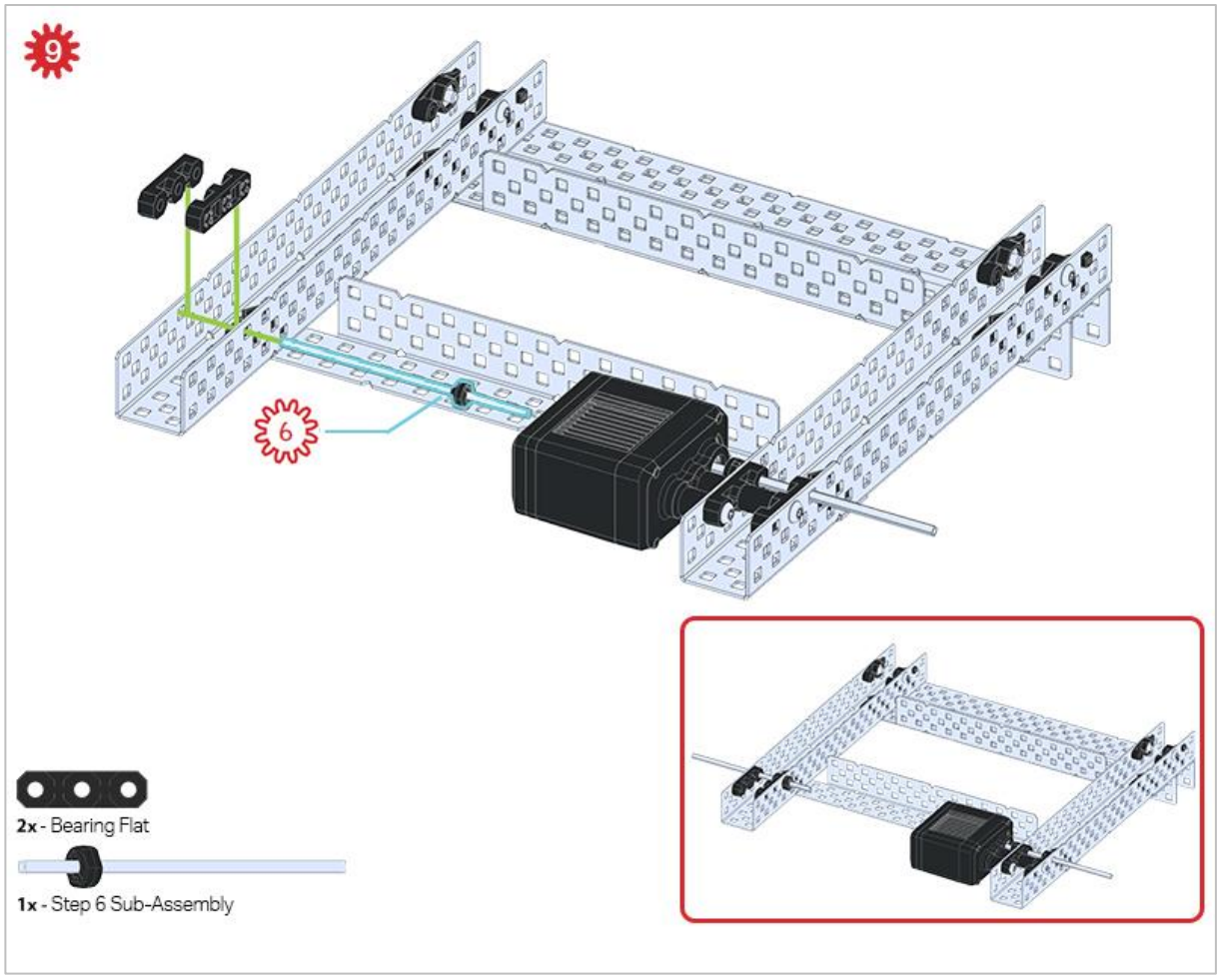
8

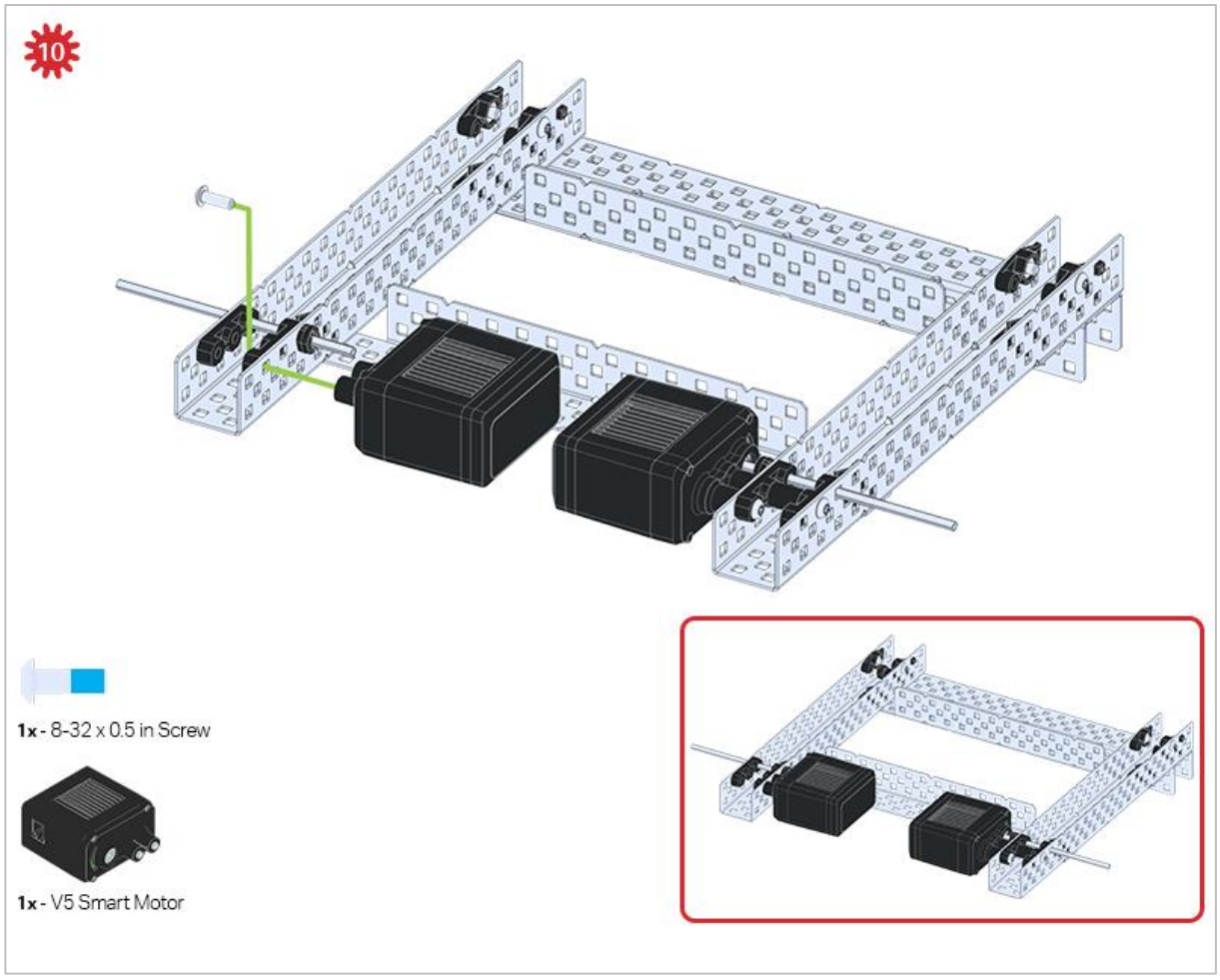


1x - 0.5 in Spacer

1x - 8-32 x 1.5 in Screw

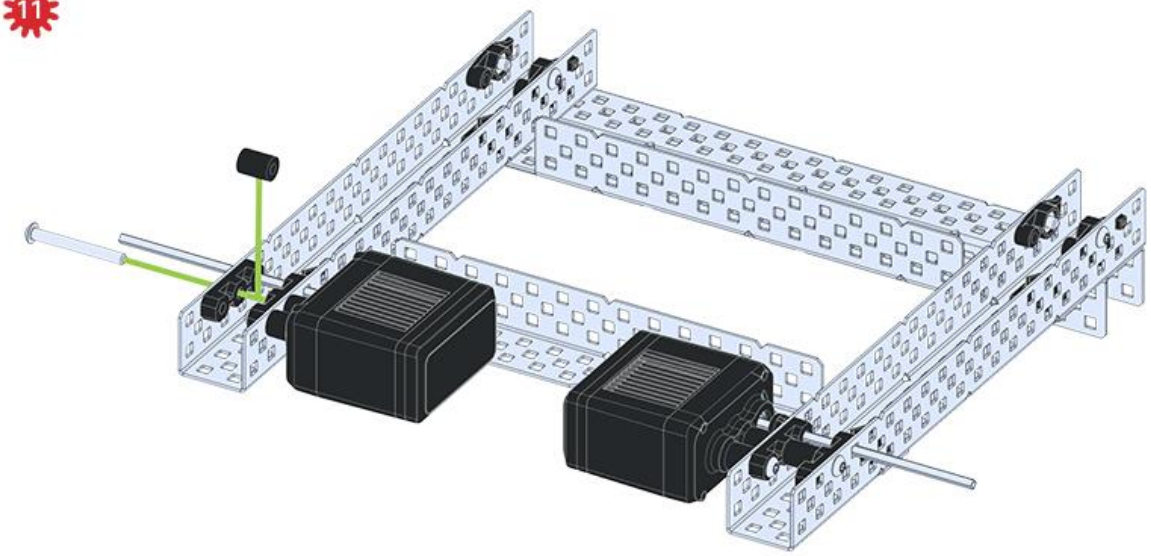



9
6
2x - Bearing Flat
1x - Step 6 Sub-Assembly



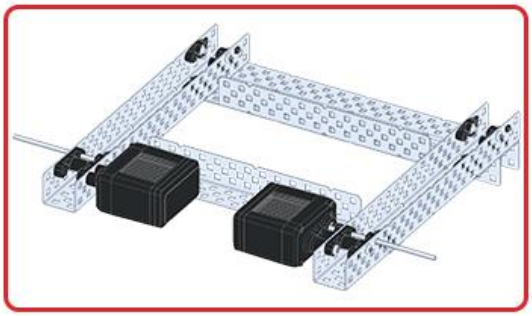
Make sure your Smart Motors are oriented in the correct direction (screw holes facing the outside of the build and the shaft hole towards the inside).

