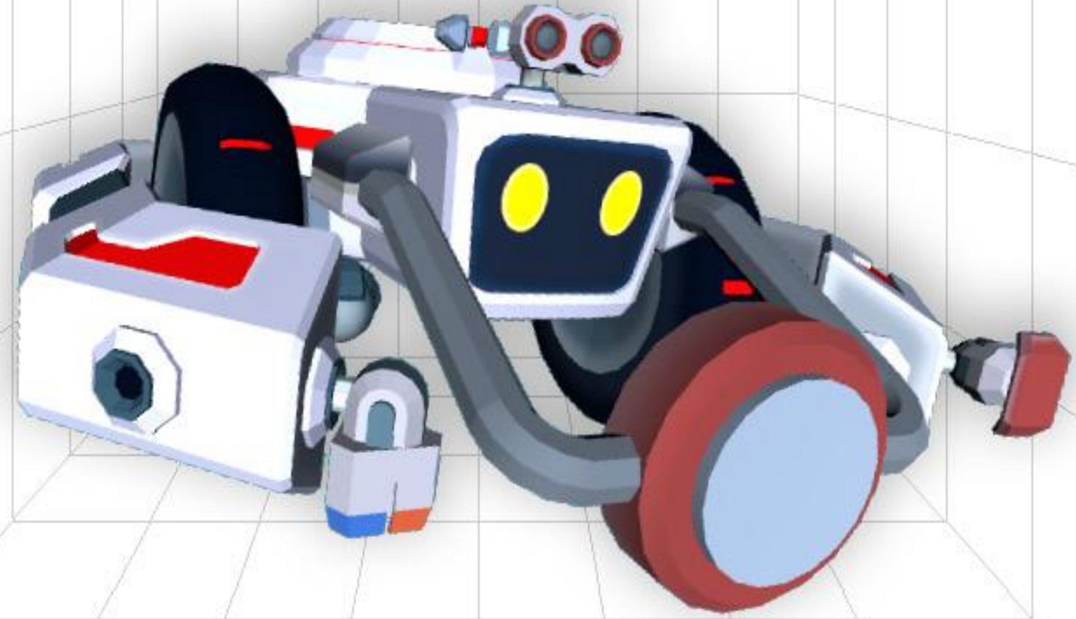
CoderZ™

CYBER ROBOTICS 102

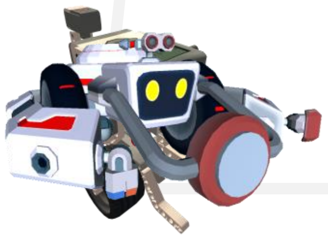
TEACHER'S GUIDE

VERSION 3.2



TOC

Content



1 About the Course

Course Description

Cyber Robotics 102 (CR102) is a continuation of the Cyber Robotics 101 course. This new course teaches STEM and coding topics using gamification and a physically based environment. CR102 introduces autonomous systems, teaches scanning and mapping the environment, error correction methods, and different system control algorithms. All the above and decision making are practiced in a random changing, physics, 3-dimensional environment as in real-life. By the end of the course the students will understand the physics forces acting on robots and their influence on the robot performance (kinematic and dynamics), and be capable of controlling and programming a robot that can interact with objects around it and can safely navigate through different changing environments.

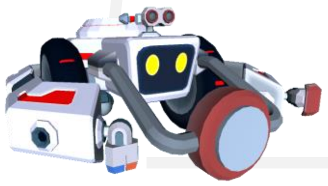
The course is powered by CoderZ's Online Robotics Learning Environment which provides besides the online physically based simulation: visual code editor, embedded content, and class management.

CoderZ's gamification environment provides an effective, informal learning environment, and helps students practice real-life situations and challenges in a safe environment. This leads to a more engaged learning experience that facilitates better knowledge retention. By completing missions, the students will learn the course curriculum effectively and yet still have fun.

The Cyber Robotics 102 curriculum teaches:

> Physics:
- Kinematics: Speed, Acceleration
- Dynamics: Forces, Moments, Levers
- Sensors and Physics

> Control systems:
- Motors
- Speed Control
- Obstacle Detection/Avoidance
- Open/Closed Loop Control
- Error Correction
- Two-state Control (On-Off Control)
- Proportional Control



About the Course

> Software:
  - Data Types and Variables
  - Operators: Mathematical and Logical
  - Conditions and Loops

> Mathematics:
  - Graphical Representation
  - Geometry and Stereometry: Distance, Angles, Coordinates
  - Spatial Cognition
  - Optimization

The course comprises of 15 sessions that fit into 1-2-hour lessons. The curriculum duration may vary depending on the student's previous experience, the time they invest in class and at home, and the time the teacher spends on theoretical subjects in class. The teacher login allows the educator to control student progress by setting the availability of program segments for students.

Course Content

> Over 25 hours of curriculum, activities, and assignments.

> Over 70 gamified missions with easy to follow walkthroughs and tips.

> Teacher resources including guides, presentations and solutions.

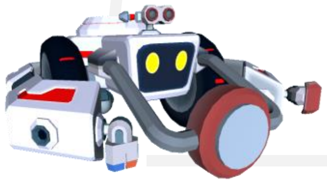
> Online support, knowledge base with articles and a call center.

Accessing the Course

> Cyber Robotics 102 is available in CoderZ Learning Center.

> All the course materials are linked from this teacher's guide.

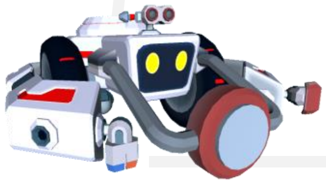
> As we update the guide every so often, we recommend you use the latest version linked on the course's notecard in the Learning Center.



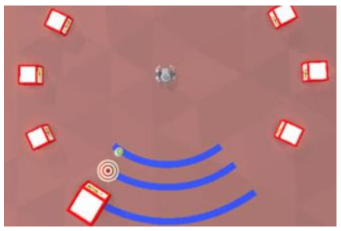
2 Course Outline

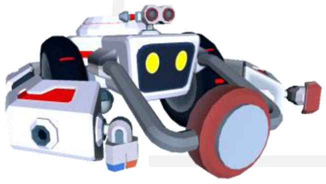
Session Topics

#	Topic	Topics Learned	
1	Plains & Hills 1	Getting started with robot control in a physically-based environment. Dynamics: distance, speed, and acceleration	
2	Plains & Hills 2	Forces and their influence on the robot: motor and gravitation Newton's second law: acceleration, mass, and forces Graphical representation	
3	Cruise Control	Understanding speed control & feedback - closed loop control Conditional statement If/Else	
4	Dangerous Curves 1	Pivot and screw turns using two motors Gyroscopic sensor	
5	Dangerous Curves 2	Smooth turns Understand advantages and disadvantages of different turns	
6	Doodling with Distance	Applying what was learned so far and carrying out creative challenges	
7	Touch, Avoid, Repeat	Obstacles recognition and avoidance Repeat loops Touch and ultrasonic sensors	



Course Outline | Session Topics

#	Topic	Topics Learned	
8	Random Obstacles Ahead	Variables: integer & double data type Mathematical operators: +, -,*, remainder of (modulo) performing missions in non-deterministic environment	
9	Radar Missions	Robotic Joints Controlling sensors' joints position for effective scanning of the environment	
10	Colorful Code	Color sensor Error correction Variables: Boolean data type Logic operators: not, and, or	
11	Repeat Again	Nested loops and their utilities. Applying what was learned so far and carrying out challenges and missions	
12	Magnetic Manipulation	Using the magnetic arm to interact and rearrange objects in the scene	
13	Line Following Logic	Following a line for accurate navigation On-Off control Proportional control	
14	A Hard Block Life	Encoders and their utility for distance measurement Y-axis Gyro Aligning to a line using two color sensors and proportional control	



Course Outline

Session Format

Learning Goals

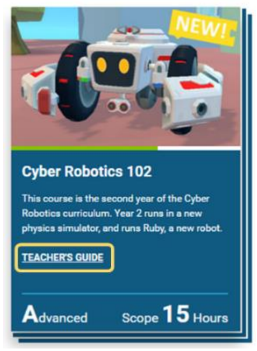
Each session is based on different topics and Learning Goals, while building on top of previous experiences and lessons learned. Actually, all the games are based on them.
We outline the Learning Goals at the start of each session.

Resources

We have developed different resources for each session for you to use in class. These resources include presentations, tutorials, and experimental games, all to ensure the best understanding of the material studied.

Teacher's Guide

The teacher's guide describes the whole course. It provides general information: course description, general guidelines, and course resources, as well as presenting a description of each session: Learning Goals, main topics, emphasis of main principles, and teacher notes which accompany the presentations.



Cyber Robotics 102

This course is the second year of the Cyber Robotics curriculum. Year 2 runs in a new physics simulator, and runs Ruby, a new robot.

TEACHER'S GUIDE

Advanced Scope 15 Hours

Look for Teacher's Guide link on the Cyber Robotics 102 notecard.

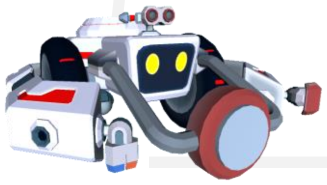
The Lesson Plans

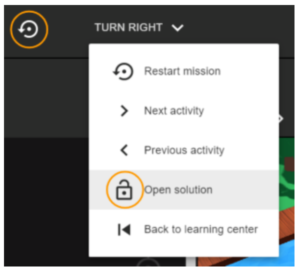
Each session is guided by a presentation that covers the entire session from theory to practice, including:

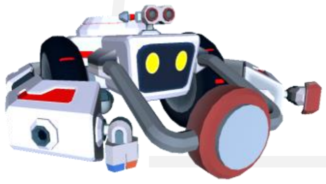
> Theoretical background
> Examples and sample code
> Discussion guidelines
> Playtime activities

Each lesson's presentation is part of the environment and is divided into several parts, so they give guidelines and an introduction to the following missions. The teacher's notes (in this teacher's guide appendix) accompany the presentations and give more emphasis, ideas for extra-activity in class, advanced information, etc.

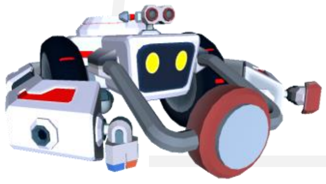
Play Time	During each session, students will be required to complete missions in CoderZ and experience problem-solving using computer code. We also mention how long this should take based on our experience. This may vary, so feel comfortable to extend play time to make sure students make the most of it. If, for whatever reason, students do not complete all missions in the allocated time, that is fine. Just ask them to complete it at home, or after class. You can use the class Heat Map to track their progress. If they complete the missions before the time runs out and the session materials are all covered, you can either go through some solutions with the class or start the next session
Reflection	We allocate a short time at the beginning of each session (except the first session) for reflection on the previous lesson. During this time, you are encouraged to engage the classroom in a discussion and get students who have solved advanced missions to present their solutions to the classroom. We have found this time for reflection to really help students deepen their understanding while highlighting key Learning Goals Here are a few steps we recommend to better prepare for the reflection part of the lesson: 1. Open the class Heat Map to see how the students are coping with the missions. 2. Identify the missions' students are most challenged by, look for columns with too many yellow/red tiles. 3. Identify students who have outperformed the class in those missions, they'll have green tiles for that mission. 4. Ask those students to present their solution OR



	Reflection	5. Solve these missions with the class. Use the “open solution” option if you like (from the mission menu). 6. Ask students to explain what they have learned by watching the solution - this will give you more insight as to what students are struggling with. For more information about reflection check out these articles: Student Progress Report Class Heat Map
	Mission Structure & Teaching Aids	Each mission has a solution, which is available only to the teacher.  TURN RIGHT Restart mission Next activity Previous activity Open solution Back to learning center > Each mission has tips. Encourage students to check them out before turning to you for help. > Each mission has a guided tour which starts automatically the first time a mission is opened. The tour can be restarted from the mission menu. > It is ok if not all your students complete the missions during the time frame suggested. Encourage them to complete the missions before the next lesson, which they can easily do after class, from any computer. > If your students complete the missions before the class ends, you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution, or help their friends to complete missions. Alternatively, you can proceed to the next session. > Make sure your students are familiar with: • Simulation controls • Manual control > We recommend you go through the session resources, missions, and solutions yourself prior to each lesson. > Before the first session create your class & invite your students. > Make sure your computer lab meets the minimum requirements, and that you’ve followed our class setup guidelines.

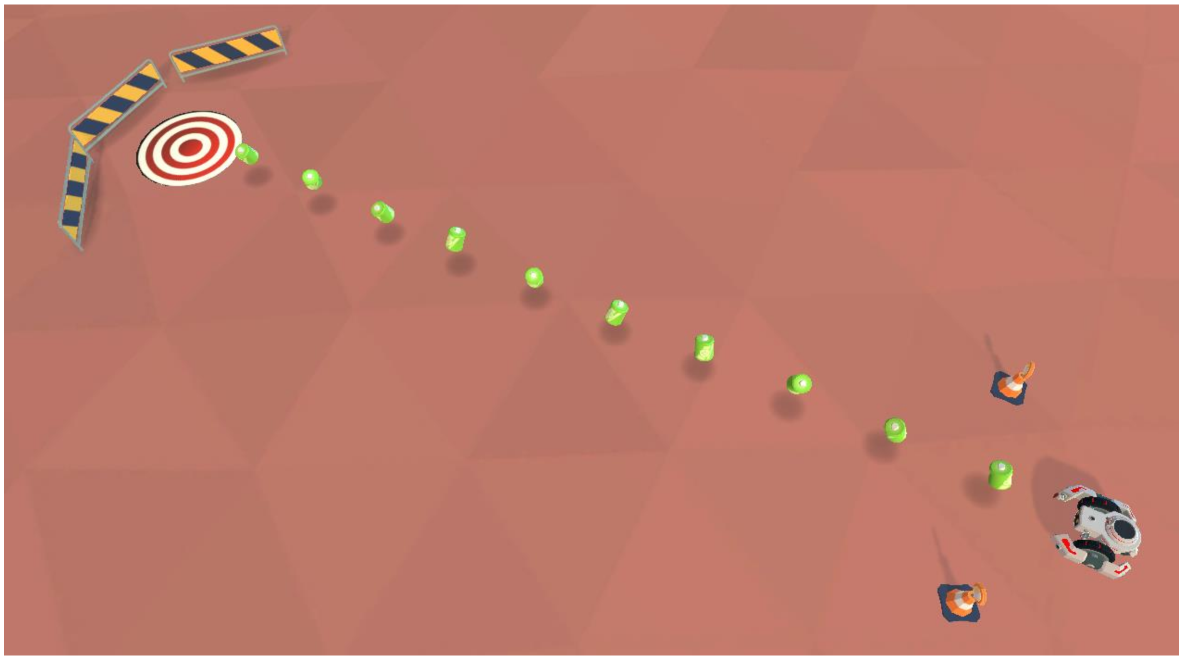


Pacing Guidelines	Dividing presentations between missions helps you know how the lesson is divided logically. Remember that timing varies from class to class so remember these are guidelines, not specific instructions to follow. If your class needs more time, gi8ve it to them. If they progress faster than expected, reflect (see below) on their progress.



Lesson 1: Plains & Hills 1

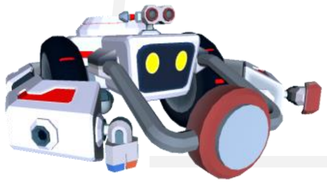
Meet Ruby and drive in the new physical environment



Learning Goals

By the end of this lesson students will:

- Understand the new physical environment
- Familiarity with Ruby - CR102's innovative robot
- Familiarity with different types of motors
- Understand the new motor power block
- Embed the concepts of speed, acceleration, deceleration
- Newton's second law: more power means greater acceleration
- Embed the concepts of momentum and braking distance
- Login to CoderZ and complete basic navigation missions



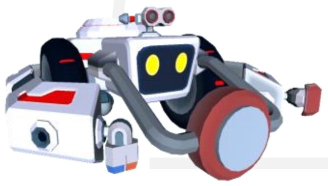
Description

The Cyber Robotics 102 course simulator produces a “real world” **physical environment** so the first goal for Lesson 1 is understanding this environment. A physical environment means an environment that simulates reality - i.e. if we command our robot to drive, it will start from speed zero and accelerate, or vice versa, if we command our robot to stop it will take some time to stop as it decelerates gradually. Another example is when we want to drive our robot uphill or downhill - it will decelerate or accelerate accordingly, under the influence of gravity.

The new physical environment provides a great opportunity for learning the challenges that engineering is dealing with while designing robots such as autonomous cars - they encounter physical rules! Therefore, Cyber Robotics 102 course teaches STEM subjects while enabling the students to develop intuition, experience relevant studies, as well as enjoying completing missions for better internalization of the curriculum.

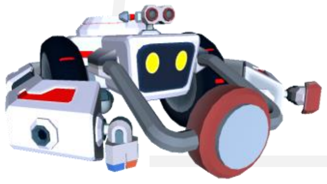
In this lesson, the students will meet Ruby the Robot. They will program her to drive forward and backward on even plains, learn to use the "**set motor power**" and "**brake**" blocks, as well as the meaning of speed and acceleration. The students will understand that power leads to acceleration and more power leads to a greater acceleration. That is according to Newton's second law, which implies that the acceleration of an object is directly proportional to the force acting on the object. Newton's second law is one of the fundamental laws in physics (dynamics). We will discuss it in the next session.

