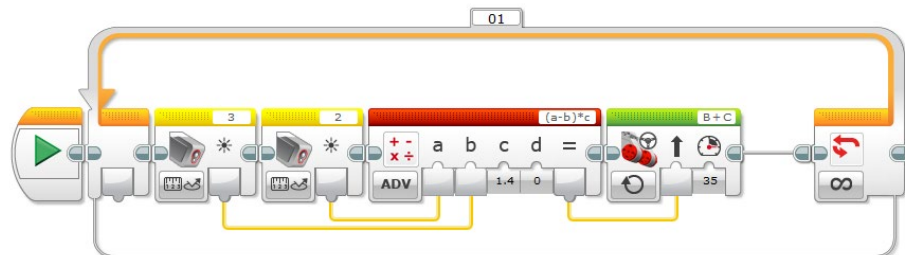


ROBOCUP JUNIOR VICTORIA

Programming the EV3

- Mindstorms (traditional)



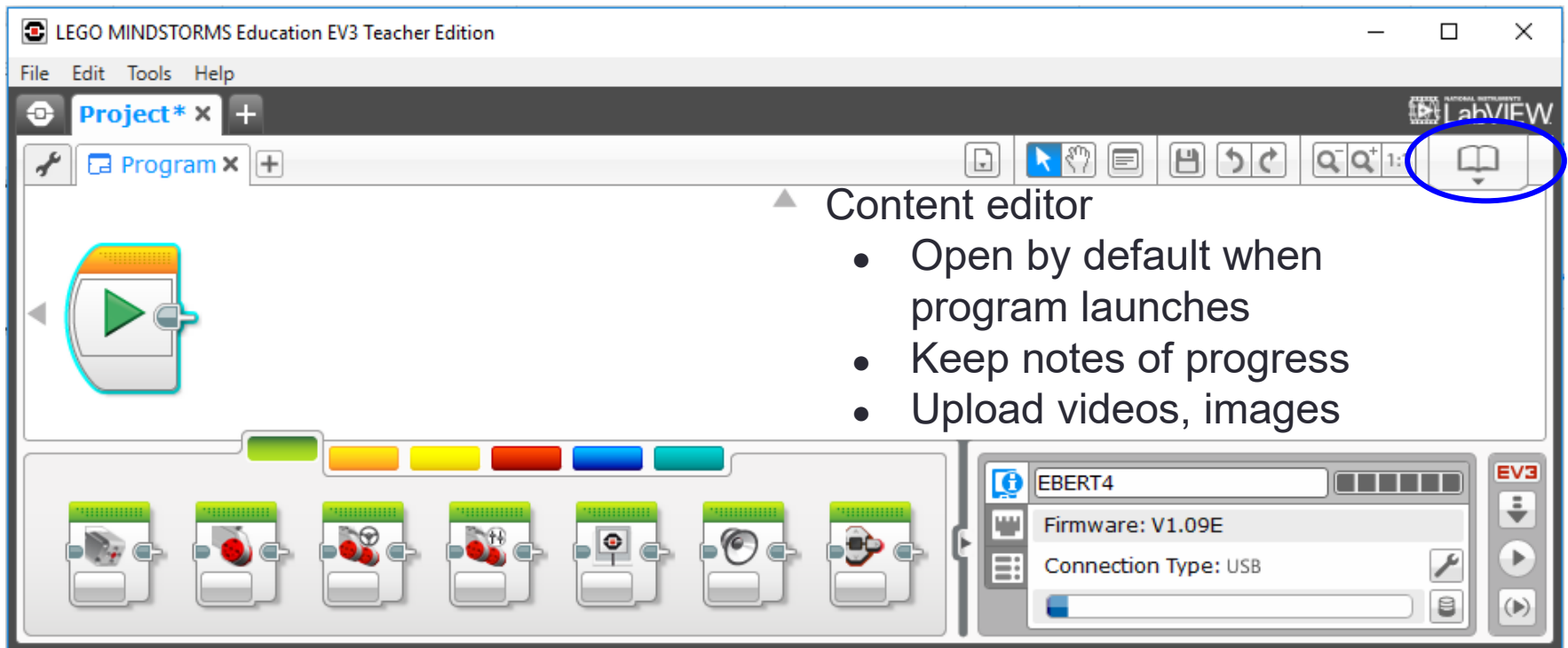
ROBOT PROGRAMMING

What are the programming challenges and how are they best approached?

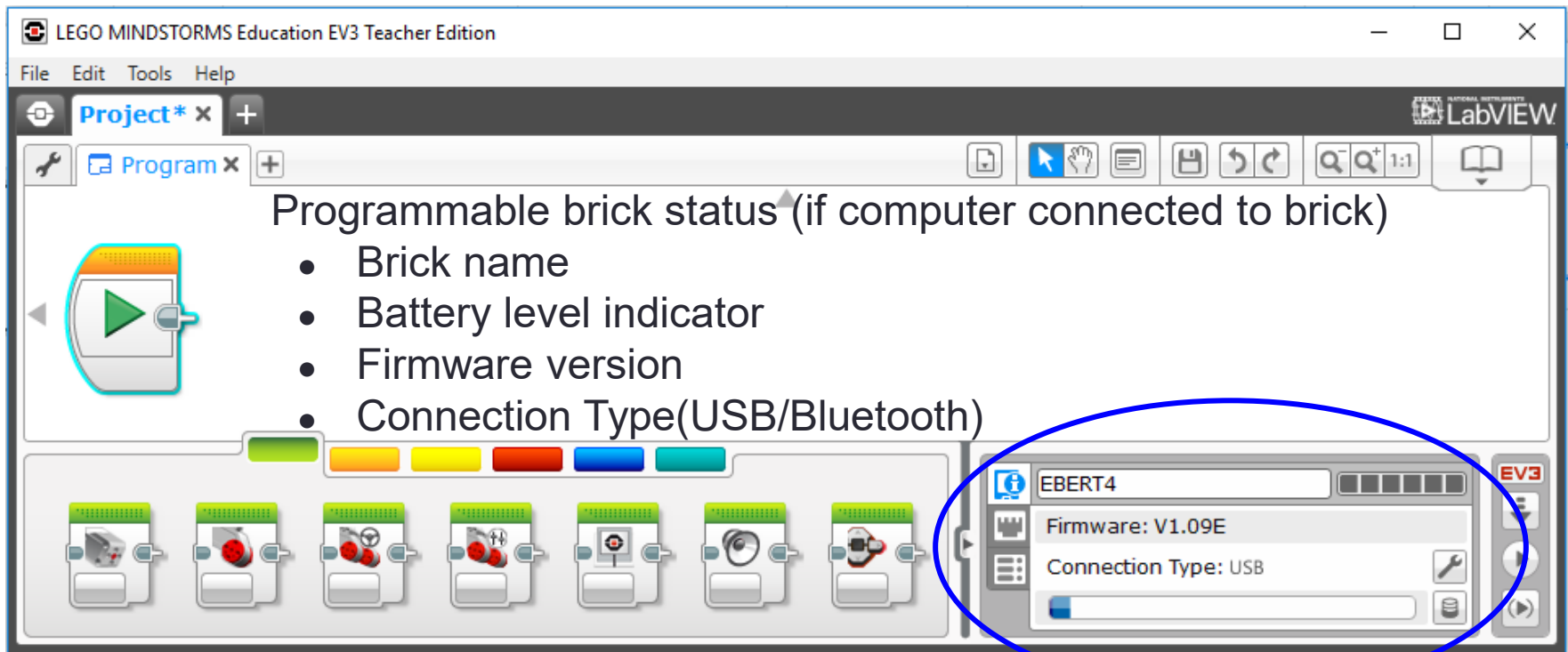
Rescue Robot Programming

- Rule #1 – Construction impacts programming
- Rule #2 – Programming impacts construction
- Most Rescue robots have been programmed using the Lego Mindstorms graphical programming languages for NXT or EV3
- These provide all the functionality you need to produce quality Rescue robot programs.
- Simplicity is the key to successful programming, especially for beginners. If it looks more complicated than necessary, it probably is.