11

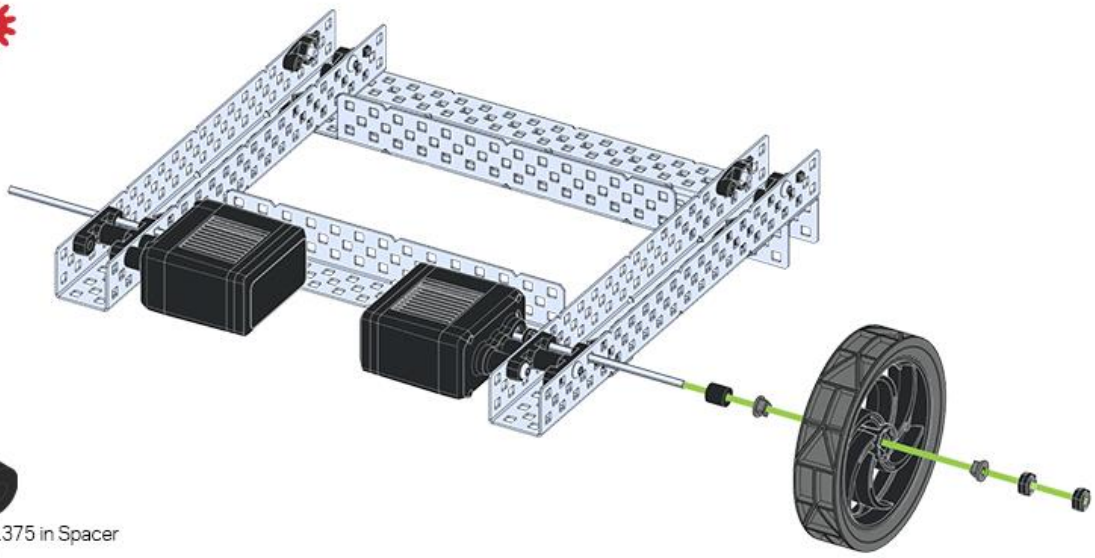


1x - 0.5 in Spacer

1x - 8-32 x 1.5 in Screw



12

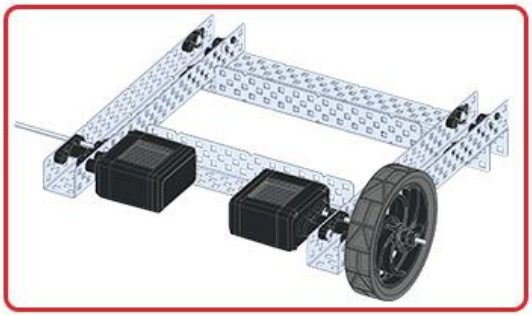


1x - 0.375 in Spacer

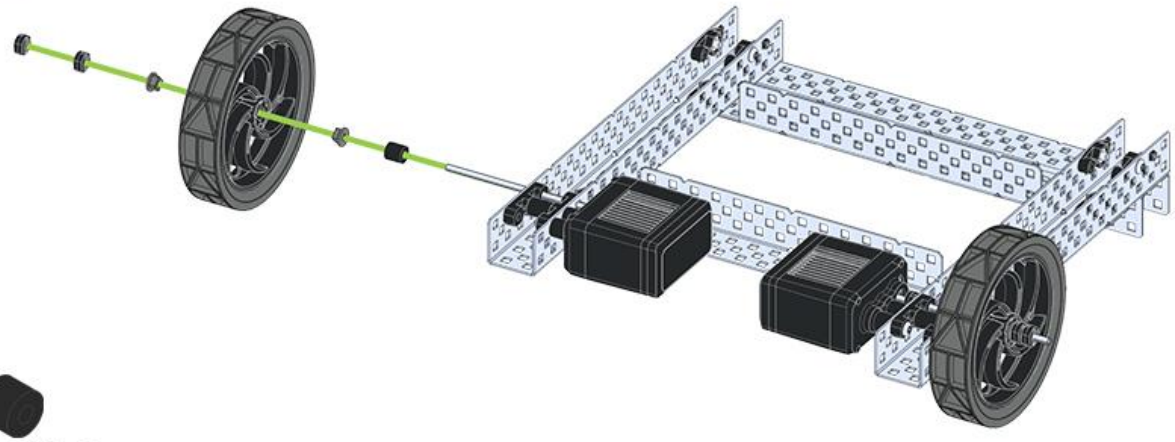
2x - Rubber Shaft Collar

1x - 4 in Wheel

2x - High Strength Shaft Insert



13

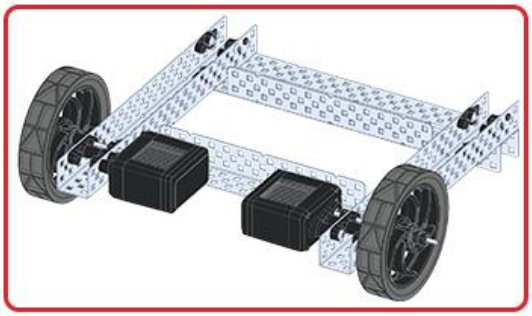


1x - 0.375 in Spacer

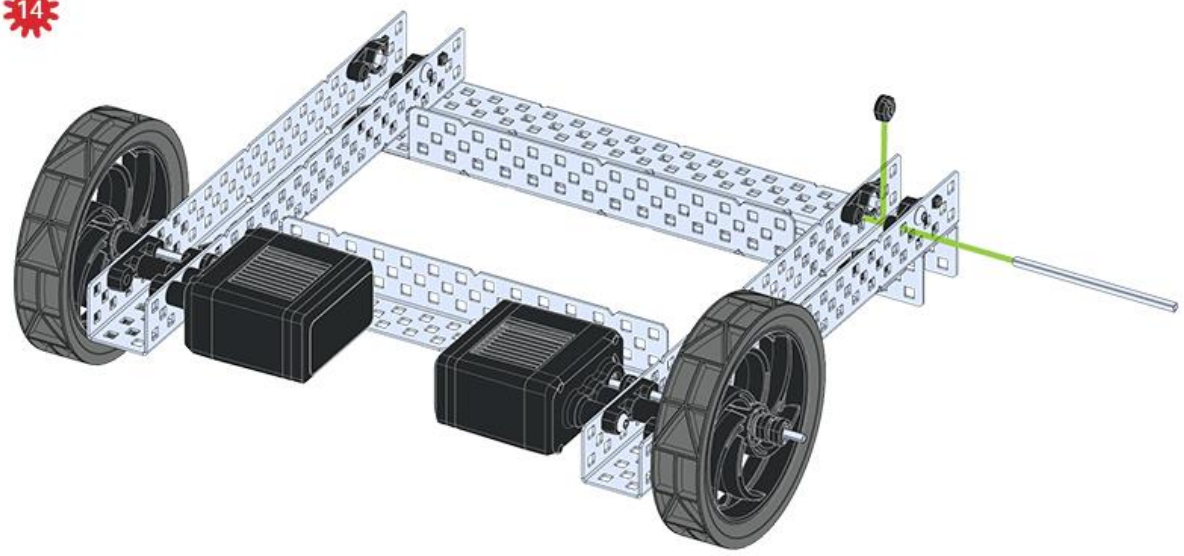
2x - Rubber Shaft Collar

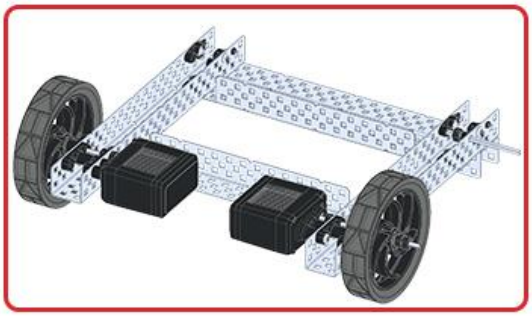
1x - 4 in Wheel

2x - High Strength Shaft Insert



14

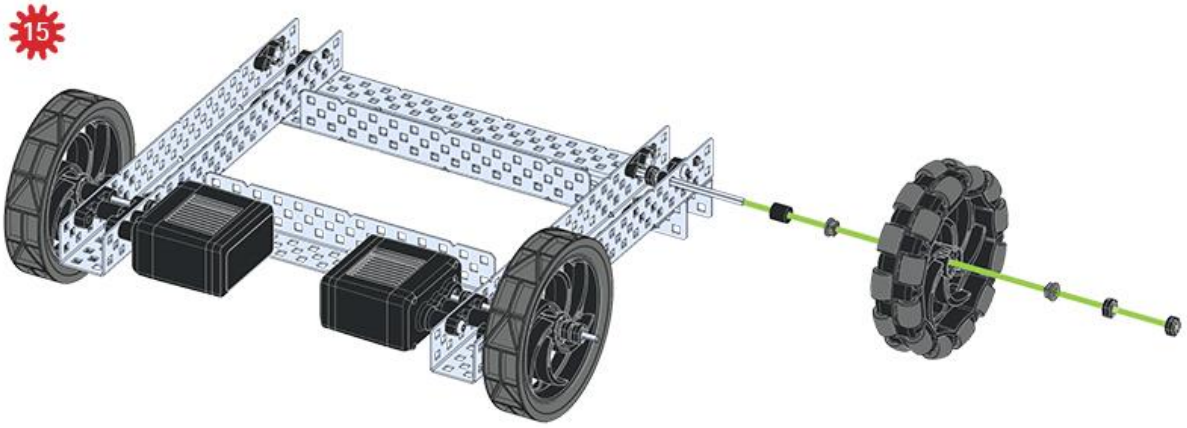




1x - Rubber Shaft Collar

1x - 3 in Shaft

15

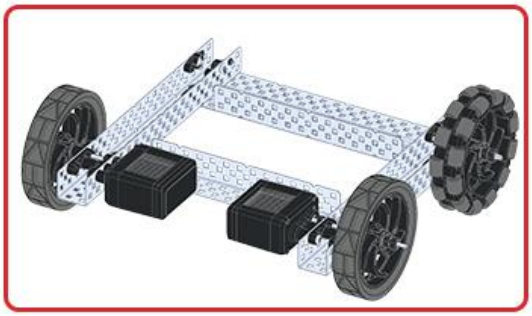


1x - 0.375 in Spacer

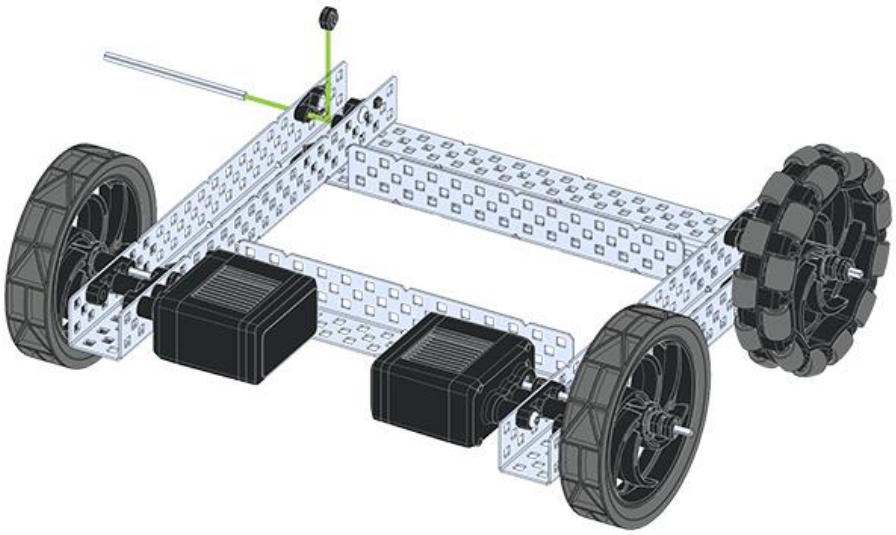
2x - Rubber Shaft Collar

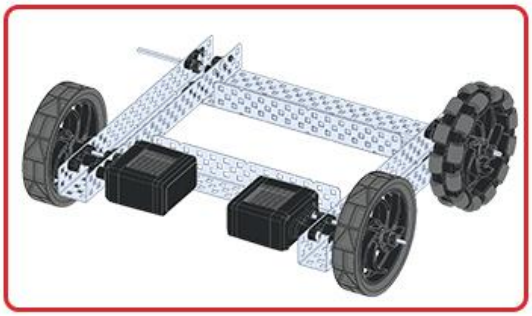
1x - 4 in Omni Wheel

2x - High Strength Shaft Insert



16

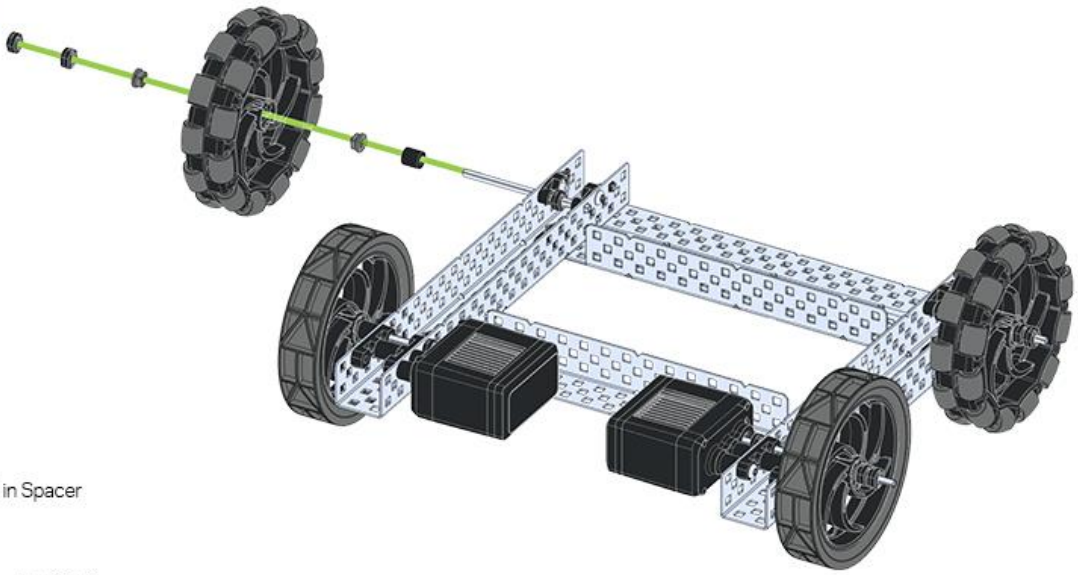




1x - Rubber Shaft Collar

1x - 3 in Shaft

17

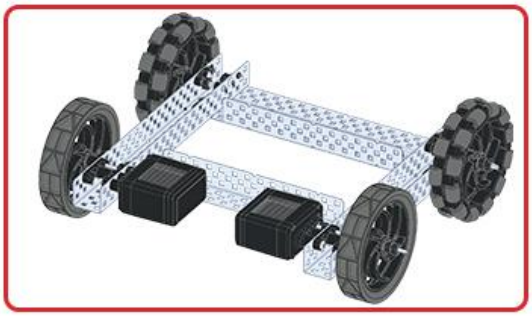


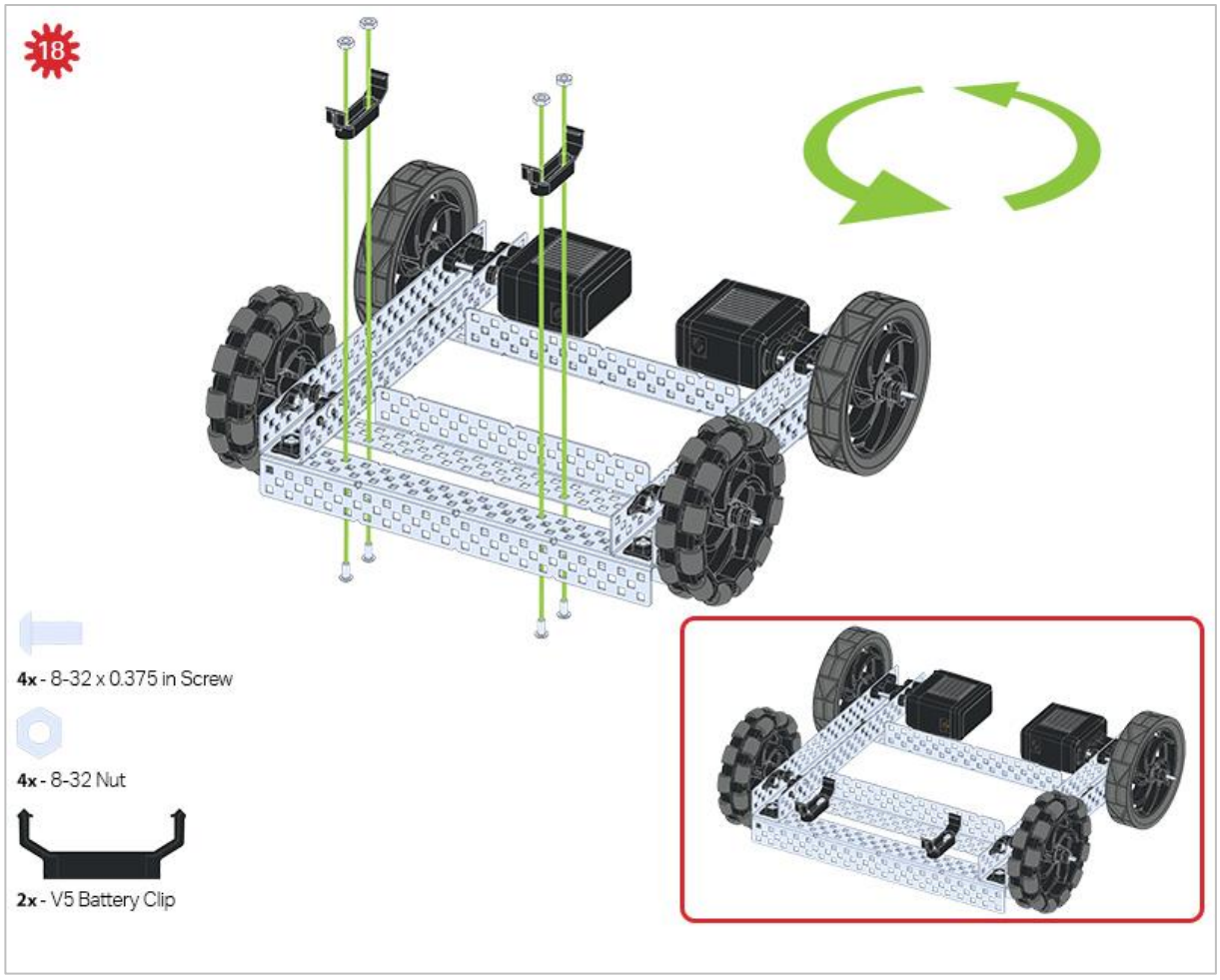
1x - 0.375 in Spacer

2x - Rubber Shaft Collar

1x - 4 in Omni Wheel

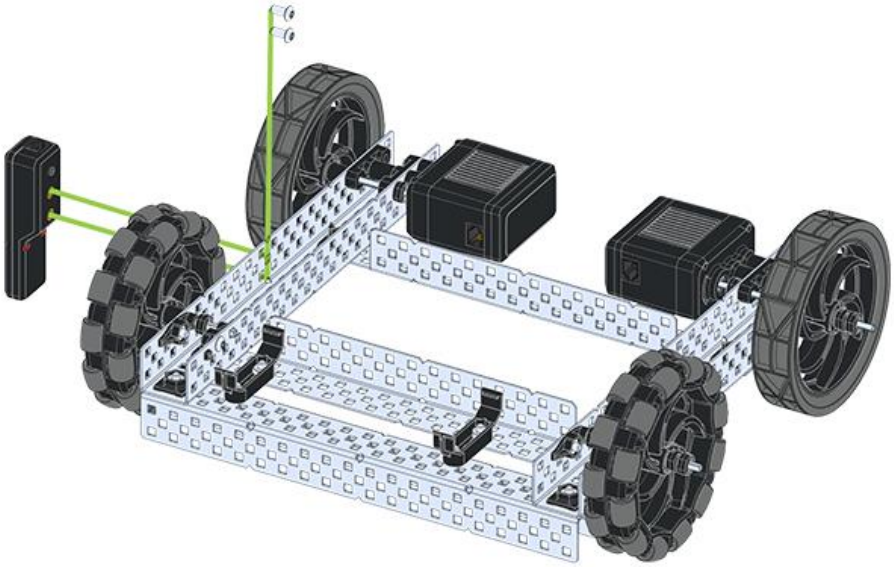
2x - High Strength Shaft Insert





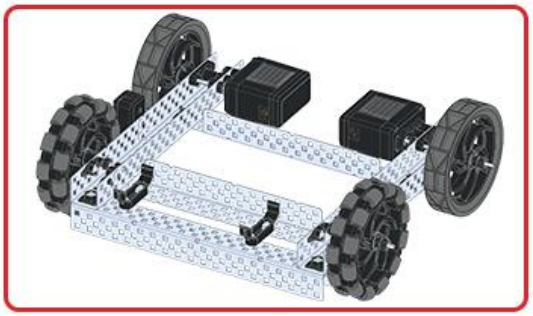
The green icon indicates that the build needs to be rotated (180 degrees).

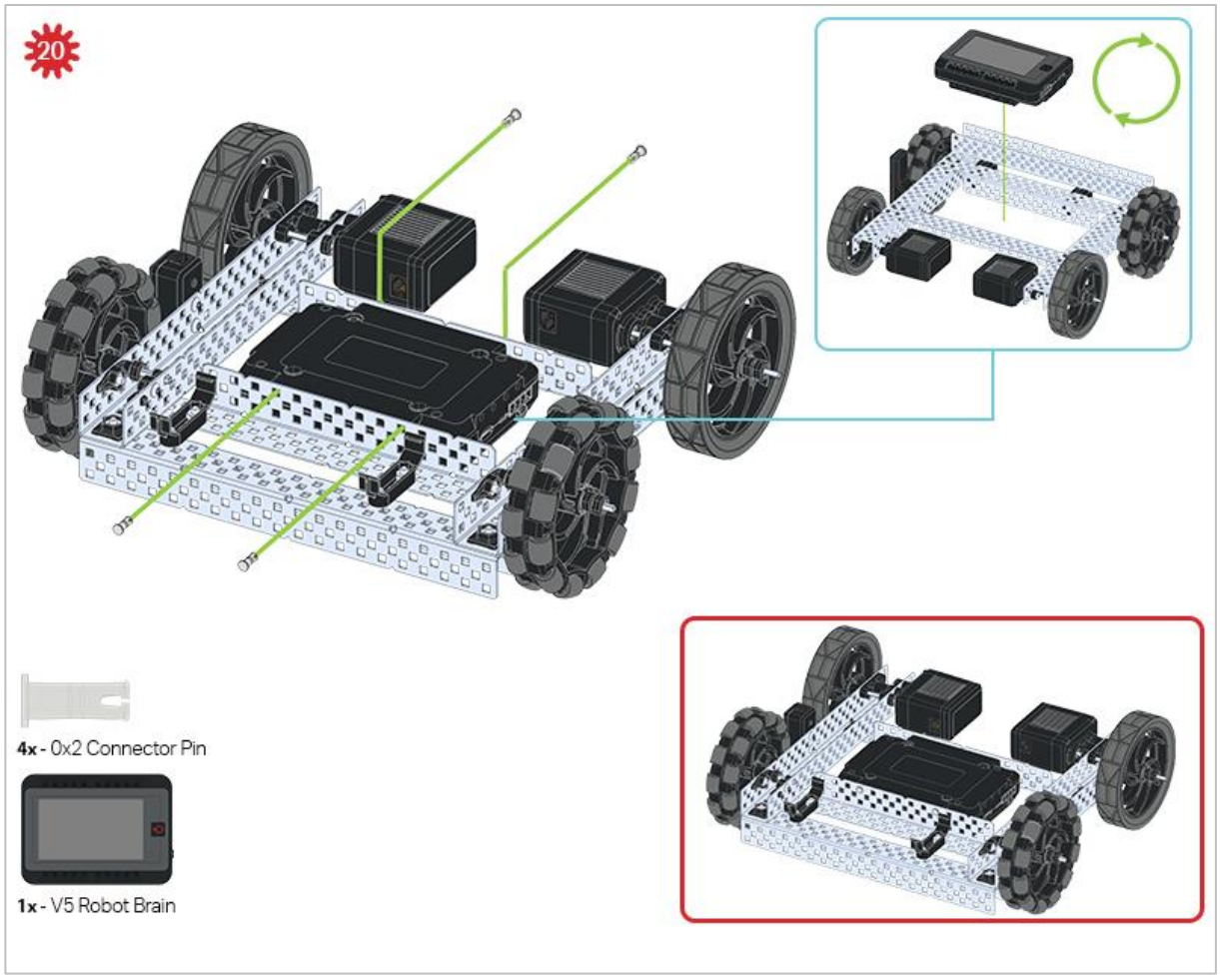
19



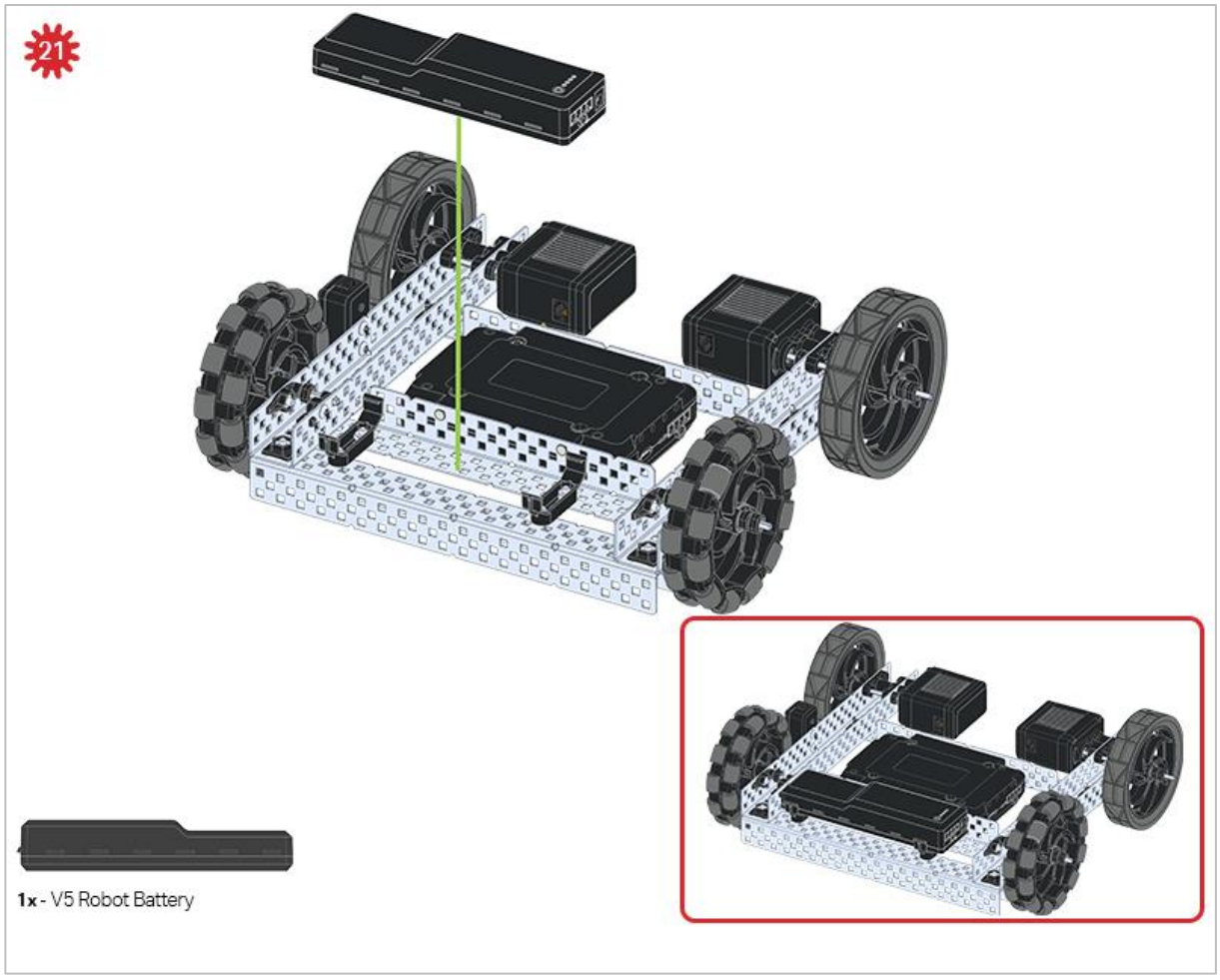
1x - V5 Radio

2x - 8-32 x 0.375 in Screw

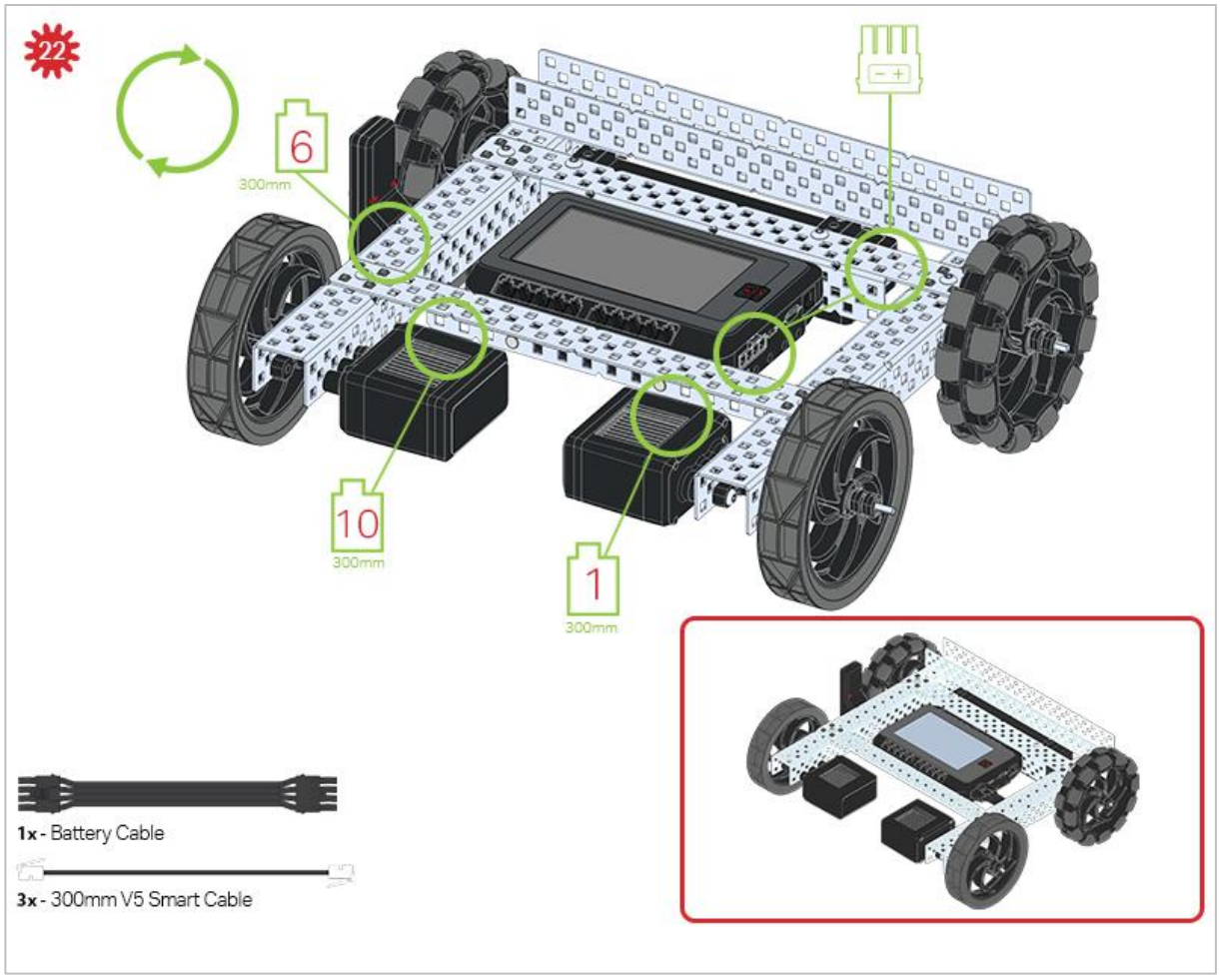




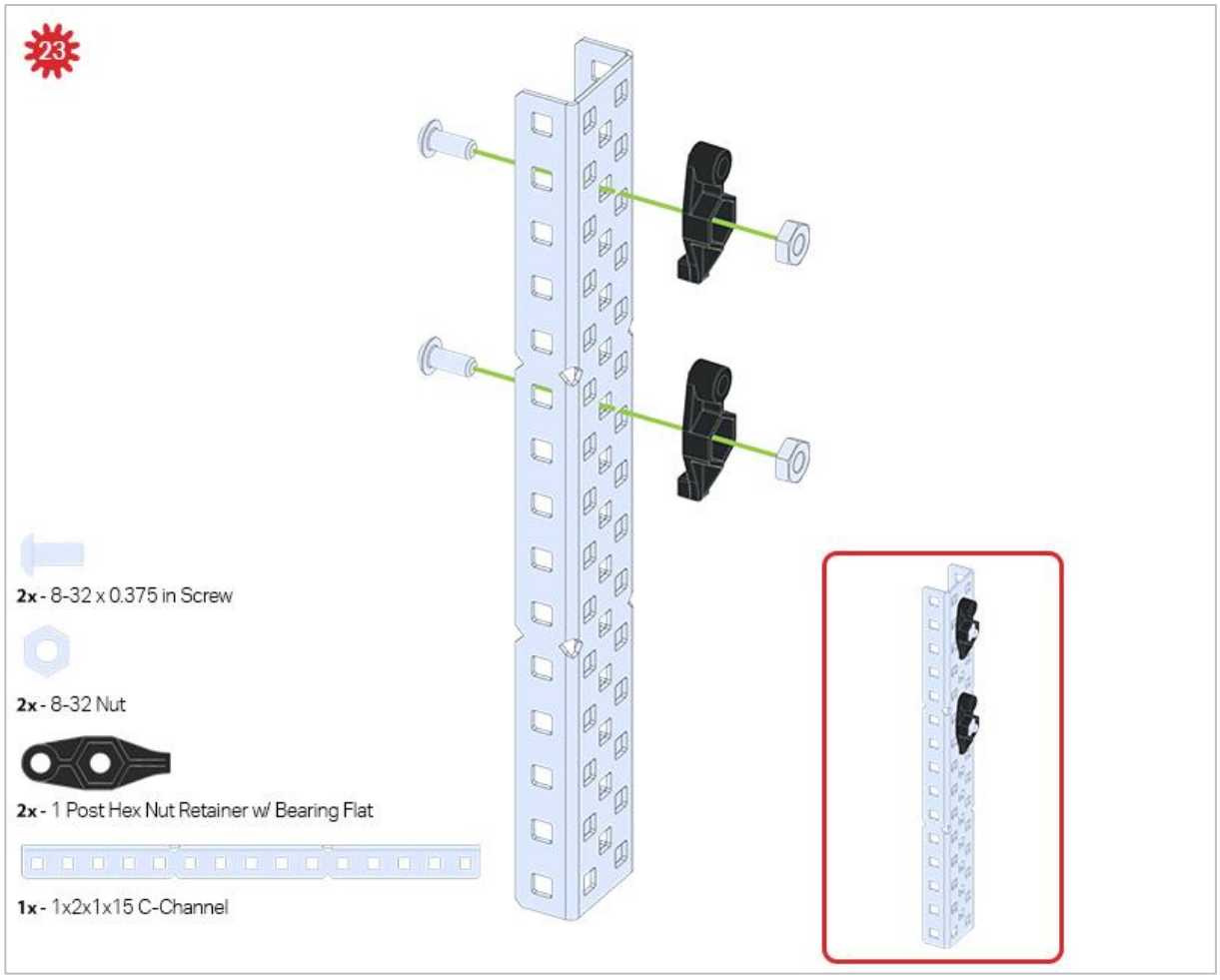
The blue call out shows what the orientation of the Robot Brain should be if the build were flipped right side up. Make sure the 3 wire ports on the Robot Brain are facing the V5 Radio!

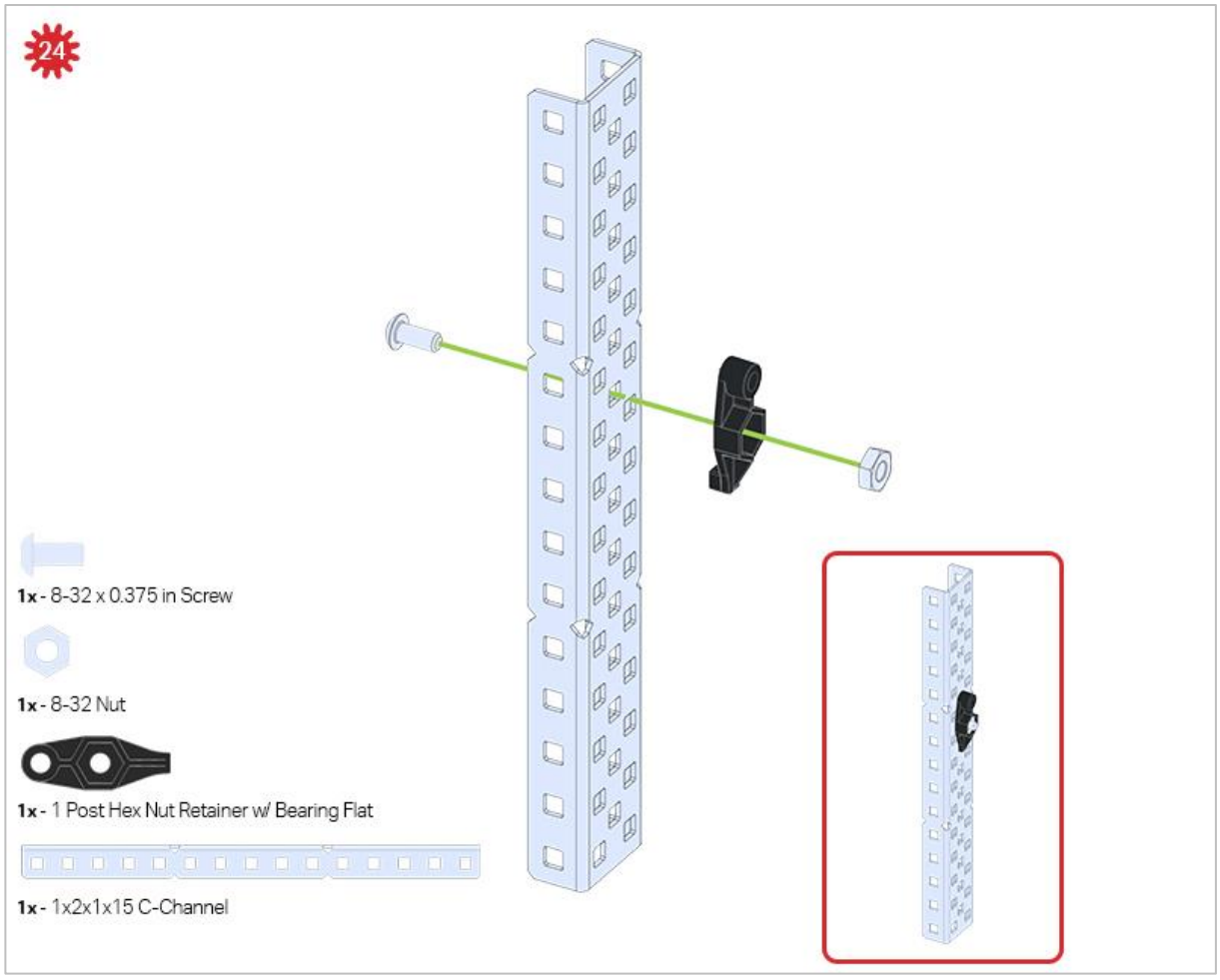


1x - V5 Robot Battery

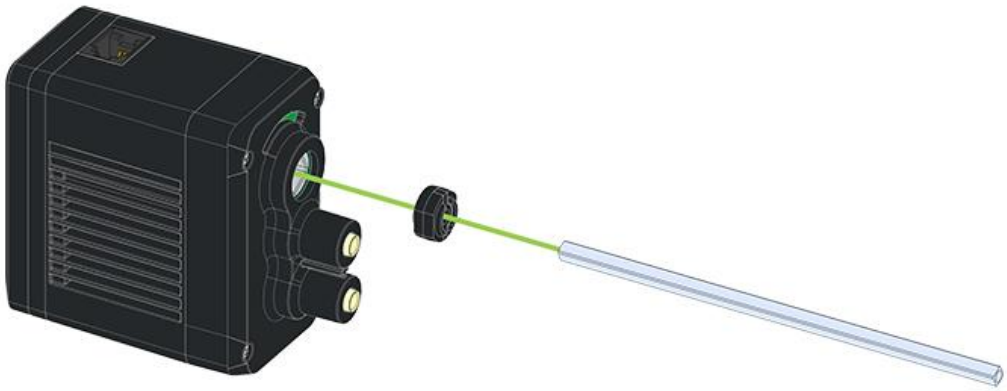


The green call outs indicate which port on the Robot Brain to plug each device into using their respective cable.