At the end of the lesson, the students are required to drive Ruby uphill and probably will encounter a difficulty - Ruby fails the mission unless it acquires momentum. The students will understand that intuitively, as well as through trial and error. In the next lesson we will discuss the physical reasons for that.



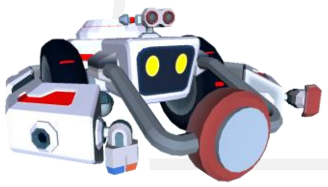
Resources

	Slideshows - within the pack
	Teacher's Notes - Plains & Hills1
	Set motor power block - Knowledge Base article
	Wait for block - Knowledge Base article
	Brake until stop block - Knowledge Base article
	Brake wheel block - Knowledge Base article
	Video about position, velocity, acceleration: https://www.youtube.com/watch?v=4dCrkp8qgLU
	Video about acceleration: https://www.youtube.com/watch?v=J3SdZwMcWhA



Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Plains & Hills 1 Part 1 slideshow	10 minutes	Introduction to Ruby, to the physical environment
2	Mission 1	5 minutes	Test Drive: “Set motor power” block
3	Go through Plains and Hills 1 Part 2 slideshow	5 minutes	Speed & acceleration “Wait for” block “Brake until stop” block Braking is a gradual process
4	Mission 2	5 minutes	Drive forward
5	Go through Plains & Hills 1 Part 3 slideshow	5 minutes	Drive forward Introducing missions 3-5
6	Complete missions 3-5	10 minutes	Back & Forth Rolling Uphill Giddy Up Doing the last 2 missions the students will understand using trial & error that they should acquire momentum (by driving back) in order to succeed
7	Lesson Summary	5 minutes	Class Conclusion Questions

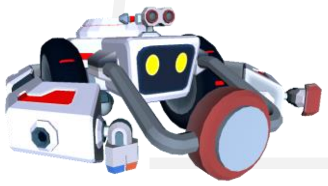


Tips

- Go through the presentation (and Teacher's notes), missions, and solutions prior to the session.
- Review the available resources suggested above.
- Don't rush things, take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.
- If your students complete the missions before the class ends, you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.

Class Conclusion Q&A

	Questions	Answers
1	What is the result of driving Ruby by supplying motors with constant power?	Constant power cause Ruby to accelerate - her speed rises continuously.
2	Why does Ruby not stop immediately when we brake?	Braking is a gradual process as objects have momentum.
3	Why does Ruby find it hard to drive uphill?	Ruby finds it hard to drive uphill because gravitational force is pulling her towards Earth.



Lesson 2: Plains & Hills 2

Speed, acceleration, and their relationship to gravity

Learning Goals

By the end of this lesson students will:

- Newton's second law: more power means greater acceleration
- Newton's second law: more mass means lower acceleration
- Familiarity with gravitational force
- Understanding gravitational force's impact on driving the robot uphill

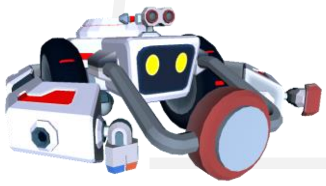
Description

The goal of Lesson 2 is to be familiar with the **concept of force and how forces influence the robots**. In Lesson 1, the students discovered that the more power they supply Ruby, the greater it accelerates, and the opposite effect of less power. When they tried driving uphill, they saw that Ruby encountered difficulty, and they had to program it to drive on a plain and gain momentum. That's because when driving uphill or downhill the **gravity** force influences Ruby.

Gravitational force is one of the fundamental forces of nature. This is an attractive force which exists between any two objects. Heavier objects always exert this force on lighter objects (both objects attract each other with the same amount of force, but lighter objects are more affected by this attraction). For example, when we throw a stone upward, it falls back down to the earth due to the gravitational force of Earth. In a similar way, the Moon orbits the Earth due to the gravitational force of the Earth, and planets are orbiting the Sun due to the gravitational force of the Sun. Gravitational force was discovered by Sir Isaac Newton in 1687.

Due to this gravitational force, Ruby finds it hard to drive uphill. Gravitational force pulls her down to earth, which causes Ruby to **decelerate**. In order to drive to the top of the hill, Ruby needs to start driving uphill with a **higher initial speed**. Therefore, she needs to drive back and away from the bottom of the hill in order to accelerate before driving up the slope.

In the last two missions, Ruby needs to carry a heavy **mass**. The students will find that increasing mass leads to reduced acceleration. That is also according to **Newton's second law of motion**, which states that acceleration of an object is inversely proportional to the mass of the object. We will mention Newton's second law in the next session.

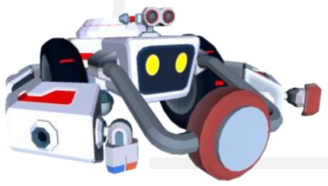


Resources

- Slideshows - within the pack
- Teacher's Notes - Plains & Hills 2
- Video about position, velocity, acceleration: https://www.youtube.com/watch?v=4dCrkp8qgLU
- Video about acceleration: https://www.youtube.com/watch?v=J3SdZwMcWhA
- Video about gravity: https://www.youtube.com/watch?v=Les9J2IQIZY
- Video about mass and acceleration: https://www.youtube.com/watch?v=WHEeGO9HVPc

Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Plains & Hills 2 Part 1 slideshow	10 minutes	Graphing Acceleration/ Deceleration. Gravity and its effect on Ruby's driving
2	Missions 1-3	10 minutes	Long Glide Going For A Dip Incremental Inclines
3	Go through Plains & Hills 2 Part 2 slideshow	5 minutes	The steeper the slope - the more force or speed we need in order to climb it. The same is for heavier mass.
4	Complete missions 4-6	15 minutes	Heavy Mass Building Up Weights Weight Lifting
5	Lesson Summary	5 minutes	Class Conclusion Questions



Tips

Go through the presentation (and Teacher's notes), missions, and solutions prior to the session.

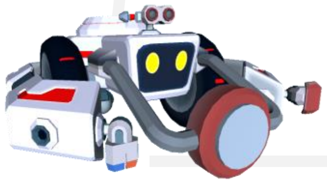
Review the available resources suggested above.

Don't rush things, take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.

If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.

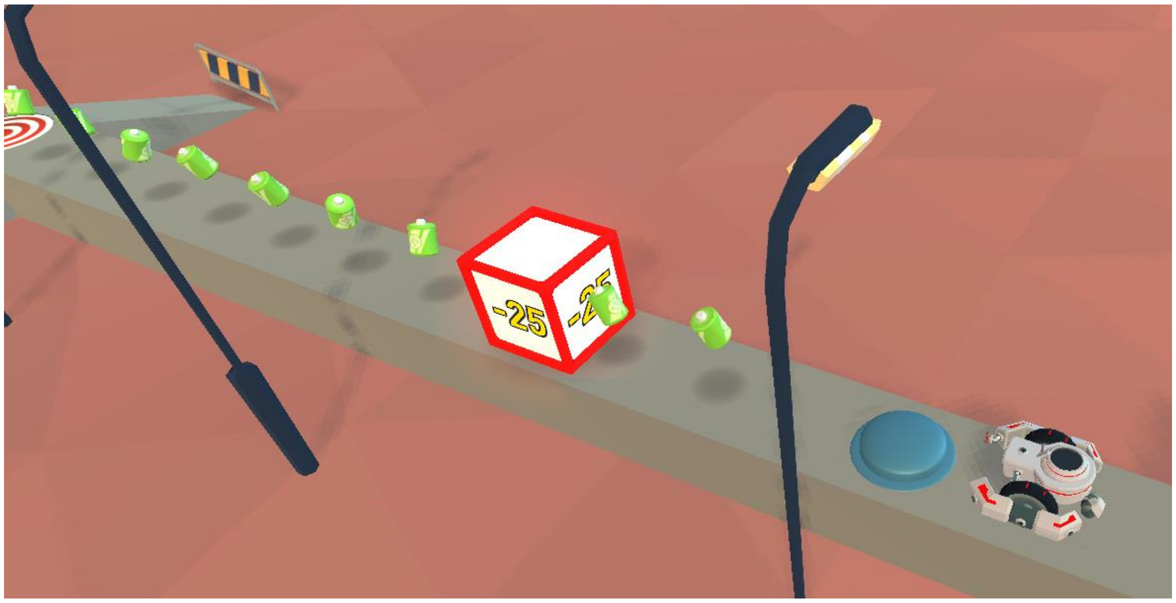
Class Conclusion Q&A

	Questions	Answers
1	How does gravitational force affect Ruby's speed?	Gravitational force causes Ruby to decelerate while driving uphill and accelerate while driving downhill.
2	Describe the relationship between the initial speed required to succeed driving uphill and the slope of the hill.	The steeper the slope is, the more initial speed we need in order to climb it.
3	Describe the relationship between the initial speed required to succeed pushing the mass and the size of the mass.	The heavier the mass is, the more initial speed we need in order to push it.
4	How can we increase the initial speed?	Greater initial speed is gained in one of two ways:



Lesson 3: Cruise Control

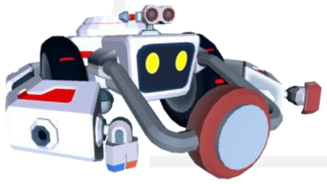
Using system control algorithms to control desired speed



Learning Goals

By the end of this lesson students will:

- The need to limit Ruby's speed
- Cruise control systems
- Control systems - open loop and closed loop
- Conditional programming: "if/else" block
- Loops Programming: "Repeat forever" loop block
- New movement block: "set wheel speed"



Description

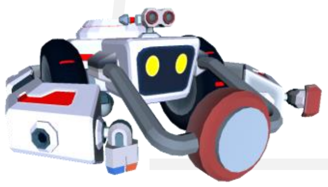
Lesson 3 introduces the students to **closed-loop systems control** while planning a simple cruise control system. The students will experience the difficulty of controlling systems, the need to have feedback from sensors (read speed value), and act accordingly (supply or cut power repeatedly in order to control the robot's speed as well as response to changing environment).

A **cruise control** system is a basic feature in cars nowadays. It is a system that automatically controls the speed of a motor vehicle by referring to a speed set by the driver and automatically speeding up or slowing down to maintain that pace. If the vehicle's speed is slower than the set speed, the system will accelerate until that speed is met. If the vehicle's speed is higher than the set speed, the system will decelerate until the vehicle slows to the set speed. Nowadays, new cars are equipped with adaptive cruise control (ACC), which allows the car to follow the car in front of it while continually adjusting speed to maintain a safe distance. ACC employs advanced radar and camera technology (ACC is not included in the CR102 course contents).

System control refers to managing or regulating the system's operation to a desirable operation. In the previous missions we controlled the robot by programming it to drive with pre-defined power for a pre-defined amount of time. Therefore, we had to write a specific program for each mission. Controlling a system without monitoring or measuring its performance is called an **Open-Loop control**.

Sometimes, mainly in an automated system, we must respond to the environment in order to operate safely and efficiently. A system control that takes into consideration feedback from the environment and operates according to that data is called a **Closed-Loop control**.

Understanding closed-loop control and experiencing programming together will make the students ready to meet the "**set wheel speed**" movement block. The "set wheel speed" movement block enables the user to control the wheel speeds easily, while a sophisticated internal cruise control system (a close-loop control algorithm) operates in the background.

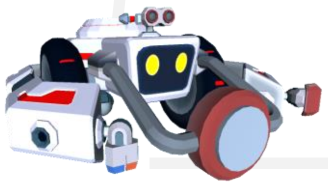


Resources

- Slideshows - within the pack
- Teacher's Notes - Cruise Control
- Set wheel speed block - Knowledge Base article
- If/else block - Knowledge Base article
- Repeat forever block - Knowledge Base article
- Video about adaptive cruise control: https://www.youtube.com/watch?v=GInSPWZRFRM

Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Cruise Control Part 1 slideshow	10 minutes	Importance of speed control
2	Missions 1	5 minutes	Speed limit
3	Go through Cruise Control Part 2 slideshow	5 minutes	Cruise Control Open loop vs. Closed loop "If/else" block "Repeat forever" block
4	Mission 2	10 minutes	Cruise Control
5	Go through Cruise Control Part 3 slideshow	5 minutes	Introducing missions 3-4
6	Missions 3-4	5+5 minutes	Uphill Scuttle Steady Climbing
7	Go through Cruise Control Part 4 slideshow	5 minutes	"Set Wheel Speed" block
8	Missions 5-6	10 minutes	Quick to the Block Speed Bump
9	Lesson Summary	5 minutes	Class Conclusion Questions



Tips

Go through the presentation (and teacher's notes), missions, and solutions prior to the session.

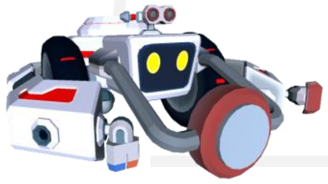
Review the available resources suggested above.

This lesson probably will take more than 45 minutes. Don't rush things and take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.

If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.

Class Conclusion Q&A

	Questions	Answers
1	Why it is important to set a speed limit?	High speed makes it difficult to control the robot: as speed increases - braking distance increases and accuracy decreases. This can cause severe accidents.
2	What is the difference between "open loop" and "close loop" system control? Which is better?	An **open-loop control** system does not take into consideration feedback from the environment and operates according to pre-defined programming. A **closed-loop control** system takes into consideration feedback from the environment and operates according to that feedback. That makes closed loop systems better as the can respond to changing environment.
3	Why did we use a "repeat forever" block in order to control our speed?	We used **"repeat forever"** block in order to measure **constantly** our speed. Without the "repeat forever" block, the "if-else" block will check speed only on program start and afterwards speed will not be monitored.
4	What is the advantage of using the "set wheel speed" block?	The "**set wheel speed**" block enables us to control Ruby's speed easily while operating in background a sophisticated internal cruise control system



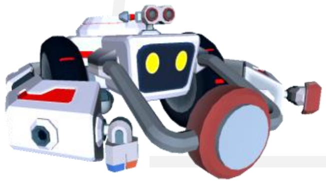
Lesson 4: Dangerous Curves 1

Making turns while measuring distances and angles

Learning Goals

By the end of this lesson students will:

- The Gyroscope sensor
- Learning the different types of turns that the robot can perform and how to execute them
- Understanding the idea that precision is required for accurate navigation
- Understanding that speed and accuracy are inversely related
- The "Drive Distance" movement block
- Using Explore Mode to measure distances and angles
- Measuring and practicing calculations of angles



Description

In Lesson 4, the goal is to understand the **kinds of turns** the robot can perform, and how to execute them. The students will meet the Gyroscope sensor and use it to perform different turns.

The **Gyroscope Sensor** measures in degrees how far the robot has turned from its starting point. Ruby, our innovative robot, has 3 gyroscope sensors for the 3 axes. In this lesson the student will use only the Y-axis (rotation).

There are several different types of turns we can execute: **screw turns**, **pivot turns**, and **smooth turns**. In Lesson 4, the students will experiment with the first two. These two types of turns require the "**break until stop**" block in order to perform an accurate turn (in order to perform accurate turns, we need to first stop the robot, as speed reduces accuracy).

While performing several consecutive turns, the students will remember the "**reset gyro**" block. Resetting the gyroscope sensor after use or before using it again makes following degrees easier.

In order to navigate accurately, the students will use **Explore Mode** to measure distances and angles. After measuring the distance, the students can drive distance accurately using the "**drive distance**" block, which enables control of Ruby's driving distance using an internal algorithm which measures driving distance and controls speed accordingly. The algorithm uses Ruby's encoders. We will learn about the encoders in the more advanced lessons.

Resources

Slideshows - within the pack

Teacher's Notes - Dangerous Curves 1

Gyro Sensor - Knowledge Base article

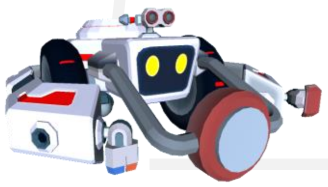
Reset gyro block - Knowledge Base article

Drive distance block - Knowledge Base article

Explore Mode - Knowledge Base article

Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Dangerous Curves 1 Part 1 slideshow	10 minutes	Using the gyro sensor Type of turns Low speed leads to a precise turn Explore Mode - measure distances and angles
2	Missions 1-2	10 minutes	L Turn V Turn
3	Go through Dangerous Curves 1 Part 2 slideshow	5 minutes	Gyro readings are continuous The "reset gyro" block
4	Mission 3-4	10 minutes	N Curve M Curve
5	Go through Dangerous Curves 1 Part 3 slideshow	5 minutes	Pivot turns
6	Missions 5-6	10 minutes	I Curve J Curve
7	Lesson Summary	5 minutes	Class Conclusion Questions



Tips

Go through the presentation (and teacher's notes), missions, and solutions prior to the session.