Introduction to the coding interface



Introduction to the coding interface



Introduction to the coding interface

LEGO MINDSTORMS Education EV3 Teacher Edition

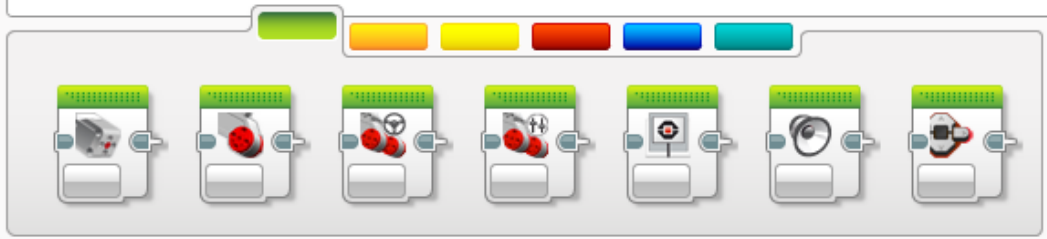

File Edit Tools Help

Project* x +

Program x +

Port view (if connected)

- Position of all connected inputs and outputs
- Readings from each port
(these values can also be read directly from the brick)

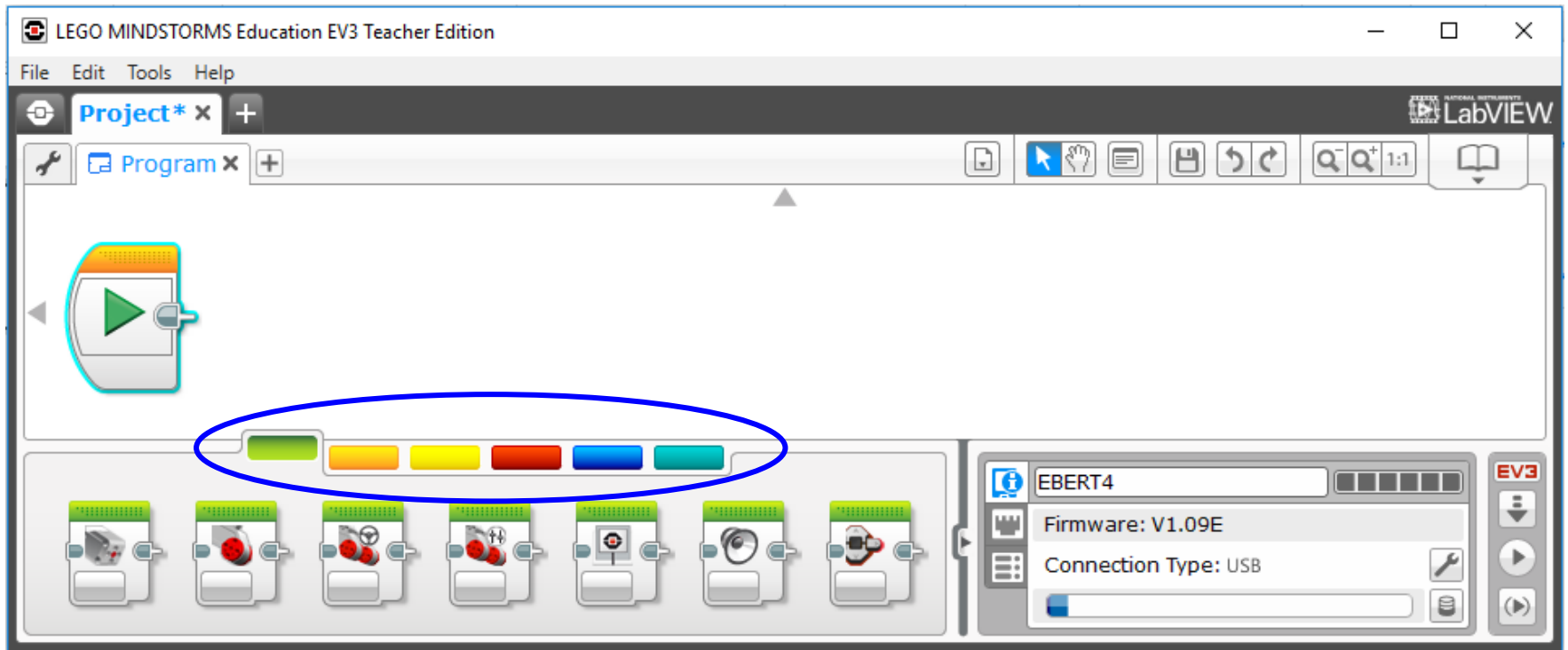


	A	B	C	D
1	0	54	64	

EV3

The image shows the LEGO MINDSTORMS Education EV3 Teacher Edition software interface. The main window displays a 'Port view' section with a list of connected inputs and outputs. Below this, there is a diagram of the EV3 brick with color-coded ports (green, yellow, red, blue, cyan) and a table showing the status of each port. The table has columns A, B, C, and D, and rows 1, 2, 3, and 4. The values in the table are: A: 0, B: 54, C: 64, D: (empty). The EV3 logo is visible in the bottom right corner. A blue circle highlights the 'Port view' icon in the top left corner of the interface.

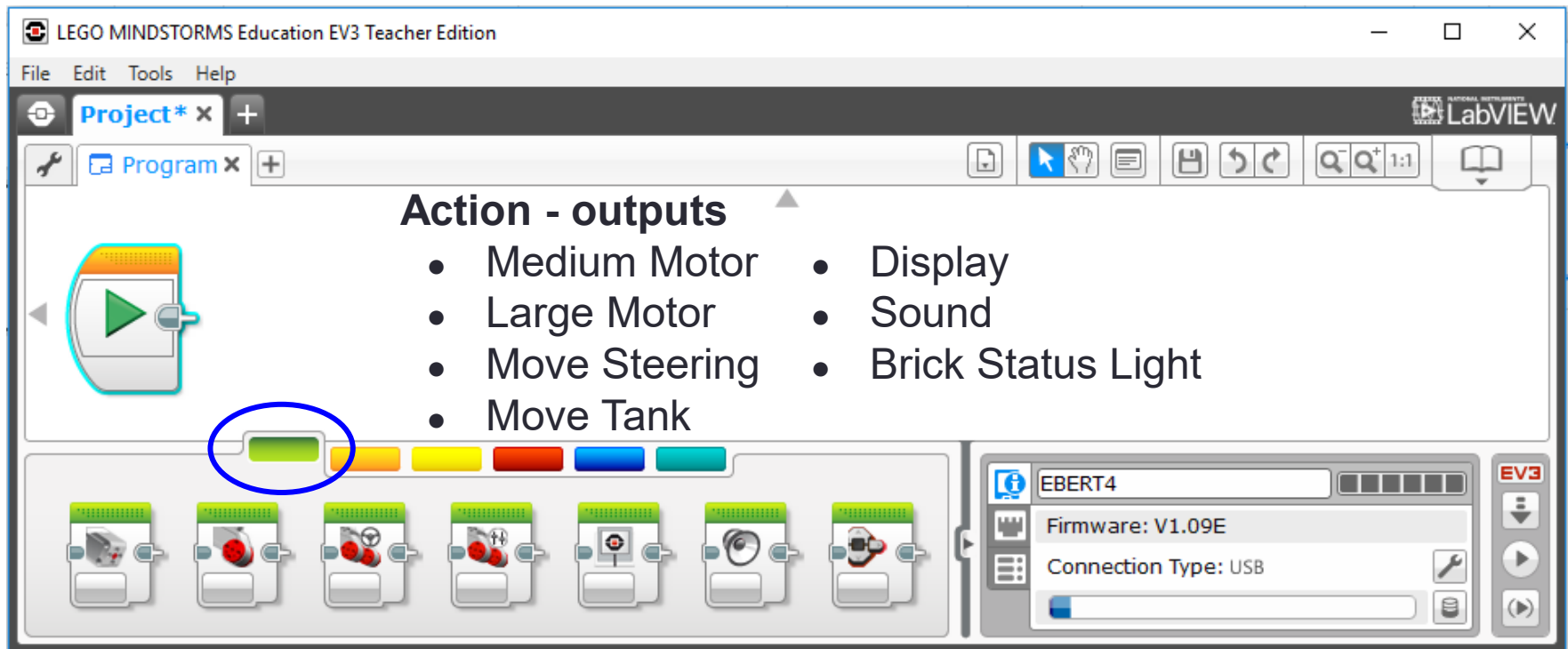
Introduction to the coding interface



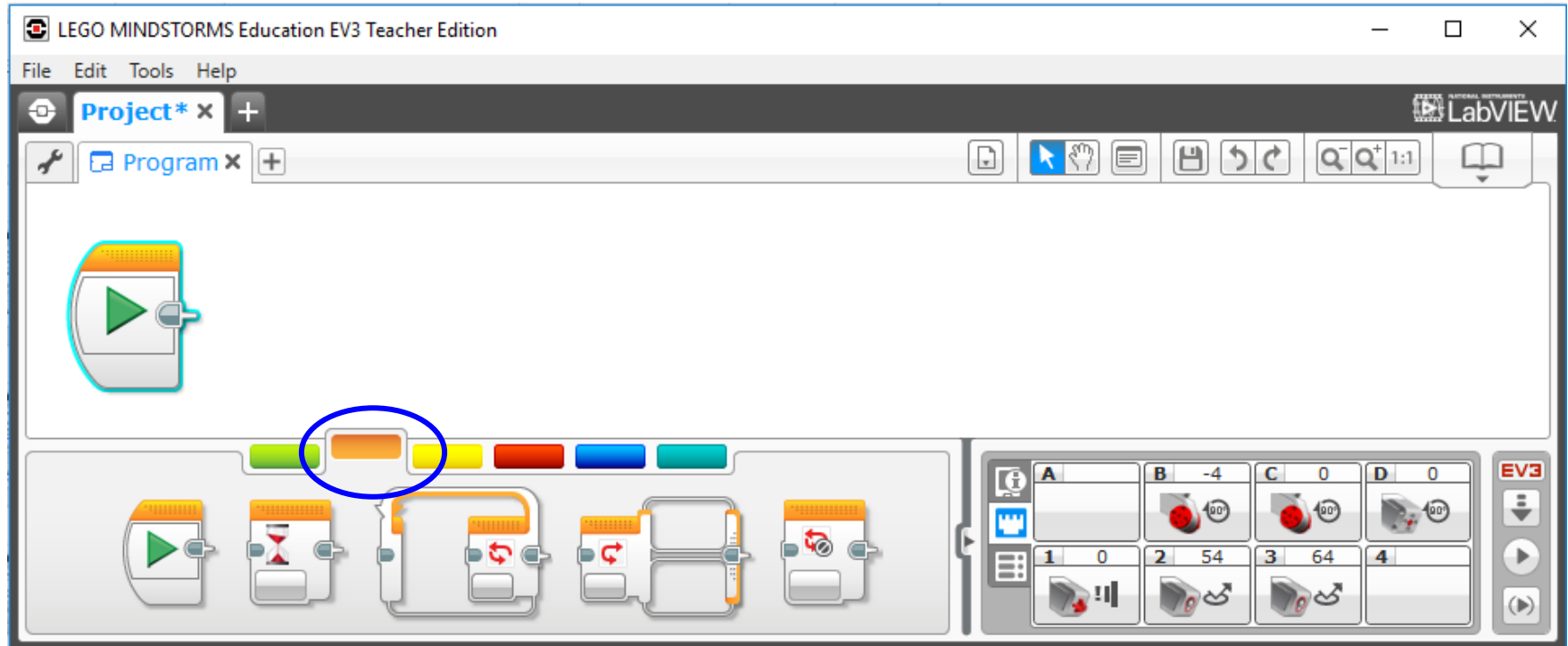
Palettes of programming blocks

- **Green** – Action
- **Orange** – Flow Control
- **Yellow** – Sensor
- **Red** – Data Operations
- **Blue** – Advanced