23
2x - 8-32 x 0.375 in Screw
2x - 8-32 Nut
2x - 1 Post Hex Nut Retainer w/ Bearing Flat
1x - 1x2x1x15 C-Channel


24
1x - 8-32 x 0.375 in Screw
1x - 8-32 Nut
1x - 1 Post Hex Nut Retainer w/ Bearing Flat
1x - 1x2x1x15 C-Channel

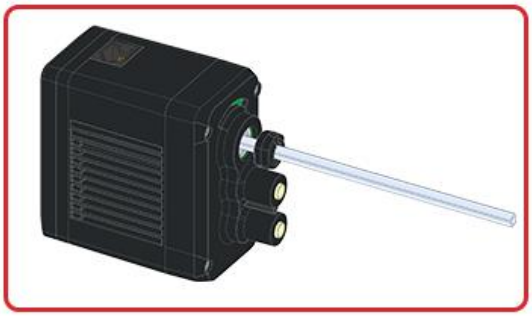
25

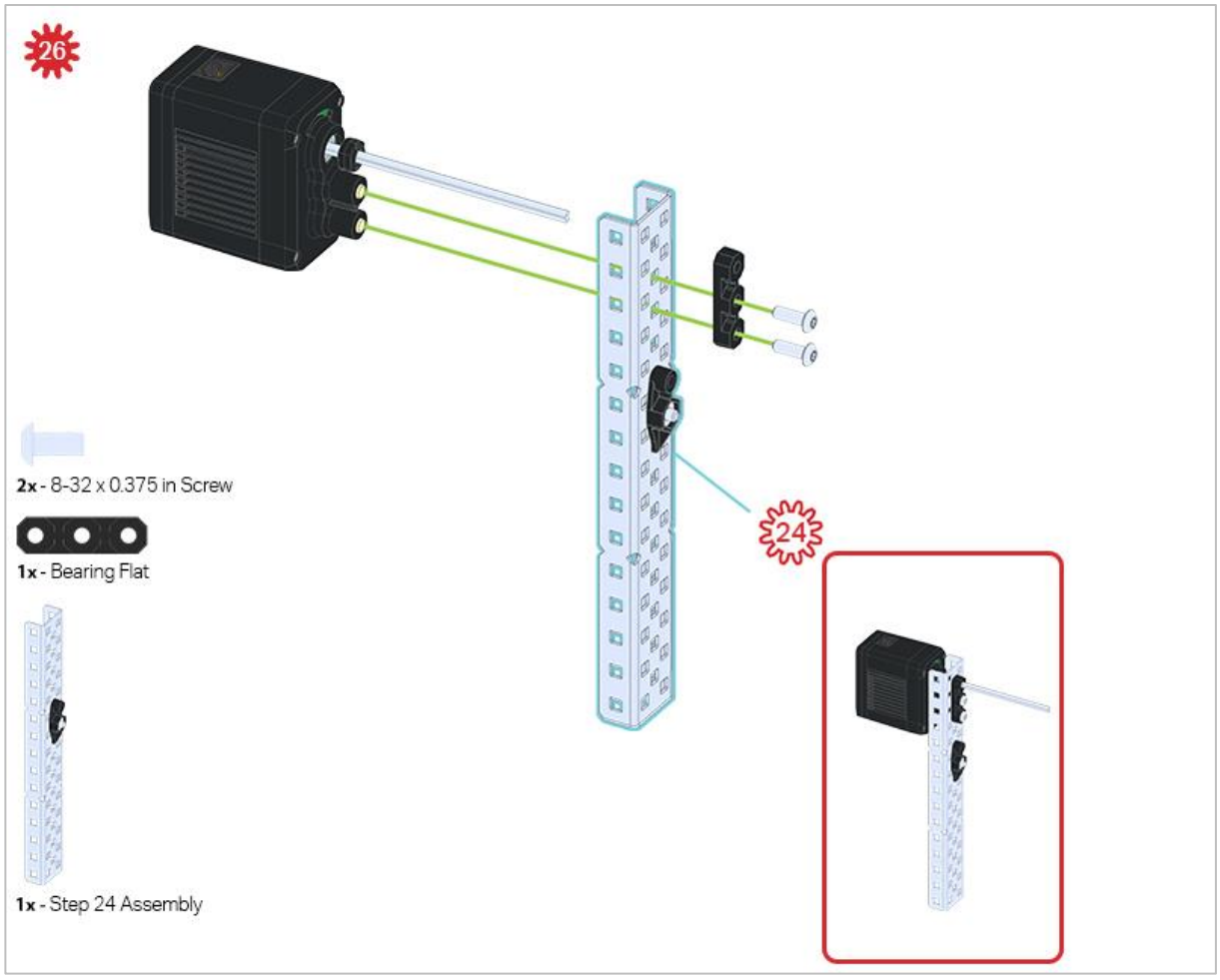


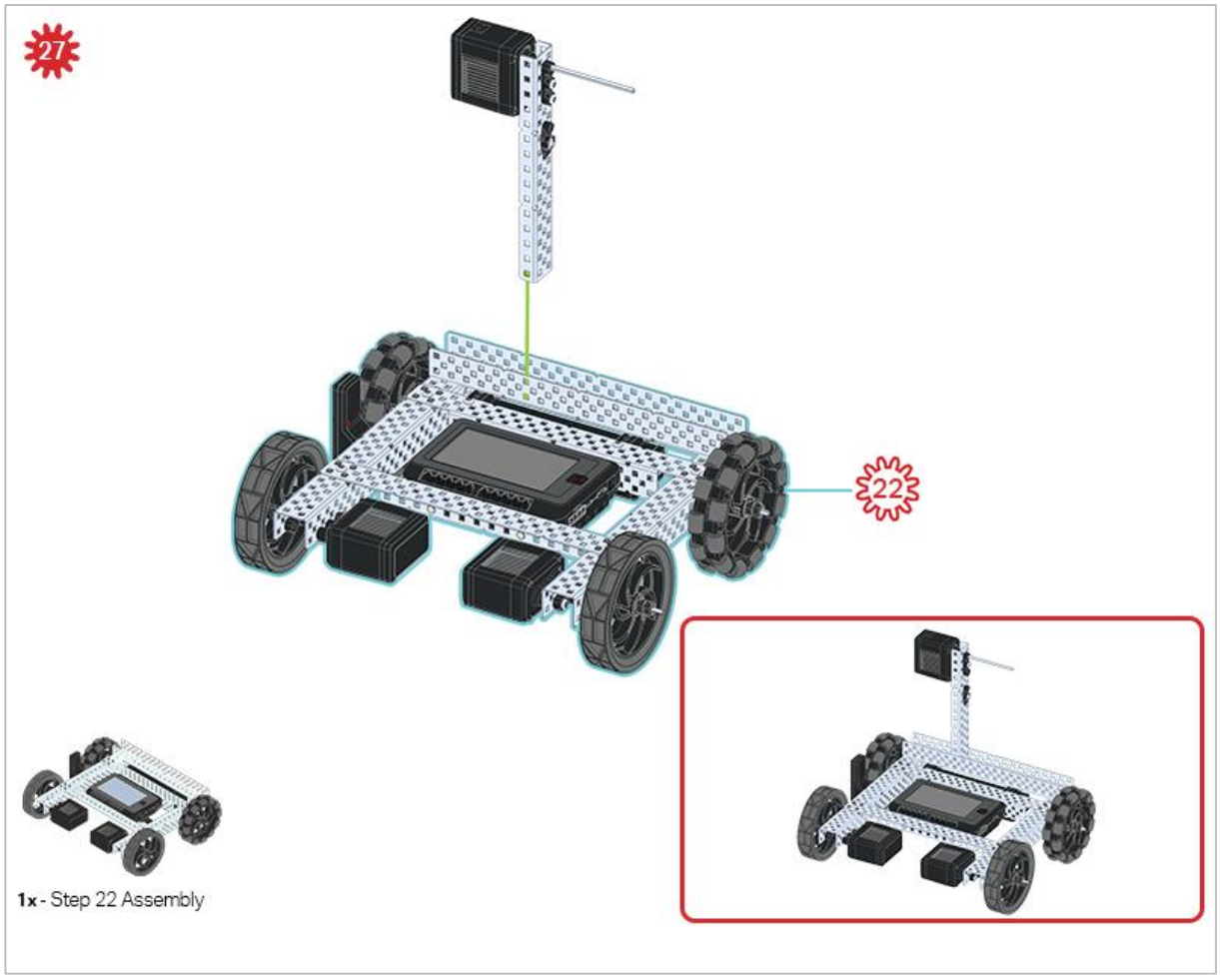
1x - Rubber Shaft Collar

1x - V5 Smart Motor

1x - 4 in Shaft




26
2x - 8-32 x 0.375 in Screw
1x - Bearing Flat
1x - Step 24 Assembly
24


27
22
1x - Step 22 Assembly

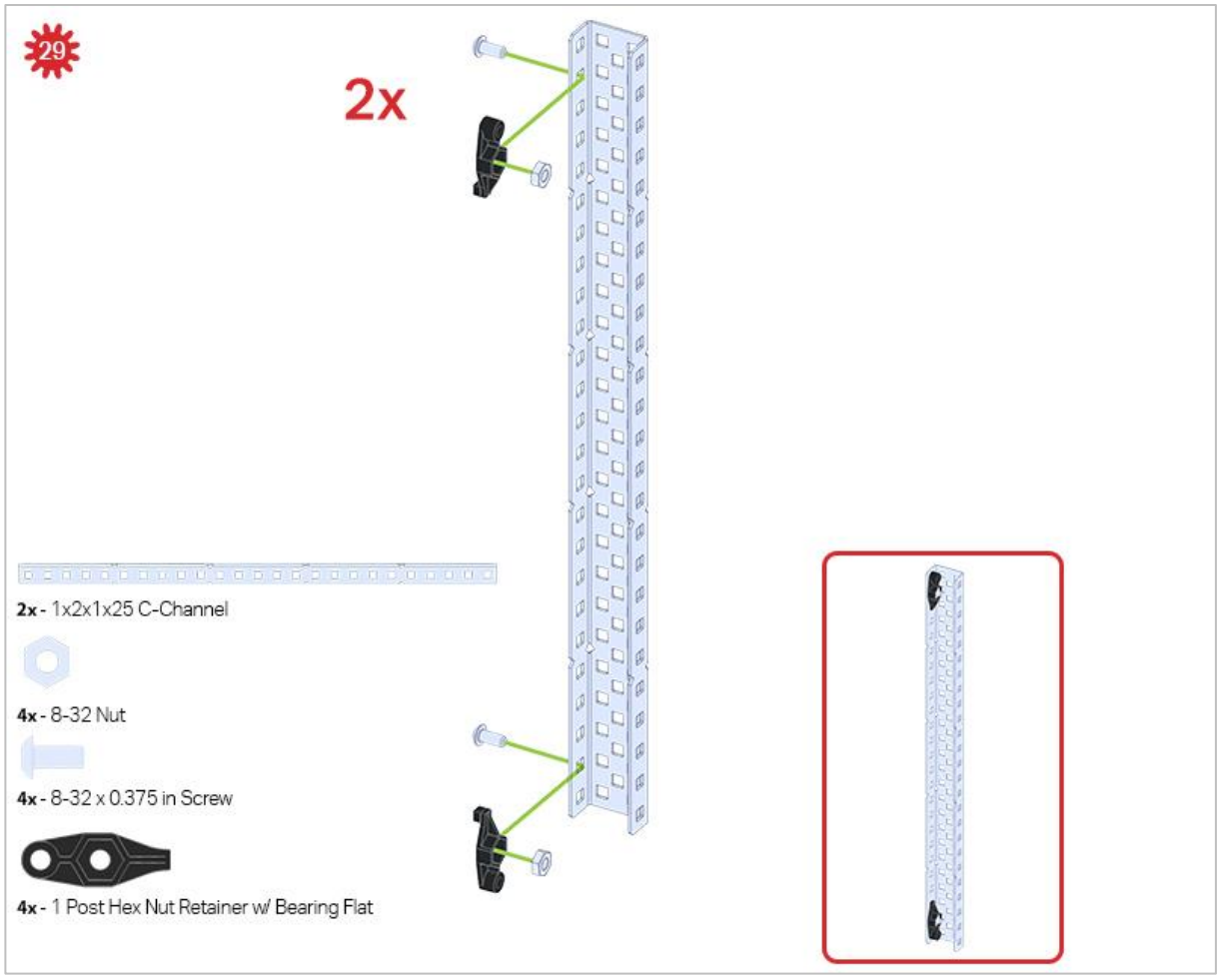
28

1x - 8-32 Nut

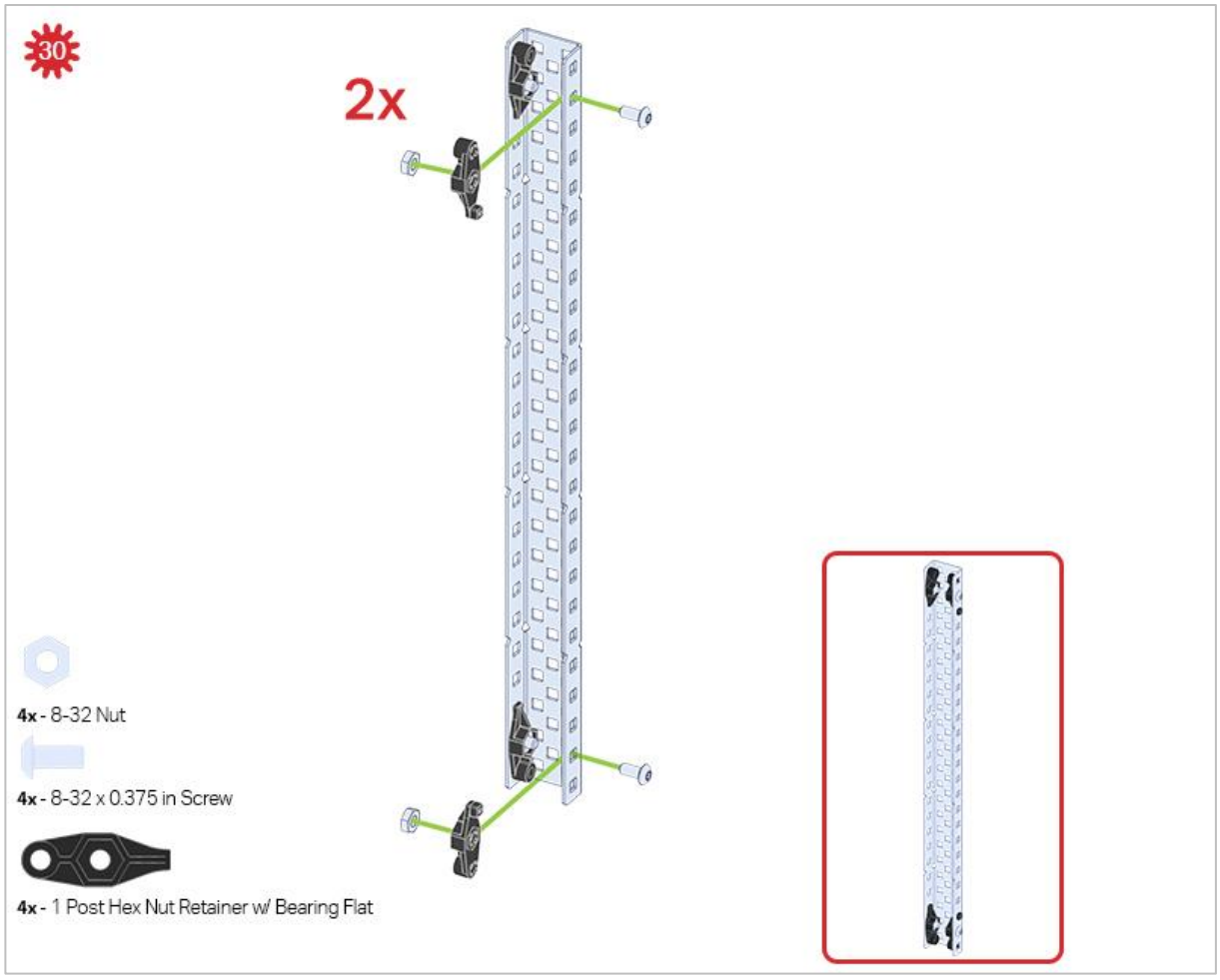
1x - 0.875 in Spacer

2x - 1 Post Hex Nut Retainer

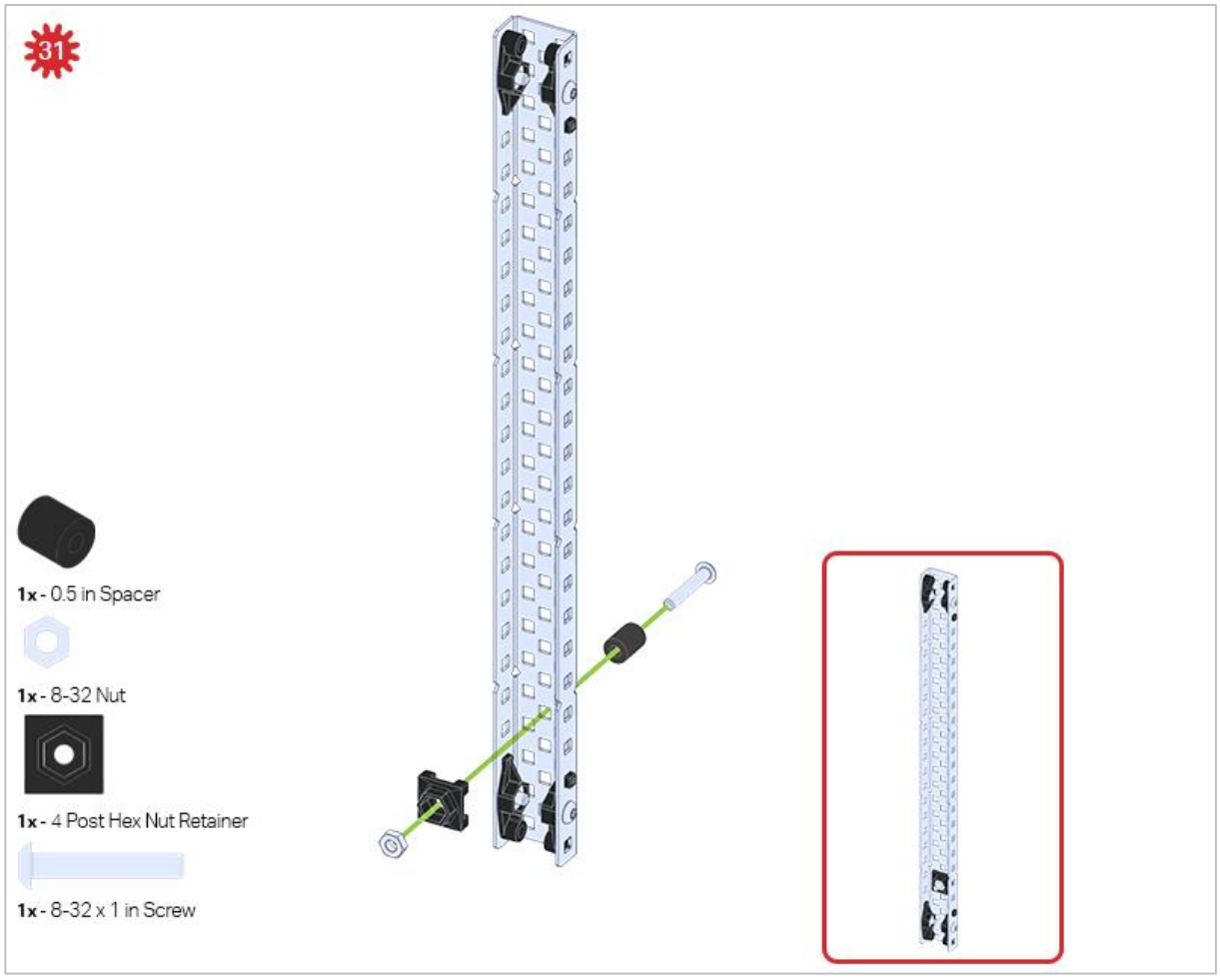
1x - 8-32 x 1.5 in Screw



Be sure to make two assemblies in this step!



This step adds onto the two assemblies started in Step 29.



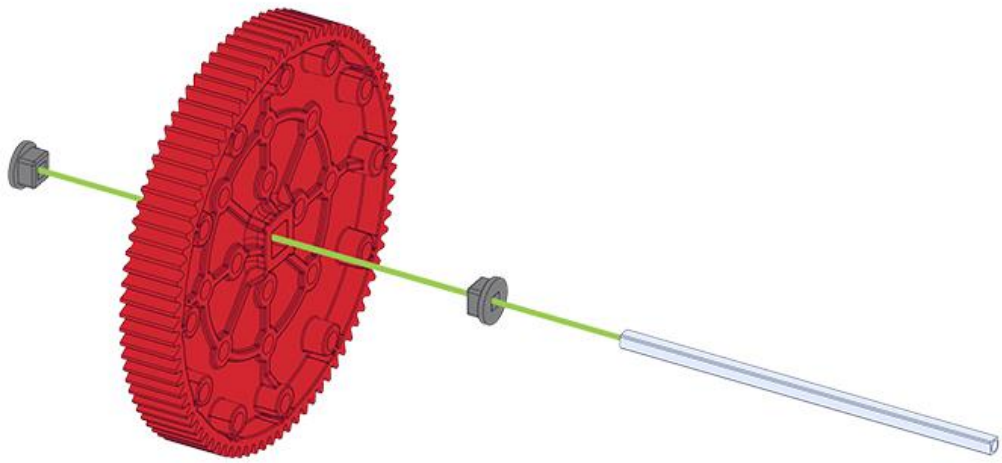
Make sure to add this to only one of the two sub-assemblies you just made.