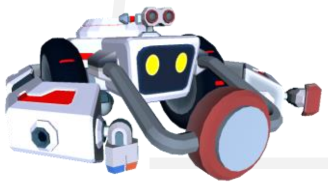
Review the available resources suggested above.

This lesson probably will take more than 45 minutes. Don't rush things, take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.

If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session

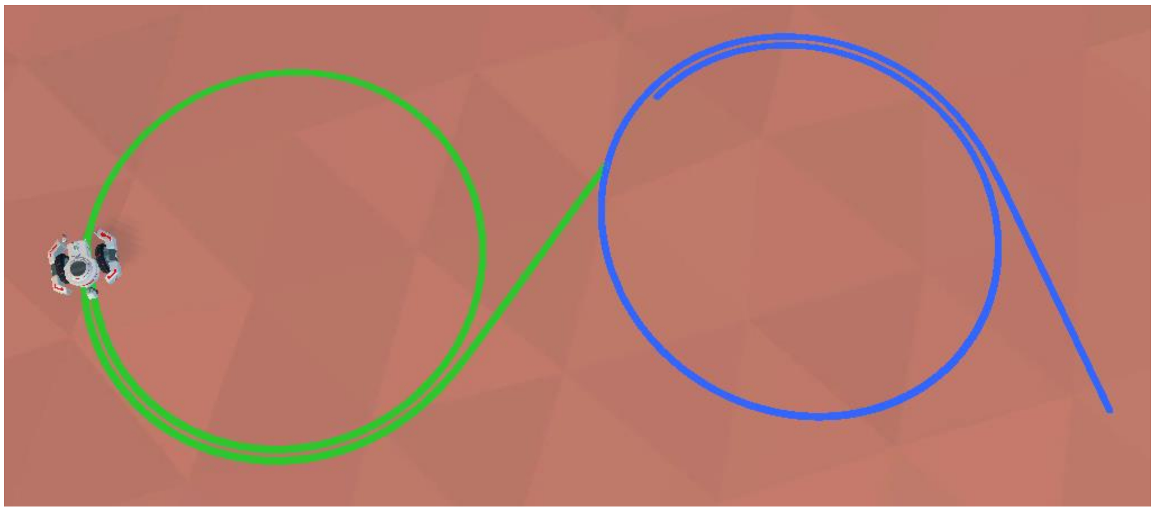
Class Conclusion Q&A

	Questions	Answers
1	What information do we get from the gyro sensor and what it is good for?	The gyro sensor measures in degrees how far the robot has turned from its starting point. Using the gyro sensor's data, we can perform turns accurately.
2	When do we use the "reset gyro" block?	We use the "gyro reset" block whenever we want to define a new reference direction. Defining a new reference direction helps us calculate angles as we do not need to remember the accumulative angle we have turned.
3	What is the difference between pivot and screw turns?	A screw turn is a turn around the center point of the robot, while a pivot turn is a turn around one of the wheels. Screw turns are better for narrow turns.
4	What is the relationship between speed and accuracy?	Speed and accuracy are inversely related - so if we must be precise, we must drive at a lower speed.



Lesson 5: Dangerous Curves 2

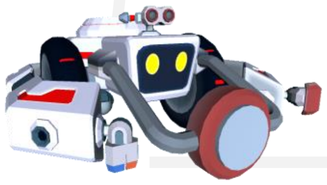
Accurate turns using the “turn to” block. Smooth Turns.



Learning Goals

By the end of this lesson students will:

- Smooth turn practice
- The new “turn to” movement block
- Practicing different kinds of turns



Description

Lesson 5 introduces a new movement block, the "**turn to**" block. Using a "turn to" block enables us to program Ruby to turn to a defined angle easily. The "turn to" block performs a screw turn using the gyro sensor and a closed-loop control algorithm which leads to an accurate turn. Remember the difficulties we encountered in Lesson 4 to perform an accurate turn because of the robot's momentum!

Lesson 5 teaches what a **smooth turn** is and how to perform it. A smooth turn is a gradual turn in which the robot's wheels both turn in the same direction at different speeds: the outer wheel turns faster than the inner wheel. The turn is in the direction of the slower wheel. A smooth turn does not require stopping and Ruby can perform it without the need to use the "break until stop" block. Therefore, it is the most rapid turn. Its main disadvantages are that it requires a wide space and it is hard to be accurate. To improve accuracy, we can reduce speed.

While performing a smooth turn, the students use an open loop control, as the wheel's speed is pre-programmed and there is no input or feedback from the environment for monitoring performance. In advanced lessons we will learn and practice methods for performing smooth turns using feedback, so performing turns will be much more accurate.

At the end of the lesson, the students will practice different missions which involve different kinds of courses - plains, hills, straight lines, and curves.

Resources

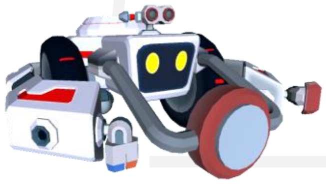
Slideshows - within the pack

Teacher's Notes - Dangerous Curves 2

Turn to block - Knowledge Base article

Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Dangerous Curves 2 Part 1 slideshow	10 minutes	"Turn To" Block
2	Missions 1-2	10 minutes	N Curve Again M Curve Again
3	Go through Dangerous Curves 2 Part 2 slideshow	5 minutes	Smooth turns
4	Mission 3-4	15 minutes	U Turn S Turn
5	Go through Dangerous Curves 2 Part 3 slideshow	5 minutes	Drawing trails
6	Missions 5-6	15 minutes	Freeform Art Star Trail
7	Lesson Summary	5 minutes	Class Conclusion Questions



Tips

Go through the presentation (and teacher's notes), missions, and solutions prior to the session.

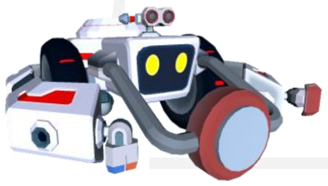
Review the available resources suggested above.

This lesson probably will take more than 45 minutes. Don't rush things, take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.

If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.

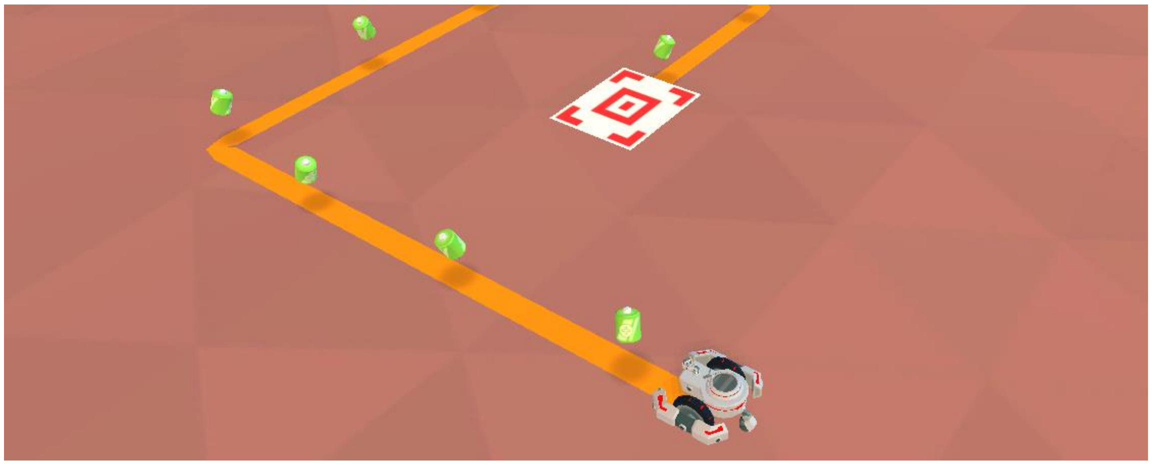
Class Conclusion Q&A

	Questions	Answers
1	What is the benefit of using the "turn to" block?	The "turn to" block enables us to program accurate screw turns easily: It uses a sophisticated system control algorithm in order to turn accurately while the user is required only to enter the desired turn angle.
2	What is a smooth turn?	A smooth turn is a gradual turn in which the robot's wheels both turn in the same direction at different speeds. Smooth turns differ from each other by the turning radius.
3	If the wheel speed is 80% for the left wheel and 60% for the right wheel, in which direction will Ruby turn?	Ruby will turn right.



Lesson 6: Doodling with Distance

Draw and doodle using 3 trail drawers.



Learning Goals

By the end of this lesson students will:

- Applying what was learned so far and carrying out creative challenges
- Using three trail drawers
- Fun and creativity

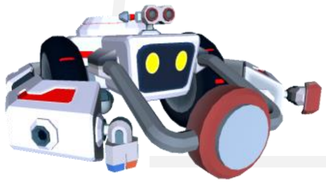
Description

The goal of Lesson 6 is to practice the skills learned in the first 5 sessions while drawing interesting shapes. Ruby will be drawing and doodling using the trail drawers.

The students will practice:

> Driving back and forth
> Turning and curving
> Using the gyro sensor and the trail drawers (left, right, bottom)
> Using Explore Mode to measure distances and angles

The 3 trail drawers enable the students to be creative and enjoy generating interesting patterns and drawings.



Resources

Slideshow - within the pack

Teacher's Notes - Doodling with Distance

Trail blocks – Knowledge Base article

Pacing Guidelines

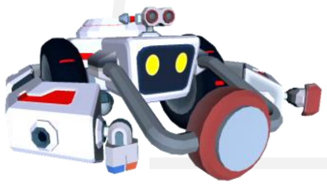
Step	Description	Timing	Links/comments
1	Go through Dangerous Curves 2 Part 1 slideshow	5 minutes	Trail drawers
2	Missions 1-6	40 minutes	Going the Distance Double Distance Drive Snake! Tracing Letters: G Ruby's Emoji Fly Your Own Flag

Tips

Go through the presentation (and teacher's notes), missions, and solutions prior to the session.

Review the available resources suggested above.

If your students complete the missions before the class ends you can ask them to present their drawings to the class or let them draw free form art. Alternatively, you can proceed to the next session



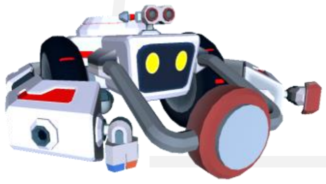
Lesson 7: Touch, Avoid, Repeat

Using a touch sensor and distance sensor while performing repetitive operations.

Learning Goals

By the end of this lesson students will:

- Robots and their sensors
- Object avoidance and recognition
- The Touch Sensor
- The Ultrasonic Sensor
- Repeat Loops
- Boolean and integers data types



Description

In Lesson 7, students will learn about the **Touch and Ultrasonic sensors** as well as **repeat loop** programming. After understanding closed-loop control systems, the students will understand the function of sensors in controlling robots - the sensors supply the feedback information in the closed loop control system.

The **Touch sensor** helps Ruby perform missions as well as navigate when there are good containers around. For example, Ruby will drive until the touch sensor is pressed and then perform a turn. The touch sensor returns a Boolean data type, i.e. can have only two possible values: true (when pressed) or false (when released).

The **Ultrasonic sensor** helps Ruby avoid obstacles. By measuring the time it takes for an ultrasonic (sound) beam to reach an object and return, and knowing the speed of sound, the sensor calculates the distance from Ruby to the obstacle. The ultrasonic sensor returns the distance from the obstacle in front of it in centimeters. By reading the distance Ruby decides how to navigate.

As software becomes more and more sophisticated, there is a requirement to write a clear and readable code. Repeat loops helps us to do so; whenever there is a scenario that Ruby needs to do several times, we can use the **repeat loop** to program Ruby to do it several times, instead of writing program code for each occurrence.

Resources

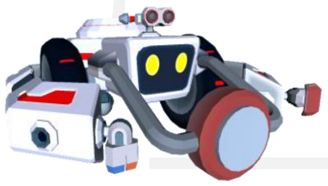
Slideshows - within the pack

Teacher's Notes - Touch, Avoid, Repeat

Repeat Loop – Knowledge Base article

Touch Sensor – Knowledge Base article

Ultrasonic Sensor – Knowledge Base article



Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Touch, Avoid, Repeat Part 1 slideshow	10 minutes	Touch Sensor "Repeat" block
2	Missions 1-3	15 minutes	Green Containers Green on the Left Repeating Green
3	Go through Touch, Avoid, Repeat Part 2 slideshow	5 minutes	The steeper the slope - the more force or speed we need in order to climb it. The same is for heavier mass.
4	Complete missions 4-5	10 minutes	Green Between the Reds Forest of Containers (bonus)
5	Lesson Summary	5 minutes	Class Conclusion Questions

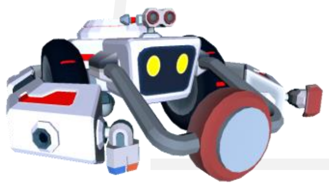
Tips

Go through the presentation (and teacher's notes), missions, and solutions prior to the session.

Review the available resources suggested above.

This lesson probably will take more than 45 minutes. Don't rush things, take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.

If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.



Class Conclusion Q&A

	Questions	Answers
1	When did we use the touch sensor?	We used the touch sensor when we could use touching obstacles for navigation – for example, with the good containers in the first 3 missions.
2	When did we use an ultrasonic sensor?	We used the ultrasonic sensor when we were required to avoid obstacles or navigate by obstacles locations. An ultrasonic sensor is much more efficient than touch sensor- it recognizes the obstacles before touching! In fact, we can use the distance sensor as a touch sensor. When doing so, we just have to remember that the ultrasonic sensor is positioned between the robot's arms and does not protrude out in the front like the touch sensor does. This means that Ruby is touching the object the ultrasonic sensor shows a small positive value of distance, rather than zero.
3	What is the advantage of using the "repeat" block?	Repeat blocks enable us to write simple and short code when we need to perform operations that occur more than one time.

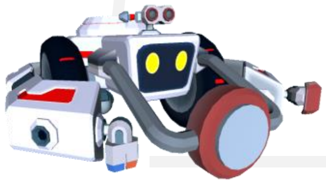
Lesson 8: Random Obstacles Ahead

Use of variables in a non-deterministic environment

Learning Goals

By the end of this lesson students will:

- Variables of integer and double data type
- Mathematical operators: +, -, x, remainder of (modulo)
- Performing missions in non-deterministic (random) environments
- Spatial cognition – identifying the start of/end of obstacles
- Making decisions in a non-deterministic environment



Description

Lesson 8 presents **variables** while practicing using them for scanning the environment and making decisions.

Variables are used to store information which we need to use later or several times in our program. There are different types of variables, which define their logical representation and size. In this lesson we will learn about the **integer** and **double data** type.

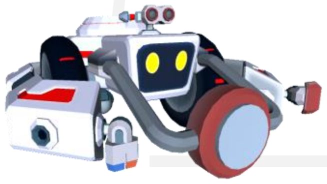
Applying **mathematical operations** on variables enables us to use variables for different missions which require mathematical calculations, such as obstacle counting, identifying objects according to their relative locations, etc.

Integer (int) data type variables store a whole positive or negative number, or a number with no fractional parts. Integers can be added, subtracted, and multiplied. Care must be taken when they are divided, as the division of two integers is not necessarily another integer.

Double data type variables store a positive or negative number with fractional parts. Therefore, it uses more computer memory, as implied by its name. The ultrasonic sensor measures distance and returns a decimal number with a decimal separator (fractional value). Therefore, the distance variable must be stored in a “double” data type variable. The gyro sensor measures angles accurately, so it also returns angle values in a double data type. The “turn to” block can turn to fractional numbers of angles as well as negative angles and can accept both integers and double data types.

Ruby will use variables to program driving distance and differentiate between objects in her surroundings using her front or side ultrasonic sensors. Furthermore, Ruby will make decisions dependent on an object’s location or number. For example, we can perform a different action on objects located every third position.

In order to identify numbers, or in our case obstacle positions, we use division and fractions. The “**remainder of**” operation returns the remainder after division. If a number is a multiple of its divisor, the remainder of operation returns 0: the remainder of 4 divided by 2 equals 0 (4 ÷ 2 = 2, no remainder).



Another example: the remainder of 8 divided by 3 is 2 (because 3 can fit into 8 twice whole – 6, with a remainder of 2). 5 divided by 4 returns 1 with a remainder of 1, therefore the remainder of a block will return a value of 1.

The remainder of an operation has many uses, such as distinguishing between odd and even numbers: if the remainder of a number divided by 2 returns 0, it is an even number; if the remainder of a number divided by 2 returns 1, it is an odd number. We will use this method to differentiate between obstacles as odd or even.

In Lesson 8 the objects' locations are not always deterministic. Objects will sometimes appear in **random locations** (as they do in real life). Autonomous systems need to accurately interpret their surroundings to make well-informed decisions about objects that are constantly changing. Using a non-deterministic environment will encourage the students to use Ruby's sensors wisely and use advanced programming.