Introduction to the coding interface



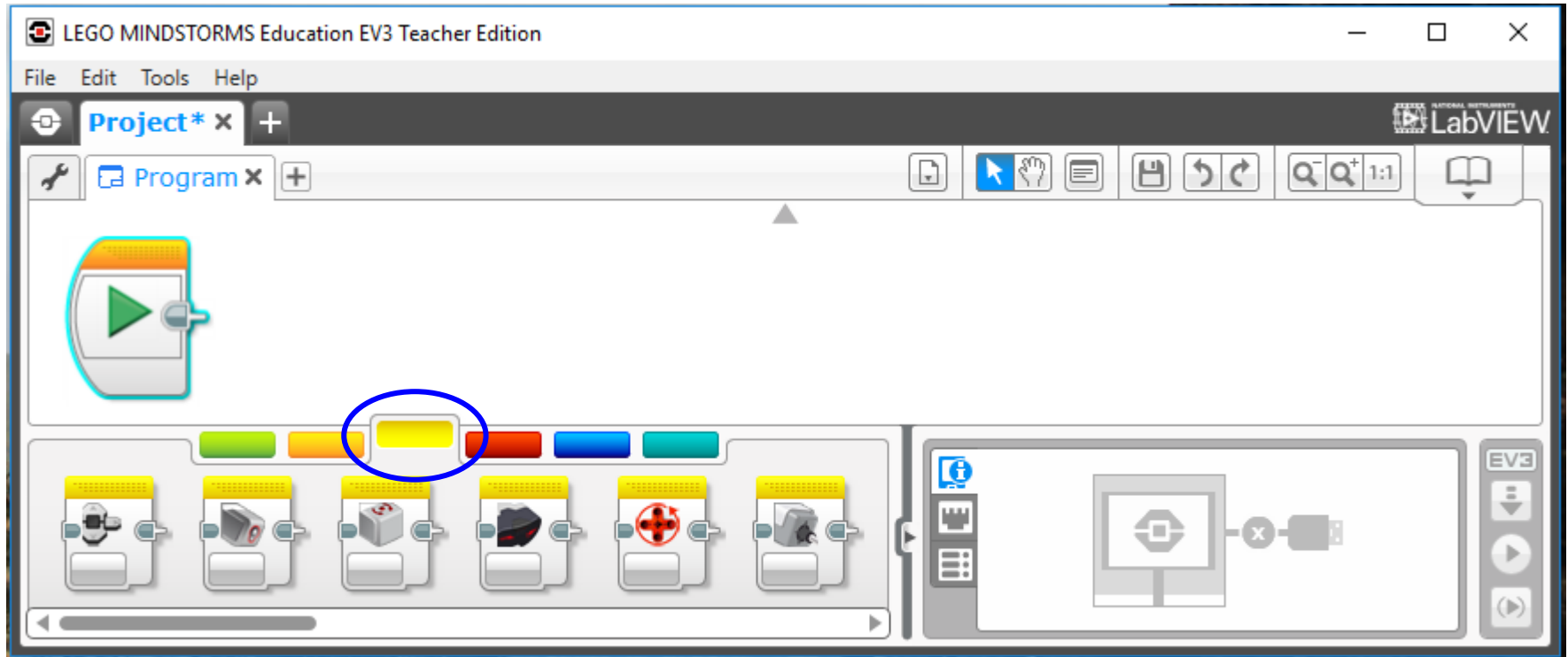
Introduction to the coding interface



Flow Control

- **Start** - required at the start of any code
- **Wait** – pause the program until...
- **Loop** – run code forever
- **Switch** – decide between two conditions
- **Loop Interrupt** – breaking out of the loop

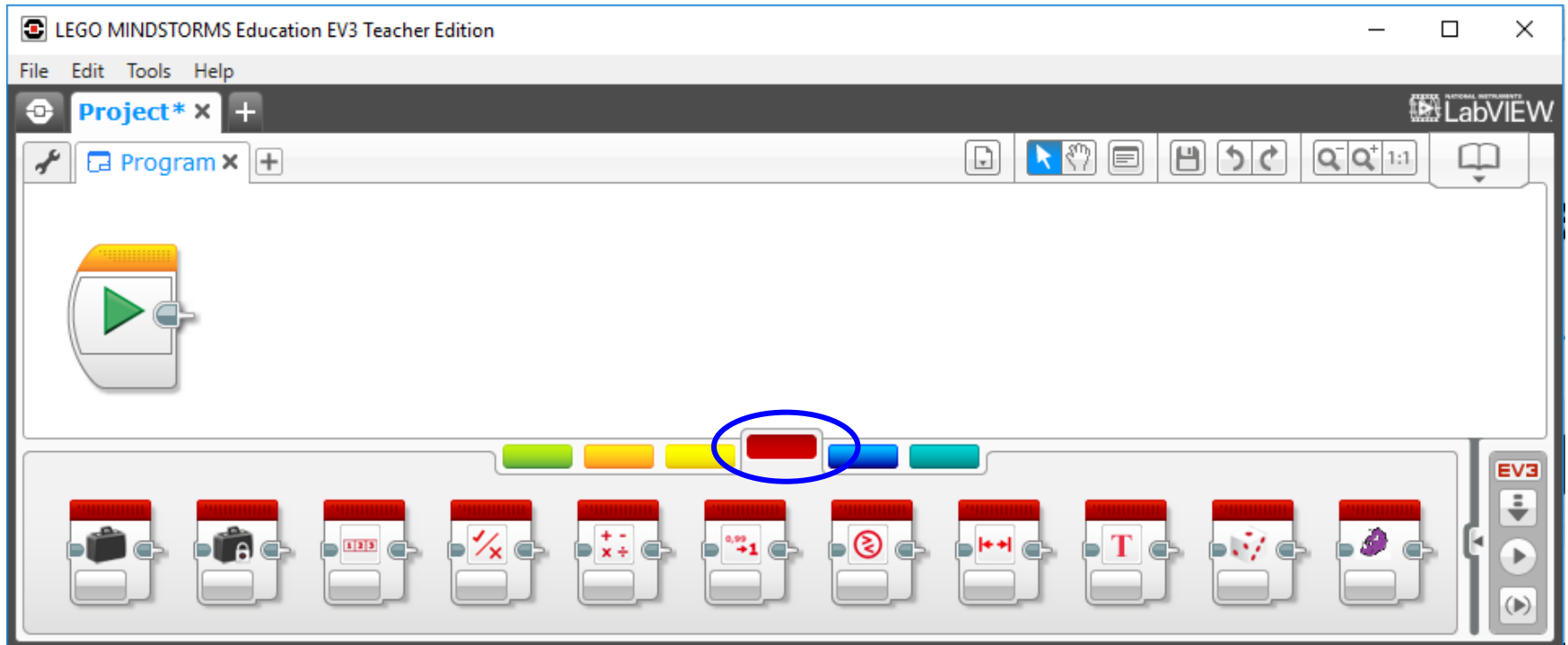
Introduction to the coding interface



Sensors

- Used for measuring values from various sensors
- Useful in conjunction with Data block