32

1x - 8-32 Nut

1x - 1 Post Hex Nut Retainer

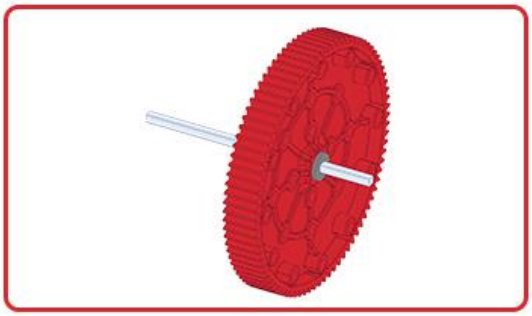
33

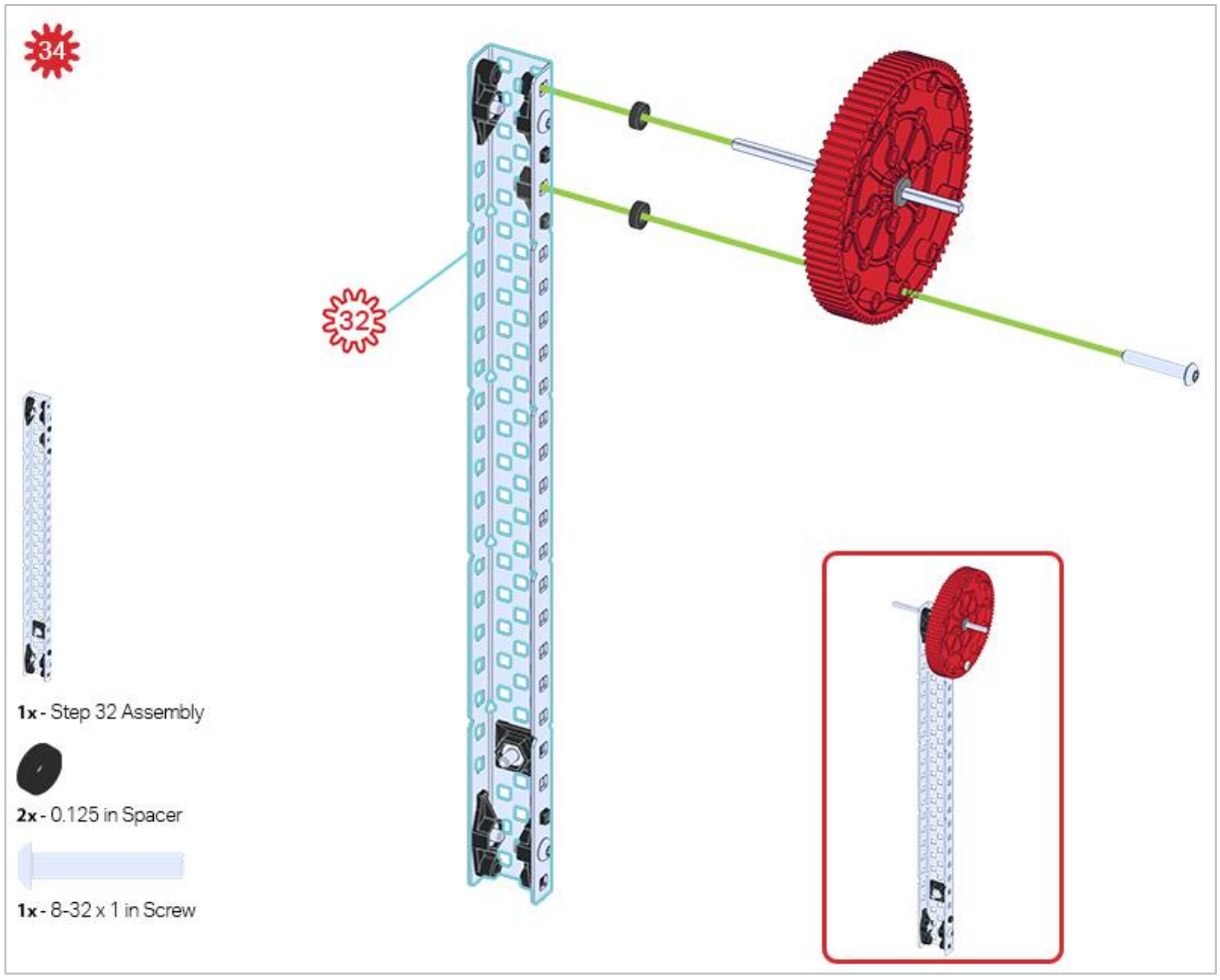


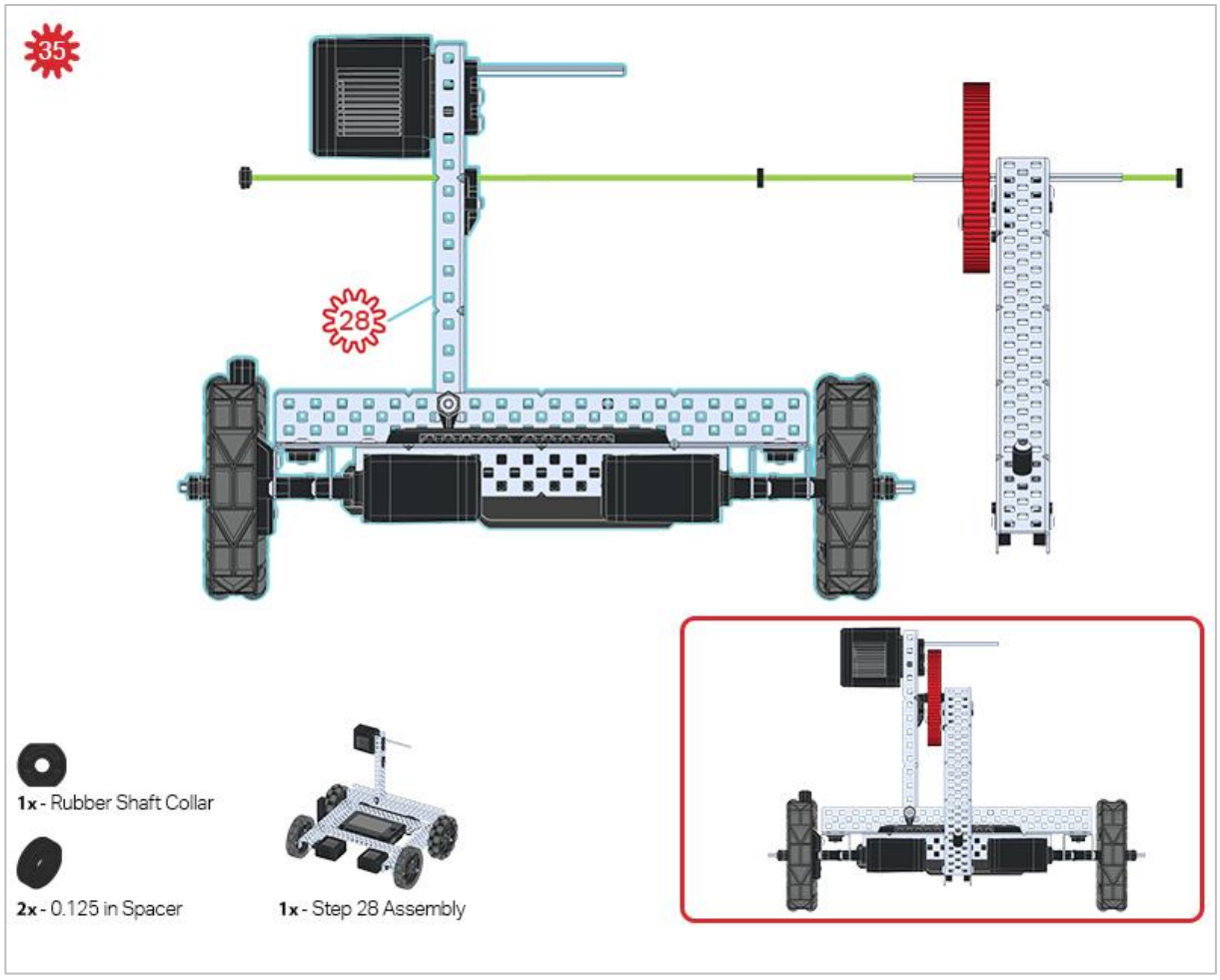
1x - High Strength 84 Tooth Gear

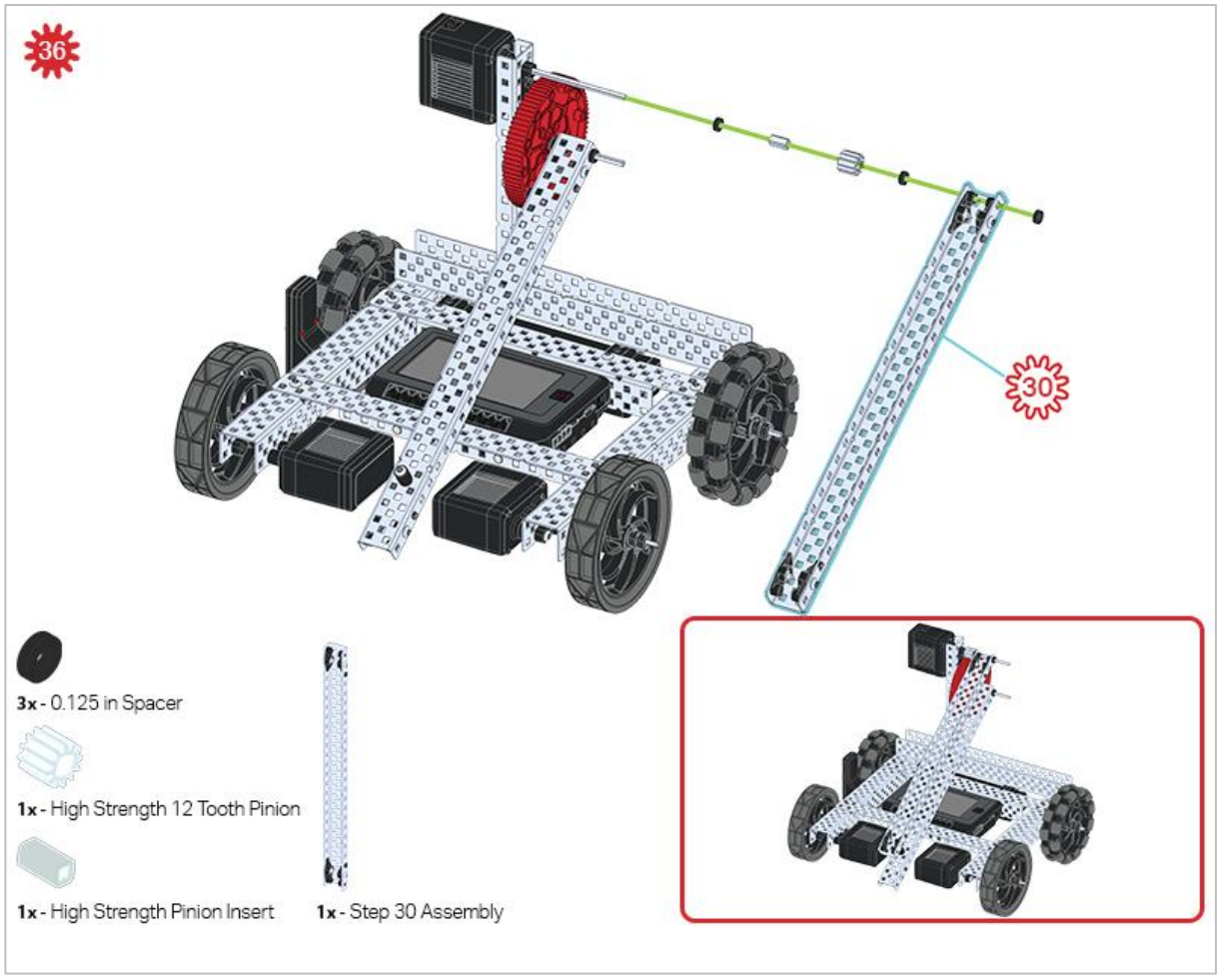
2x - High Strength Shaft Insert

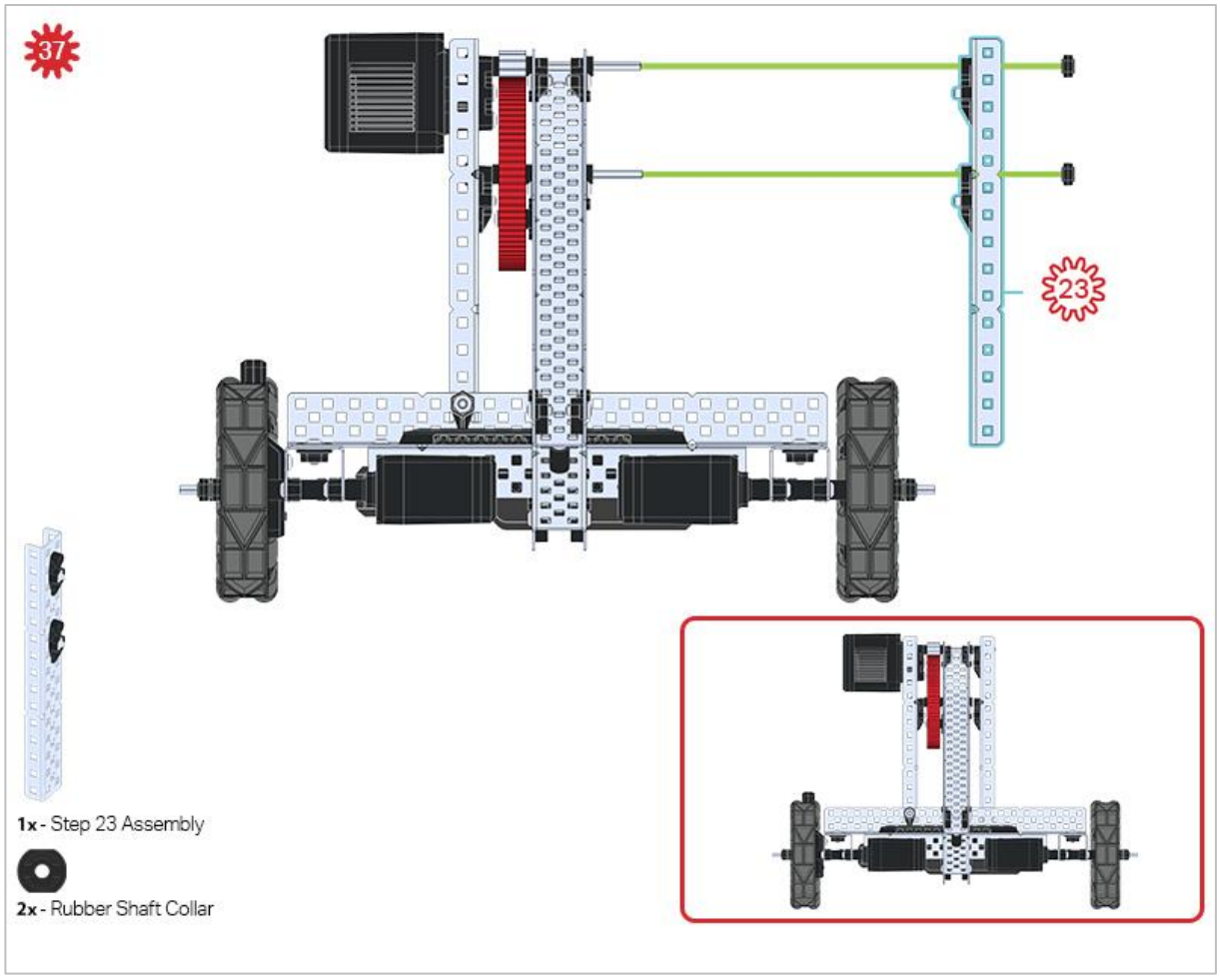
1x - 3.5 in Shaft




34
32
1x - Step 32 Assembly
2x - 0.125 in Spacer
1x - 8-32 x 1 in Screw


35
28
1x - Rubber Shaft Collar
2x - 0.125 in Spacer
1x - Step 28 Assembly

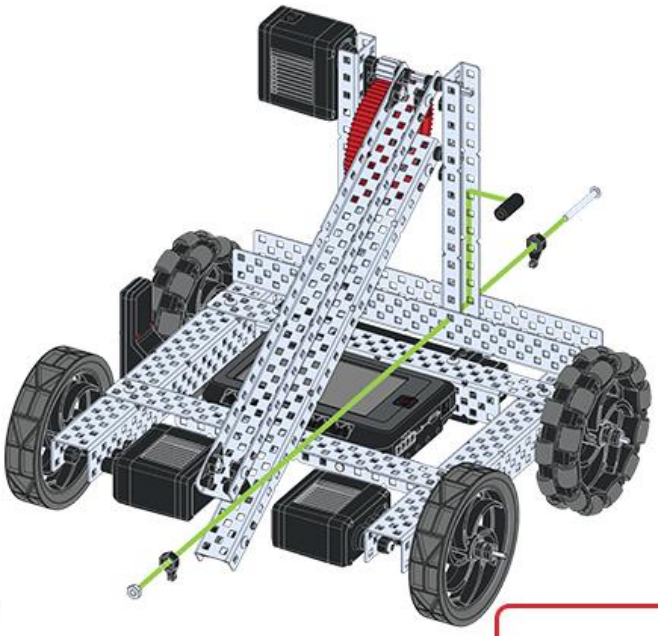

36
30
3x - 0.125 in Spacer
1x - High Strength 12 Tooth Pinion
1x - High Strength Pinion Insert
1x - Step 30 Assembly



1x - Step 23 Assembly

2x - Rubber Shaft Collar

38

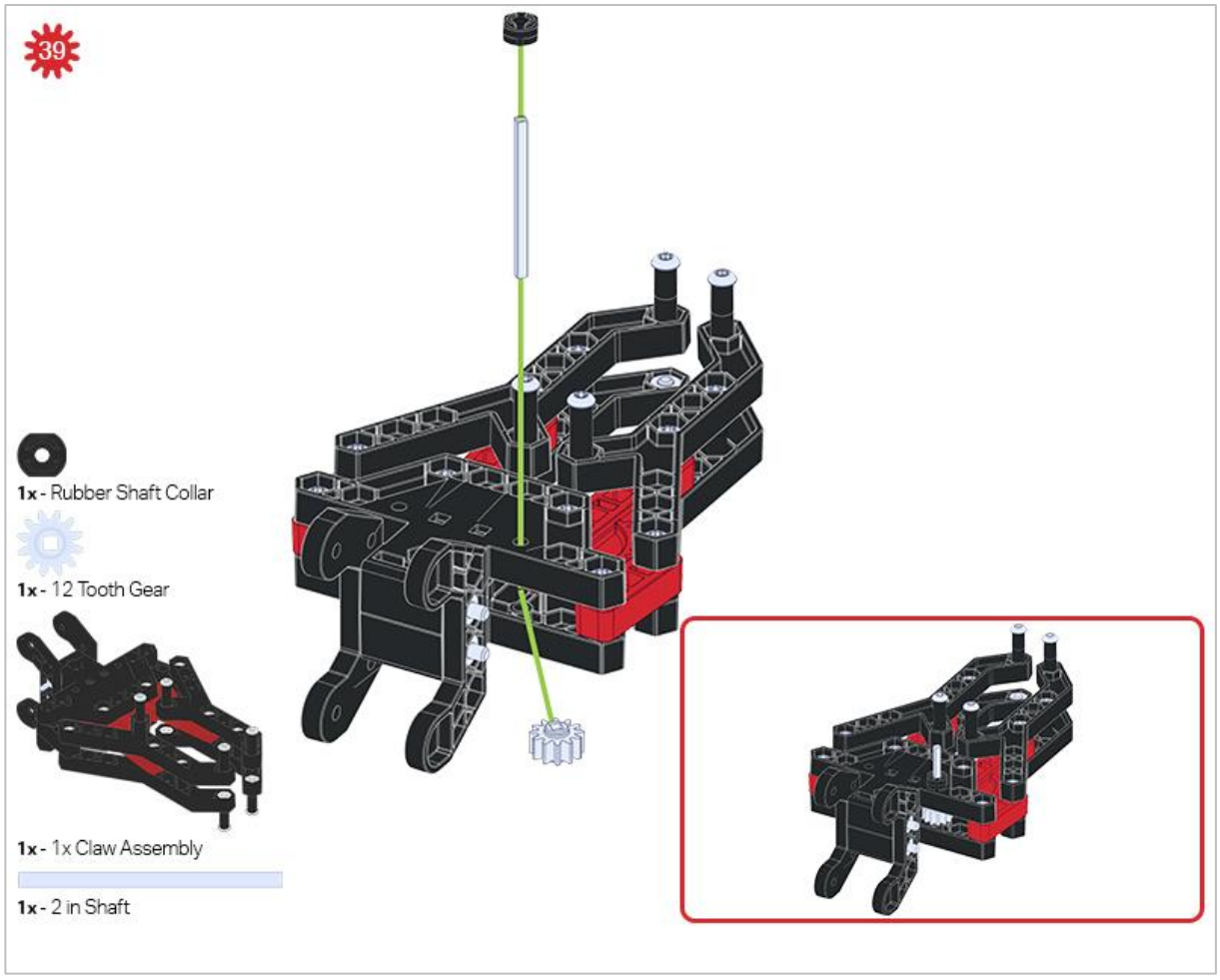


1x - 0.875 in Spacer

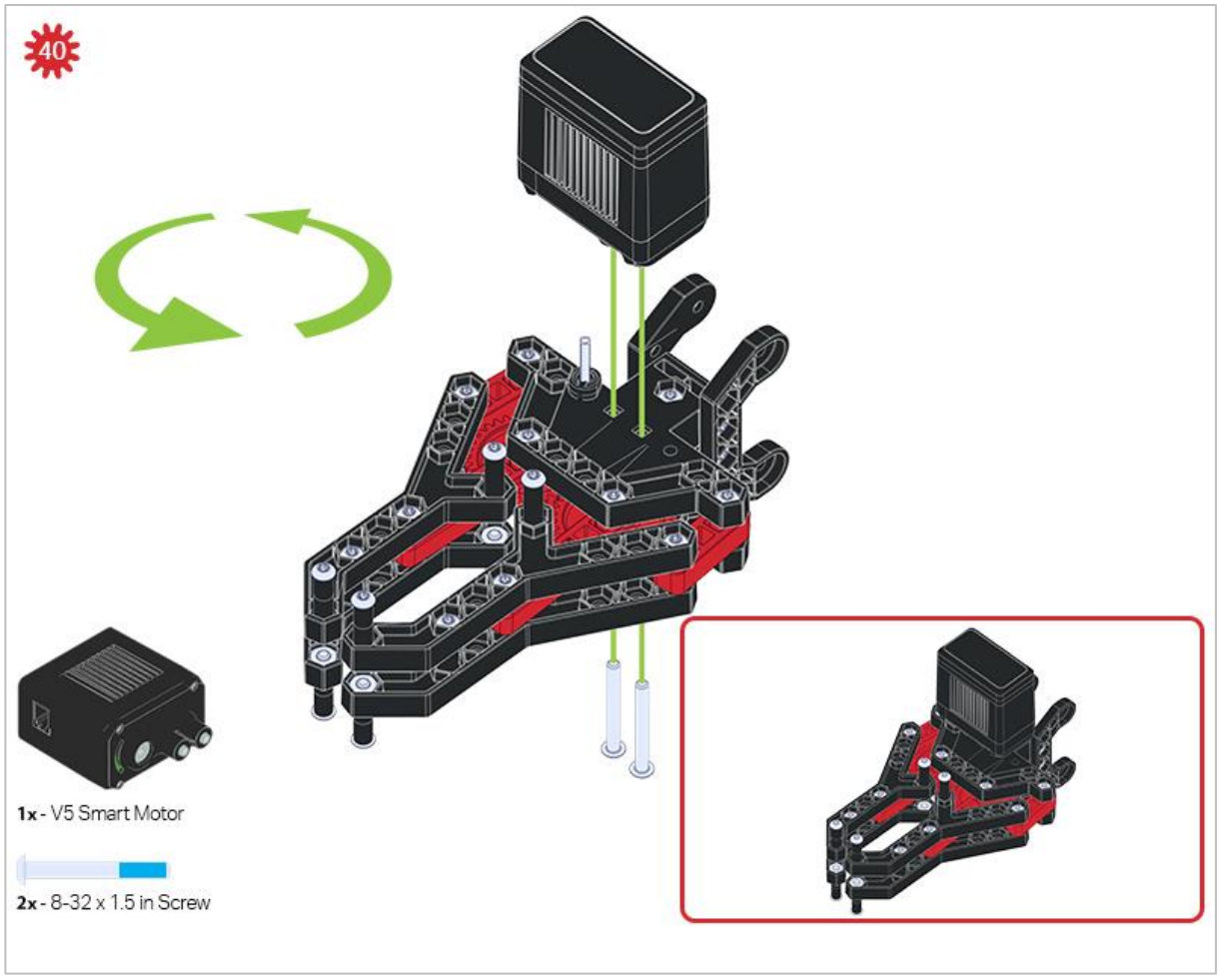
1x - 8-32 Nut

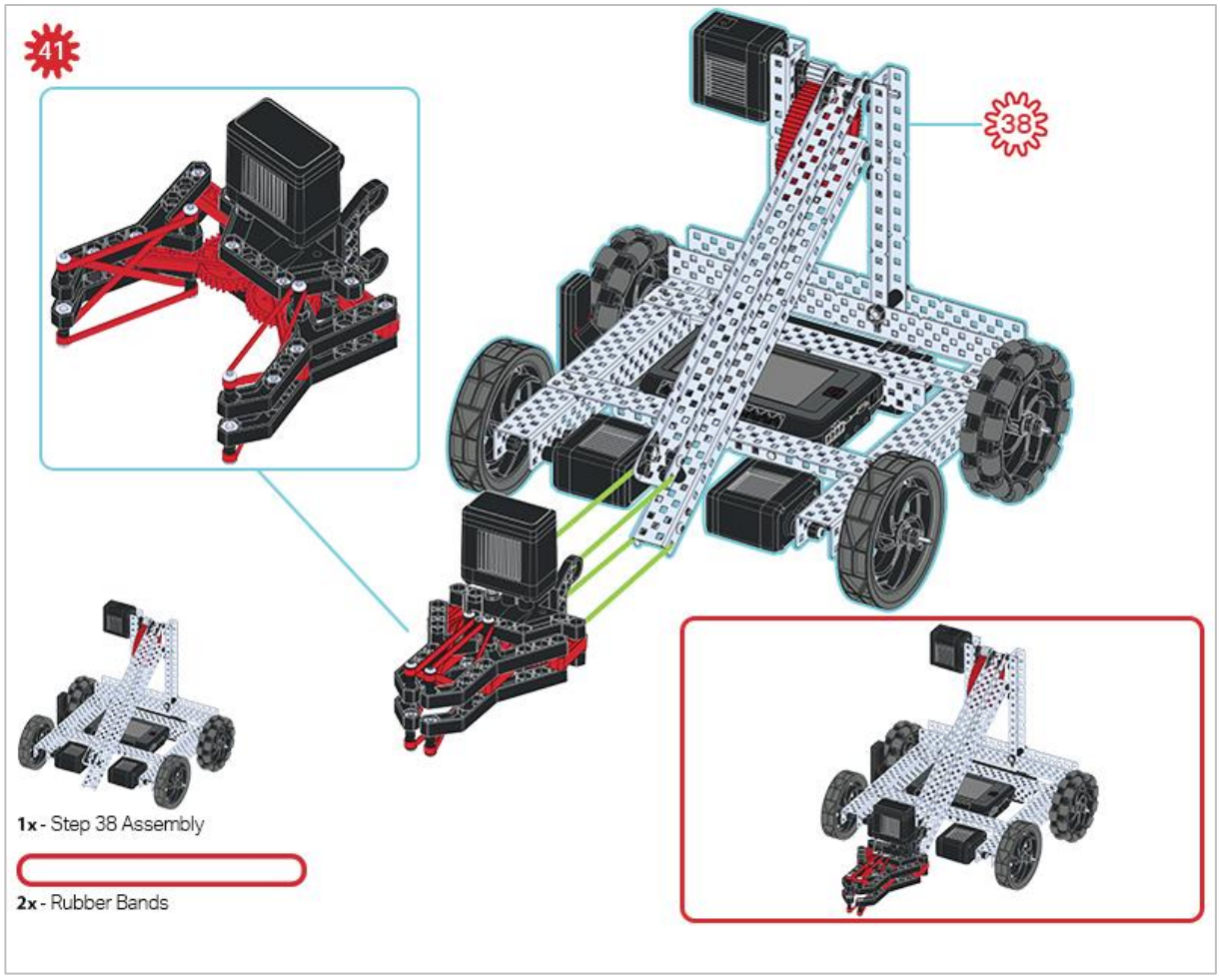
2x - 1 Post Hex Nut Retainer

1x - 8-32 x 1.5 in Screw



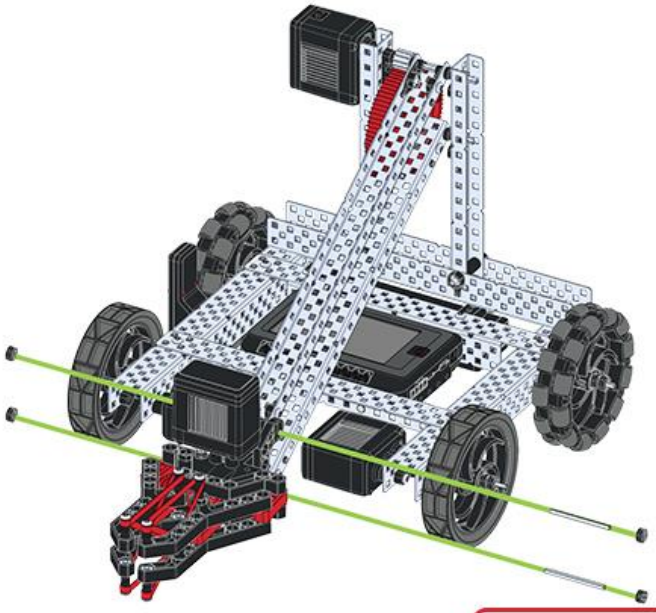
Make sure the 12- tooth gear is installed on the right side of the claw.


40
1x - V5 Smart Motor
2x - 8-32 x 1.5 in Screw



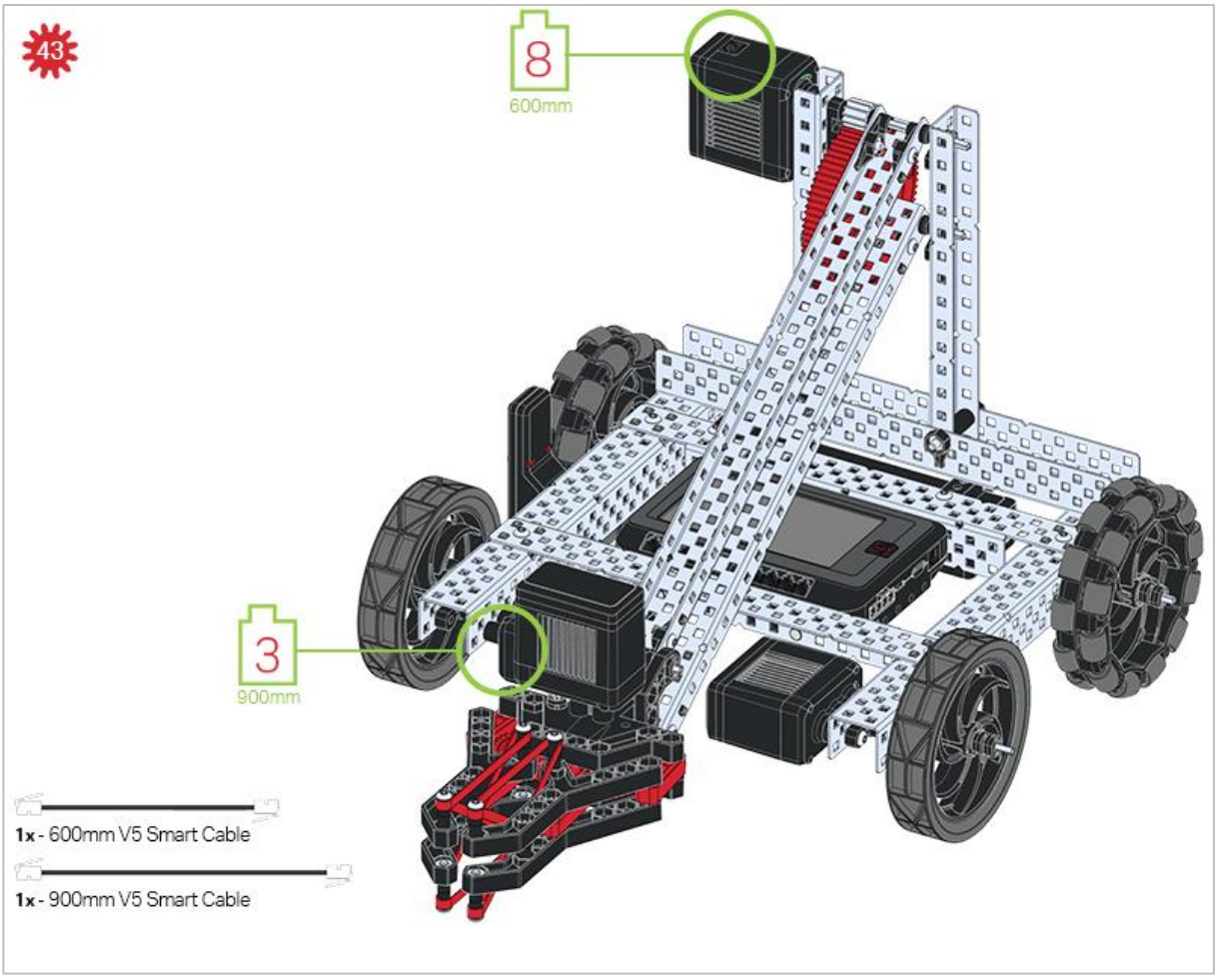
Make sure that the port on the Smart Motor is facing the right side of the robot when the claw is installed (the same side as the V5 Radio).