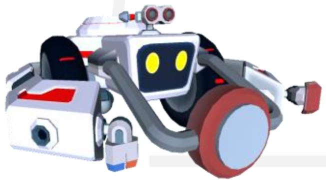
Resources

Slideshows - within the pack

Teacher's Notes - Random Obstacles Ahead

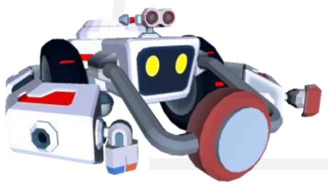
Variables – Knowledge Base articles

Various data blocks – Knowledge Base articles



Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Random Obstacles Ahead Part 1 slideshow	10 minutes	Variables- Kind of variables, Using variables Mathematic operators
2	Missions 1-2	10 minutes	Growing Distance Two Times The Distance
3	Go through Random Obstacles Ahead Part 2 slideshow	10 minutes	Random events Counting obstacles
4	Missions 3-4	10 minutes	Third is Random Random Corner
5	Mid-session Summary	5 minutes	Class Conclusion Questions 1-4
6	Go through Random Obstacles Ahead Part 3 slideshow	10 minutes	Counting a circle of obstacles
7	Missions 5-6	10 minutes	The Fifth Obstacle Obstacle Counter
8	Go through Random Obstacles Ahead Part 4 slideshow	5 minutes	"Remainder of" block
9	Missions 7-8	15 minutes	Odd or Even Third is Good
10	Lesson Summary	5 minutes	Class Conclusion Questions 5-7



Tips

Go through the presentation (and teacher's notes), missions, and solutions prior to the session.

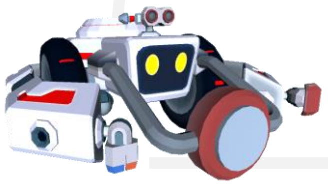
Review the available resources suggested above.

This lesson is long and will probably take 2 hours. We suggest splitting it into two sessions and performing a mid-class question or discussion at the end of the first session.

If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.

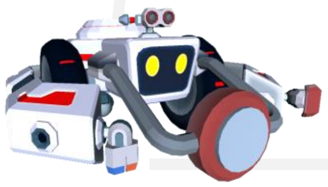
Class Conclusion Q&A

	Questions	Answers
1	When should we use variables?	We need variables whenever we need to store data or information that will be later used, referenced, or manipulated.
2	Describe 3 types of variables you know.	**Integer** - a whole positive or negative number, a number with no fractional parts. Examples: 1, 2, -5, 341, 1,002. **Boolean** - has one of two possible values: true or false. **String** - a sequence or string of connected characters, surrounded by quotation marks. This type of variable has no numerical value and it cannot be used for mathematical calculations even if it contains number characters; it is only text. Examples: "Hello world", "this is a string of characters", "fourteen" (stress that this last example is a word and not the number 14).



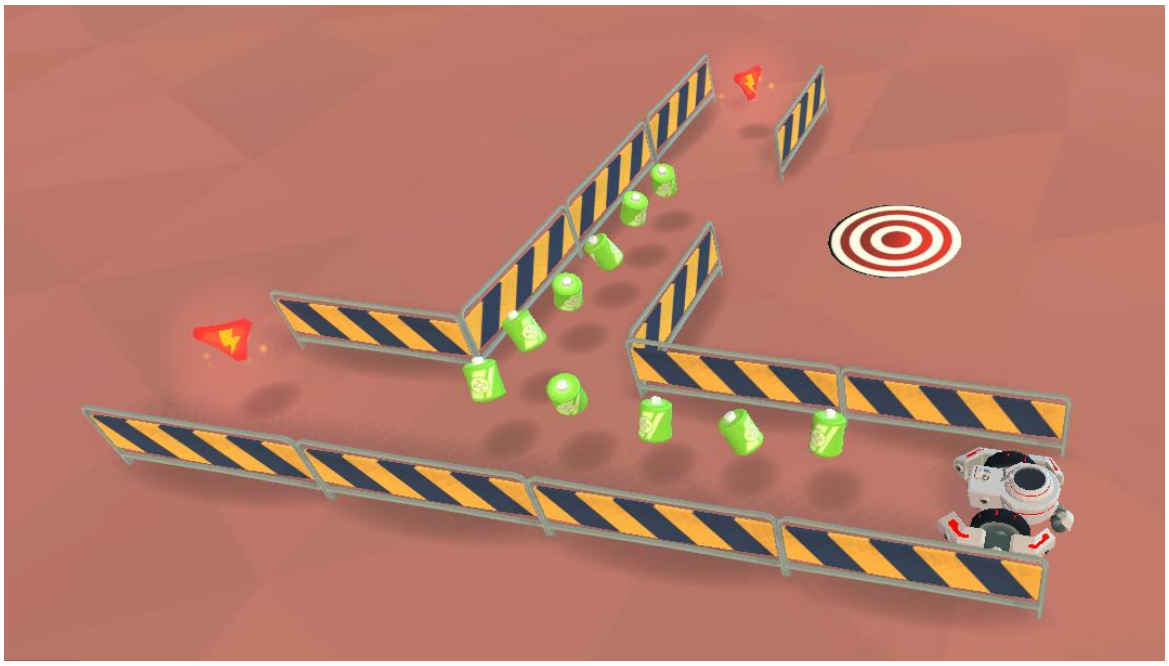
Class Conclusion Q&A

	Questions	Answers
3	What is a random event? Give an example of a random event.	A random event is an event that has some scenarios that do not appear exactly in constant order. Therefore, it is impossible to predict exactly the next immediate scenario. The opposite of random is deterministic, which means that scenario order or values are predictable exactly. The toss of a coin, the throw of dice, and lottery draws, as well as our obstacle locations, are all examples of random events.
4	Using the distance sensor, how do we detect that we passed an obstacle?	We detect first the beginning of the obstacle (distance is smaller than…), afterwards we detect passing of the obstacle (distance is greater than…)
5	How can we tell if a number is odd or even by using the “remainder of” block?	With the help of the “remainder of” calculation we can tell if one number is a multiple of the other. If a number is a multiple of 2 then the remainder of its division by 2 equals 0. By comparing the remainder of the division to 0 we can perform different action on even and odd numbers.
6	How can we tell if a number is a multiple of 3 by using the “remainder of” block?	We can determine if a number is divisible by 3 by comparing the remainder of its division by 3 to zero.
7	As we turned from the center of the circle towards an object, how did we know to drive back to the midpoint?	As we turned from the center of the circle towards the obstacle, we knew how to drive back to the midpoint by measuring the distance from the obstacle and saving it as a variable before we drive towards it, then using the variable to drive back the same distance we drove forward.



Lesson 9: Radar Missions

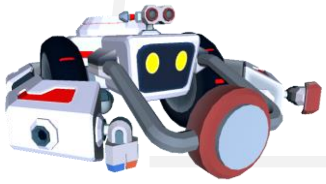
Rotating Top Ultrasonic Sensor for Scanning



Learning Goals

By the end of this lesson students will:

- Meet Ruby’s sensor ports
- Adjustment of port joints for efficient performance
- Using the top ultrasonic sensor
- Scanning the environment while keeping driving forward

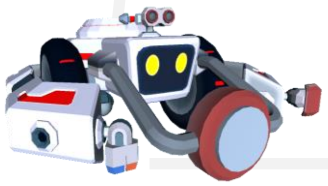


Description

Lesson 9 introduces Ruby's **top ultrasonic sensor**. The top ultrasonic sensor is built with several **joints**, quite like a human's arm. Different joints move in different ways - sideways or up and down. In Lesson 9, we will use joint number 0 to rotate sideways and scan the surroundings.

In Lesson 8, Ruby scanned her surroundings by using her front ultrasonic sensor which pointed forward, so scanning the environment forces her to stop driving or turn around in place. Lesson 9 shows the configurability option of the sensors' location so Ruby can use her top ultrasonic sensor as a radar to scan her environment while she keeps moving.

Using the top ultrasonic sensor and programming the position to which it is pointing, will enable us to identify turns even if they are in a random location or random direction. Ruby will drive forward while scanning around her sides to identify corners and the right location to turn. In Lesson 9 there will always be one possible turn, but its direction can be random.



Resources

Slideshows - within the pack

Teacher's Notes - Radar Missions

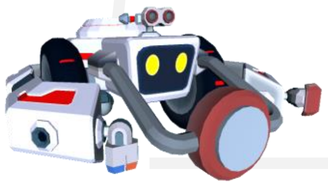
Ultrasonic Sensor – Knowledge Base article

Set Sensor Rotation block – Knowledge Base article

Get Sensor Rotation Block – Knowledge Base article

Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Radar Missions Part 1 slideshow	10 minutes	Top Ultrasonic sensor joints
2	Missions 1-3	30 minutes	Pre-Radar Set and Turn Out in The Open
3	Go through Radar Missions Part 2 slideshow	10 minutes	Constantly scanning for random turns
4	Complete missions 4-6	30 minutes	Set and Turn and Set and Turn Scan Scan Some More
5	Lesson Summary	5 minutes	Class Conclusion Questions



Tips

Go through the presentation (and teacher's notes), missions, and solutions prior to the session.

Review the available resources suggested above.

This lesson probably will take more than 45 minutes. Don't rush things, take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.

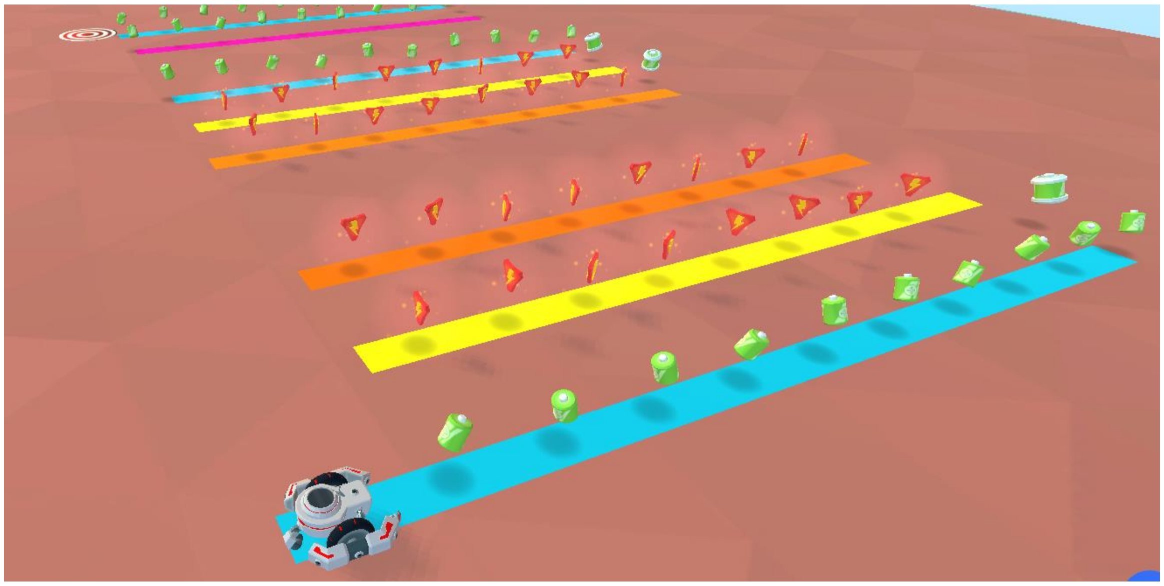
If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.

Class Conclusion Q&A

	Questions	Answers
1	What are the advantages of using the top ultrasonic joints for scanning?	Using the top ultrasonic joints allows Ruby to scan the environment without interrupting her current operation (such as driving).
2	How can we use the top ultrasonic sensor to turn right without hitting the wall?	We will adjust the top ultrasonic sensor to 50 degrees backwards so Ruby can safely turn right without hitting the fence.
3	How can we turn safely when turn direction is random (left or right)	We will turn the sensor left and right regularly until turn is detected. Upon detection we will determine the turn direction.

Lesson 10: Colorful Code

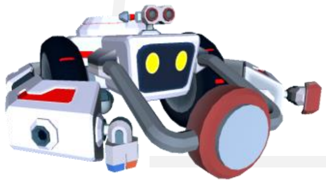
A: Color Sensor and Logic Operators: AND, OR, NOT.



Learning Goals

By the end of this lesson students will:

- The color sensor
- String data type
- Logic operators: NOT, AND, OR
- Error correction



Description

The goal for Lesson 10 is to recall the **color sensor** and use it to teach **logical operators: NOT, AND, OR.**

Ruby's color sensor has two functions:

1. Color Detection
2. Reflected Light Measurement

In Lesson 10 we will use the **Color Detection mode**. Color detection works by shining a white light at an object (actually, three beams of light - red, green, blue - RGB) and then decoding the reflected color. The color sensor returns 9 possible **string** data type values – Black, White, Red, Green, Cyan, Blue, Magenta, Yellow, and Brown (Note: the values are capitalized). We can compare the result to the desired color name and make decisions accordingly.

Lesson 10 also teaches **error correction**. Comparison to a desired color is sometimes essential but not enough. In previous sessions we have learned that Ruby cannot stop immediately and there is a stop distance ("drifting") that is dependent on driving speed. Therefore, when we want to be accurate and stop exactly at the end of a line drawn on the floor, we need to do some corrections and drive backwards. The requirement for correction is critical as driving speed grows.

Some missions require checking if two conditions occur simultaneously. For that we will use the **AND** operator which returns TRUE only if both conditions are met. Other missions require checking two conditions and acting if at least one of the conditions is met. For that we will use the **OR** operator which returns TRUE if at least one condition is met. When there is a requirement that a condition does not occur, we will use the **NOT** operator. Below you can find truth tables of AND, OR, NOT logic operators:

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

AND

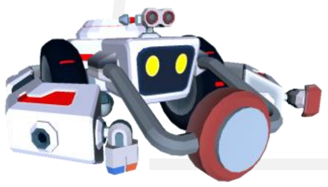
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

OR

A	Q
0	1
1	0

NOT

The NOT operator can be used for toggling value of Boolean variable: not(true) = false, not(false) = true. We will use it to differentiate between odd or even rows: not(odd) = even.

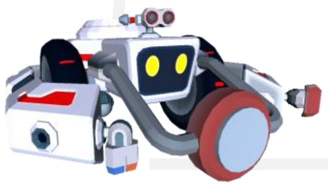


Resources

- Slideshows - within the pack
- Teacher's Notes - Colorful Code
- Color Sensor – Knowledge Base article
- Various data blocks – Knowledge Base articles
- Video about Boolean operators help searching with google: https://www.youtube.com/watch?v=bCAULDuMcso

Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Colorful Code Part 1 slideshow	10 minutes	Color sensor String Data type Error Correction
2	Missions 1-2	15 minutes	Three Yellow Spots Color Line Correction
3	Go through Colorful Code Part 2 slideshow	5+10 minutes	Operators: NOT Introducing driving along several rows
4	Mission 3-5	20 minutes	Yellow Brick Roads Roads and Operators Weightboxes on Green Lines
5	Go through Colorful Code Part 3 slideshow	5 minutes	Operators: AND, OR
6	Missions 6-7	15 minutes	Amazing Colors A Celebration of Colors
7	Missions 8-9 (Bonus)	10 minutes	Colored Circles 1 Colored Circles 2
8	Lesson Summary	5 minutes	Class Conclusion Questions

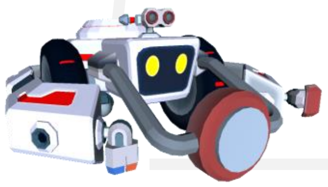


Tips

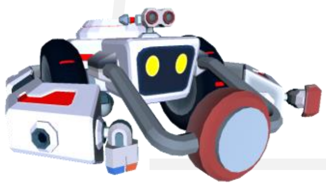
- Go through the presentation (and teacher's notes), missions, and solutions prior to the session.
- Review the available resources suggested above.
- This lesson probably will take more than 45 minutes. Don't rush things, take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.
- If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.

Class Conclusion Q&A

	Questions	Answers
1	What kind of data does the color sensor return when working in color detection mode?	String data type (text inside quotation marks)
2	When we wanted to travel on a colored line, we programmed Ruby to stop when detecting the end of the line and then drive back. This is called "error correction". Can you explain what was the error and how did we correct it?	The error is the fact that even though Ruby stopped power to her motors when her color sensor no longer sees the colored line, she continued to move forward a little bit and off the line because of her high speed. We corrected by driving backwards at low speed until Ruby's color sensor sees the colored line on the floor again.



3	How can we use the logic operator “NOT” to distinguish between odd or even lines?	By declaring a boolean type variable set to TRUE for the odd line at the start of the program, and then at the end of the loop using the NOT block to set the variable’s value to NOT ITSELF. This effectively toggles the variable between true and false, and so we can use TRUE as odd (1, 3, 5) and FALSE as even (2, 4, 6).
4	We have learned to distinguish between odd or even numbers using a mathematical operation (“Random Obstacles Ahead” pack). Do you remember how?	In “Random Obstacles Ahead” we used the “remainder of” block to tell if a number is odd or even. If a number is a multiple of 2 then the remainder of its division by 2 equals 0, it is even. If not, it is odd.
5	We want to do something only if 2 conditions are met at the same time. Which operator will we use?	The AND operator.
6	We want to do something if at least one of two conditions is met. Which operator will we use?	The OR operator.
7	We want to do something only if a certain condition is not met. Which operator will we use?	The NOT operator.