Introduction to the coding interface

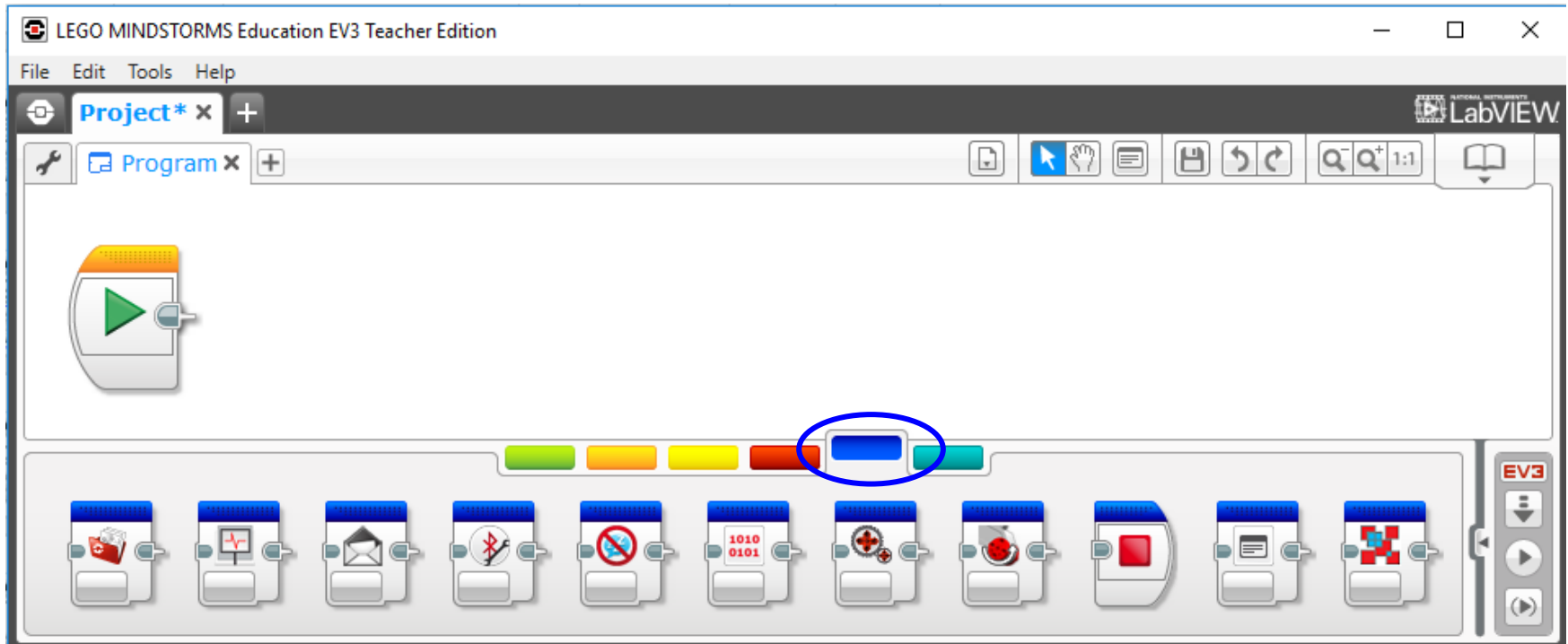


Data Operations

- These blocks allow you to do things with data values
- The “briefcase” block is a container for holding the value of a variable
- The “✓/x” block contains Boolean operators (returns True/False values)
- The “mathematical operators” block allows calculations to be done
- The “dice” block is for generating random numbers



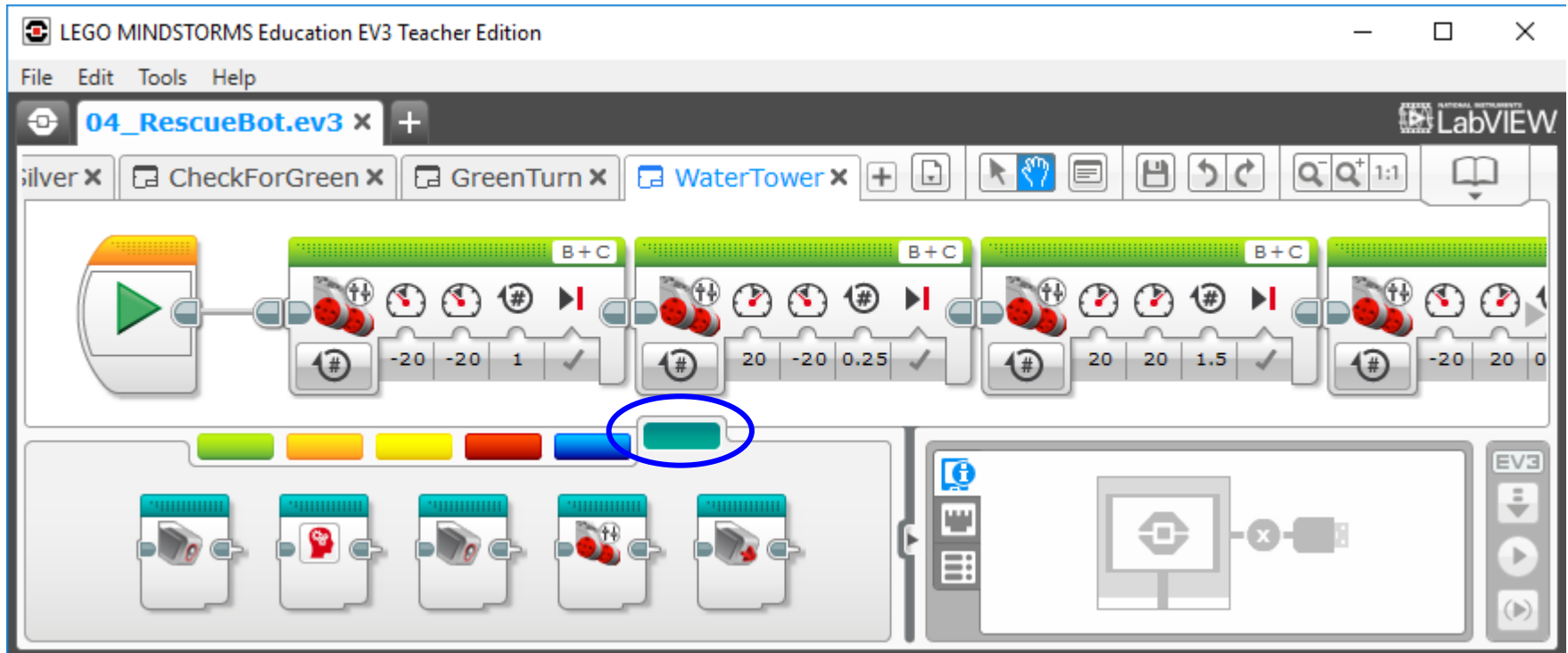
Introduction to the coding interface



Advanced

- These are specialised blocks that go beyond what most students are likely to have time for
- It includes blocks for data logging, communication, etc.

Introduction to the coding interface



My Blocks

- Useful tool for generating your own blocks of code
- Important for breaking up complexity of complex code
- Not essential for EV3 (but it is for more complex codes with NXT)

Example of simple program

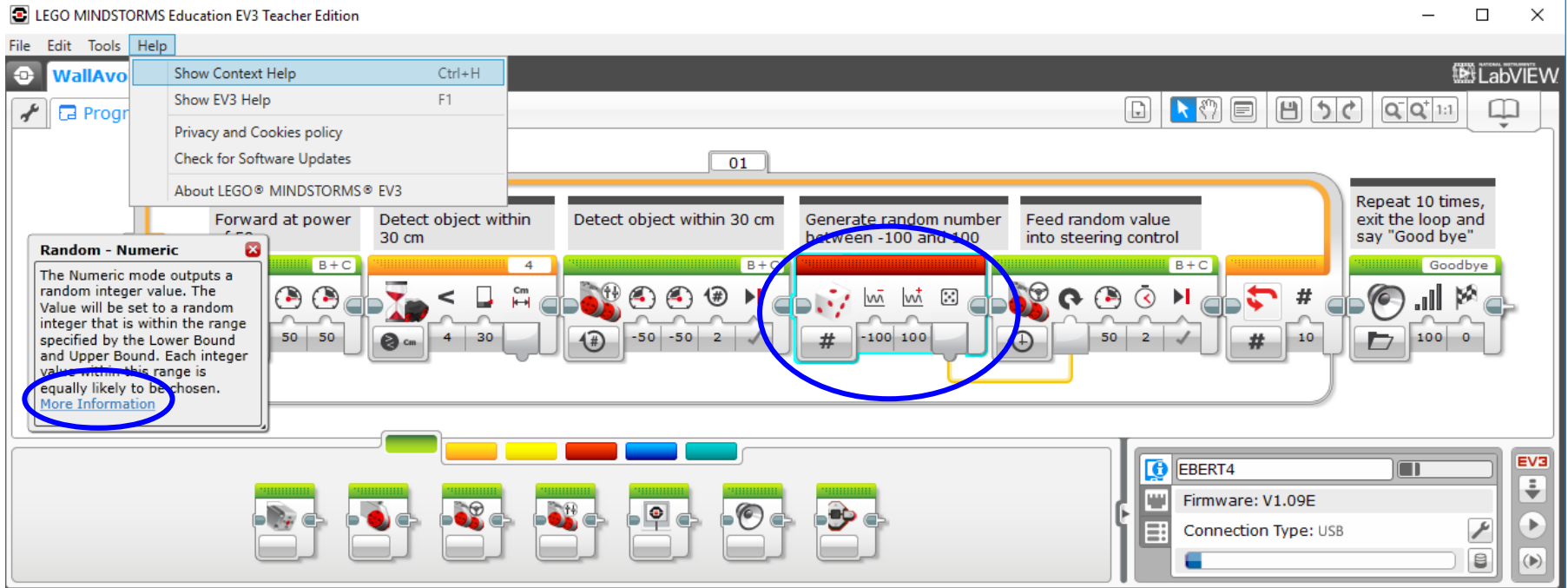
Select objects tool Pan tool Add comments tool

The screenshot shows the LEGO MINDSTORMS Education EV3 Teacher Edition interface. The program 'WallAvoider2.ev3' is open. The main workspace displays a sequence of blocks: a 'Forward at power of 50' block, a 'Detect object within 30 cm' block, a 'Back up for two rotations of the motor' block, a 'Generate random number between -100 and 100' block, a 'Feed random value into steering control' block, and a 'Repeat 10 times, exit the loop and say "Good bye"' block. The 'Repeat' block is highlighted with a yellow box. The bottom of the screen shows a toolbar with various tools, including the 'Select objects tool', 'Pan tool', and 'Add comments tool', which are pointed to by yellow arrows from the text labels above. The bottom right corner shows the connection status for 'EBERT4' with firmware 'V1.09E' and connection type 'USB'.