42



4x - Rubber Shaft Collar

2x - 2 in Shaft



Build Instruction Tips

Check the Appendix for information on how to use the new Hex Nut Retainers.

Exploration

Now that you've finished the build, test what it does. Explore your build and then answer this question in your engineering notebook.

How would the speed of the arm change if the High Strength 84 Tooth Gear in the build's Step 33 was changed to a smaller diameter High Strength 60 Tooth Gear?

For help with this question, compare the rotational speed (RPM) of the High Strength 12 Tooth Pinion (from Step 36 of the build) to the rotational speed of the 84 Tooth Gear on the robot by gently moving the arm up and down. Be sure to justify your answer with your observation.

SPARK
PLAY

Test your build, observe how it functions,
and fuel your logic and reasoning skills
through imaginative, creative play.

Decision Making-VEXcode V5 Blocks

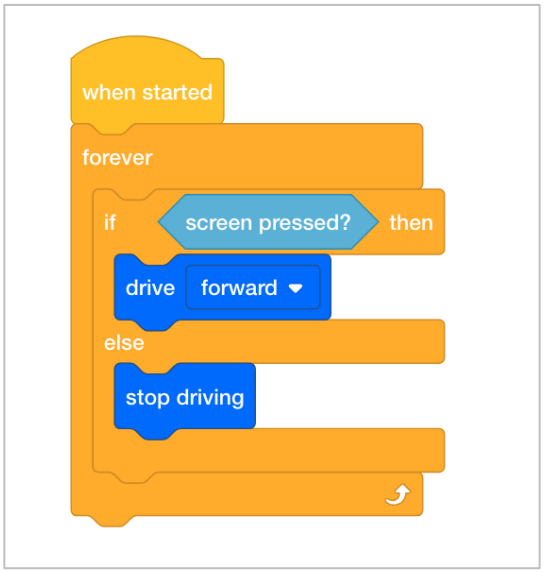
TRUE and FALSE paths

Decision Making

At their most basic level, programs are written to accomplish simple sequences of behavior. For example, you might want your robot to drive forward and also make some turns to reach a destination. But, what if you want your robot to wait for the right time to start driving forward and complete its route? That would require programming with conditional statements. You would use a conditional statement to define what the "right time to start" is within your project. Maybe the "right time" is after a button is pressed or when a sensor detects a specific level and then it starts driving. When you watch the robot's behavior, it will seem like it is deciding when to start driving but it's because you set the condition for when driving should start.

Conditional statements are powerful programming statements that use a boolean (TRUE or FALSE) condition. Using the same example scenario as above, you could program your robot to repeatedly check if its brain screen is pressed and drive forward when it is. The conditional statement in that project may read something similar to, "If the screen detects that it is pressed (TRUE), run the driving sequence." This statement does not mention any behavior if the condition is FALSE (the screen is not pressed) so the robot takes no action when FALSE. Conditional statements allow you to develop projects that have the robot behave differently depending on what it senses.

In the following example, if the Brain's screen is pressed (TRUE) the robot will drive forward. If the Brain's screen is not pressed (FALSE) the robot will stop driving. This shows the robot only driving forward when the Brain's screen is pressed, otherwise the robot stops.



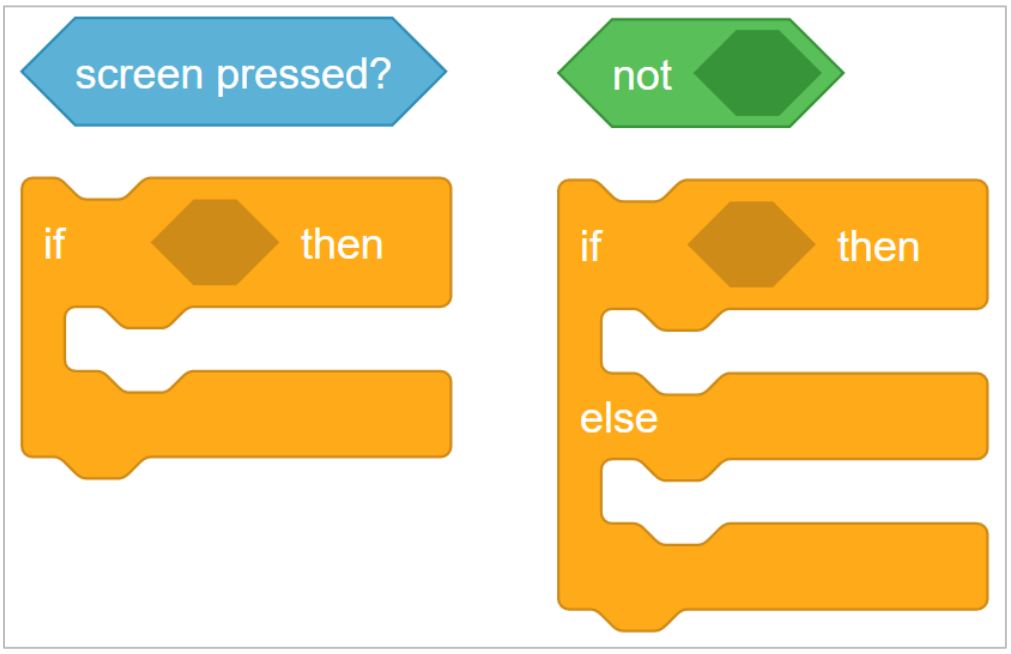
Programming with Conditionals - VEXcode V5 Blocks

Hardware/Software Required:

Amount	Hardware/Software
1	VEX V5 Classroom Starter Kit (with up-to-date firmware)
1	VEXcode V5 Blocks (latest version, Windows, MacOS, Chromebook)
1	Engineering Notebook
1	Clawbot (Drivetrain 2-motor, No Gyro) Template

The Clawbot is ready to make decisions!

This activity will give you the tools to program your robot with conditional behaviors. The *if then* and *if then else* blocks are the main focus within the activity but Operators and Sensing blocks are also used.



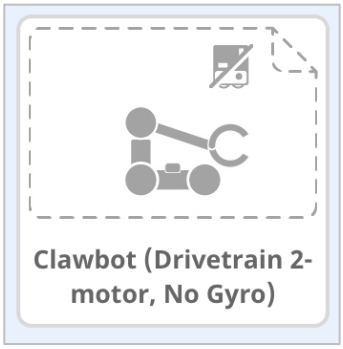
You can use the Help information inside of VEXcode V5 Blocks to learn about the blocks. For guidance in using the Help feature, see the Using Help tutorial.

1. Let's start with an understanding of conditional statements.

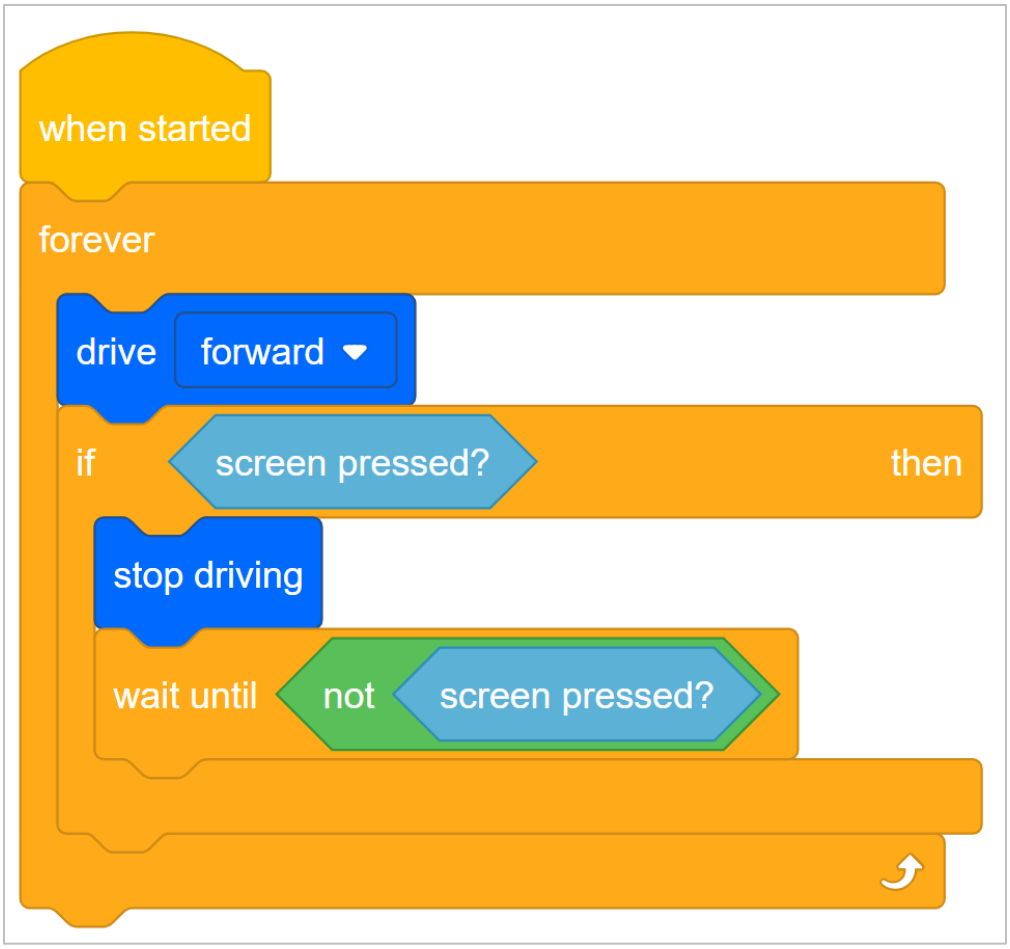
Before you begin programming with conditionals, first watch the If-Then-Else tutorial video below. It can also be found as a Tutorial video in VEXcode V5 Blocks.

2. Let's start programming with conditional statements.

- Open the Clawbot (Drivetrain 2-motor, No Gyro) template example project.

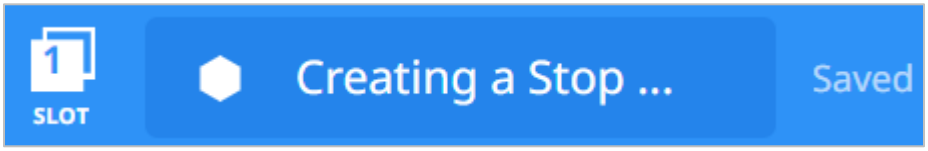


- Build the project below.



Do the following in your engineering notebook:

- Explain what the project has the Clawbot do. You will need to explain more than the fact that it creates a stop button. Explain which blocks make the Clawbot do what.
- Write a one sentence summary that captures what the project does.

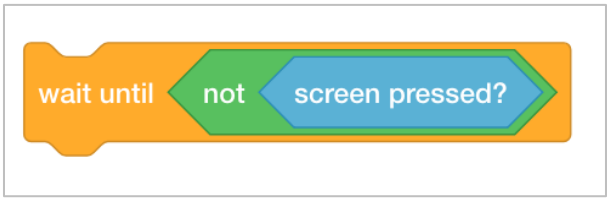


- Test to see if your prediction of what the project has the Clawbot do is correct.
- Save and download the project as **Creating a Stop Button** to Slot 1 on the Clawbot, and then run it.
- For help downloading a project, see the tutorial in VEXcode V5 Blocks that explains how to Download and Run a Project.
- Check your explanations of the project and add notes to correct them as needed.

3. Understanding the *wait until* block.

Notice that if the Brain's screen is pressed, the flow of the project moves so quickly that the project will move to the next block, which is the *stop driving* block.

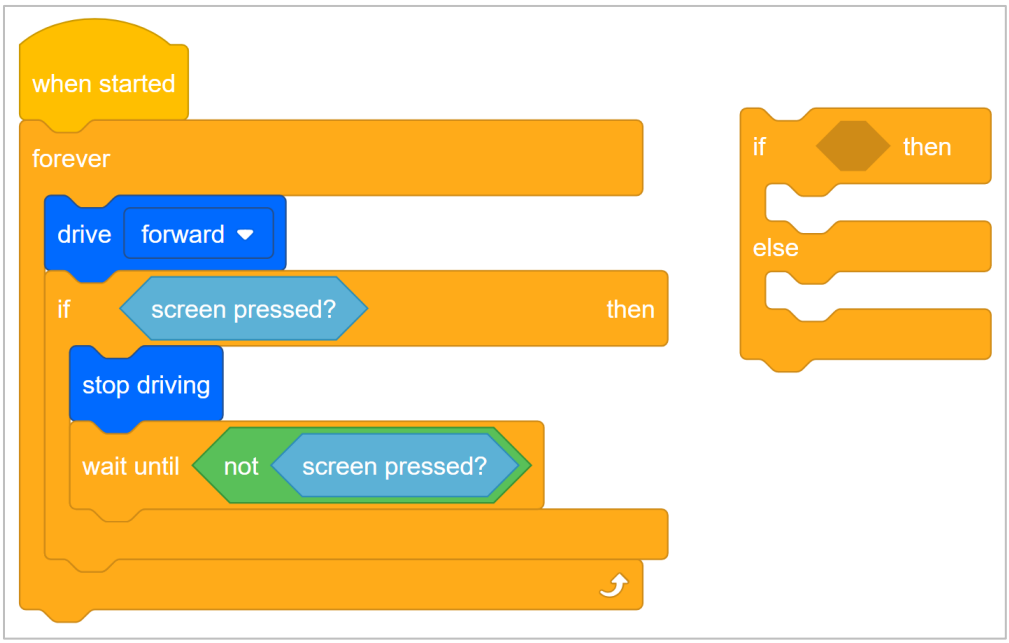
Thus, the project needs a *wait until* block that tells the robot to remain stopped until the Brain's screen is released. Otherwise, the *forever* block would cause the project to begin again with the *drive* block.



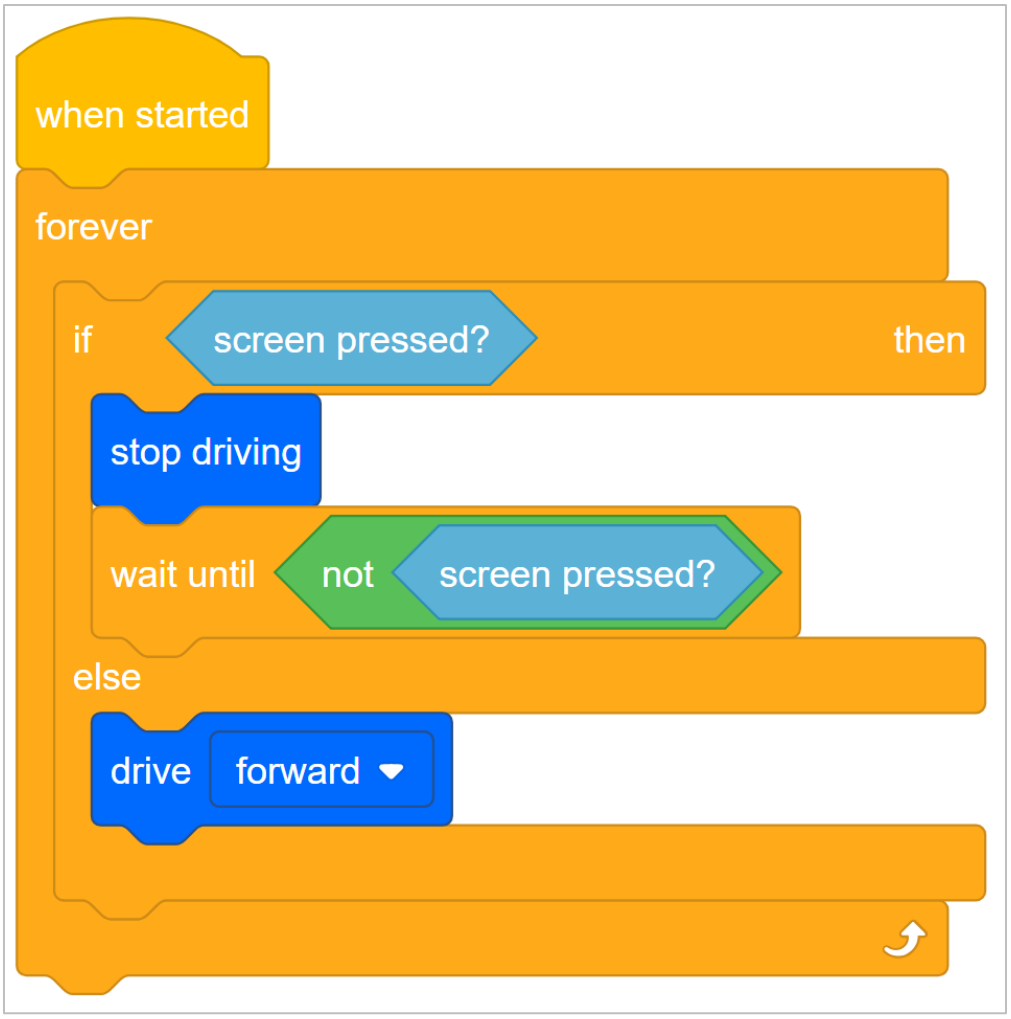
The *wait until* block is necessary because of the speed of the project's flow. If it was not there, the project would move to the next block before the robot ever had time to respond.

4. Change the project.

Our next step is changing the *if then* block to an *if then else* block.



- Start by saving **Creating a Stop Button** as the new project, **StopOrDrive**.
- If you need help saving a project, see the Naming and Saving Your Project tutorial in VEXcode V5 Blocks.
- Then build the **StopOrDrive** project shown below.



- Download **StopOrDrive** to Slot 2 on your Clawbot.
- For help downloading a project, see the tutorial in VEXcode V5 Blocks that explains how to Download and Run a Project.
- Test **Creating a Stop Button** (Slot 1) and then test **StopOrDrive** (Slot 2) and compare them to see if there are any difference in the robot's behavior. Note any differences in your engineering notebook.

The two projects have the Clawbot behave the same way. The only difference is the use of the *if then else* block in the **StopOrDrive** project.

Using the *if then else* block will allow you to add additional buttons to the screen in upcoming activities.

Adding a Second Button to the Brain's Screen-VEXcode V5 Blocks

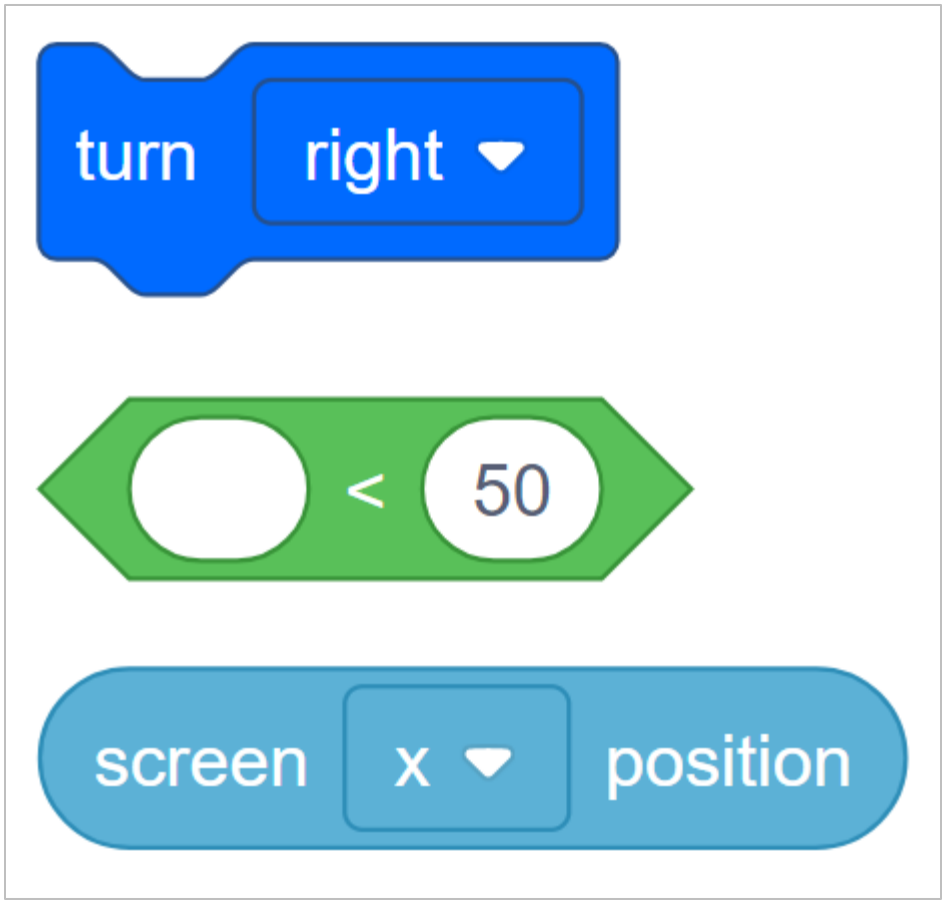
Hardware/Software Required:

Amount	Hardware/Software
1	VEX V5 Classroom Starter Kit (with up-to-date firmware)
1	VEXcode V5 Blocks (latest version, Windows, MacOS, Chromebook)
1	Engineering Notebook
1	StopOrDrive project from the previous Play page

The brain's screen can have more than one button.

This activity will let you program the robot to drive forward and turn left or right depending on which side of the brain's screen is pressed.

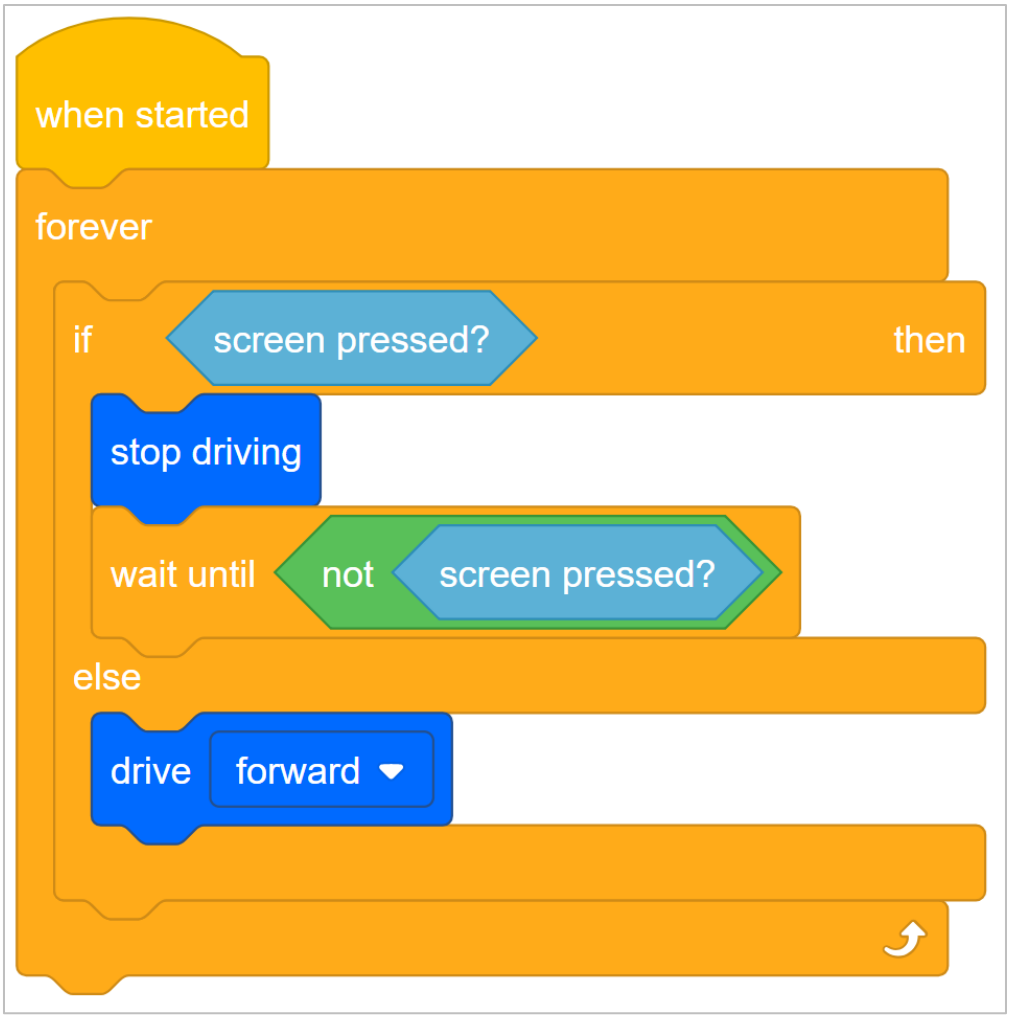
The three additional types of blocks that you will need during this activity are the following:



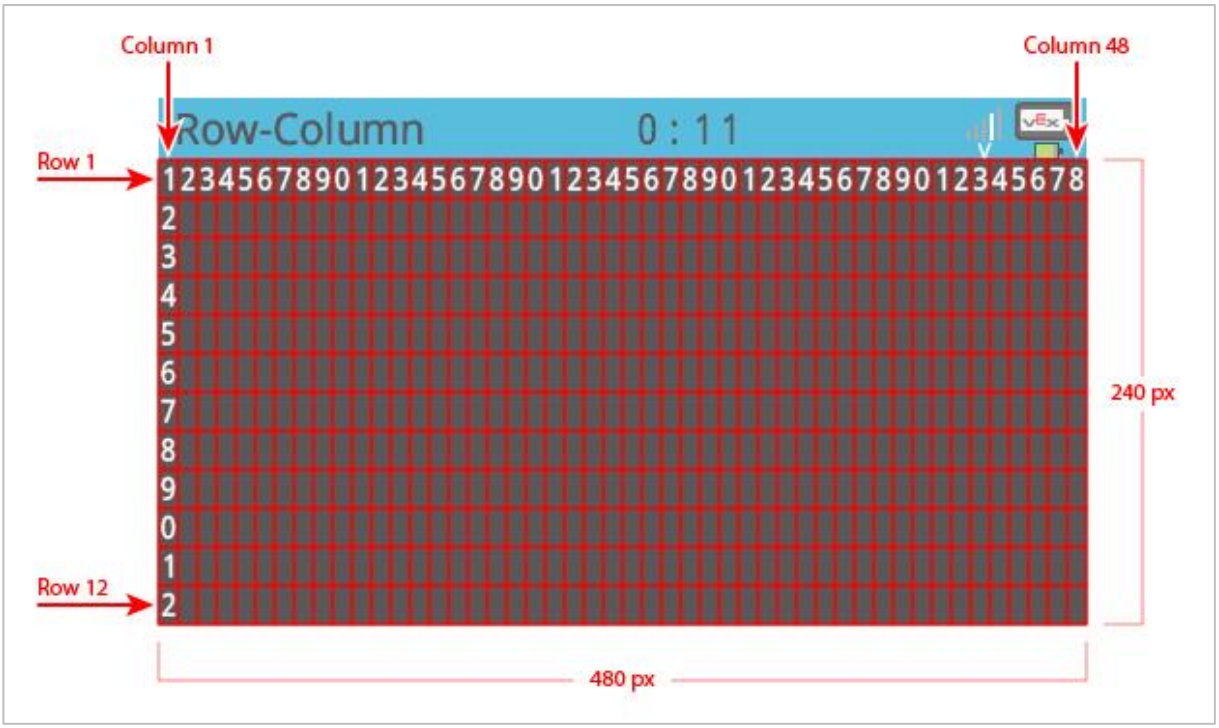
You can use the Help information inside of VEXcode V5 Blocks to learn about the blocks. For guidance in using the Help feature, see the Using Help tutorial.