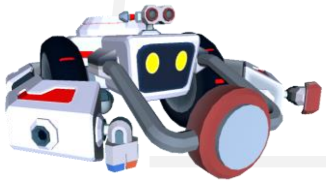
Lesson 11: Repeat Again

Nested Loops and Using X-Gyro to Identify Plain/Uphill/Downhill

Learning Goals

By the end of this lesson students will:

- Nested Repeat loops
- X-axis gyro
- Applying what was learned so far and carrying out challenges and missions



Description

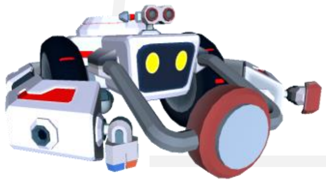
Lesson 11 teaches students to use nested loops. Nested loops, like “simple loops”, help us to write clear and short code.

Nested loops are used whenever there is a scenario that involves performing an action several times in succession, and each action contains several inner repeated actions.

We can understand using nested loop by looking on the calendar: the outer loop can be considered as the month and the inner loop can be considered the days of that month. When a month starts, counting the days restarts and continues until all the month’s days are counted. When day counting ends, we go back to the outer loop and change the month.

In order to complete missions, Ruby is required to drive uphill or downhill and perform missions on different plains. Ruby can identify driving uphill/downhill driving by using data from the x-axis gyro: the gyro will return positive values when driving uphill and negative values when driving downhill. Driving on a plain will show 0 on the x-axis. Ruby can turn her sensors using the “**get sensor rotation**” block we introduced in pack 9 “Radar Missions” so she can notice obstacles ahead, for example.

As missions become more complicated and the students’ knowledge increases, there are (as in real life) many ways to solve a problem or complete a mission. The students can use whatever they like - sensors, signs, objects in the environment, etc. in order to complete the mission. The teacher’s solution suggests one option of many and is not necessarily the only option.



Resources

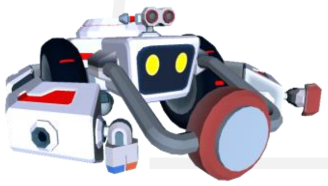
Slideshows - within the pack

Teacher's Notes - Repeat Again

Get Gyro Axis block - Knowledge Base article (note x-axis)

Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Repeat Again Part 1 slideshow	10 minutes	Nested loops
2	Missions 1-2	15 minutes	Hilltop Trees Trees on Hills
3	Go through Repeat Again Part 2 slideshow	10 minutes	X-Gyro
4	Missions 3-4	20 minutes	Downhill Drive Through Downhill Drive Two
5	Go through Repeat Again Part 3 slideshow	10 minutes	Multiple nested loops Color-based decision making
6	Missions 5-6	15 minutes	Hilltop Parking Hilltop Mayhem
7	Lesson Summary	5 minutes	Class Conclusion Questions



Tips

Go through the presentation (and teacher's notes), missions, and solutions prior to the session.

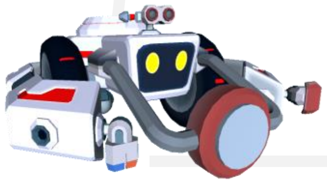
Review the available resources suggested above.

This lesson probably will take more than 45 minutes. Don't rush things, take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.

If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.

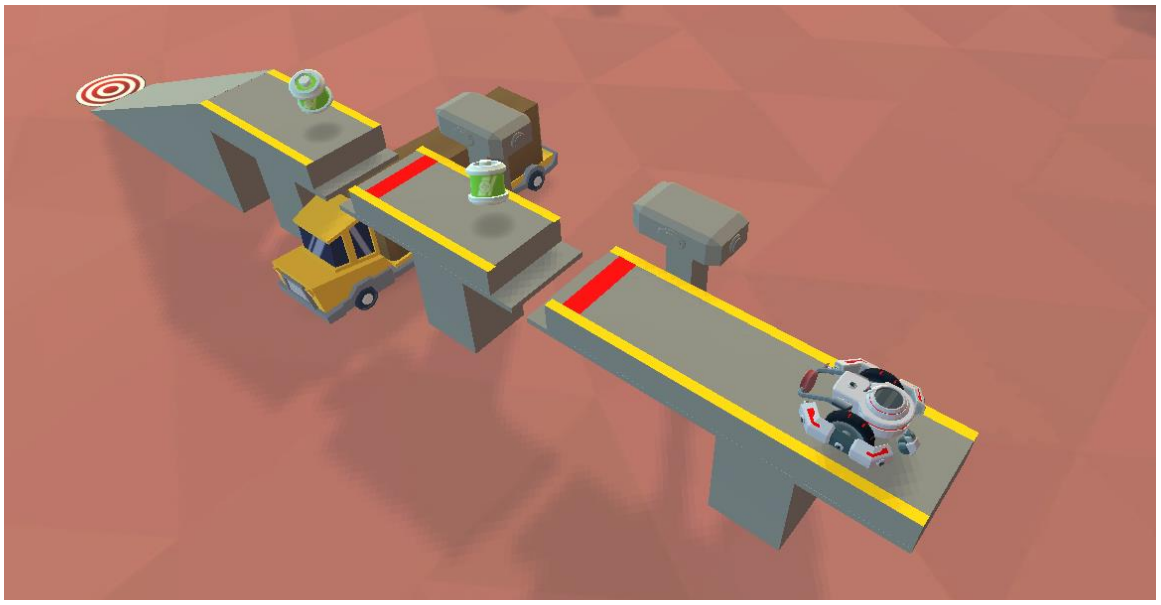
Class Conclusion Q&A

	Questions	Answers
1	What are nested loops and when are they used?	Nested loops are loops within loops. For example an outer repeat loop might repeat for 3 times, while the inner (nested) loop holds code that drives the robot in a square shape by repeating a 'drive forward, turn 90 degrees right' sequence 4 times. The result will be the robot driving in a square shape 3 times.
2	Describe how the gyro sensor is used to detect driving uphill.	The gyro sensor's x-axis shows positive degrees (above 0) when Ruby is driving uphill. By using a Get Gyro Axis block set to the x axis and checking if its value is greater than 0, we can be sure we are driving uphill.



Lesson 12: Magnetic Manipulation

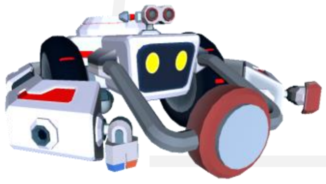
Meet Ruby's Magnetic ARM



Learning Goals

By the end of this lesson students will:

- Meeting Ruby’s magnetic arm
- Applying what was learned so far and carrying out challenges and missions



Resources

Slideshows - within the pack

Teacher's Notes - Magnetic Manipulation

Magnet Arm Position Block - Knowledge Base article

Magnet Arm Cargo Connected Block - Knowledge Base article

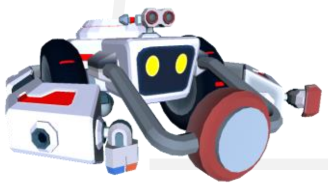
Get Magnet Arm Position Block - Knowledge Base article

Magnet Arm Release Block - Knowledge Base article

Description

Lesson 12 introduces Ruby's magnetic arm. Ruby can raise and lower the arm, hold magnetic objects, and release them. The arm even has a sensor that detects if an item is magnetized to it.

In this session Ruby will use her arm to rearrange objects in her surroundings: move cargo out of the way, use cargo to fill in pits and gaps in the road, or even press/release buttons which activate rising bridges. To succeed in the mission, the students will have to use all they have learned so far: driving skills, all kind of sensors (touch, color, distance or gyro), use variables, loops, math, and logic operators etc.



Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Magnetic Manipulation Part 1 slideshow	10 minutes	Meet Ruby's Arm
2	Missions 1-3	20 minutes	Meet the Magnet Arm Wrestling Ruby Pit stop
3	Go through Magnetic Manipulation Part 2 slideshow	5 minutes	Introduction to missions 4-5 X-Axis Gyro, color-based decisions
4	Missions 4-5	15 minutes	A Block Too Far Blocking Up the Bridge
5	Go through Magnetic Manipulation Part 3 slideshow	10 minutes	Introducing mission 6-7 Repeat loops, variables, operators, color-based decisions
6	Missions 6-7	20 minutes	The Buttonless Pit Magnet Slide
7	Lesson Summary	5 minutes	Class Conclusion Questions

Tips

Go through the presentation (and teacher's notes), missions, and solutions prior to the session.

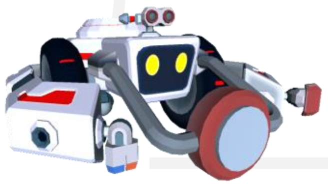
Review the available resources suggested above.

This lesson probably will take more than 45 minutes. Don't rush things, take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.

If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.

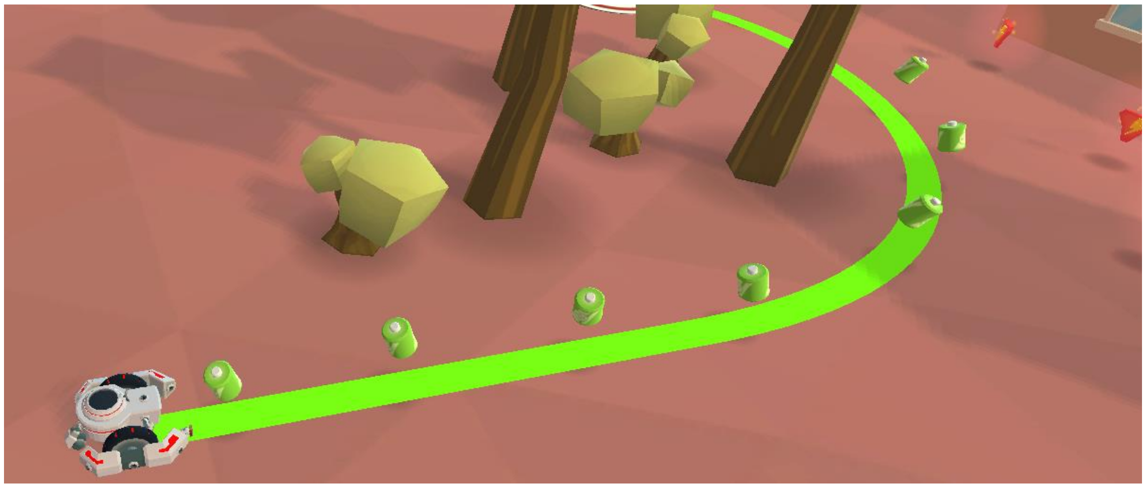
Class Conclusion Q&A

	Questions	Answers
1	After raising or lowering Ruby's magnetic arm with a "Magnet arm position block", why should we use a "wait until" block with the "Get magnet arm position" block and compare it to "up" / "down"?	It is good practice to wait until a physical action (such as raising/lowering a mechanical arm in this case) has completed before driving. This will ensure that Ruby's arm and her magnetic load do not bump into objects, which might happen if she drives while still raising or lowering her arm.
2	In the mission Magnet Slide, does it matter if we turn to the button on the right or on the left first? And accordingly – should we declare our variable to be 90 or -90 degrees at the start of our code?	No, as long as we start with one and finish with the other.
3	How did we switch direction after the first iteration of the loop?	By using a mathematical operator to set the direction variable to itself in the opposite direction – by multiplying it by -1. So, if we declared the variable's value to be 90 degrees at the start of the code, then at the end of the loop we will toggle the value of the variable to -90. If we declared the value to be -90 degrees at the start, this action will toggle the value to +90.



Lesson 13: Line Following Logic

Using the color sensor and proportional control to follow a line



Learning Goals

By the end of this lesson students will:

- Line following logic
- On-off control / Two-state control
- Three-state control
- Proportional control
- The color sensor - reflection mode

Description

The goal of Lesson 13 is to teach on-off control and proportional control. These two methods are used for line following.

Lines marked on the ground help Ruby navigate. We have already used colored lines for navigation in previous lessons – according to these marks, Ruby knew when to stop or turn. In Lesson 13 Ruby will use such marks to accurately drive along routes that consist of straight lines, smooth curves, and sharp turns. Ruby will drive on the **edge of the colored line** so she can differentiate between the line and background (the floor) colors or light reflection.

Description

At first, Ruby will follow the line using two-state control: she will drive slightly to the left and slightly to the right, using her color sensor and correcting herself continuously. This method will make Ruby drive slowly and in a funny (zigzag) but accurate way.

Using the color sensor in **reflection mode** will enable Ruby to define more states, as reflection mode returns numerical values with a range of 0-100 values. That range enables Ruby to react differently to different values. Ruby will calculate error values, i.e. the distance from a desired value (threshold) and react accordingly.

Threshold is calculated as the mean of line and background reflection values. Error is the reflection value minus the threshold value. If we use the left-side color sensor and drive along the left edge of a white line, negative error values will indicate that Ruby drove too far to the left (onto the floor) and positive error values will indicate that Ruby drove too far right (onto the white line). The correction will be **proportional** to the error - the correction will be bigger as the error increases and vice versa.

Proportional control will improve the way Ruby drives along the line - it will be much smoother. Still, Ruby cannot drive fast on different shapes of roads. To make driving efficient, Ruby will be assisted by colored marks that will signal if the road is smooth or has sharp turns ahead.

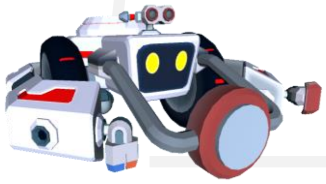
Resources

Slideshows - within the pack

Teacher's Notes – Line Following Logic

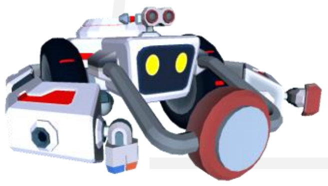
Color Sensor - Knowledge Base article

Get Color Name/Reflection Value Block - Knowledge Base article



Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through Line Following Logic Part 1 slideshow	10 minutes	Using color sensor to identify the edge of a line On-off control / two-state control
2	Missions 1-3	10 minutes	Going Green Green Again Question Marked Code
3	Go through Line Following Logic Part 2 slideshow	5 minutes	Color sensor- reflection mode
4	Mission 4	15 minutes	Following White
5	Go through Line Following Logic Part 3 slideshow	5 minutes	Three-state control
6	Mission 5	15 minutes	Go to the Light
7	Go through Line Following Logic Part 4 slideshow		Continuous proportional control
8	Mission 6		Ruby on the Road
9	Go through Line Following Logic Part 5 slideshow		Color sensor colored marks along the road
10	Missions 7-10		Sharp Turn Situation Traffic Corner Diagonal Drive Traffic Circle Trial
11	Lesson Summary	5 minutes	Class Conclusion Questions

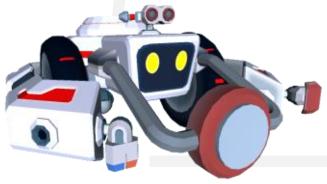


Tips

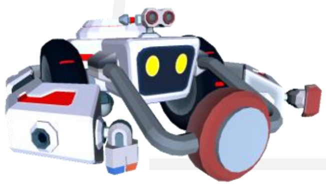
- Go through the presentation (and teacher's notes), missions, and solutions prior to the session.
- Review the available resources suggested above.
- This lesson probably will take more than 45 minutes. Don't rush things, take your time. If students do not complete all the missions required, either allow them to complete it from home or reserve some time at the beginning of the next session.
- If your students complete the missions before the class ends you can ask them to present their solutions to the class and discuss the efficiency and compatibility of their solution. Alternatively, you can proceed to the next session.

Class Conclusion Q&A

	Questions	Answers
1	Describe what is two-state control and how we used it with Ruby in the first few missions.	Two-state control is a method in which the robot drives in a zigzag way along the edge of a line, riding the edge between the colored line and the floor. The two states are the correct color (the color of the line, like green or white) or the wrong color (the floor). Ruby does this by checking if the color her sensor sees is above or below a threshold value, which is the mean number between the color of the line and the color of the floor's respective reflection values.
2	Explain why we followed the **edge of the line** and not the middle of the line, for example. Does the position of the sensor matter? (Left / right / center color sensor)	If Ruby has left the line behind completely, or is completely on the line itself so that both her sensors see **only the floor color**, then Ruby can't know if she is on the floor to the left or the line or the right of the line. She will not know in which direction to turn in order to correct course and stay along the line.