Things to note:

- comments should be used to explain code
- no unused coding blocks should be left lying around

Getting help



- To get help, select a block then select Help>Show Context Help.
- A brief description will appear; select [More Information](#) for detailed description.

Getting help

David

— □ ×

Programming with LEGO × LEGO MINDSTORMS EV3 ×

← → ↻ ⌂ ⓘ localhost:58401/localizedMapping_B908DB05-F70E-4B0B-8CEA-031DCF197215/en-GB/editor/page.html?Path=block... ☆ ⓘ ⚙

LEGO MINDSTORMS education EV3

SEARCH

INDEX

Home

► General

► Tools

▼ Programming Blocks

> Action Blocks

> Flow Blocks

> Sensor Blocks

▼ Data Block

Constant

Variable

Array

Logic

Mathematics

Round

Compare

Range

Text


Random

Simple PID Control

> Advanced Blocks

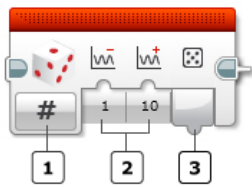
► Data Logging

Random Block



The Random block can output a random Numeric or Logic value. You can use the result of the Random block to make your robot randomly choose from different actions.

> CHOOSE THE OUTPUT TYPE



- 1 Mode Selector
- 2 Inputs
- 3 Output

Use the Mode Selector to choose whether to output a random Numeric value or a random Logic value. After selecting the mode, you can choose the [Inputs](#). The inputs control the range and probability of the [Value](#) output.

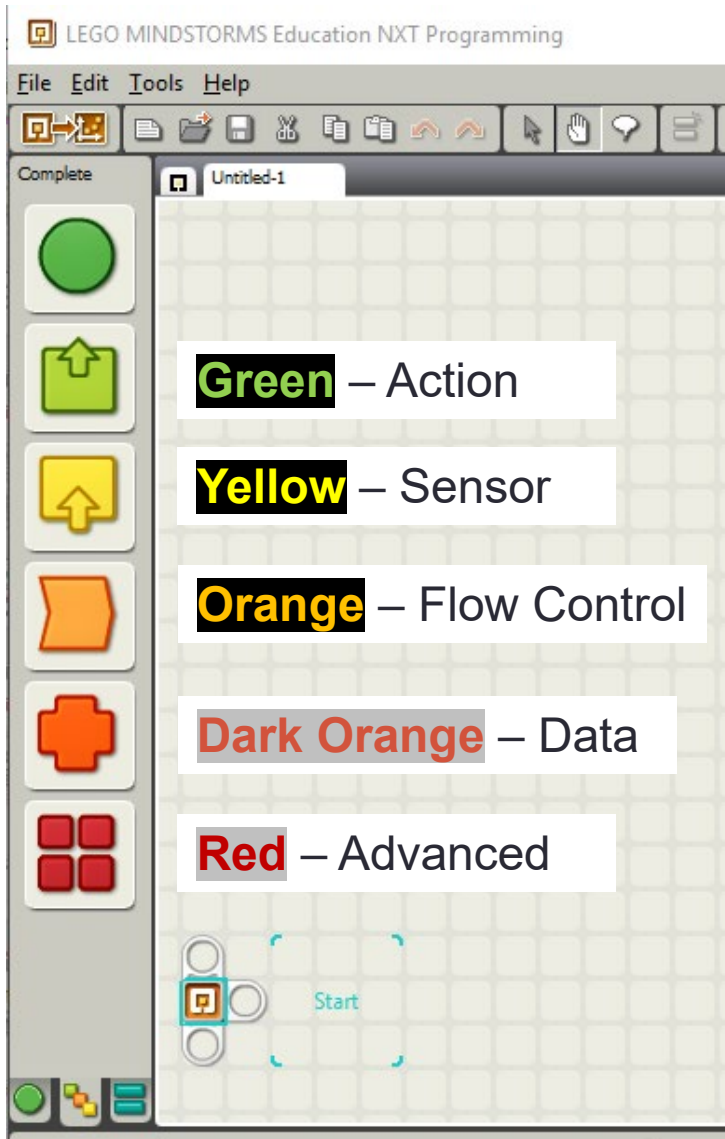
Modes: [Numeric](#) , [Logic](#)

Random

Quick links

- Choose the Output Type
- Modes
- Numeric
- Logic
- Inputs and Outputs

NXT Interface



Different layout, colour scheme and order, but mostly the same functions

LINE FOLLOWING

What are the programming challenges and how are they best approached?

Rescue Robot Programming

Where to start?

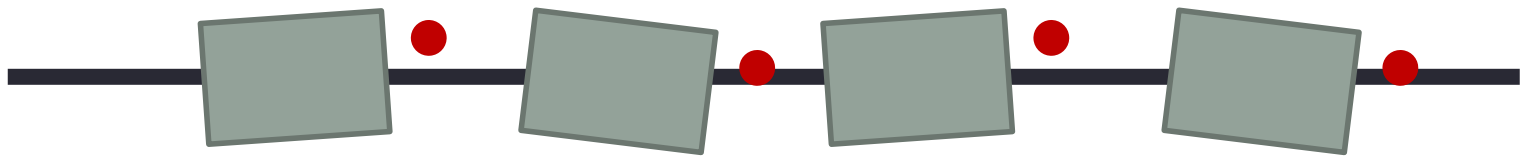
All Rescue divisions require the robot to:

- Follow a line
- Locate victim

One light sensor or two?

- Riley Rover Rescue (Victoria only) only needs one
- All other divisions require two

Principles of Line Following



Case 1

While True: `# robot on`

 While sensor sees white:

 Turn right

 While sensor sees black:

 Turn left

Case 2

While True: `# robot on`

 If sensor sees white:

 turn right

 If sensor sees NOT white:

 turn left

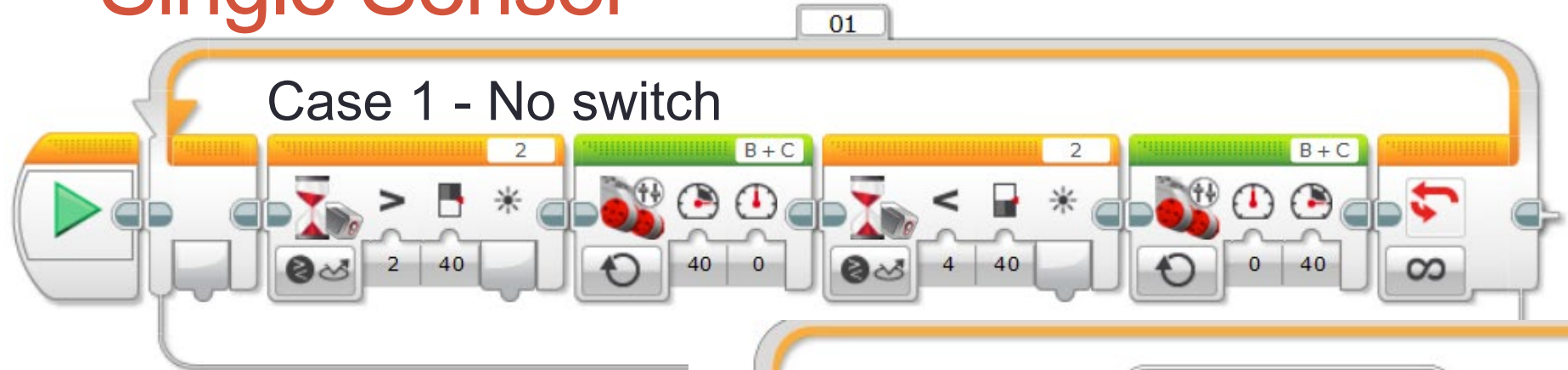
What does this look like in code?

Please note: I have not included comments with all example codes. A good exercise for the students is for them to add comments. It forces them to think through the code and also helps them to see the important role that comments play.