1. Let's start by reviewing the StopOrDrive project.

The StopOrDrive project had the Clawbot stop if the screen was pressed, or else it had it drive forward.
The entire screen was one big button but in this next project, we want half the screen to be one button and the other half to be the other.



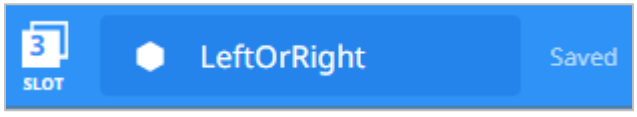
In order to split the screen into two buttons, we need to understand more about the layout of the screen.



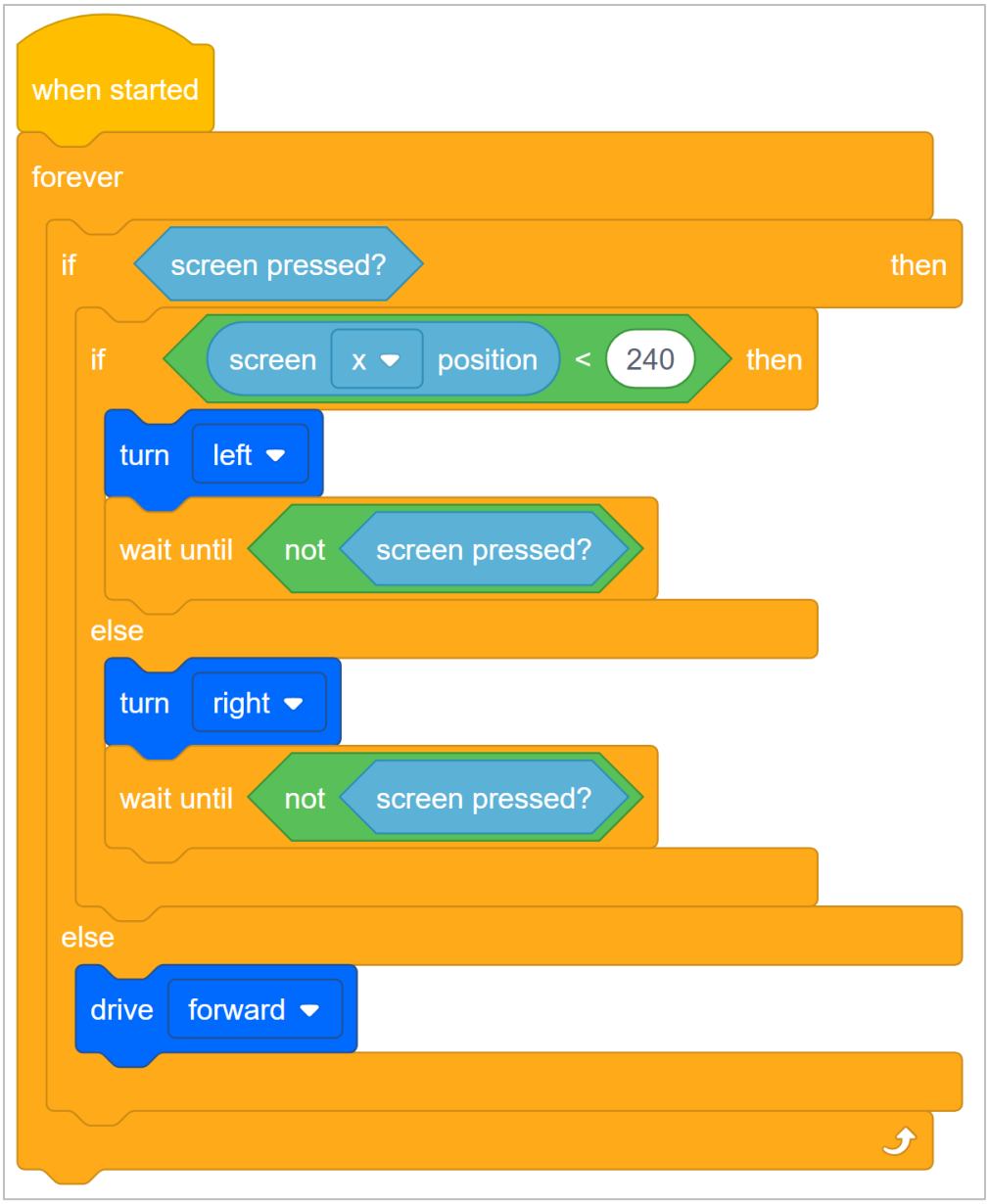
- Notice that the columns increase in number from left to right. The number of columns is 48 and the screen is 480 pixels wide.
- Write down in your engineering notebook that the x-value on the screen is equal to the number of pixels measured from left to right.
- What is the x-value of the center of the screen? For this activity, you can focus on the x-axis alone because you only need a left and right button.

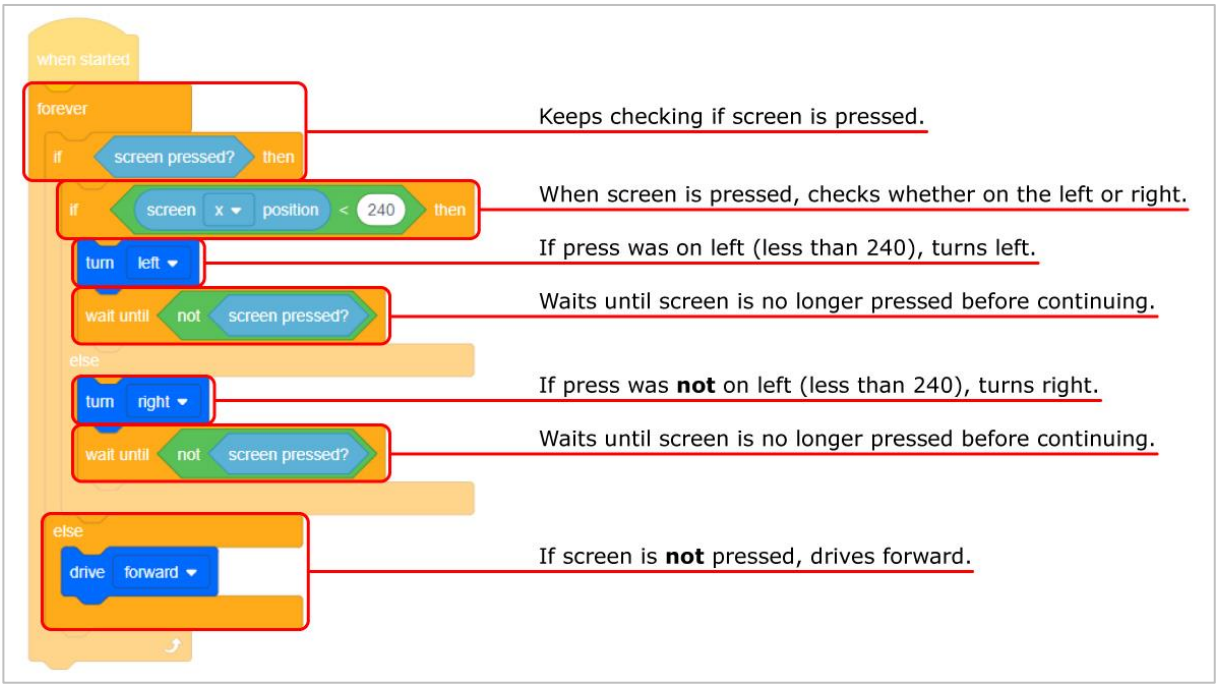
2. Programming for two buttons.

- Save **StopOrDrive** as the **LeftOrRight** project.
- Remember, if you need help opening, naming, or saving projects, watch the Tutorials in VEXcode V5 Blocks.



- Build the project below. It will have the Clawbot turn left or right when the screen is pressed, depending on the side it is pressed on.


when started
forever
if
screen pressed?
then
if
screen
x
position
<
240
then
turn
left
wait until
not
screen pressed?
else
turn
right
wait until
not
screen pressed?
else
drive
forward



- Let's review what this project does.

  It keeps checking if the screen is pressed. If the screen isn't pressed it drives forward but if it is, it checks where the screen is pressed.

  If the press was on the left side (less than 240), it turns left. Otherwise, it turns right. We don't need another condition for when the x-value is greater than 240 because if it isn't less than 240 (turn left), it must be greater (turn right). We only have two buttons to worry about.

  The *wait until* Control blocks after each turn have the project wait until the screen is no longer being pressed before continuing.

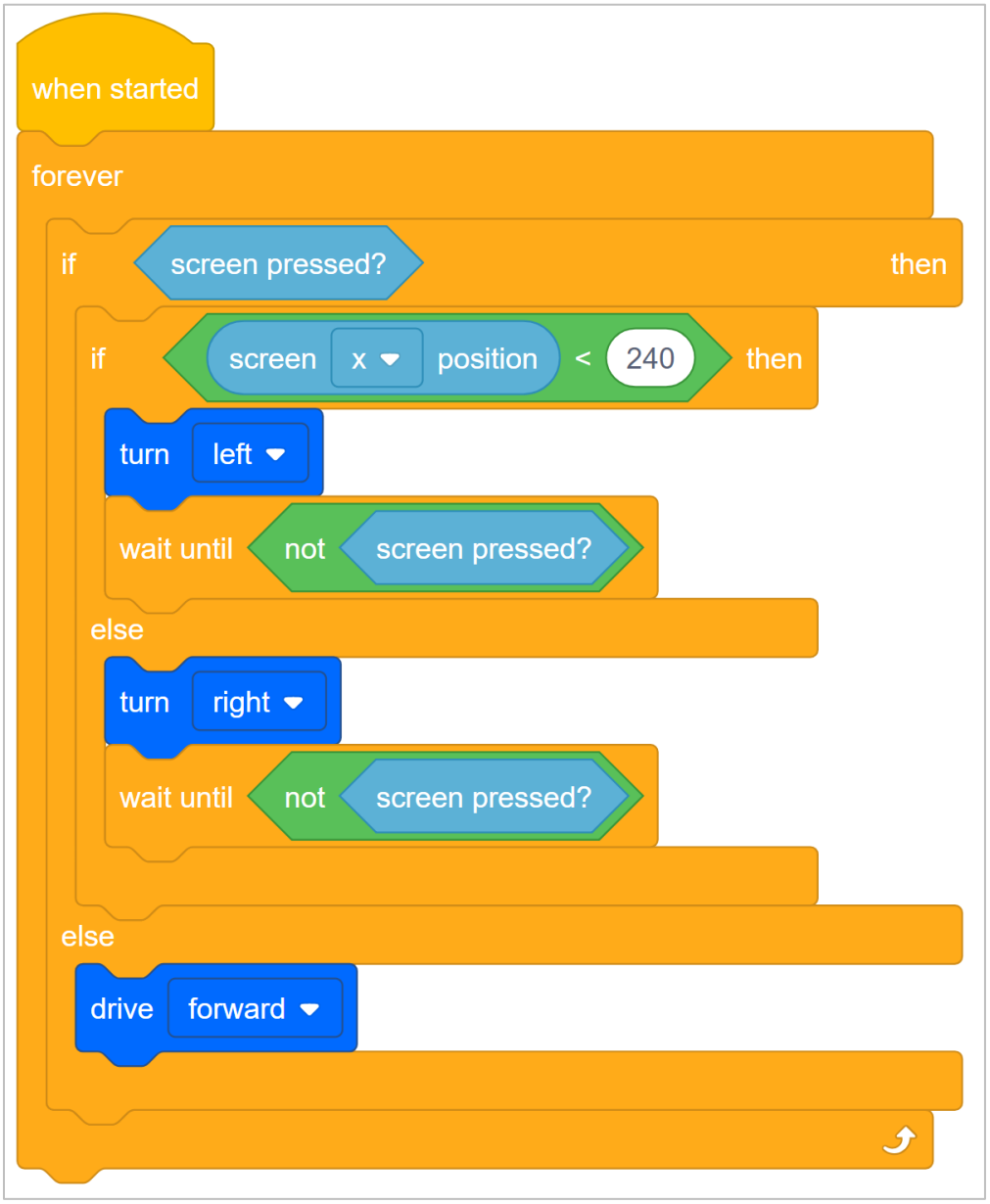
- Now that the project is done, download and run it to test how it works.
- For help, watch the Download and Run a Project tutorial video in VEXcode V5 Blocks.
- Take notes in your engineering notebook about how the buttons control the movements of the Clawbot.

3. Adjust the project for a better User Experience.

When pressing the screen's buttons from behind the Clawbot as it drove forward, you pressed on the right side of the screen to turn left and on the left side of the screen to turn right. That is not a good User Experience. A User Experience is how well a user can interact

with a User Interface to control a computer system. There is more information about User Interfaces in the Apply section of this lab.

In this case, we need to improve the User Interface in order to improve the User Experience.



- Review the LeftOrRight project and revise it so that when the user presses the buttons from behind the Clawbot, the robot turns right when the user presses the left side of the screen. Or else, the Clawbot turns left.
- Plan, test, and iterate on this project in your engineering notebook so that the project makes the Clawbot turn toward the side of the screen that the user is pressing from behind the Clawbot.

Decision Making-VEXcode V5 Text

TRUE and FALSE paths

Decision Making

At their most basic level, programs are written to accomplish simple sequences of behavior. For example, you might want your robot to drive forward and also make some turns to reach a destination. But, what if you want your robot to wait for the right time to start driving forward and complete its route? That would require programming with conditional statements. You would use a conditional statement to define what the "right time to start" is within your project. Maybe the "right time" is after a button is pressed or when a sensor detects a specific level and then it starts driving. When you watch the robot's behavior, it will seem like it is deciding when to start driving but it's because you set the condition for when driving should start.

Conditional statements are powerful programming statements that use a boolean (TRUE or FALSE) condition. Using the same example scenario as above, you could program your robot to repeatedly check if its brain screen is pressed and drive forward when it is. The conditional statement in that project may read something similar to, "If the screen detects that it is pressed (TRUE), run the driving sequence." This statement does not mention any behavior if the condition is FALSE (the screen is not pressed) so the robot takes no action when FALSE. Conditional statements allow you to develop projects that have the robot behave differently depending on what it senses. For more information on Boolean Logic, click here.

In the following example, if the Brain's screen is pressed (TRUE) the robot will drive forward. If the Brain's screen is not pressed (FALSE) the robot will stop driving. This shows the robot only driving forward when the Brain's screen is pressed, otherwise the robot stops.

```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();

  while (true) {
    if (Brain.Screen.pressing()){
     Drivetrain.drive(forward);
     }else{
      Drivetrain.stop();
    }
  }
}

```

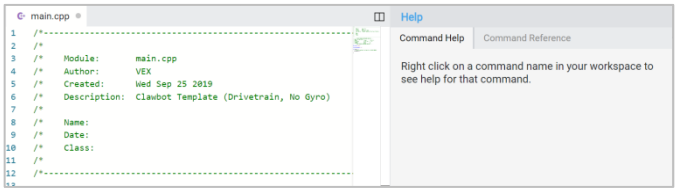
Programming with Conditionals - VEXcode V5 Text

Hardware/Software Required:

Amount	Hardware/Software
1	VEX V5 Classroom Starter Kit (with up-to-date firmware)
1	VEXcode V5 Text (latest version, Windows, MacOS)
1	Engineering Notebook
1	Clawbot (Drivetrain 2-motor, No Gyro) Template

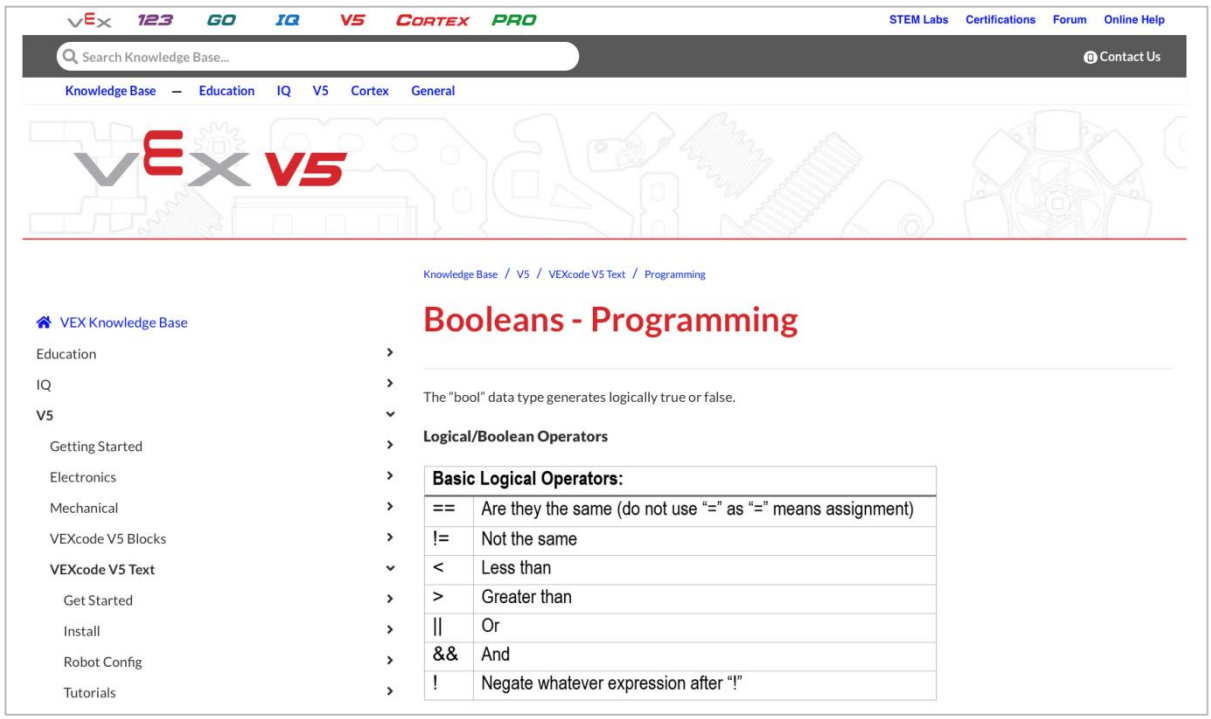
The Clawbot is ready to make decisions!

During this exploration you can access help right in the VEXcode V5 Text program.



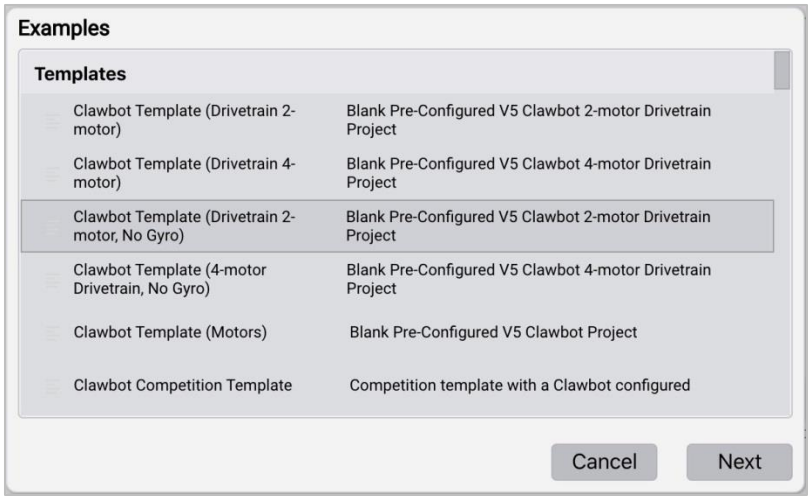
1. Let's start with an understanding of conditional statements.

Before you begin programming with conditionals, read the VEX Knowledge Base Article explaining *If Then Else* Statements. The article can be found here.For a list of operators to use in the *If Then Else* statements, read the VEX Knowledge Base Article explaining Booleans. This article can be found here.



2. Let's start programming with conditional statements.

- Open the Clawbot (Drivetrain 2-motor, No Gyro) template example project.



- Build the project below.

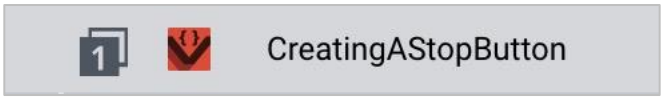
```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();
  while (true) {
    Drivetrain.drive(forward);
    if (Brain.Screen.pressing()) {
      Drivetrain.stop();
      waitUntil(!Brain.Screen.pressing());
    }
  }
}
```

Do the following in your engineering notebook:

- Explain what the project has the Clawbot do. You will need to explain more than the fact that it creates a stop button. Explain which instructions make the Clawbot do what.
- Write a one sentence summary that captures what the project does.
- Test to see if your prediction of what the project has the Clawbot do is correct.



- Save and download the project as **CreatingAStopButton** to Slot 1 on the Clawbot, and then run it.
- For help downloading a project, see the tutorial in VEXcode V5 Text that explains how to Download and Run a Project.
- Check your explanations of the project and add notes to correct them as needed.

3. Understanding the *wait until* statement.

Notice that if the Brain's screen is pressed, the flow of the project moves quickly and the project will move to the next instruction, which is the *Drivetrain.stop()* instruction.

Thus, the project needs a *waitUntil()* instruction that tells the robot to remain stopped until the Brain's screen is released. Otherwise, the *forever* statement would cause the project to begin again.

```
waitUntil(Brain.Screen.pressing());
```

The *waitUntil*() instruction is necessary because of the speed of the project's flow. If it was not there, the project would move to the next instruction before the robot ever had time to respond.

4. Change the project.

Our next step is changing the *if then* statement to an *if then else statement.*

- Start by saving **CreatingAStopButton** as the new project, **StopOrDrive**.
- If you need help saving a project, click here.
- Then build the **StopOrDrive** project shown below.

```
int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();
while(true) {
  if (Brain.Screen.pressing())  {
    Drivetrain.stop();
    waitUntil(!Brain.Screen.pressing());
  }
  else {
  Drivetrain.drive(forward);
  }
}
}
```

- Download **StopOrDrive** to Slot 2 on your Clawbot.

- For help downloading a project, see the tutorial in VEXcode V5 Text that explains how to Download and Run a Project.
- Test **CreatingAStopButton** (Slot 1) and then test **StopOrDrive** (Slot 2) and compare them to see if there are any difference in the robot's behavior. Note any differences in your engineering notebook

The two projects have the Clawbot behave the same way. The only difference is the use of the *if then else* statement in the **StopOrDrive** project.

Using the *if then else* statement will allow you to add additional buttons to the screen in upcoming activities.

Adding a Second Button to the Brain's Screen-VEXcode V5 Text

Hardware/Software Required:

Amount	Hardware/Software
1	VEX V5 Classroom Starter Kit (with up-to-date firmware)
1	VEXcode V5 Text (latest version, Windows, MacOS)
1	Engineering Notebook
1	StopOrDrive project from the previous Play page

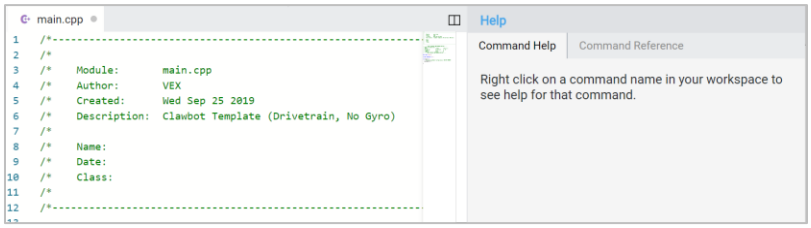
The brain's screen can have more than one button.

This activity will let you program the robot to drive forward and turn left or right depending on which side of the brain's screen is pressed.

The three additional types of instructions that you will need during this activity are the following:

- *Drivetrain.turn(right)*;
- *number* < 50
- *Brain.Screen.xPosition()*;

You can use the Help information inside of VEXcode V5 Text to learn about the instructions.



1. Let's start by reviewing the StopOrDrive project.

The StopOrDrive project had the Clawbot stop if the screen was pressed, or else it had it drive forward.

The entire screen was one big button but in this next project, we want half the screen to be one button and the other half to be the other.