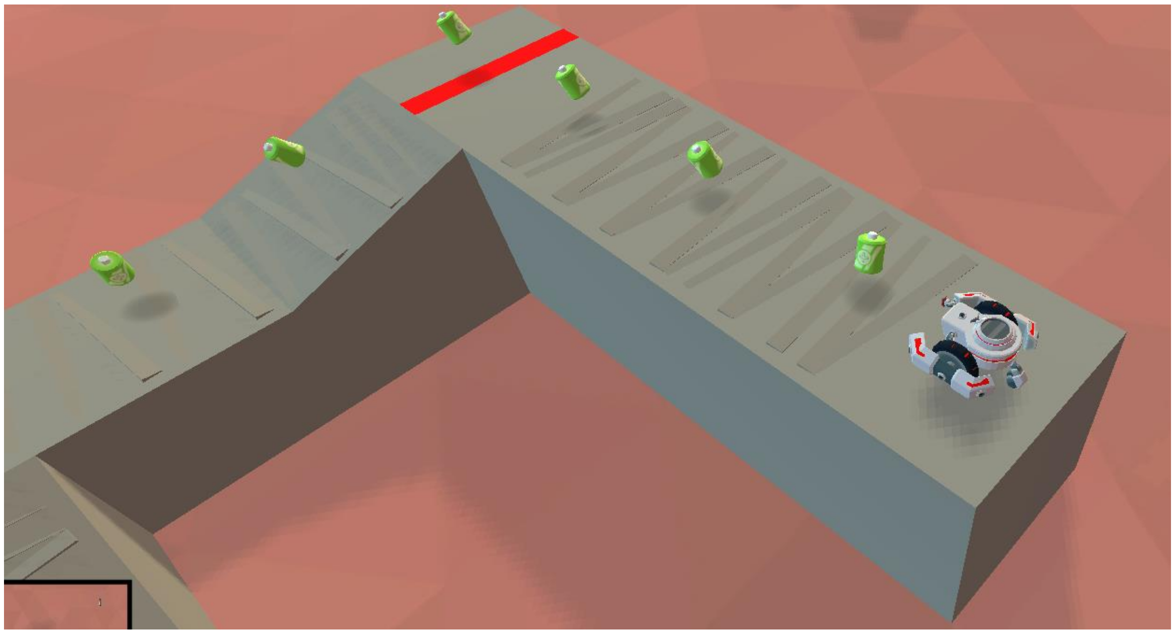


3	What are the advantages of using the color sensor in the reflection mode over color detection?	For this reason, it is better to focus on one of the edges of the line, and according to which edge, we will use the readings from the respective side sensor: if we follow the left edge of the line – we will use the left side color sensor. If we follow the right edge of the line – right side color sensor. This way, the very moment Ruby leaves the edge of the colored line and moves onto the floor, she will correct herself by driving slightly in the opposite direction.
4	Explain the benefits of continuous, proportional control over a small number of states control (two-state or three-state control).	The **color name** mode allows for only 9 color values, as string values ("Black" , "White" , "Red" , "Yellow" , "Green" , "Cyan" , "Blue" , "Magenta" , "Brown"). This limited number of options also limits the code. By using **reflection value** mode, we can use 100 different numerical reflection values and thus achieve much greater precision, as numerical values can also be manipulated and used in mathematical equations (while string values cannot). This is ideal for all types of control, and especially for proportional control.
5	Describe what the colors on the floor marked for Ruby in the last few missions, and why does Ruby need these signs.	Proportional control is better because it is a closed-loop system which a very high degree of control and precision. If we program it correctly, the correction will always be as small or as big as the error is – meaning, that usually the code will 'catch' the error sooner rather than later, and thus the error will be small, and the correction also small. This will make for a much cleaner, smoother drive for Ruby, and in real-life engineering, will also conserve energy.



Lesson 14: A Hard Block Life

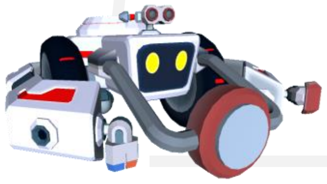
Using encoder, gyro and reflection values for aligning according to marks on the floor



Learning Goals

By the end of this lesson students will:

- Encoders and their utility for distance measurement
- Recalling the Y-axis Gyro
- Aligning to a line using two color/light sensors and proportional control
- Proportional gain

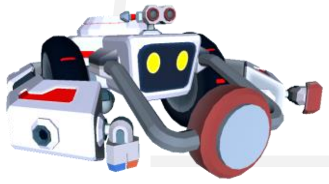


Description

Lesson 14 introduces the encoders and how to use them for measuring distance. An encoder is an electromechanical device that converts the angular position of the motor axle to an electrical signal that is used for speed and/or position control. Ruby will use the encoder to remember positions in her environment. Resetting the encoder marks a position as a reference point. If Ruby drives forward and wants to return to this reference point, she should drive back until the encoder shows zero.

In Lesson 14, Ruby needs to drive on narrow, bumpy roads. This can be done successfully only if Ruby stays aligned straight ahead. Using the Y-axis gyro and proportional control will enable Ruby stay aligned: using the Y-axis gyro Ruby will measure the deviation from driving straight ahead (deviation from Y=0) and using proportional control in the code will straighten Ruby's driving trajectory. The correction will be proportional to the error (deviation).

The last subject that Lesson 14 introduces is aligning to a straight line using 2 color sensors in reflection mode. As we saw in previous lessons, marks on the ground can help us navigate. This time, straight lines will help us turn accurately. Ruby will drive until she detects a line in one of her side color sensors (left or right). When the line is detected, Ruby will use proportional control to align herself to the line: she will turn slowly towards the direction of the sensor which detected the line until the light reflection value of both sensors is equal. That means the alignment is completed. It is important that the students understand the point is not necessarily both color sensors are aligned to the white line - the point is that both sensors detect the same value. It will probably be the edge of the line, so a mean of line and background reflection values. Instead of checking if both sensors sense a specific numeric value (such as 67), we will check if the values of both sensors are equal.

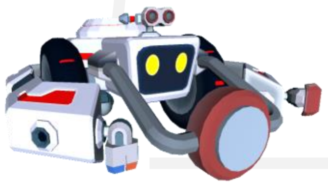


Resources

Slideshows - within the pack
Teacher's Notes – A Hard Block Life
The Gyro sensor - Knowledge Base article
Get Gyro Y block - Knowledge Base article
Color Sensor - Knowledge Base article
Get Color Name/Reflection Value Block - Knowledge Base article

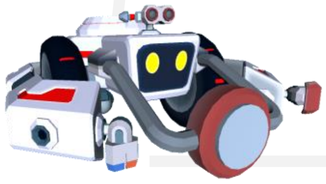
Pacing Guidelines

Step	Description	Timing	Links/comments
1	Go through A Hard Block Life Part 1 slideshow	10 minutes	Using encoders to measure exact movement and positions
2	Missions 1-2	10 minutes	Decoding Encoders Encoded Stroll
3	Go through A Hard Block Life Part 2 slideshow	15 minutes	Y-axis gyro Proportional control and gain
4	Missions 3-5	15 minutes	Rocky Road Ruby and the Treacherous Heights Ruby the Roving Robot
5	Go through A Hard Block Life Part 3 slideshow	15 minutes	Aligning to colored lines Comparing reflection values Logic operators
6	Missions 6-7	15 minutes	Adjusting Left Reflection Redirection
7	Go through A Hard Block Life Part 4 slideshow	15 minutes	Putting it all together
8	Missions 8-10	20 minutes	Downhill Tumble Slope Slide Correction Narrow Maneuvers
9	Lesson Summary	10 minutes	Class Conclusion Questions



Class Conclusion Q&A

	Questions	Answers
1	Why should we use variables in our code in the missions of this pack?	When using two- or three-state control, and especially whenever using proportional control, we must use variables for our speed values as well as our error and correction / gain values, since they continuously change. That is, after all, the meaning of **proportional** control: correction values change according to/in proportion with error values.
2	What coding 'trick' did we use in the mission "Ruby the Roving Robot" in order to keep our code short and efficient?	Repeat loops.
3	In "Adjusting Left" and "Reflection Redirection" missions, why did we only wait until the values of the two-color sensors were equal to each other's, instead of waiting for them both to equal the color/reflection value of the white line?	We **can** wait for the values to equal the color/reflection value of the white line, but that will require more time, and more corrections, and thus more energy, and possibly longer code as well. By waiting only until the two sensor's values are equal, we make sure that Ruby is aligned well enough to the line, and that will happen quicker and with less energy.



4 Frequently Asked Questions

1	What should I do before the first lesson?	Make sure the computers that will be used meet the minimum requirements to run CoderZ. Activate your CoderZ teacher account, create a class, and invite students to your class. If you can have them complete the sign-in process before the first session, it will save time. Review this Teacher's Guide and complete the relevant missions for the first session.
2	Do we need special computers or software?	CoderZ is only compatible with Chrome browser. It can be downloaded here. Users should make sure webGL is enabled. If it is not enabled, here is how to enable it.
3	How can I add students to my class?	Here is a support article with instructions for adding students to your classes is in our Knowledge Base.
4	How do I get help?	Teachers have full access to mission solutions, support articles and support staff. You can contact support@gocoderz.com with any technical issues
5	Can my students access this from home?	Yes, as long as they have an internet connection and their computers meet the minimum requirements.
6	What should I do if my students cannot finish all the assigned missions during class time?	We recommend that they finish the rest as homework. Teachers can also dedicate more lessons to allow students to complete all the missions during class.
7	How can I grade my students?	Teachers can use the Class Heat Map and Student Progress Reports to view their students' progress, including total missions completed, average attempts, score and more. Check this video tutorial to learn more.
8	Do you have any additional curriculums?	Yes! **Python Gym** is the continuation of Cyber Robotics 102. We are always updating our content, so have a look at our website or at this article in our Knowledge Base.
9	How can I provide feedback?	Use the purple Help Widget or contact us at feedback@gocoderz.com.

5

Credits

Course Materials

Nissim Hania

Gila Klein

Nitzan Green

Missions Development and Design

Nissim Hania

Daniel Ho

Ofer Zivony

CoderZ Learning Environment

Ahmed Kawasmi

Mohammad Qashua

Maayan Blum

David David

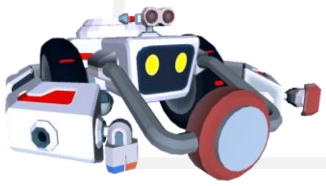
Quality Assurance

Nir Eini

Support

Justin Almeida

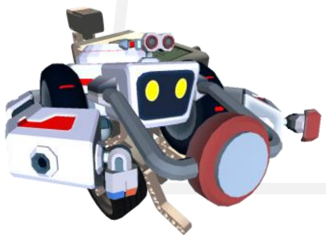
Trevor Pope



6 Appendix - Teacher's Notes

Session 1: Plains and Hills 1

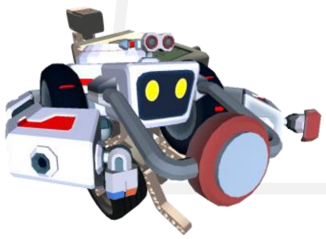
Part #	Slide #	Notes
1	**15**	Ask the students: what types of motors do you know?
	16	Steam engine – uses steam to push a piston back and forth inside a cylinder. Internal combustion engine – uses fuel like gasoline or diesel. Electric engine – uses electricity as its fuel. Jet engines – a type of Internal combustion engine that uses fast-moving jets of air. **Ask** the students what type of motor they think powers cars and buses nowadays.
	18	**Ask** the students what type of machines they know of which use motors. **Examples**: Cars, washing machine, vacuum cleaner, refrigerator, toy cars, airplanes, ships, cars, motorcycles, electric bicycles, etc. Explain that not all motors are the same shape or size. Some motors can be very small, for example the ones used to power toy cars or other toys; while a spaceship's motor can be as big as a house and weigh several tons.
	19	**Ask** the students what type of machines they know of which use motion. **Examples**: A washing machine needs to spin a drum around and around to wash our clothes. A vacuum cleaner power an air pump to create a vacuum. And automatic vacuums like Roombas need to drive around to clean the house. Ships need to move through the ocean. Motorcycles, bicycles, cars, and trucks need the wheels to spin to drive forwards.



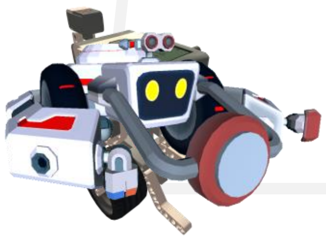
Appendix - Teacher's Notes

Session 1: Plains and Hills I

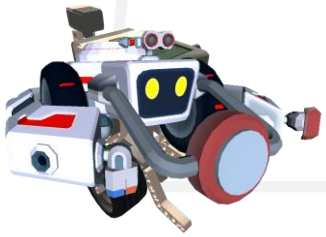
Part #	Slide #	Notes
1	**20**	At the beginning of course the Ruby will not have its arm/ forklift yet. It will be added (and used) later in the course.
	21	Explain that for the first few missions, we will work them out together. As we advance in the course, the students will get some time to try the missions themselves before receiving the solution.
	23	Movement > "set motor power" block > drag and snap to "program start" block.
	27	Have the students play around for a little bit (5 minutes) with the different power values in an empty scene. Ask them if they have used the same value (same number, same sign) for both left and right motors And if not – what happened? The robot didn't drive straight! Maybe it curved, or even spun in place! Remind them that in order for the robot to drive straight (whether forward or backward) both left and right motors must turn in the same direction (positive/negative value in both left and right data block) and at the same power (same value in both left and right data block)
2	**1**	**Explain** to the students about acceleration: Constantly giving the robot the same amount of power does not necessarily mean it will continue driving at the same speed. The longer the robot is driving, the more momentum it picks up, and the faster it drives. Of course, it can't exceed the maximum speed the robot is physically capable of, which is 100% power. But even when starting at 50% power, eventually the robot will accelerate to 100% velocity.
	2	Mention that in the HUD we can see the robot's velocity both **rising** and **falling**
	3	Explain that touching the container will cost us 5 batteries, so we need to make sure we stop before we touch it.
	4	Explanation about "wait" block is in next slide.



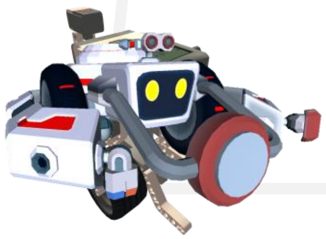
Part #	Slide #	Notes
2	**5**	Control Flow > "wait" block > drag and snap to "set power" block **Explain**: if the previous command in the program was to drive forward, (set power block) then the robot will continue to perform the last command (driving forward at XX power) and wait for 2500 milliseconds (2.5 seconds) before it moves on to the next command, whatever it is. The end result? The robot will drive forward for 2500 milliseconds. Explain that 2500 milliseconds is just a guess at this point – it might be too much. We'll have to test and see if we might need less or more time. Run the simulation. The robot will overshoot the target and hit the obstacle. Try a different (smaller) value for the "wait" block and see if that works.
	6	Explain to the students that touching the bolt will cause loss of batteries.
	7	Movement > "break until stop" block > drag and snap to "wait" block Explain that this is the same way that our parents' cars brake: pushing the brake pedal means we've taken our foot off the gas pedal (that's stopping the power to the motors) and also that the car's brake engages, and applies power on the wheels or the wheels' axis in order to stop it from turning.
	8	Explain to the students that energy takes time to dissipate, which is why the robot will continue drifting forward even after the motors stop powering the wheels. Have them look at the HUD in the gif – explain that it takes time for the velocity to drop to 0 (full stop). Explain that the faster the robot is going (higher velocity), the longer it will take to stop fully. **Human drift exercise**: ask a student to run a few meters in the class in your direction and pass near you. When he is just in front of you tell him to stop, of course he will not stop next to you, it takes time for him to stop. The distance that he passes from the moment he starts to stop until he stops fully shows the same idea of the influence of the momentum.



Part #	Slide #	Notes
2	**9**	Explain that trial and error, making little changes, and testing again and again is part of the engineering and STEM process!
	10	Have the students play around for a little bit (5 minutes) with the different power values in an empty scene. Ask them if they have used the **same value** (same number, same sign) for both left and right motors And if not – what happened? The robot didn't drive straight! Maybe it curved, or even spun in place! Remind them that in order for the robot to drive **straight** (whether forward or backward) **both left and right motors must turn in the same direction** (positive/negative value in both left and right data block) **and at the same power** (same value in both left and right data block).
3	**1**	Ask the students what they think. If one of them suggests driving backward and only then forward – praise them and let them complete the mission. **If not, explain the solution**: This is exactly why we've learned how to drive in both directions – backward and forward. For this mission we will have to use both directions – each time we will use a combination of power block and wait block, with one "brake until stop" block between the back and forth. Explain that the mission will be completed the moment we collect all the batteries, so for the 'forth' part we don't even need a "wait" block or a "brake" block.

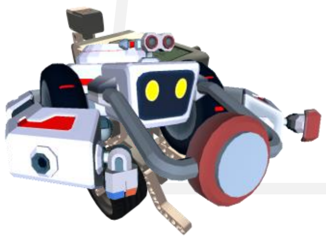


Part #	Slide #	Notes
3	**2**	Have the students play around for a little bit (5 minutes) with the different power values in an empty scene. Ask them if they have used the **same value** (same number, same sign) for both left and right motors. And if not – what happened? The robot didn't drive straight! Maybe it curved, or even spun in place! Remind them that in order for the robot to drive **straight** (whether forward or backward) **both left and right motors must turn in the same direction** (positive/negative value in both left and right data block) **and at the same powe**r (same value in both left and right data block)
	3	In the next mission we will have to deal with driving uphill!
	4	Let the students complete the following missions of the Pack: Plains and Hills Driving Uphill Giddy Up **NOTE: Do not explain how to solve the missions.** The idea is to let the students experience the uphill slopes and see if there is or isn't enough power to drive up the slope without driving backward to gain momentum. Explain that in the next lesson we will learn about gravity, mass, and weight, and understand how to solve these missions using physical concepts.