Single Sensor

01

Case 1 - No switch



Case 1

While True: # robot on

While sensor sees white:

Turn right

While sensor sees black:

Turn left

Case 2

While True: # robot on

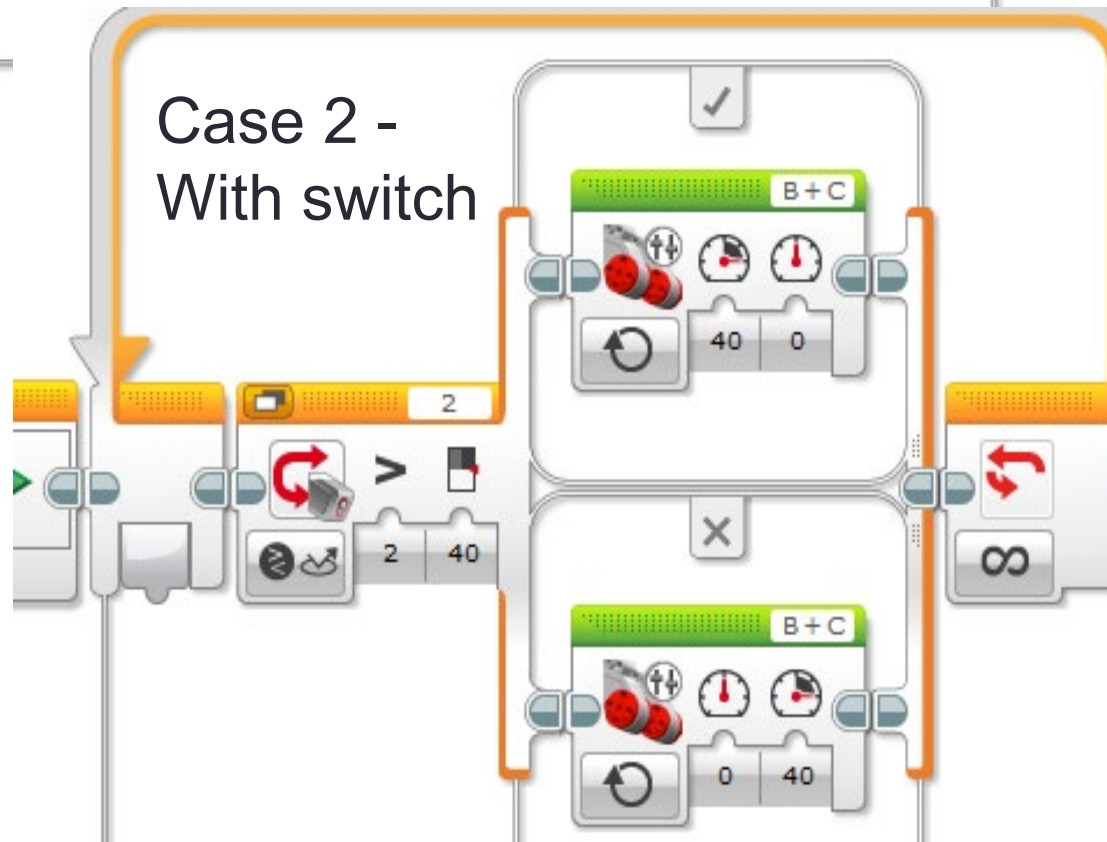
If sensor sees white:

turn right

Else sensor sees NOT white:

turn left

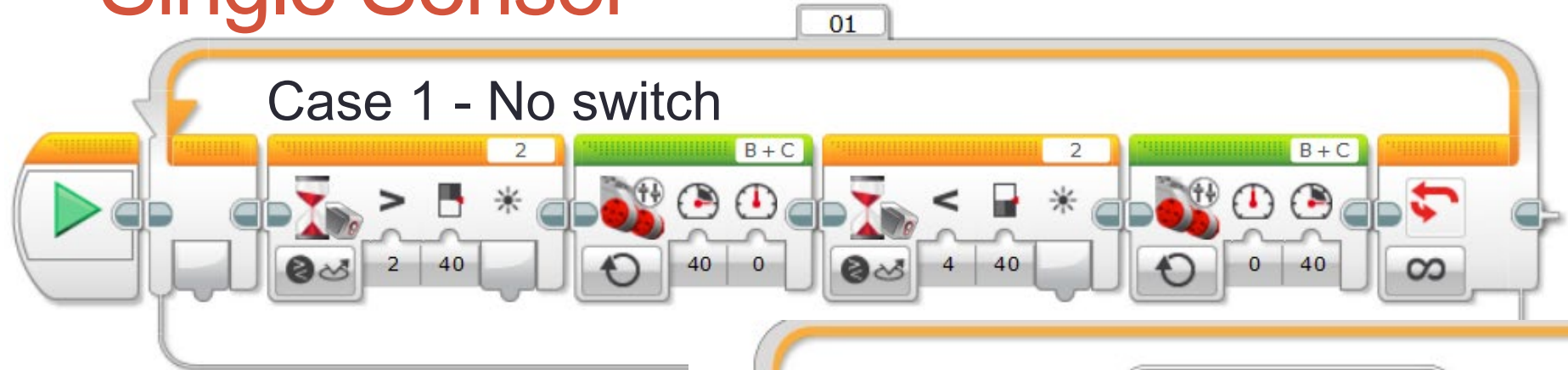
Case 2 - With switch



Single Sensor

01

Case 1 - No switch



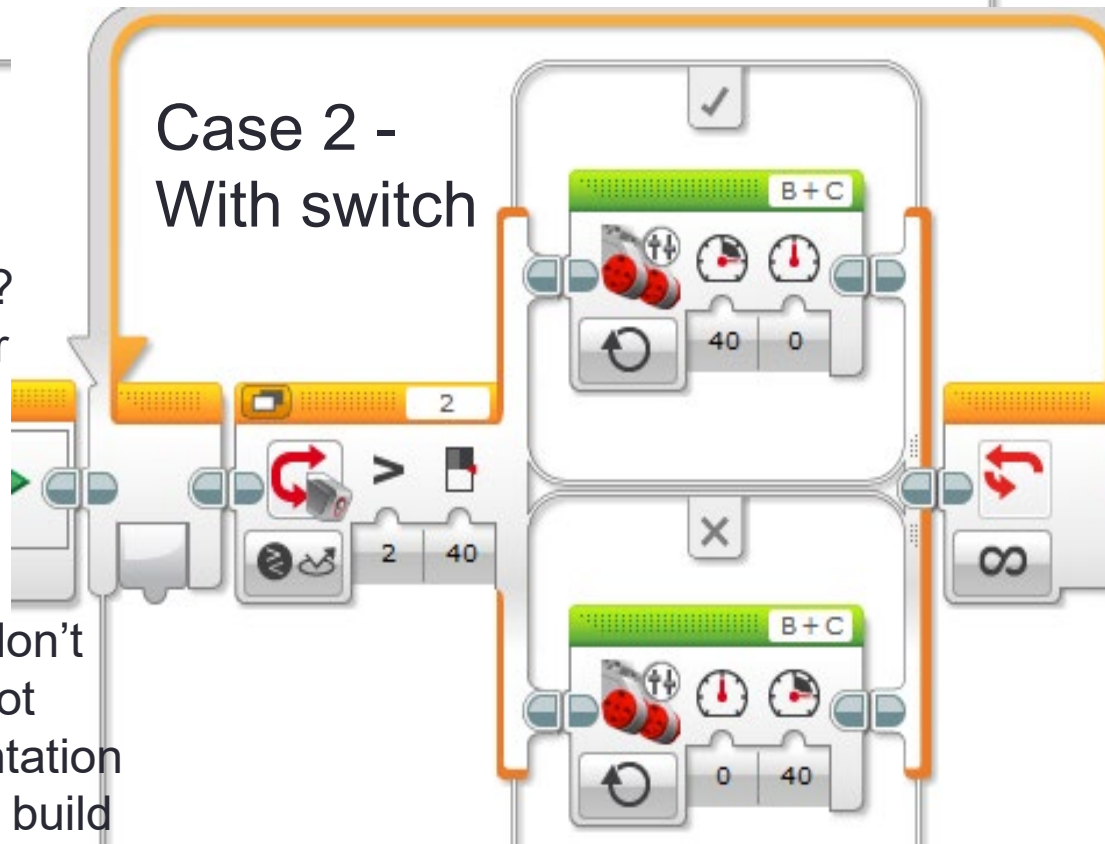
I have used reflected light

- What cut-off value should be used?
- Could this be done using colour?
- Why would you choose one over the other?
- Does motor power (speed) matter?

Common pitfalls

- Sensor/motor ports in program don't correspond to ports used on robot
- Motors mounted in reverse orientation
- This code may not work for your build

Case 2 - With switch



Adding a second sensor



Case 1

While True: `# robot on`

 While sensorLeft sees black:

 Turn left

 While sensorRight sees black:

 Turn right

What does this look
like in code?

Case 2

While True: `#robot on`

 If sensorLeft sees white AND sensor Right sees white:
 go straight

 If sensorLeft sees white AND sensorRight sees black:
 turn right

 If sensorLeft sees black AND sensorRight sees white:
 turn left

 If sensorLeft sees black AND sensorRight sees black:

`# Will this ever occur??? What should happen???`

Double sensor – Case 1



Case 1

While True: # robot on

While sensorLeft sees black:

Turn right

While sensorRight sees black:

Turn left

I have used the **Move Tank** block

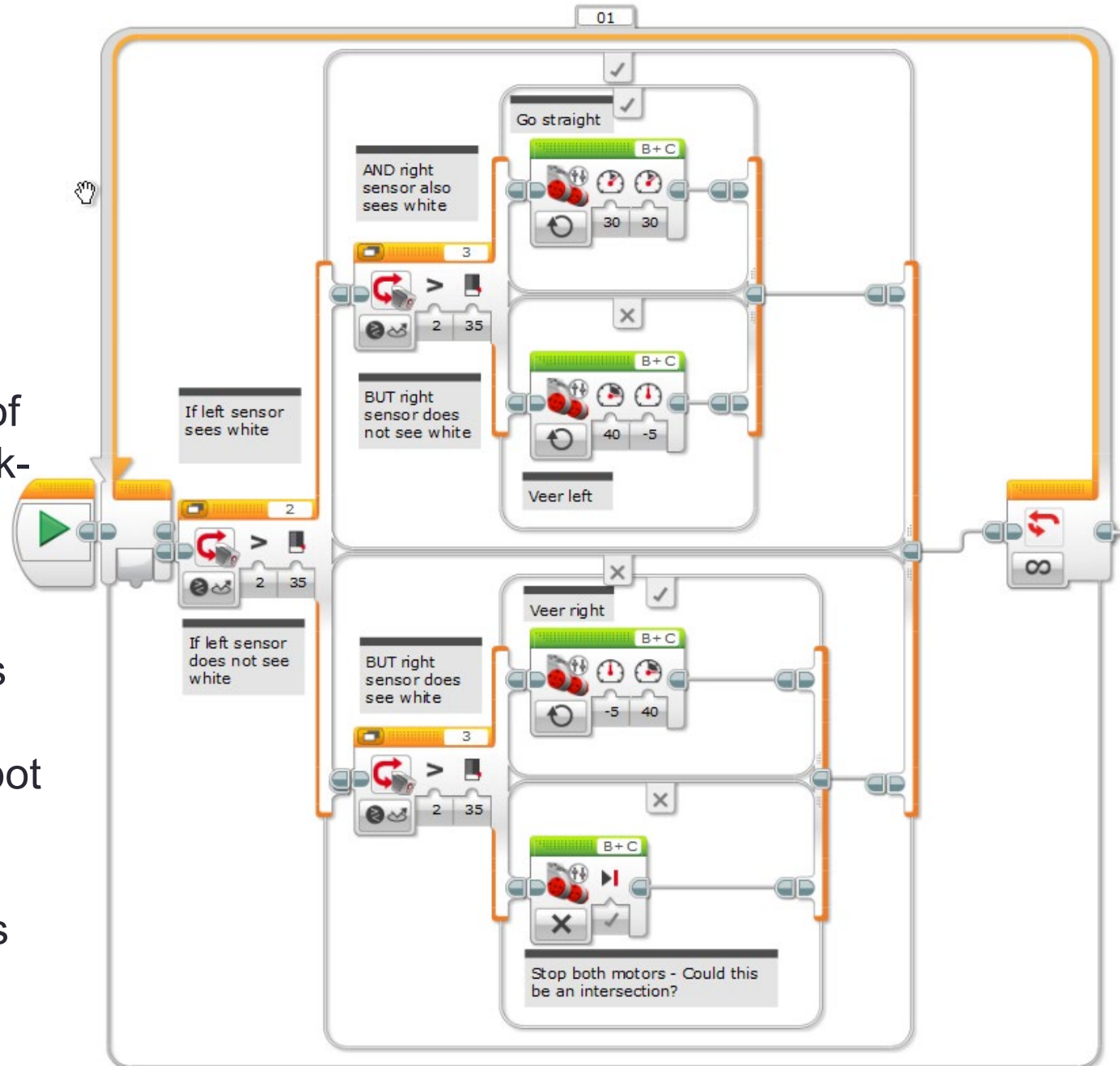
- Could I have used the **Move Steering** block?
- Could the robot look for white instead of black?

Also consider colour vs reflected light, cut-off values, motor power

Double sensor – Case 2

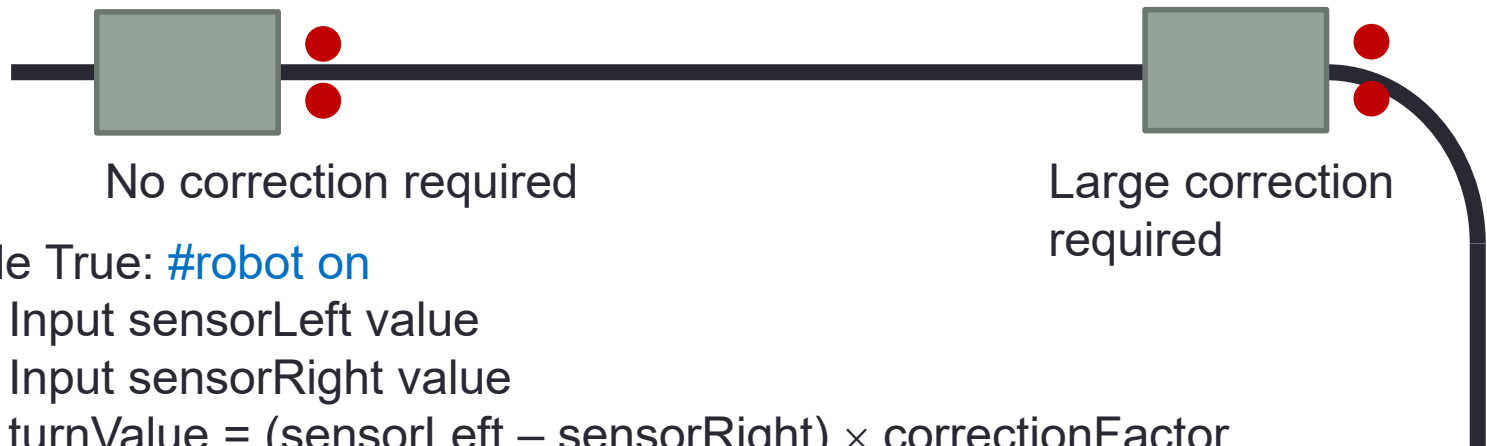
At the very bottom of the code is the black-black situation

- Does your robot ever encounter this? If it does, as written, this code will cause the robot to stop.
- What happens if the motor block is removed?



Proportional line follower

- What if I could adjust the amount the robot turns by how far off the line it is?
- Would this make for a much smoother run along the line?



While True: **#robot on**

Input sensorLeft value

Input sensorRight value

$\text{turnValue} = (\text{sensorLeft} - \text{sensorRight}) \times \text{correctionFactor}$

Input turnValue to Move Steering block

if sensorLeft > sensorRight, turnValue > 0 and robot turns right

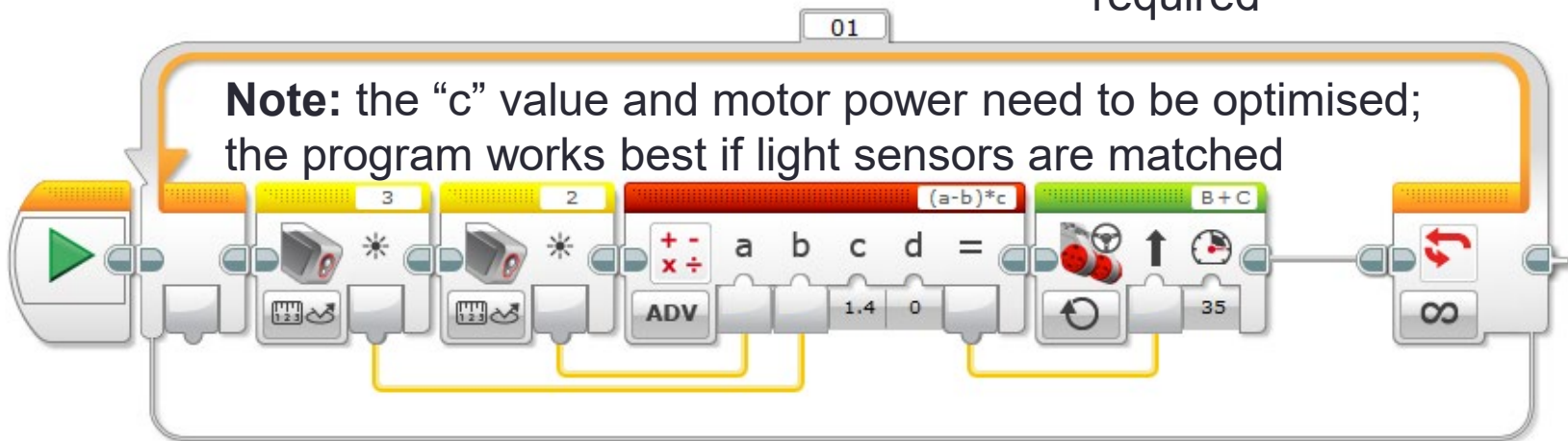
if sensorLeft < sensorRight, turnValue < 0 and robot turns left

Proportional line follower



No correction required

Larger correction required



Example 1

- Both sensors on white
- $60 - 60 = 0$
- $\text{turnValue} = 0$
- Robot goes straight

Example 2

- Left sensor on black (10)
- Right sensor on white (60)
- $10 - 60 = -50$
- $\text{turnValue} = -50 \times 1.4 = -70$
- Robot turns sharp left

Example 3

- Left sensor on edge (50)
- Right sensor on white (60)
- $50 - 60 = -10$
- $\text{turnValue} = -10 \times 1.4 = -14$
- Robot turns slight left

RESCUING THE VICTIM

What are the programming challenges and how are they best approached?

Rescue

Riley Rover (Victoria only)

- Push victim completely out of chemical spill

Primary Rescue

- Push victim completely out of chemical spill
- Exit chemical spill and recapture line (new in 2021)

Secondary Rescue

- Control and release victim in an upright position outside the swamp
- Exit chemical spill and recapture line

Open Rescue

- Lift victim onto rescue platform in upright position
- Exit chemical spill and recapture line

Entering chemical spill and detecting the victim

How can the robot detect chemical spill tile?

- Highly reflective tape at entrance

How can the robot detect the victim?