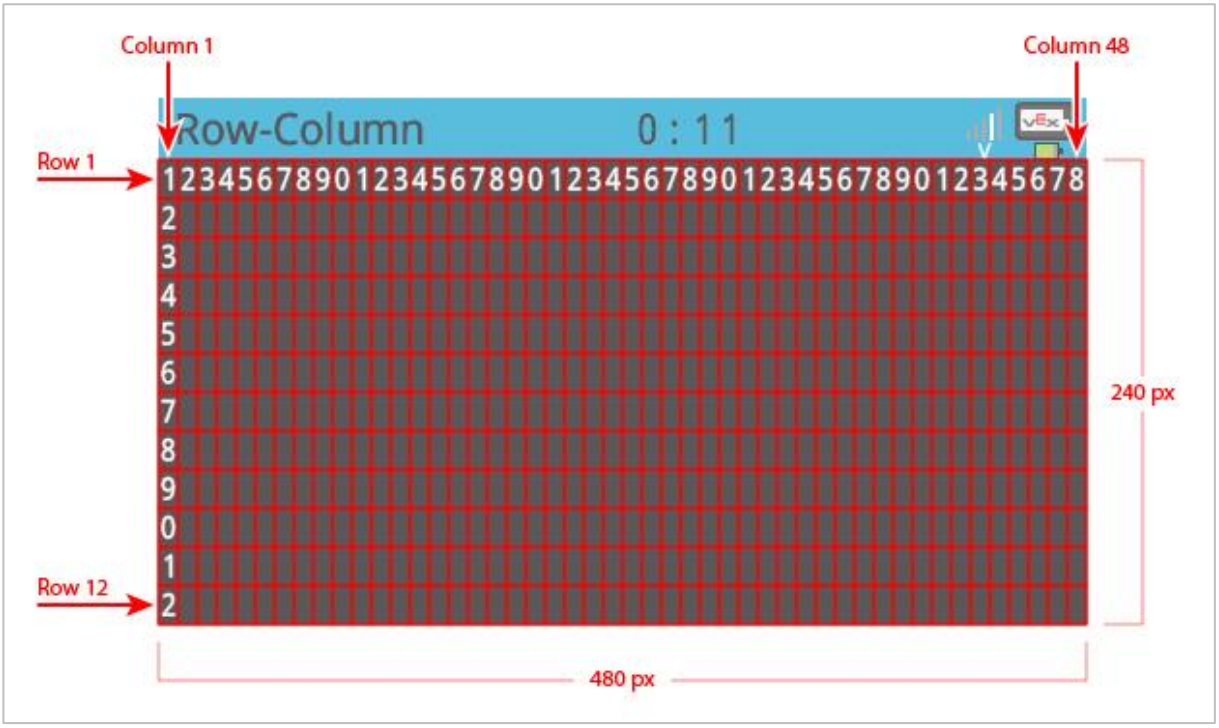
```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();
while (true) {
  Drivetrain.drive(forward);
  if (Brain.Screen.pressing()){
    Drivetrain.stop();
    waitUntil(!Brain.Screen.pressing());
    } else {

    }
  }
```

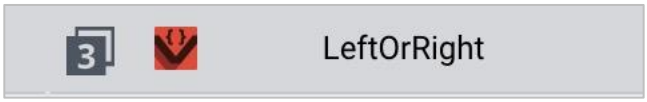
In order to split the screen into two buttons, we need to understand more about the layout of the screen.



- Notice that the columns increase in number from left to right. The number of columns is 48 and the screen is 480 pixels wide.
- Write down in your engineering notebook that the x-value on the screen is equal to the number of pixels measured from left to right.
- What is the x-value of the center of the screen? For this activity, you can focus on the x-axis alone because you only need a left and right button.

2. Programming for two buttons.

- Save **StopOrDrive** as the **LeftOrRight** project.



- Build the project below. It will have the Clawbot turn left or right when the screen is pressed, depending on the side it is pressed on.

```
19
20  int main() {
21    // Initializing Robot Configuration. DO NOT REMOVE!
22    vexcodeInit();
23    while (true) {
24      if (Brain.Screen.pressing()) {
25        if (Brain.Screen.xPosition() < 240) {
26          Drivetrain.turn(left);
27          waitUntil((!Brain.Screen.pressing()));
28        } else {
29          Drivetrain.turn(right);
30          waitUntil((!Brain.Screen.pressing()));
31        }
32      } else {
33        Drivetrain.drive(forward);
34      }
35    }
36  }
37
```

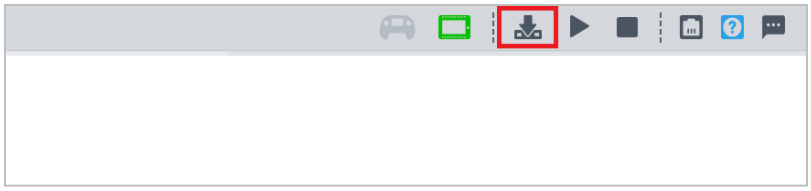
```
19
20  int main() {
21    // Initializing Robot Configuration. DO NOT REMOVE!
22    vexcodeInit();
23    while (true) {                                   Keeps checking if screen is pressed
24      if (Brain.Screen.pressing()) {
25        if (Brain.Screen.xPosition() < 240) {        When screen is pressed, checks whether on the left or right.
26          Drivetrain.turn(left);                     If press was on left (less than 240), turns left.
27          waitUntil((!Brain.Screen.pressing()));     Waits until screen is no longer pressed before continuing.
28        } else {
29          Drivetrain.turn(right);                    If press was not on left (less than 240), turns right.
30          waitUntil((!Brain.Screen.pressing()));     Waits until screen is no longer pressed before continuing.
31        }
32      } else {                                       If screen is not pressed, drives forward.
33        Drivetrain.drive(forward);
34      }
35    }
36  }
```

- Let's review what this project does.

  It keeps checking if the screen is pressed. If the screen isn't pressed it drives forward but if it is, it checks where the screen is pressed.

  If the press was on the left side (less than 240), it turns left. Otherwise, it turns right. We don't need another condition for when the x-value is greater than 240 because if it isn't less than 240 (turn left), it must be greater (turn right). We only have two buttons to worry about.

  The *wait until* Control instructions after each turn have the project wait until the screen is no longer being pressed before continuing.



- Now that the project is done, download and run it to test how it works.
- Take notes in your engineering notebook about how the buttons control the movements of the Clawbot.

3. Adjust the project for a better User Experience.

When pressing the screen's buttons from behind the Clawbot as it drove forward, you pressed on the right side of the screen to turn left and on the left side of the screen to turn right. That is not a good User Experience. A User Experience is how well a user can interact with a User Interface to control a computer system. There is more information about User Interfaces in the Apply section of this lab.

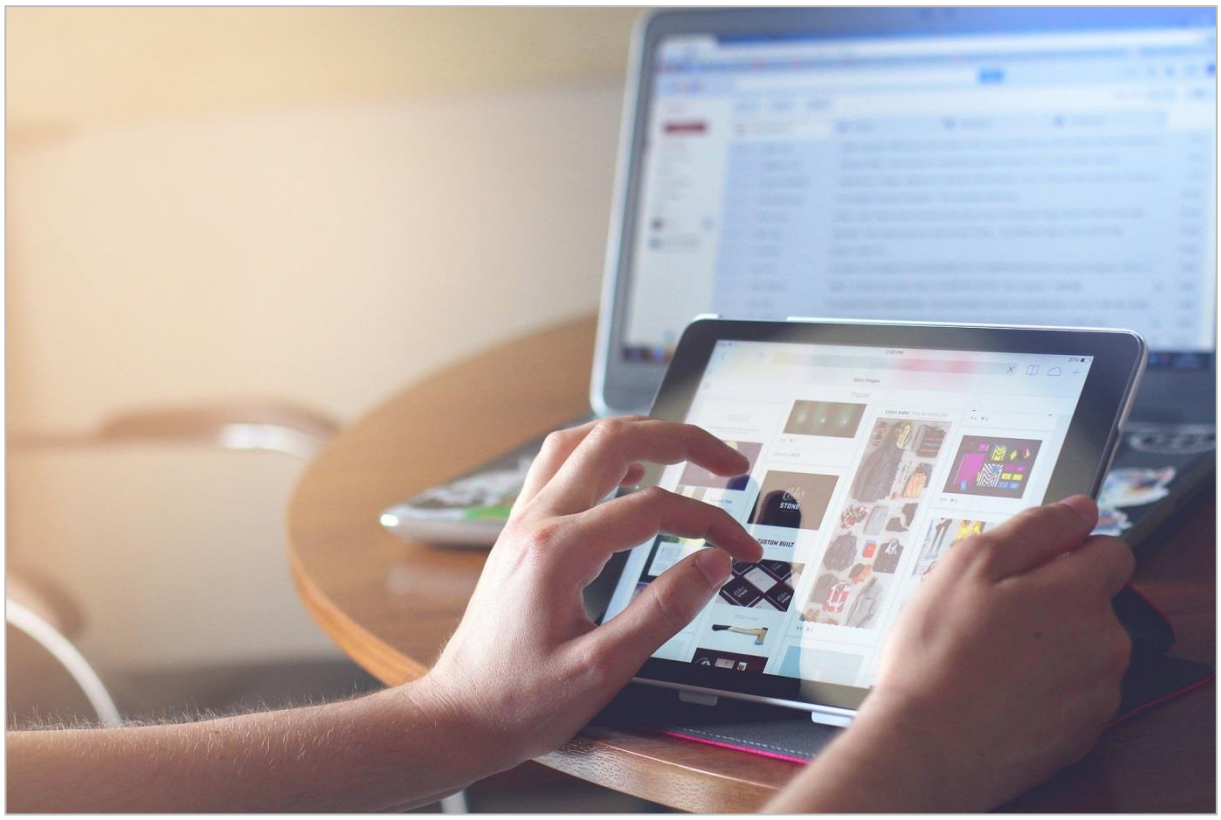
In this case, we need to improve the User Interface in order to improve the User Experience.

```
19
20  int main() {
21    // Initializing Robot Configuration. DO NOT REMOVE!
22    vexcodeInit();
23    while (true) {
24      if (Brain.Screen.pressing()) {
25        if (Brain.Screen.xPosition() < 240) {
26          Drivetrain.turn(left);
27          waitUntil((!Brain.Screen.pressing()));
28        } else {
29          Drivetrain.turn(right);
30          waitUntil((!Brain.Screen.pressing()));
31        }
32      } else {
33        Drivetrain.drive(forward);
34      }
35    }
36  }
37
```

- Review the LeftOrRight project and revise it so that when the user presses the buttons from behind the Clawbot, the robot turns right when the user presses the left side of the screen. Or else, the Clawbot turns left.
- Plan, test, and iterate on this project in your engineering notebook so that the project makes the Clawbot turn toward the side of the screen that the user is pressing from behind the Clawbot.

Become a 21st century problem solver
by applying the core skills and concepts
you learned to other problems.

User Interfaces



Interacting with Computer Systems

The buttons you created on the brain's screen are the beginning of a basic Graphical User Interface (GUI). There are other types of User Interfaces (UIs), but we will focus on GUIs because they are the type we use most.

A UI is a space that allows the user to interact with a computer system (or machine). When you programmed the buttons on the brain's screen, you gave users a way to interact with the Clawbot so they could make it stop or turn left or right. When you interact with a touchscreen on one of your devices (tablet, smartphone, smartwatch), those screens are often the only interface you have. Maybe your device has volume or power buttons as well but you mainly interact with the screen.

After programming your own buttons on the brain's screen, you should have a better sense of how a touchscreen might be programmed to detect which icon or button you want to select. Of course, there are more sophisticated ways of programming those features that professionals use instead of hard programming exactly where a button should be.

Professional programs for GUIs are more adaptive to moving buttons and icons and other variables, but they share some of the same underlying principles.

Those principles form the foundation of the User Experience (UX) while using a UI. The User Experience is how well the interface lets me, as the user, do what I'm trying to do. Is the interface working as I expect it to? Is it responsive to what I'm trying to communicate with my presses? Is it organized well, or can buttons/icons/menus be moved around to make it easier? What does the interface look like in general? Is it pleasing to look at and does it make me want to use it more often? When a UI is still being developed and undergoing iterations, the developers collect data on what works as planned and what needs to be fixed or enhanced. That data then informs the next round of iterative design. Some of the UX changes recommended occur before the release of the device. But, the device might also be sold as is and those changes are made later before the next version is offered to the public consumer.

The Controller as a User Interface

Students react to a successful Driver Controlled match.

Remotely Controlling the Robot

We most often use remote controls to interact with our televisions. We press buttons that make the television display a channel or information/access screen that we want. Technically, your television's remote control is a UI. However, it is a much less sophisticated UI than the one that your smartphone uses. Because it's less sophisticated, it is usually electrical engineers, not UI engineers, that design television remotes. Because of their training, electrical engineers look at the problem of adding new features to a remote control as a circuit problem: how to add a new button to control some new feature on your television. They don't consider the usability of the new button in relation to the other buttons.

Programming your V5 Controller is much more sophisticated. During the Driver Controlled matches of a competition, you want your driver/team to have as many advantages as possible. So you can program the buttons and joysticks to do more than one simple behavior. And, you can program them to do complex behaviors when buttons/joysticks are

used in combination-similar to how some gaming controllers work. As the programmer of your Controller, you consider-like a UI Engineer would-which buttons to use in combination by figuring out how your fingers and hands would need to be placed in order to reach all of the buttons involved. You wouldn't want the driver controlling your robot to end up with cramps in her hands.

The programming for your V5 Controller has it check repeatedly on which button(s) is being pressed so that they can have the robot perform the appropriate behavior(s). Consider that there could be very many nested conditional statements within the Controller's project when using combinations of presses, like the following example: If the A button is pressed and the B button is pressed, do this behavior. If the A button is pressed, the B button is pressed, and the left joystick is pushed downward, do this behavior. Else (only A is pressed), do this behavior. Consider how many more combinations of conditionals are needed to take into account all of the other buttons and their combinations that are available on the Controller.

Of course, as you program more complex behaviors into the functioning of the Controller, the project gets closer to having the robot be autonomous. So a competition team needs to figure out which are the best behaviors to program into their Controllers as complex sequences and which behaviors are best left decomposed into multiple parts so that the Controller lets the driver (user) have more control over the speed and accuracy of the behavior.

Is there a more efficient way to come to the same conclusion? Take what you've learned and try to improve it.

Prepare for the User Interface Challenge

An Interface to Grab and Lift!

In the User Interface Challenge, you need to program your robot so that a user can use buttons on the brain's screen to pick up a variety of objects.

Your Brain's screen will need to have four buttons:

- A button for opening the claw
- A button for closing the claw
- A button for raising the arm
- A button for lowering the arm

To complete the challenge you will need:

- A Clawbot
- Objects to be picked up: an empty can or water bottle, a VEX cube, an unused piece from the VEX kit, or anything else that your teacher can provide

Design, Develop, and Iterate on your Project-Blocks

Answer the following questions in your engineering notebook as you design your project.

- What do you want to program the robot to do? Explain with details.
- How many conditions will your project need to check in the *if then else* blocks?

  Remember that the LeftOrRight project only needed one more condition after a press of the screen was detected.

  This project uses four buttons: left upper, left lower, right upper, and right lower. How many conditions will the project need to check after detecting that the screen has been pressed? Explain with details.

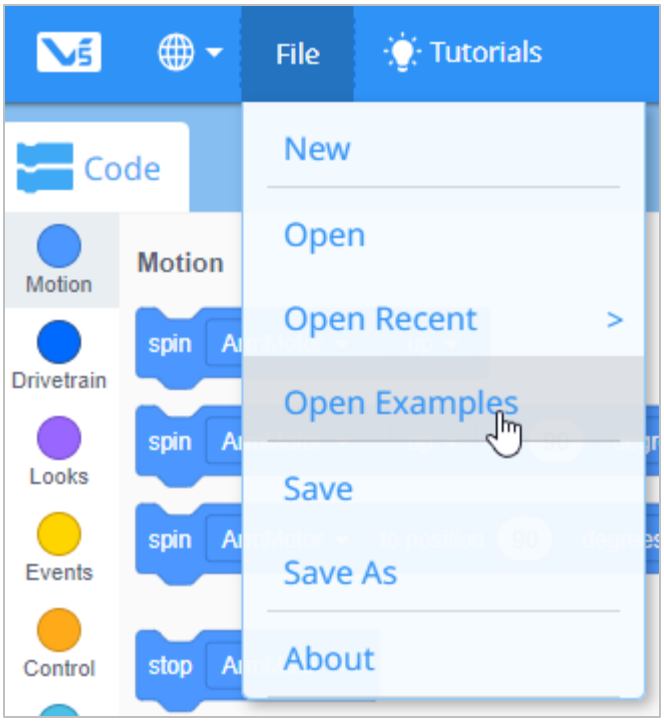
  Hint: The project can check if the screen is pressed by using an *if then* block. Then you will need to nest three *if then else* blocks within the *if then* block, with some nested inside of each other.

Follow the steps below as you create your project:

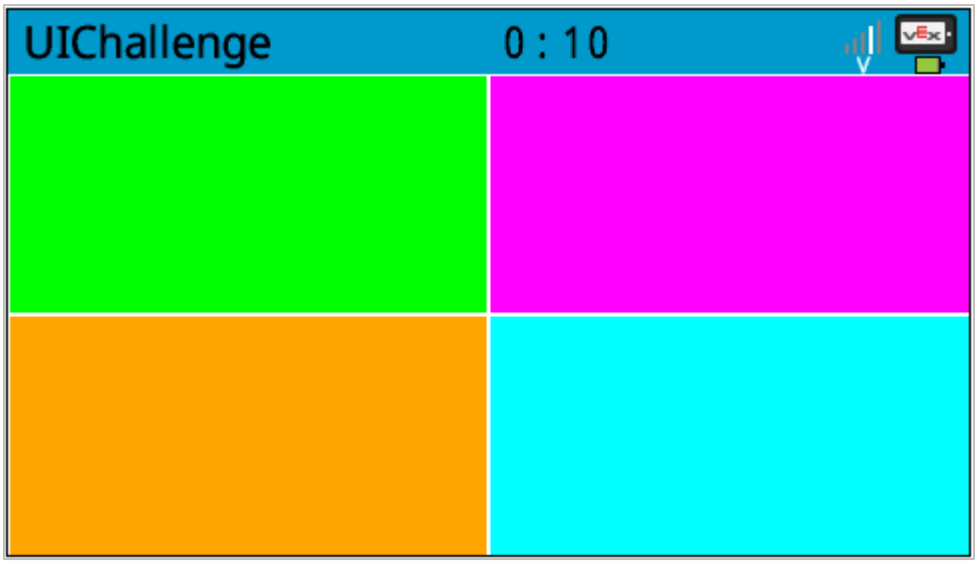
- Plan out the conditions that your project needs to check using drawings and pseudocode. Also, plan for the part of your project that will draw the four buttons on the screen. Decide on their colors.
- Use the pseudocode you created to develop your project.
- Test your project often and iterate on it using what you learned from your testing.
- What could you add to your project to better control the Claw and Arm Motors? Explain with details.
- Share your final project with your teacher.

If you're having trouble getting started, review the following in VEXcode V5 Blocks:

- Previous versions of your "Creating a Stop Button" project
- If then else tutorial
- Previous versions of your project (LeftOrRight)
- The Help feature to learn more about the blocks


File
Tutorials
New
Open
Open Recent
Open Examples
Save
Save As
About
Code
Motion
Drivetrain
Looks
Events
Control
spin
stop

The User Interface Challenge- VEXcode V5 Blocks



The User Interface Challenge

In the User Interface Challenge, you will program the Clawbot so that a user can press the brain's screen to control the arm and claw motors. Then the four buttons on the screen will be used to pick up and replace a variety of ten objects. This challenge does not require the Clawbot to drive or turn. The objects are picked up and then replaced to the same spot on the table or floor.

Rules:

- Each of the four buttons must only do one of the four actions: open the claw, close the claw, lift the arm, or lower the arm.
- Using the Controller is not allowed.
- Each Clawbot will need to lift and replace as many objects as possible within one minute and without dropping them. Lifting and replacing one object at a time is recommended.
    - The one-minute round ends at the 1-minute mark or if any object is dropped - even if the round is only a few seconds in. Dropping an object disqualifies the team from the full minute of the round but any points earned prior to the drop are counted.
- If all of the provided objects have been lifted before the one-minute round is over, objects can be re-used until time is called.
- The object needs to be lifted higher than the arm's motor before it is replaced on the table.

- Each object successfully grabbed and lifted up then down and replaced is worth one point.
- Between rounds, roles can be changed but only one user per Clawbot can play each round.
- The group with the most points at the end of all of the rounds, wins!

Round Number	Clawbot ID	Number of Items Successfully Lifted and Replaced

Roles in the Challenge:

- There should only be one user (the ***lifter***) lifting the objects during each round but groups can switch users between rounds.
- There should be a designated ***scorekeeper*** who tallies how many objects each Clawbot is able to lift successfully. Each object successfully replaced is worth one point. After an object is dropped, the Clawbot's turn is over. Multiple rounds can be played. A scoring table is included here but rows can be added for additional groups and rounds. Each group can use their own scoring table or everyone can record on the same one. If multiple groups are using the same score table, write each groups' Clawbot ID or group name in the second column. Also, the scorekeeper should combine the points from each round at the end of the User Interface Challenge for a total score.
- There should be a person assigned to switch objects out from the table: the ***switcher***. Because the Clawbot is not turning or driving, the lifted object will need to be removed and replaced with a new object after each attempt.

- There should be a person assigned to keep track of the time: the ***timekeeper***. Each round is one minute.
- The teacher should provide the objects approved for this challenge prior to starting.

Design, Develop, and Iterate on your Project-Text

Answer the following questions in your engineering notebook as you design your project.

- What do you want to program the robot to do? Explain with details.
- How many conditions will your project need to check in the *if then else* statement?

  Remember that the LeftOrRight project only needed one more condition after a press of the screen was detected.

  This project uses four buttons: left upper, left lower, right upper, and right lower. How many conditions will the project need to check after detecting that the screen has been pressed? Explain with details.

  Hint: The project can check if the screen is pressed by using an *if then* statement. Then you will need to nest three *if then else* statements within the *if then* statement, with some nested inside of each other.

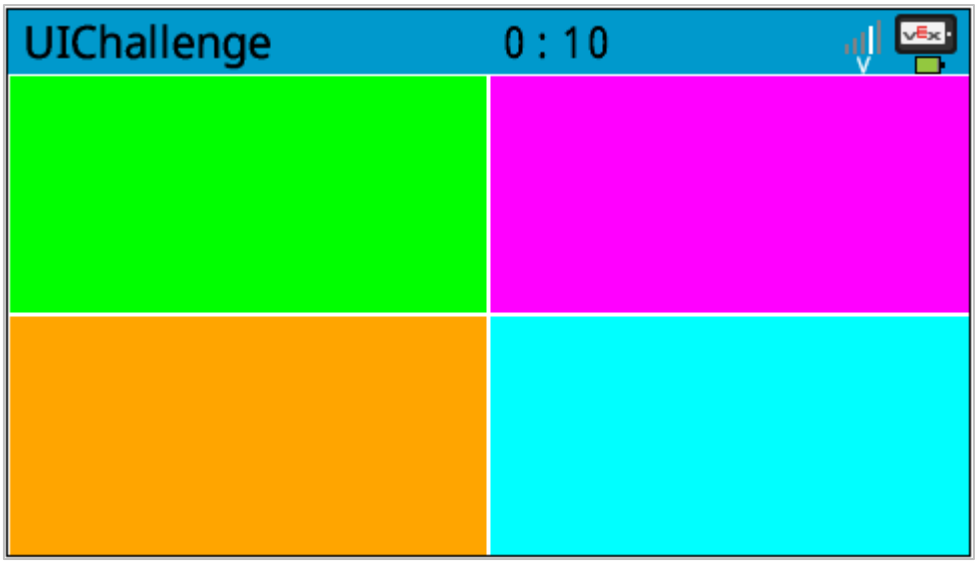
Follow the steps below as you create your project:

- Plan out the conditions that your project needs to check using drawings and pseudocode. Also, plan for the part of your project that will draw the four buttons on the screen. Decide on their colors.
- Use the pseudocode you created to develop your project.
- Test your project often and iterate on it using what you learned from your testing.
- What could you add to your project to better control the Claw and Arm Motors? Explain with details.
- Share your final project with your teacher.

If you're having trouble getting started, review the following in VEXcode V5 Text:

- Previous versions of your "Creating a Stop Button" project
- If then else tutorial
- Previous versions of your project (LeftOrRight)

The User Interface Challenge- VEXcode V5 Text



The User Interface Challenge

In the User Interface Challenge, you will program the Clawbot so that a user can press the brain's screen to control the arm and claw motors. Then the four buttons on the screen will be used to pick up and replace a variety of ten objects. This challenge does not require the Clawbot to drive or turn. The objects are picked up and then replaced to the same spot on the table or floor.

Rules:

- Each of the four buttons must only do one of the four actions: open the claw, close the claw, lift the arm, or lower the arm.
- Using the Controller is not allowed.
- Each Clawbot will need to lift and replace as many objects as possible within one minute and without dropping them. Lifting and replacing one object at a time is recommended.
    - The one-minute round ends at the 1-minute mark or if any object is dropped - even if the round is only a few seconds in. Dropping an object disqualifies the team from the full minute of the round but any points earned prior to the drop are counted.
- If all of the provided objects have been lifted before the one-minute round is over, objects can be re-used until time is called.
- The object needs to be lifted higher than the arm's motor before it is replaced on the table.

- Each object successfully grabbed and lifted up then down and replaced is worth one point.
- Between rounds, roles can be changed but only one user per Clawbot can play each round.
- The group with the most points at the end of all of the rounds, wins!

Round Number	Clawbot ID	Number of Items Successfully Lifted and Replaced

Roles in the Challenge:

- There should only be one user (the ***lifter***) lifting the objects during each round but groups can switch users between rounds.
- There should be a designated ***scorekeeper*** who tallies how many objects each Clawbot is able to lift successfully. Each object successfully replaced is worth one point. After an object is dropped, the Clawbot's turn is over. Multiple rounds can be played. A scoring table is included here but rows can be added for additional groups and rounds. Each group can use their own scoring table or everyone can record on the same one. If multiple groups are using the same score table, write each groups' Clawbot ID or group name in the second column. Also, the scorekeeper should combine the points from each round at the end of the User Interface Challenge for a total score.
- There should be a person assigned to switch objects out from the table: the ***switcher***. Because the Clawbot is not turning or driving, the lifted object will need to be removed and replaced with a new object after each attempt.

- There should be a person assigned to keep track of the time: the ***timekeeper***. Each round is one minute.
- The teacher should provide the objects approved for this challenge prior to starting.

Understand the core concepts and how to apply them to different situations. This review process will fuel motivation to learn.