Session 2: Plains and Hills 2

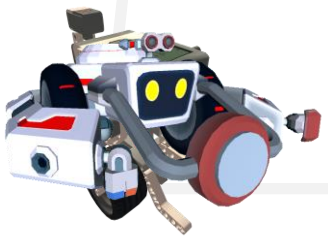
Part #	Slide #	Notes
1	2	Remind the students of what we did in the previous lesson: We learned about motors and motion We learned about our robot's motors and how to use the "power" block to drive the robot forwards and backwards We completed two missions.
	3	**Ask** the students if anyone had any difficulty or has any questions before we learn anything new today.
	9	Play the video – explanation on forces and movement
	10	If further explanation is needed: The force of gravity is affected by the mass of the bodies and their distance from each other and is more acutely felt when the bodies are large (like planets, as opposed to apples. This is the reason why it looks like only the Earth is pulling the apple to itself, when in fact the apple is pulling the Earth toward it too – just with a much weaker force). Gravitational force was formulated by Sir Isaac Newton, who determined that any two bodies with mass are attracted to one another directly proportional to their masses, and inversely proportional to the square of distance between them.
	11	**Ask** the students: do all objects fall to the ground? The answer is YES. Every object on Earth is drawn down to the ground by gravity, unless some other force is holding it up. Apples, for example, are connected by their stems to the tree's branches, which are holding them up. Even then, the apples are pulled **downward** toward the ground – just not strongly enough to fall. But this is the reason they **hang down** from the tree, rather than facing up or drifting off towards the sky. And once the stem or the branch breaks – the apple will fall.



Appendix - Teacher's Notes

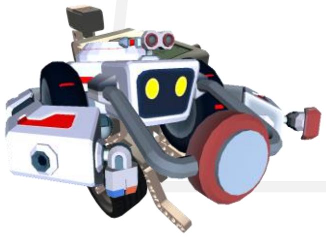
Session 2: Plains and Hills II

Part #	Slide #	Notes
1	15	Like in the mission "rolling uphill", the robot needs to be further away from the slope in order to have enough time driving forwards and gaining speed – enough speed to allow it to climb up the hill
	16	Have the students complete the following missions (10-15 minutes): Pack: Plains and Hills Rolling Uphill Giddy Up
2	**2**	On a plain, the floor/ground stops gravity from pulling the robot to the ground anymore. The robot doesn't need to exert much power to move forwards. On a moderate slope, gravity's pull on the robot is stronger, but the motor's power is still enough to climb. On a steep slope, gravity's pull is even stronger – the motor's power will not be enough. We will need to compensate for power with speed! If our surface is entirely vertical, then even with a lot of speed we will not be able to climb for more than a few seconds. Without alternative power (like a rocket!) we will not be able to climb at all.
	3	Let the students try these missions on their own. Ask them, what happens when the robot is pushing a heavy weight? Does this change the robot's speed? Does this change the way the robot should act?
	4	Heavy Mass Building Up Weights Weight Lifting



Session 3: Cruise Control

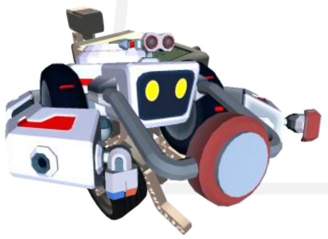
Part #	Slide #	Notes
1	**4**	**Explain** that the longer we drive, and the more we accelerate, the harder it will be to brake, and the longer time we will need in order to brake completely and make a full stop.
	6	Break-down / demonstration in next slide
	7	Ask the students if they remember specific places that have speeds limits. Examples: the street they live on will probably have a strict speed limit such as 40 or 50 kph. Densely populated areas like near shopping centers, pedestrian walks, and schools might have even stricter speed limits. Interstates and highways have a higher speed limit, like 90 kph. The **Autobahn**, the highway in Germany, has a speed limit as high as 130 kph!
	9	Ask the students: if the bad containers are moving, and we are in between them, how can we drive forward without touching the block before us or without the block behind us catching up with us and touching the robot from behind?
	10	Open the "Speed Limit" mission and turn on Manual Control. Open the HUD and open the VELOCITY value. Drive the robot forward and try to maintain a steady speed. Have the students note the best velocity in the HUD. **Explain** that using the up/down arrow keys with manual control is like using a "power" block – the robot accelerates, and it's hard to maintain a steady speed.
	11	Have the students complete the following missions **(MAX 3 minutes)** Pack: Cruise Control Speed Limit – **using manual control**.



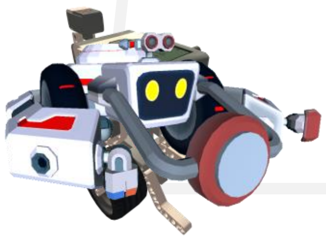
Appendix - Teacher's Notes

Session 3: Cruise Control

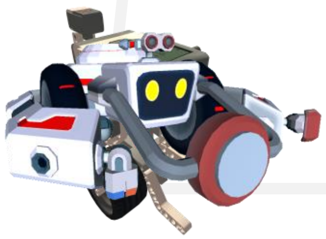
Part #	Slide #	Notes
2	**1**	Explain that many cars nowadays have an automatic Cruise Control option, which makes it easier for drivers to maintain steady speeds. The driver must bring the vehicle up to speed manually and use a button to set the cruise control to the current speed, and the system automatically turns off if the driver presses on the brake pedal. This option is best used in long stretches of road, like interstate highways.
	5	Explain the examples: **Streetlights** – the city sets a specific time of day for the street lamps to turn on and off. Even if the day is especially dark and cloudy, the streetlights will not turn on until the time they are programmed to arrives. **Irrigation systems** – again, the city will set a specific time of day for the water sprinklers to turn on and off in the park. Even if the day is very hot and dry, the system operates automatically and waters the park for a set length of time, even if it is not enough for the flowers and trees. It works the other way around too – even if it is pouring rain, and the garden does not need to be watered – the sprinklers will still turn on, because they do not know it is raining. They just know they were told to work for a specific length of time, at a specific time of day. **A toaster** – once you set the level of toasting you want, the toaster will toast your bread for a specific amount of time, and only once that time is up – the toast will pop up. Even if the toast burns – the toaster will not stop working before its set time. **Explain** that these systems are usually simple and cost less to design and maintain.



Part #	Slide #	Notes
2	**6**	Ask the students to brainstorm ideas how to turn each of the open loop systems from the previous slides to closed loop system. Most answers will require sensors of some sort: **Street lights** – instead of setting a specific time of day for the street lamps to turn on and off, installing a light sensor would turn it into a closed loop system: whenever it gets dark, the street lights will turn on. This could be at 7pm, or at 6pm, or even at 2pm if it is a stormy winter day. **Irrigation systems** – instead of setting a specific time of day for the water sprinklers to turn on and off in the park, installing a moisture sensor to determine when the soil is dry or wet would make the system more sophisticated. The system would sense when the soil is dry and turn on the water, until the sensor recognizes that the soil is wet to a degree the city decides, and only then will the system shut off the water. If it has rained today, the moisture sensor will recognize that the soil is already wet, and so the sprinkler system doesn't need to be turned on at all. **A toaster** – any ideas? Play the video for an explanation.
	7	**Ask the students** which of these we need to build in CoderZ in order to drive forward and avoid touching the bad containers in front of and behind the robot: an open loop or a closed loop?
	8	**The answer is:** a closed loop, because we need the robot to check its velocity(speed) and act according to it, change its behavior according to that data (velocity).
	9	Explain that **Pseudo code** is an easy way to plan our code, using normal words rather than blocks. Later we will translate our statements into blocks of code in CoderZ. Write these steps out on the classroom board so that the students can still see them after we move on from this slide. OR Ask one of the students to copy these steps to their notebook and then read them out to the class later. Explain that we will refer to these steps while we are building our closed loop.



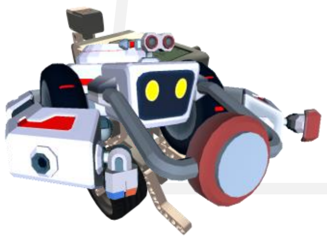
Part #	Slide #	Notes
2	**10**	Explanation about **If/Else block** is in next slide
	13	Explain that we need an **If/Else loop**, to **continuously** check the robot’s velocity, compare it to a certain value (the speed limit), and act accordingly – If the answer to the condition (in this slide, it is a comparison) is YES, the robot needs to act a certain way (this is the “do” segment of the loop); Else (in other words, if the answer to the comparison is NO) the robot needs to act in a different way (the “else” segment)
	14	Explain that a comparison block is already in place as our condition for the if/else block. We want to compare the value of the robot’s velocity (data from a sensor that might be changing all the time) to a specific value (the speed limit – this is a constant value and doesn’t change) But we don’t want to compare them as equals, we want to see if one of them is lower (less) than the other. So, we need to change the equals sign to a < (**less than**) sign. And then we can place the two blocks on either side of the sign.
	16	Explain that it’s best to not drive at full speed (100%) but to start out at half-speed (50%)
	17	Explain that if we were driving a car, we’d take our foot off the acceleration pedal (gas) therefore cutting the power to the motor. (0% power to both motors) We might also brake, but for now 0% power is enough.



Appendix - Teacher's Notes

Session 3: Cruise Control

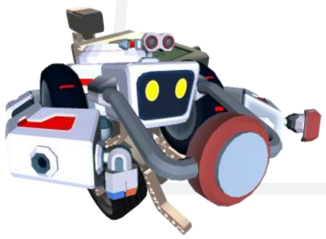
Part #	Slide #	Notes
2	**19**	Have the students complete the following missions (MAX 3 minutes) Pack: Cruise Control Cruise Control **NOTE: the code is not yet complete.** It is missing a "repeat forever" loop. In its current state, the code makes the robot drive forward indefinitely, because it doesn't loop back to the condition after the first run. The idea is for the students to figure it out on their own that the loop element is missing. It is step 4 in the pseudo code. If one of the students catches onto this and says, "wait, we forgot step 4!" then praise them and say we will take care of it now. If not, try to lead up to it, then move on to next slide.
	20	Ask the students: which step did we forget? **The answer is**: Step 4: repeat until we reach the target! Explain that without a **loop** to make the code **repeat itself**, the robot will only perform the comparison in the condition once, and only act according to its data once. That is what caused the robot to drive forward and accelerate beyond the speed limit.
	23	Have the students complete the following missions **(MAX 3 minutes)** Pack: Cruise Control 2. Cruise Control **NOTE: the code is not yet complete**. It is missing a "repeat forever" loop. In its current state, the code makes the robot drive forward indefinitely, because it doesn't loop back to the condition after the first run. The idea is for the students to figure it out on their own that the loop element is missing. It is step 4 in the pseudo code. If one of the students catches onto this and says, "wait, we forgot step 4!" then praise them and say we will take care of it now. If not, try to lead up to it, then move on to next slide.



Appendix - Teacher's Notes

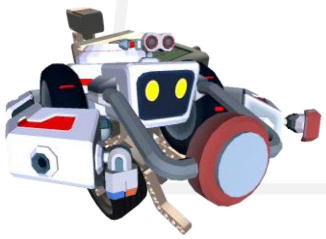
Session 3: Cruise Control

Part #	Slide #	Notes
3	**2**	Explain that now we don't have to build a complicated block with a loop and several conditions. Whatever speed we set the robot's motors to, it will maintain that speed, even when driving uphill or downhill. Try it and see!
	3	Have the students complete the following missions (10-15 minutes) Pack: Cruise Control Quick to the Block Speed Bump **Note:** these missions are the same as missions 3-4 (Uphill Scuttle + Steady Climbing) but now the students can complete them much easier thanks to the "set speed" block.



Session 4: Dangerous Curves 1

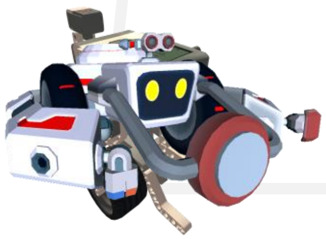
Part #	Slide #	Notes
1	**2**	Remind the students of what we did in the previous lesson: **Ask** the students if anyone had any difficulty or has any questions before we learn anything new today
	6	**Ask** the students: who here lives on a **straight** street? (show of hands) Well once you get to your house, you probably have to **turn** in order to face your house, or to pull into your driveway/garage, right? And who lives in a street that **curves**? (show of hands) See, life isn't always just straight. Sometimes you have to be flexible, and to curve this way or that. This is what our lesson today will be about.
	10	**Ask** the students to remember which direction is clockwise and counterclockwise. If there is an analogue clock in the classroom, use it to demonstrate. Clockwise = rotating to the right Counterclockwise = rotating to the left
	13	A screw turn can be useful for making very tight, precise turns. During a screw turn the robot stays in place and doesn't move forward or backward or to the sides. If the right wheel is spinning backward and the left forward (like in this example) the robot will spin to the right – clockwise. If the left wheel is spinning backward and the right forward, the robot will spin to the left – counterclockwise. In other words: the robot will spin in the direction of the wheel that is spinning backwards.



Appendix - Teacher's Notes

Session 4: Dangerous Curves 1

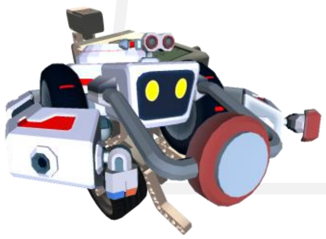
Part #	Slide #	Notes
1	**14**	Movement menu > "set speed" block > snap to "program start" block > change left motor to 50 and right motor to -50. This will result in a screw turn to the right.
	16	For turns we will always use a low speed, so that the turn is slow. Ask the students: but how do we know when to stop? The answer is: we use the gyro sensor.
	17	Break-down / demonstration in next slide
	19	Explain the building of the command block: Step 1: Add a "wait until" block. Step 2: Wait until what? Until some comparison becomes true. Step 3: What is that comparison? The first element, the one we will compare, is the value from the robot's gyro sensor. Step 4: What are we comparing the gyro sensor's value to? **Explain** that after this turning block we created we need to brake again, otherwise the robot will continue turning.
	20-21	**Now here's the tricky part**. Technically, we want the robot to stop at 90 degrees. But we need to use a ≥ **sign** (greater than and equal to) in order for the robot to recognize the angle and stop there. So, actually, this block says to wait until the gyro sensor recognizes an angle that is **greater than or equal to** 90 degrees for a right turn. If we are turning to the left, then the values of the gyro sensor are going down from 0, into the negatives. So, we need the gyro sensor to recognize an angle that is **smaller than -90**, like -91, -95, -100 etc. * Make sure the students understand the negative values: -100 is smaller than -90.