- For Riley Rover, it doesn't need to but more efficient if it does

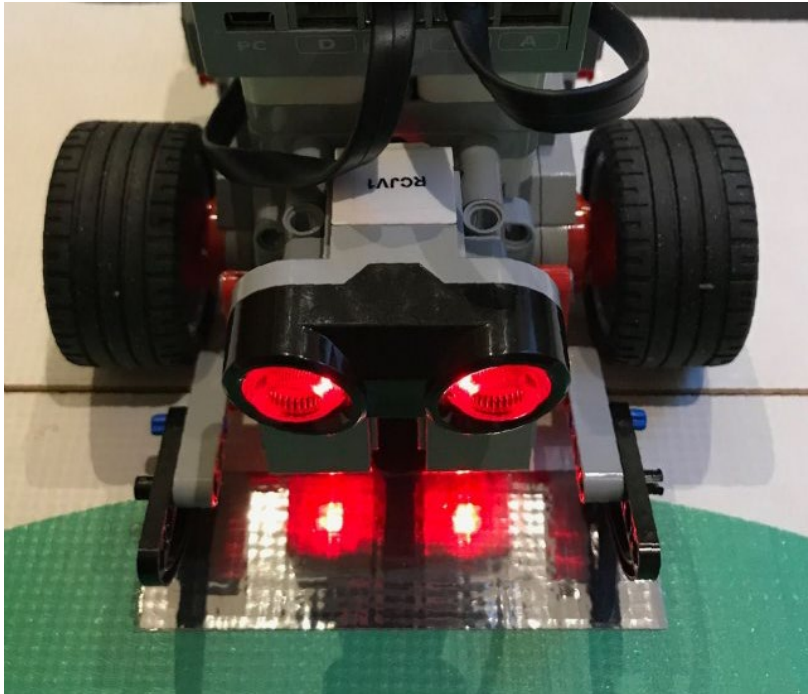
How can the robot control the victim?

- What level of control is needed for each division?

How can the robot exit the spill and regain the line?

- Not required for Riley Rover

Think through problem – Detecting the spill tile



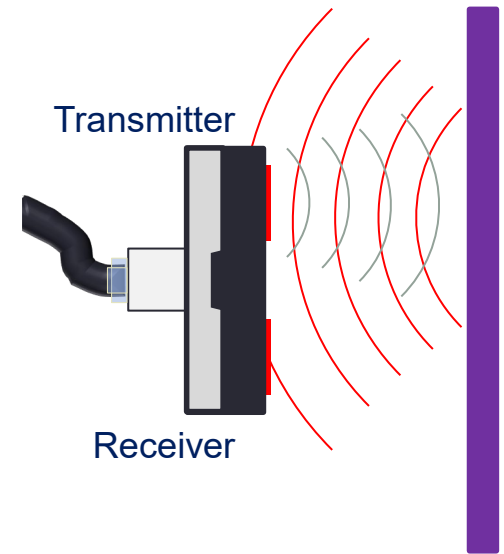
Detecting chemical spill

- Does the reflected value for the foil tape differ from white?
- Does the tape have a colour value?
- Are measured values for reflected light and colour consistent?

Think through problem – Finding the victim

Finding victim

- Ultrasonic sensor measures distance from object
- Where is the best place to position the robot to begin checking?
- What is the maximum distance the victim could be from the robot?
- What happens if the sensor is too close to the victim?
- Is the ultrasonic sensor able to detect curved surfaces as easily as flat surfaces?
- What if alignment is not perfect?



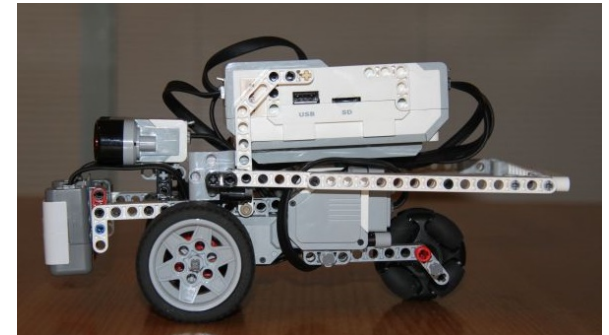
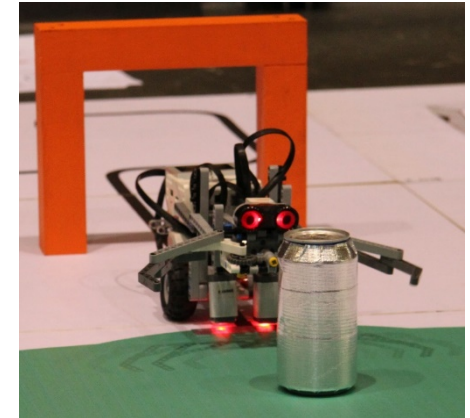
Thinking through problem – Rescuing the victim

Controlling the victim (secondary)

- Grabber mechanism
 - **Positive:** Can get away with not being perfectly lined up since grabber will gather victim in
 - **Negative:** Can be bulky and add to length of robot resulting in course navigation problems
- Cage mechanism
 - **Positive:** Much more compact
 - **Negative:** Need precise alignment

Rescuing the victim

- Push/drag rescue capsule to white and release
- What happens if the robot has missed or lost control of the rescue capsule?



Think through problem – Finding exit and regaining the line

Finding exit (3 options – are there others?)

1. Use single light sensor line follower algorithm to follow edge of green until reflective tape is reached
 - Where would you position the robot relative to the edge?
2. Random or systematic “walk” until reflective tape is detected
3. Record and retrace steps

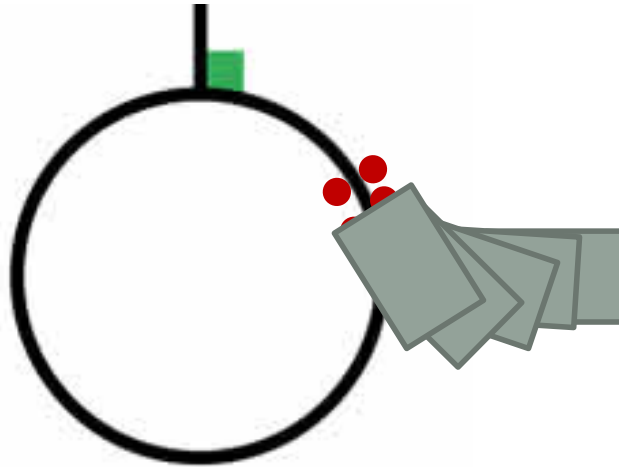
ADDITIONAL CHALLENGES

Detecting intersections

Navigating around water tower

Detecting green at intersections

Robots should turn in the direction of the green marker



Detecting green at intersections

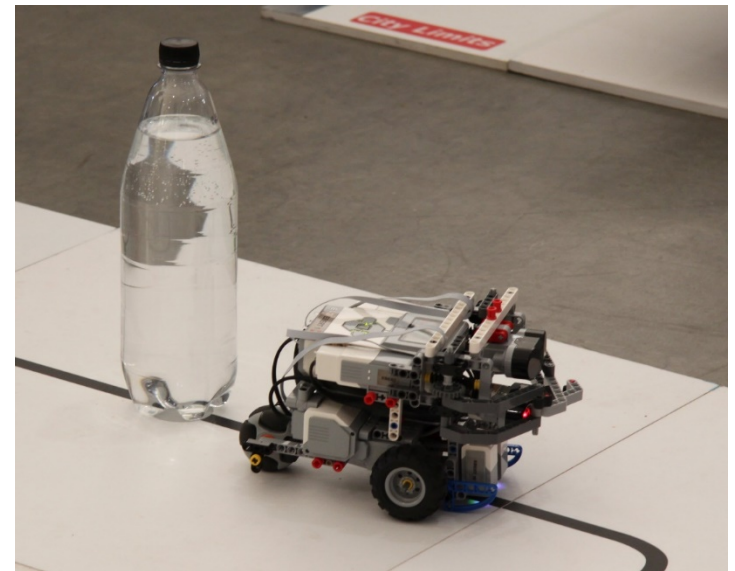
NOTE: The green on Rescue challenge mats used in the Victorian competitions are detected as green by Lego EV3 colour sensors. This is not necessarily the case for all Rescue mats and may not be true if sensors change. There is nothing in the rules that specifies the shade of green.

Thinking through the problem:

- Does the robot turn correctly using a basic line following program? Always? Most of the time? Rarely? Never?
- Do the colour sensors detect the “green” as “green”?
- What are the reflected light values when over the green squares? Are they unique individually? As a sum? As a difference? Can you use any of this to reliably detect green?

Navigating around water tower

- What should be used to detect the water tower (Ultrasonic? Touch?)?
- It is relatively easy to pre-program a route around tower, but ...
 - What happens if robot isn't perfectly aligned with water tower?
 - How does the robot know when it has found the line again?
- Must recapture the line on the same tile to get points



Troubleshooting

Check that:

- Port settings in program **match ports** on robot
- **Correct sensor** type is being used in switch/wait/loop blocks
- **Actual reflected light/colour readings** correspond to values set in program
- Motor power values are reversed in program if **motors are in reverse** orientation
- **Loop exit conditions** are set correctly
- **Motor “On” conditions** are set correctly
- **Sounds** aren’t affecting program flow

And a few tips

- Program in small increments
- Use the brick status display blocks or sounds to help identify if a particular part of the program is being executed
- Run the program while the robot is connected to the computer to show which block(s) are being executed
- Use My Blocks to help organise more complex programs
- **Save programs with significant changes as a new version, so stable older versions are not lost**
- On competition days make sure that programs loaded on the brick are functional programs (don’t leave rubbish programs on the brick that could be run by accident)