Review-**VEXcode V5 Blocks**

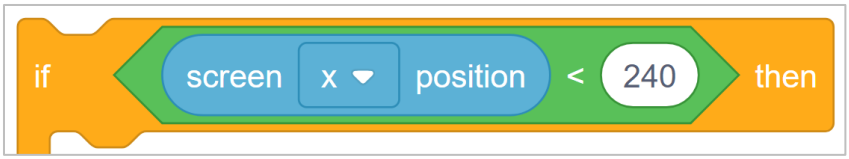
1. **True or False: Blocks inside of the *forever* loop block are run in order, from top to bottom.**
   - True
   - False

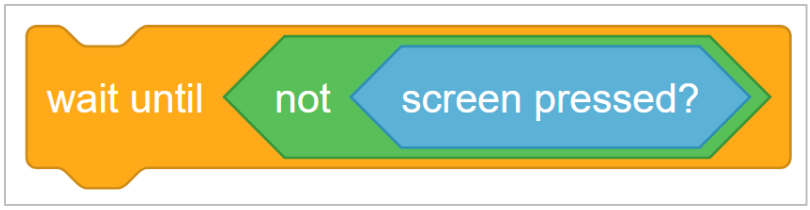
2. **True or False: When the project reads an *if then else* block, if the first condition is met it does not proceed to the *else* condition.**
   - True
   - False

3. **Conditional statements are powerful programming statements that you can use to make decisions based on the outcome of a_____ condition.**
   - boolean
   - holding
   - motor
   - temperate

4. **For the following pseudocode, which block would best be used to program the last line?**

   Keeps checking if screen is pressed
   When screen is pressed, check whether it is pressed on the right or left
   If it was pressed on the left (less than 240), turn left
   Waits until screen is no longer pressed before continuing
   If press was not on the left, turn right





5. **True or False: A conditional statement allows a project to run different blocks depending on whether some condition(s) is met.**
   - True
   - False

6. **Which of the following is the term used for how well the interface lets me, as the user, do what I'm trying to do?**
   - Graphical User Interface (GUI)
   - User Interface (UI)
   - User Experience (UX)
   - None of these answers are correct.

7. **True or False: Developers of User Interfaces (UIs) already know how to optimize the User Experience and therefore do not need to collect data from users.**
   - True
   - False

Review-VEXcode V5 Text

8. **True or False: Instructions inside of the *forever* loop structure are run in order, from top to bottom.**
   - True
   - False

9. **True or False: When the project reads an *if then else instruction*, if the first condition is met it does not proceed to the *else* condition.**
   - True
   - False

10. **Conditional statements are powerful programming statements that you can use to make decisions based on the outcome of a_____ condition.**
    - boolean
    - holding
    - motor
    - temperate

11. **For the following pseudocode, which insruction would best be used to program the last line?**

    Keeps checking if screen is pressed
    When screen is pressed, check whether it is pressed on the right or left
    If it was pressed on the left (less than 240), turn left
    Waits until screen is no longer pressed before continuing
    If press was not on the left, turn right

```
else{
Drivetrain.drive(forward);
```

```
if (Brain.Screen.xPosition() < 240) {
```

```
waitUntil(Brain.Screen.pressing());
```

```
else{
 Drivetrain.turn(right);
```

12. **True or False: A conditional statement allows a project to run different instructions depending on whether some condition(s) is met.**
    - True
    - False

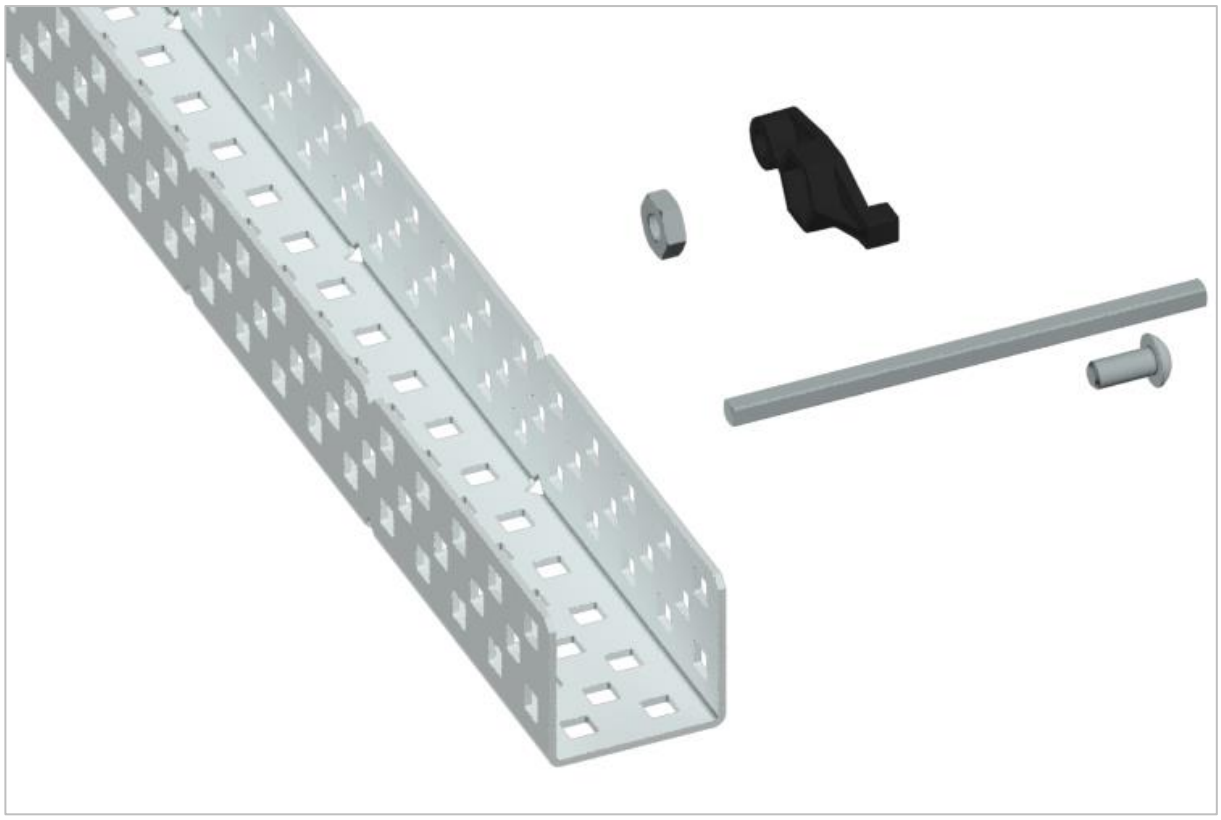
13. **Which of the following is the term used for how well the interface lets me, as the user, do what I'm trying to do?**
    - Graphical User Interface (GUI)
    - User Interface (UI)
    - User Experience (UX)
    - None of these answers are correct.

14. **True or False: Developers of User Interfaces (UIs) already know how to optimize the User Experience and therefore do not need to collect data from users.**
    - True
    - False

APPENDIX

Additional information, resources, and materials.

Using the 1 Post Hex Nut Retainer w/ Bearing Flat



1 Post Hex Nut Retainer w/ Bearing Flat

Using the 1 Post Hex Nut Retainer w/ Bearing Flat

The 1 Post Hex Nut Retainer w/ Bearing Flat allows shafts to spin smoothly through holes in structural components. When mounted, it provides two points of contact on structural components for stability. One end of the retainer contains a post sized to securely fit in the square hole of a structural component. The center hole of the retainer is sized and slotted to securely fit a hex nut, allowing a 8-32 screw to easily be tightened without the need for a wrench or pliers. The hole on the end of the Retainer is intended for shafts or screws to pass through.

To make use of the retainer:

- Align it on a VEX structural component such that the end hole is in the desired location, and the center and end sections are also backed by the structural component.

- Insert the square post extruding from the retainer into the structural component to help keep it in place.
- Insert a hex nut into the center section of the retainer so that it is flush with the rest of the component.
- Align any additional structural components to the back of the main structural component, if applicable.
- Use an 8-32 screw of appropriate length to secure the structural component(s) to the retainer through the center hole and hex nut.

Using the 4 Post Hex Nut Retainer

4 Post Hex Nut Retainer

Using the 4 Post Hex Nut Retainer

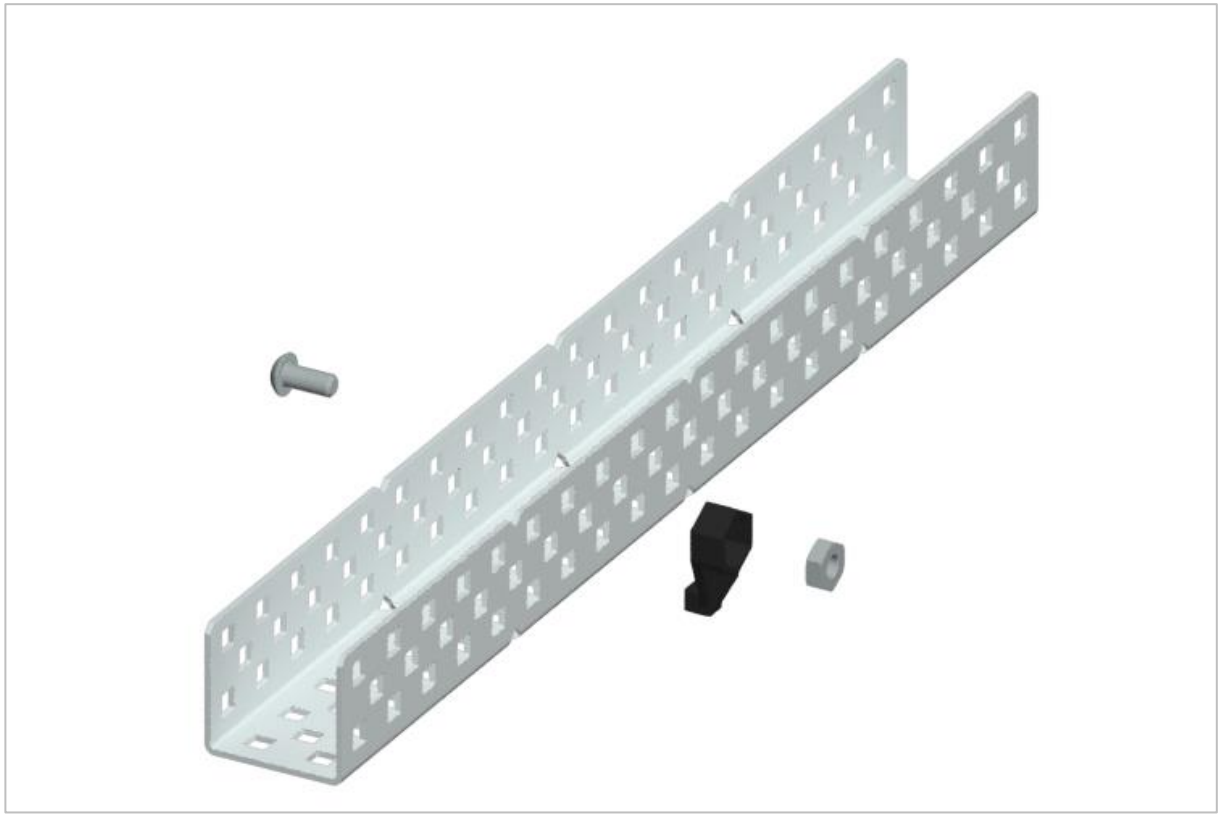
The 4 Post Hex Nut Retainer provides five points of contact for creating a strong connection between two structural components using one screw and nut. Each corner of the retainer contains a post sized to securely fit in a square hole within a structural component. The center of the retainer is sized and slotted to securely fit a hex nut, allowing a 8-32 screw to easily be tightened without the need for a wrench or pliers.

To make use of the retainer:

- Align it on a VEX structural component such that the center hole is in the desired location, and each corner is also backed by the structural component.
- Insert the square posts extruding from the retainer into the structural component to help keep it in place.
- Insert a hex nut into the center section of the retainer so that it is flush with the rest of the component.

- Align any additional structural components to the back of the main structural component, if applicable.
- Use an 8-32 screw of appropriate length to secure the structural component(s) to the retainer through the center hole and hex nut.

Using the 1 Post Hex Nut Retainer



1 Post Hex Nut Retainer

Using the 1 Post Hex Nut Retainer

The 1 Post Hex Nut Retainer provides two points of contact for connecting a structural component to another piece using one screw and nut. One end of the retainer contains a post sized to securely fit in the square hole of a structural component. The other end of the retainer is sized and slotted to securely fit a hex nut, allowing a 8-32 screw to easily be tightened without the need for a wrench or pliers.

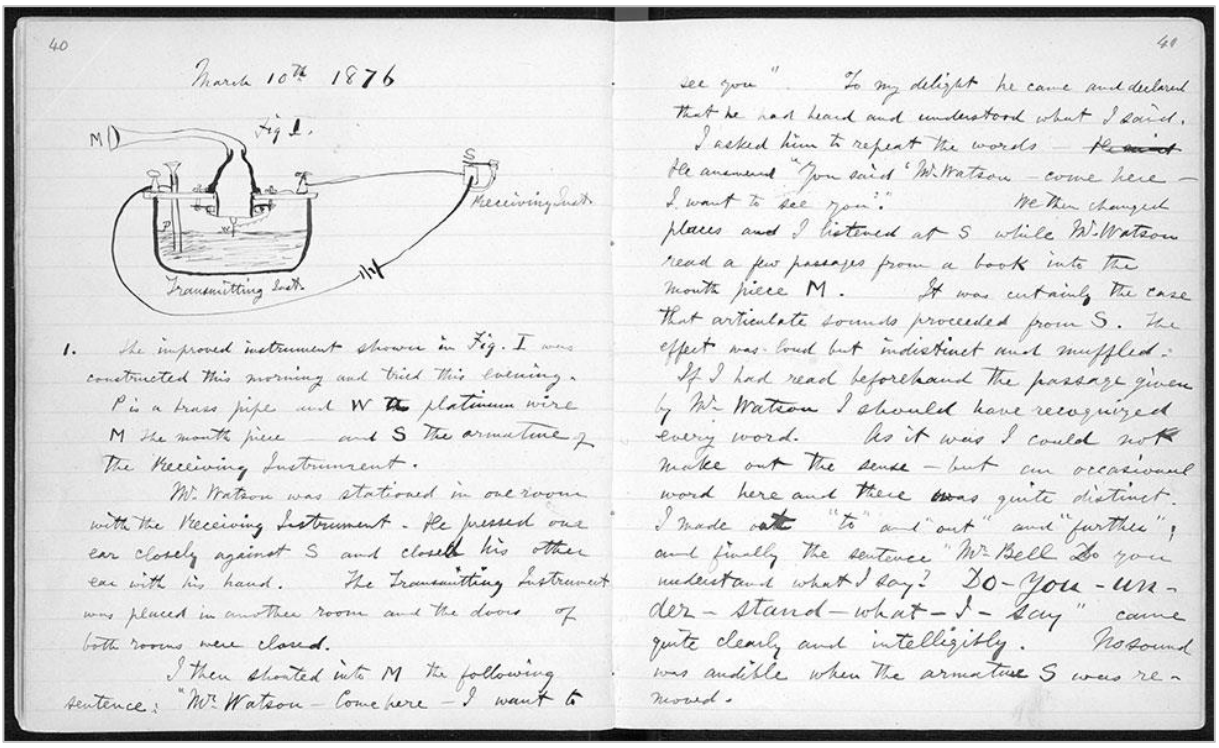
To make use of the retainer:

- Align it on a VEX structural component such that both ends are backed by the structural component and positioned to secure the second piece.
- Insert the square post extruding from the retainer into the structural component to help keep it in place.
- If the retainer is being used to secure two structural components, insert a hex nut into the other end of the retainer so that it is flush with the rest of the component. If used to secure

a different type of component, such as a standoff, it may be appropriate to insert the screw through this side.

- Align any additional components to the back of the main structural component, if applicable.
- If the retainer is being used to connect two structural components, use an 8-32 screw of appropriate length to secure the structural components through the hole and hex nut. If used to connect a different type of component, such as a standoff, secure it directly or with a hex nut.

Engineering Notebooks



40

March 10th 1876

Fig 1.

M

S

Receiving Instr.

Transmitting Instr.

1. The improved instrument shown in Fig. I was constructed this morning and tried this evening. P is a brass pipe and W the platinum wire M the mouth piece — and S the armature of the Receiving Instrument.

Mr. Watson was stationed in one room with the Receiving Instrument - He pressed one ear closely against S and closed his other ear with his hand. The Transmitting Instrument was placed in another room and the doors of both rooms were closed.

I then shouted into M the following sentence: "Mr. Watson — Come here — I want to

41

see you" To my delight he came and declared that he had heard and understood what I said.

I asked him to repeat the words He said He answered "You said 'Mr. Watson — come here — I want to see you.'"

We then changed places and I listened at S while Mr. Watson read a few passages from a book into the mouth piece M. It was certainly the case that articulate sounds proceeded from S. The effect was loud but indistinct and muffled.

If I had read beforehand the passage given by Mr. Watson I should have recognized every word. As it was I could not make out the sense — but an occasional word here and there was quite distinct. I made out "to" and "out" and "further", and finally the sentence "Mr. Bell Do you understand what I say? DO-YOU-UN-DER-STAND-WHAT-I-SAY" came quite clearly and intelligibly. No sound was audible when the armature S was removed.

Alexander Graham Bell's notebook entry from a successful experiment with his first telephone

An Engineering Notebook Documents your Work

Not only do you use an engineering notebook to organize and document your work, it is also a place to reflect on activities and projects. When working in a team, each team member will maintain their own journal to help with collaboration.

Your engineering notebook should have the following:

- An entry for each day or session that you worked on the solution
- Entries that are chronological, with each entry dated
- Clear, neat, and concise writing and organization
- Labels so that a reader understands all of your notes and how they fit into your iterative design process

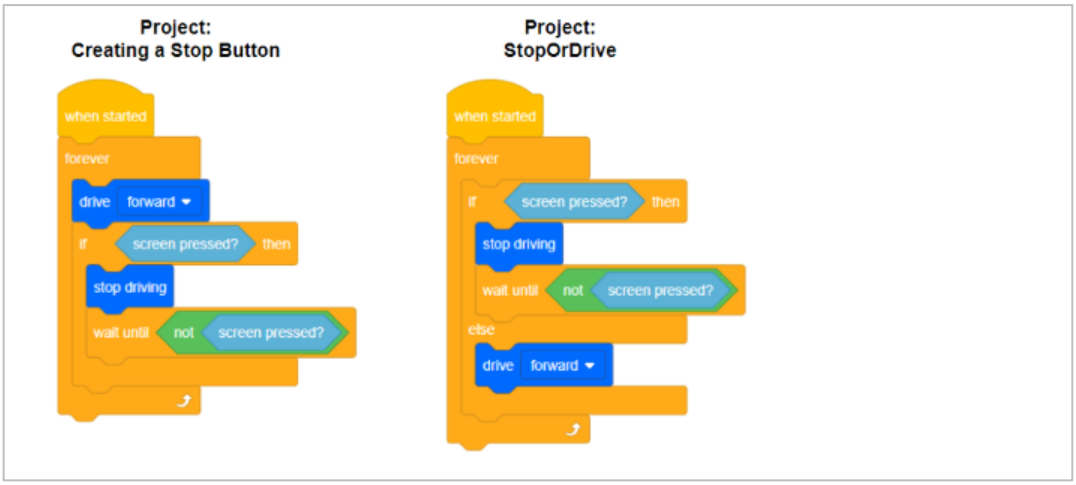
An entry might include:

- Brainstorming ideas
- Sketches or pictures of prototypes

- Pseudocode and flowcharts for planning
- Any worked calculations or algorithms used
- Answers to guiding questions
- Notes about observations and/or conducted tests
- Notes about and reflections on your different iterations

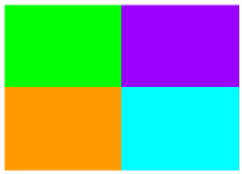
Popups

Creating a Stop Button vs. StopOrDrive-VEXcode V5 Blocks



Pseudocode for the User Interface Challenge

Brain's Screen

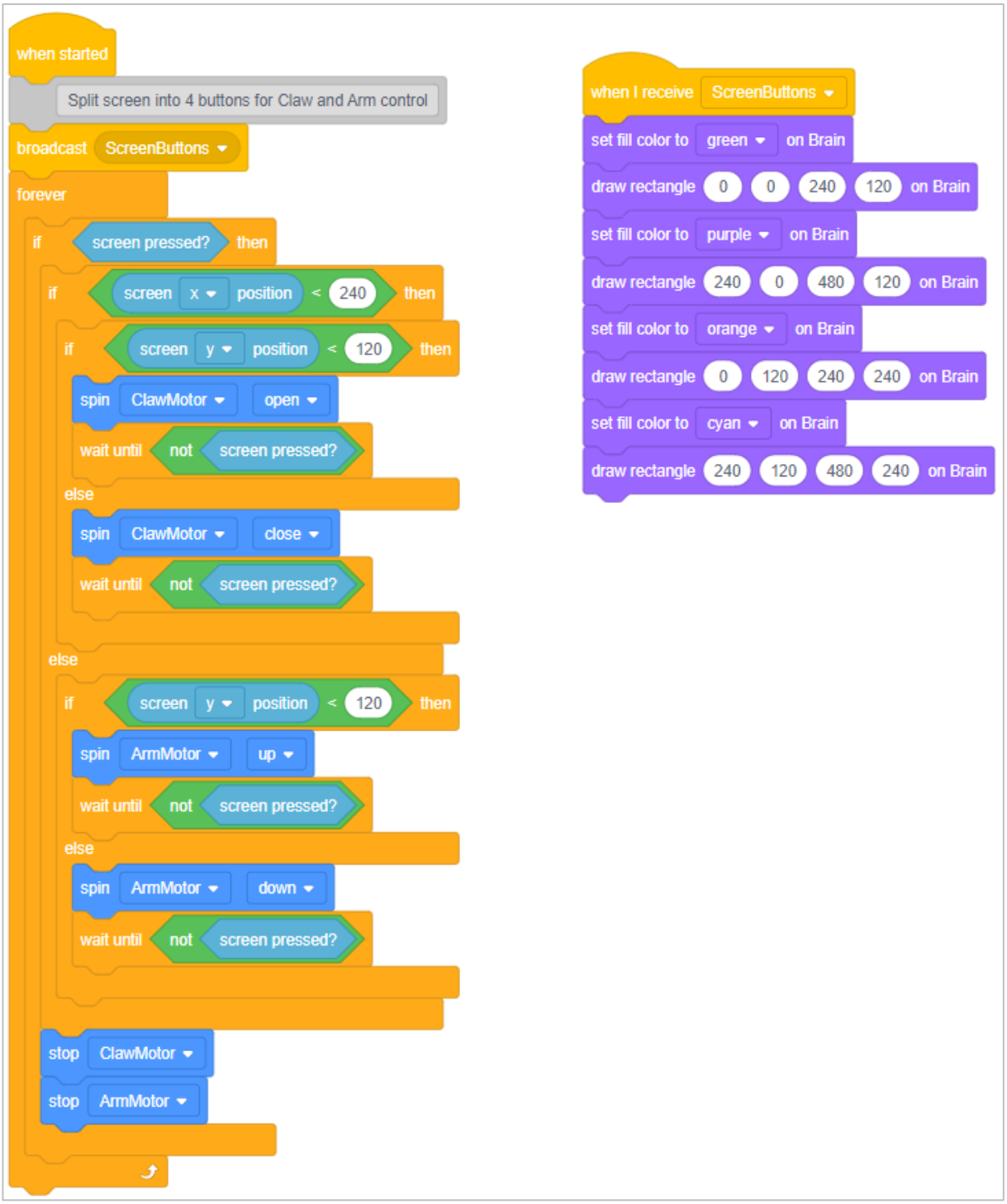


Config:
Clawbot w/
drivetrain

```
Broadcast: ScreenButtons
Forever
    If the screen is pressed
        If x<240, then
            If y<120, then
                Open ClawMotor
                Wait until screen isn't pressed
            Else
                Close ClawMotor
                Wait until screen isn't pressed
        Else  If y<120, then
                Lift ArmMotor
                Wait until screen isn't pressed
            Else
                Lower ArmMotor
                Wait until screen isn't pressed
        Stop Motors
```

```
Receive ScreenButtons
Choose green
Draw button in upper left
Choose purple
Draw button in upper right
Choose orange
Draw button in lower left
Choose cyan
Draw button is lower right
```

Example Solution to User Interface Challenge-VEXcode V5 Blocks



Creating a Stop Button vs. StopOrDrive-VEXcode V5 Text

Project: Creating a Stop Button

```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();
while (true) {
  Drivetrain.drive(forward);
  if (Brain.Screen.pressing()){
    Drivetrain.stop();
    waitUntil(!Brain.Screen.pressing());
    } else {

    }
  }
```

Project StopOrDrive

```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();
  while (true) {
    if (Brain.Screen.pressing()) {
      Drivetrain.stop();
      waitUntil(!Brain.Screen.pressing());
    } else {
      Drivetrain.drive(forward);
    }
  }
}
```

Example Solution to User Interface Challenge-VEXcode V5 Text

```
17    #include "vex.h"
18
19    using namespace vex;
20    event ScreenButtons = event();
21
22    void HasScreenButtons() {
23      Brain.Screen.setFillColor(green);
24      Brain.Screen.drawRectangle(0, 0, 240, 120);
25      Brain.Screen.setFillColor(purple);
26      Brain.Screen.drawRectangle(240, 0, 480, 120);
27      Brain.Screen.setFillColor(orange);
28      Brain.Screen.drawRectangle(0, 120, 240, 240);
29      Brain.Screen.setFillColor(cyan);
30      Brain.Screen.drawRectangle(240, 120, 480, 240);
31      wait(1, sec);
32    }
33
34    int main() {
35      // Initializing Robot Configuration. DO NOT REMOVE!
36      vexcodeInit();
37      ScreenButtons(HasScreenButtons);
38      while (true) {
39        ScreenButtons.broadcast();
40        if (Brain.Screen.pressing()) {
41          if (Brain.Screen.xPosition() < 240.0) {
42            if (Brain.Screen.yPosition() < 120.0) {
43              ClawMotor.spin(forward);
44              waitUntil((!Brain.Screen.pressing()));
45            } else {
46              ClawMotor.spin(reverse);
47              waitUntil((!Brain.Screen.pressing()));
48            }
49          } else {
50            if (Brain.Screen.yPosition() < 120.0) {
51              ArmMotor.spin(forward);
52              waitUntil((!Brain.Screen.pressing()));
53            } else {
54              ArmMotor.spin(reverse);
55              waitUntil((!Brain.Screen.pressing()));
```