Appendix - Teacher's Notes

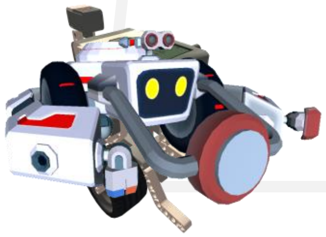
Session 4: Dangerous Curves 1

Part #	Slide #	Notes
1	**22**	Ask for 5 volunteers. Have 1 student stand in the middle, surrounded by the other 4 students at 90-degree angles (like the points of a compass) Ask the student in the middle to spin in place (direction doesn't matter) for several seconds – at least 3-4 spins. Ask the student to try to stop facing one of his fellow students directly. Most likely the student will not succeed and will 'drift' a bit further. Explain that because of their spin and momentum, it took a while to stop completely.
	26	See, time is tricky. If we were to drive for 3 seconds at full speed, we would go a further distance than if we were to drive for 3 seconds at 20% speed. Because time * speed = distance traveled, any change in speed results in a different distance that Ruby will drive. By using **distance** to tell Ruby exactly how far to drive, we are making it much simpler, and more importantly, consistent.
	28	If animation is unclear to students, draw the figure on the classroom board and explain each angle (first 180, then a= inner, then b=outer) Explain that 180 = straight line, driving straight forward, and that the two angles (a=inner + b=outer) together equal 180 degrees.
	29	Open mission V Curve and open Explore Mode. Show how to use the tool to discover angles. If necessary, move the robot around the scene first via manual control.
	31	Have the students complete the following missions (10-15 minutes): Pack: Dangerous Curves 1 L Turn V Turn



Session 4: Dangerous Curves 1

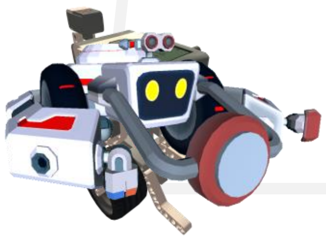
Part #	Slide #	Notes
2	**1**	Explain that the values of the gyro sensor are continuous, and **relative to the robot's original starting point**.
	2	Explain that using Explore Mode we can discover that the inner angles at both points of the N are both 45 degrees, meaning we must turn 135 degrees twice This is the same calculation we used for the letter V: 180 – 45 = 135. If animation is unclear to students, draw the figure on the classroom board and explain each angle and each turn (outline directions of turns) **Ask the class**: After the second turn, what will our gyro sensor show? **The answer is**: 0 degrees, because we turned the same amount as we did the first turn, but in the opposite direction. So, basically, the second turn 'cancelled out' the first turn and brought us (the robot) back to its original direction (facing forwards) If we would have turned in the same direction, the gyro sensor would have shown the sum of the two angles (135 + 135 = 270)
	3	Have the students complete the following missions (10-15 minutes): Pack: Dangerous Curves N Curve M Curve



Appendix - Teacher's Notes

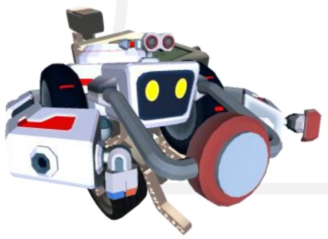
Session 4: Dangerous Curves 1

Part #	Slide #	Notes
3	**1**	Explain that using a "reset gyro" block after each turn, or right before performing a turn, means that we will only ever have to calculate the angle of the turn itself, starting from 0 degrees before each turn. We don't ***have*** to do this. But if we do not use the "reset gyro" block then we have to remember that all the turns' degrees add up, and calculate accordingly.
	2	Explain that there still might be some movement of the stationary wheel, because it is not locked in place. However it is only 'being moved' by the fact that the robot itself is moving – the engine controlling this wheel is not powered.
	4	Explain that finding the correct angle is a matter of trial and error and depends on the robot's speed. 20% speed and an angle of 55 might be enough for a 90-degree turn in the end, or it might not be enough, or it might be too much. If the robot drives faster, at 50%, you'll probably need to put a smaller value in the gyro block. It balances out.
	6	Have the students complete the following missions (10-15 minutes): Pack: Dangerous Curves I Curve J Curve



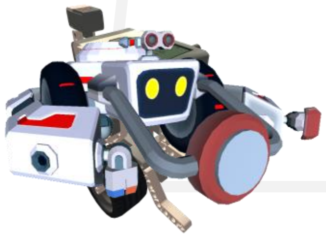
Session 5: Dangerous Curves 2

Part #	Slide #	Notes
1	**2**	Remind the students of what we did in the previous lesson: We learned about two types of turns: the screw turn, and the pivot turn. We completed several missions using these two types of turns. We learned about the gyro sensor and how we can reset it to 0 after a turn, and how if we don't, then all the turns' angles add up.
	6	Explain that this block uses its own control system behind the scenes, checking the robot's current gyro reading, comparing it to the angle we want it to turn to, deciphering if it needs to turn clockwise or counter-clockwise, and then turning very carefully and precisely to exactly that angle. It means we don't have to take into consideration the robot's momentum, and we won't have to play around with "oh, we want to turn to 135 degrees but the robot's momentum will carry it further, so let's try 110 degrees and see if that works..." If you tell it to turn to 135 degrees – it will turn exactly to 135 degrees and no further!
	8	By the difference in speeds of the two wheels (motors) we'll determine the route the robot will take – the radius of the turn. The larger this difference, the tighter the turn will be. For example, if the left wheel is turning at 20% speed, and the right at 80%, it means that the left wheel is moving very slowly, and the left side of the robot will travel a very short distance. The right wheel is travelling very fast, meaning the right side of the robot will travel more distance – this will result in the robot turning a very tight turn to the left. If we let the robot continue its smooth turn indefinitely, it will basically drive in a circle.
	9	Movement menu > "set speed" block > snap to "program start" block > change left motor to 20 and right motor to 80. This will result in a smooth turn to the left



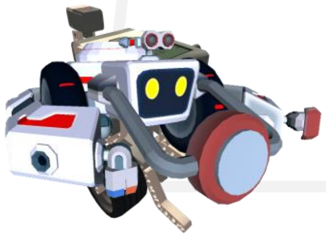
Session 5: Dangerous Curves 2

Part #	Slide #	Notes
2	**2**	After the U-turn robot animation completes - **Ask the students**: What angle does a U-turn have? The right answer is: 180 degrees. The robot needs to be facing in the opposite direction, so the gyro sensor needs to reach +180 degrees throughout the turn.
	3	**Ask the students:** What angle does a U-turn have? The right answer is: 180 degrees. The robot needs to be facing in the opposite direction, so the gyro sensor needs to reach +180 degrees throughout the turn. Explain that now that we know how far (in degrees) to turn, and how to perform a smooth turn, we should be able to navigate this U-turn mission with ease.
	6	Explain that using a "reset gyro" block after each turn, or right before performing a turn, means that we will only ever have to calculate the angle of the turn itself, starting from 0 degrees before each turn. We don't ***have*** to do this. But if we do not use the "reset gyro" block then we have to remember that all the turns' degrees add up, and calculate accordingly. For example, in the S-Turn, we are turning first 180 degrees to the right, and then at the second turn, 180 degrees to the left. If we **do not** use the gyro reset block, then we must calculate the second turn **back to 0 degrees**, since we are calculating 180 – 180 = 0. If we **do** use the gyro reset block, then we need to calculate the second turn to **-180 degrees**, since we are calculating 0 – 180 = -180 (turns left/ccw from 0 run into the negative values)
	7	Have the students complete the following missions (10-15 minutes): Pack: Dangerous Curves 2 U Turn S Turn



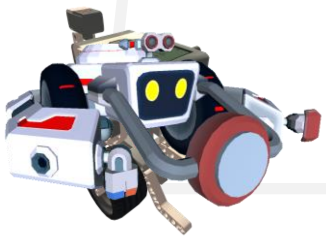
Session 5: Dangerous Curves 2

Part #	Slide #	Notes
3	**4**	Use a loop for drawing the star, it'll be easier. Note that because of the placement of Ruby's trail drawer, Ruby draws a little squiggle when she turns or rotates. Therefore, you might want to stop the trail while Ruby turns, then start it up again when Ruby starts driving forward again.
	5	Have the students complete the following missions (10-15 minutes): Pack: Dangerous Curves 2w Freeform Art Star Trail



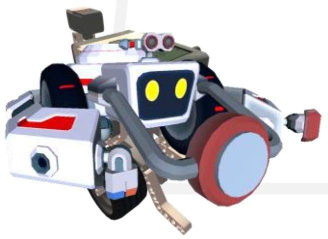
Session 6: Doodling with Distance

This pack is a great opportunity to use **Repeat Loops** (especially for drawing patterns). Technically this subject has not yet been taught in-depth in CR102, but any student who remembers Repeat Loops and wishes to use them is welcome to.



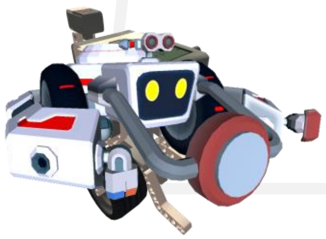
Session 7: Touch, Avoid, Repeat

Part #	Slide #	Notes
1	**9**	Ruby's touch sensor is located on her front left arm for the first few missions of the pack. It might be located in a different position in future missions!
	10	Explain that this type of sensor, which gives a true/false reading, is called a **Boolean** sensor. It's like binary code – either 1 or 0, 1 being true/on and 0 being false/off
	11	Most of the time, the touch sensor will read False, because most of the time Ruby is not touching anything. In this GIF, Ruby is facing the wall dead-on, which is why both sensors are pressed at the same time. If Ruby were touching the wall at an angle, with only her left touch sensor or her right sensor touching the wall, the readings would show only the left or right sensor reading TRUE, respectively.
	13	Explain that "until we touch the container/tree" actually means "until the touch sensor gives us a **TRUE** value"
	14	Whichever sensors are configured for Ruby for any particular mission, these are the sensors that will appear in the Sensors menu. They are usually numbered, or called meaningful names like "fwd_left" so that we can easily understand which sensor we are dealing with
	21	It doesn't matter if Ruby is touching a container, a tree, or a fence – all that's necessary is to check if the touch sensor is saying TRUE. In order for "90 degrees counter-clockwise" to always work, we must use a "reset gyro" block after each turn, **inside the loop**, so that Ruby will always turn 90 degrees counter-clockwise relative to where she is facing right now.

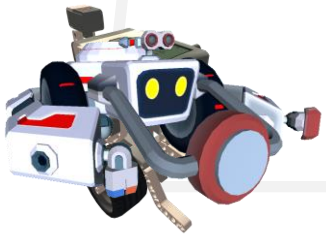


Session 7: Repeat Touching or Avoiding Obstacles

Part #	Slide #	Notes
1	**22**	Let the students write pseudo-code on their own if they want to.
	23	Have the students complete the following missions (10-15 minutes): Pack: Apple Picking Red Apple Trees Trees on The Left **Note:** The second loop technically only needs 2 repeats, and then a block to drive forward towards the target. Explain that by adding another repetition to the loop (3 instead of 2) we can utilize the 'drive forward' block in the loop to drive forwards! Repeat Picking
2	**4**	Explain that this type of sensor, which gives a numerical value, is an **integer** sensor – it returns values in whole numbers. Also explain that if it is placed in Ruby's front, then it is "inside" her arms' range, it does not stick out as much as the other sensors. This means that when the robot hits a wall, the ultrasonic sensor is still a few centimeters away from the wall and its reading will show a value of a few centimeters.
	5	Explain that for this mission, we have the bad containers placed closer to us. We want to keep driving forward until we reach a point where the sensor recognizes something placed further than 80 cm from us – the good container which we want to touch.
	6	Ask the students which trick that we learned in the last few missions we should also use here? The answer is: a **repeat loop**! Ruby needs to pass a bad container which is closer to her (we can use manual control and Explore Mode to determine the exact distance) and recognize a container that is further away from her (a good container), turn to 90 degrees, drive forward until she touches the good container, then drive backward until she is out of range of the bad containers, stop, then turn back to face forward (0 degrees). Ruby needs to repeat this process 5 times before she can drive to the target!

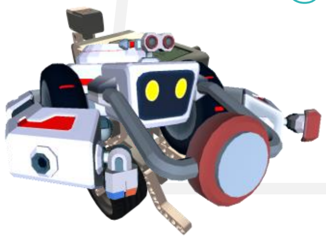


Part #	Slide #	Notes
2	**7**	This mission is quite similar to the previous one (Trees on the Right) except that now Ruby has to go both left and right. But does she really have to go in both directions? Hint: once she has turned right, she can drive forward to the container on the right, then drive backward to the container on the left!



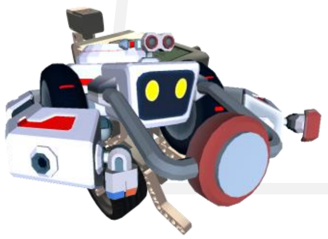
Session 8: Random Obstacles Ahead

Part #	Slide #	Notes
1	**5**	It is helpful to think of variables as containers that hold information.
	7	Ask the students: Have we learned about any of these variables before? The answer is: We have learned about **Boolean** variables – the touch sensor gives us a true/false value.
	10	Software programmers usually give variables a descriptive name, which implies the value stored in the variable. (So the variable can be understood more clearly by us and by any outside reader looking at our code).
	16	Explain that when using "wait for ___ milliseconds" blocks, it might actually translate to different distances depending on the wheel speed or motor power. If Ruby drives forward at 100% speed/power for 2000 milliseconds, she will travel further than if she drives at 25% speed/power for 2000 milliseconds. This makes "wait for time" blocks less precise. When we want to be very precise and absolutely accurate, we will measure distance using Explore Mode, and use 'drive distance' blocks.
2	**5**	Since the first command inside our loop is to drive forward at 100% speed, the two wait until commands can come directly one after the other and not interfere. All this time, Ruby is still driving forward, and she will wait until the ultrasonic registers the distances we need before she moves on to the next part of our code.
	8	If we have placed the "print" block inside a loop, then the value of the "containers passed" variable will be printed into the console each time the code reaches the block in the loop.
	9	Explain that a "repeat while" loop will make Ruby repeat the code inside it until the condition is met. Therefore, our condition is "while the value of the variable containers_passed is smaller than 3".
3	**5**	Since the first command inside our loop is to drive forward at 100% speed, the two wait until commands can come directly one after the other and not interfere. All this time, Ruby is still driving forward, and she will wait until the ultrasonic registers the distances we need before she moves on to the next part of our code.



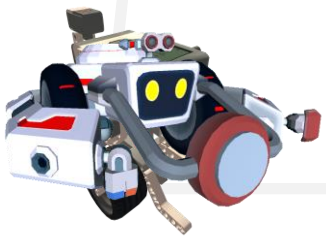
Session 9: Radar Missions

Part #	Slide #	Notes
1	**9**	Since the first command in our code is to drive forward at 25% speed, the two wait until commands can come directly one after the other and not interfere. All this time, Ruby is still driving forward, and she will wait until the ultrasonic registers the distances we need before she moves on to the next part of our code.
	12	0 is the first number in programming, rather than 1. So, the first joint will be joint 0.
2	**6**	Technically, a double type variable could also work. However, doubles take up more computing space, and since we are working with whole numbers (no decimals), an integer would work just fine.



Session 10: Colorful Code

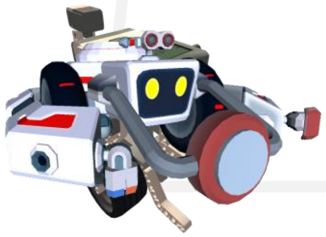
Part #	Slide #	Notes
1	**11**	Explain that "one" and "1" are both string values and are not the same as an integer variable's value of 1. Using string values in an equation will result in completely different values than using integers or doubles. Examples: ["1"] + ["1"] = 11 (but it does not equal eleven! It's a textual 1 and next to it another textual 1) ["1"] + ["5"] = 15 (not 6! But also, not fifteen. It's a textual 1 and next to it a textual 5)
	13	Ruby's color sensor has two functions: • Color Detection • Reflected Light Measurement For this lesson we will only be using the color detection mode.



Appendix - Teacher's Notes

Session 11: Repeat Again

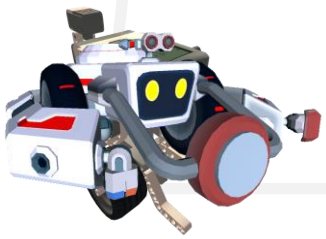
Part #	Slide #	Notes
1	**6**	Think of Russian **nesting dolls** (Matryoshka/Babushka) where there are smaller dolls inside bigger dolls. The concept is the same.
	8	We could go even deeper and say that within every month are four weeks, and within each week are 7 days.



Appendix - Teacher's Notes

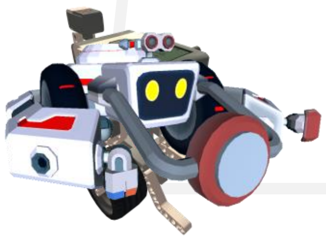
Session 12: Magnetic Manipulation

Part #	Slide #	Notes
1	**9**	It's important to put a **wait until get magnet arm position = ["up" / "down"]** blocks directly after the pink **magnet arm position** block, before the next command. By doing this, we make sure that Ruby does not move on to the next command (which might be driving forward or backward, braking, turning, etc.) until the magnet arm has finished rising or lowering. This could make or break a mission!
	12	Since there are 3 magnet boxes and Ruby needs to perform the same actions for each one, advise the students to use a **repeat loop**. It's possible to use **repeat 3 times**, but this will necessitate an extra block to drive forward after the loop. The better option is to use a **repeat forever** loop, which will save us that extra block, and the mission will simply end early in the 4th iteration when Ruby reaches the target.
2	**5-6**	This is also a good place to use a **repeat loop**, since Ruby needs to magnetize two magnet boxes and drop them in the correct places to fill the gaps in the bridge. Note that the yellow lines are only for decoration and the code does not need to include any reference to them.
3	**4**	If the students do not remember logic operators, you can show them Lesson 10 Part 2
	7	Since the scene is symmetrical, it doesn't matter if Ruby goes first for the magnet boxes on the right and then on the left, or first to the left and later to the right - so long as we keep our degrees straight and don't mix up 90 and -90.
	8	In order to keep our directions straight, we'll use a **variable** for our direction and using this variable as the value for a **turn to** block. This way it will be easy to toggle between right and left at the end of the loop, by setting the direction to itself x (-1), therefore changing from 90 to -90 [90 x (-1) = -90], or from -90 to 90 [(-90) x (-1) = 90]. This code can work the same way if the student uses 90 or -90 for the initial value of the variable. If there is time left in the lesson, demonstrate this by switching the initial value.



Session 13: Line Following Logic

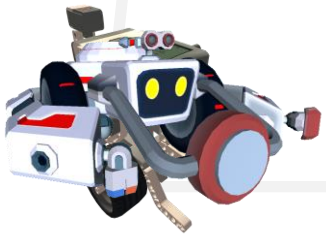
Part #	Slide #	Notes
1	**13**	At first, Ruby is centered exactly on the middle of the line, in a way that both her sensors detect the same color - "Green". Once we decide we want to follow the **left** edge of the line, we need to start working with the left color sensor and focus on its readings. For this reason, the first statement inside the if/else loop (do) will be to drive slightly to the **left**. To do this, we have to give Ruby's right wheel more speed than the left wheel, forcing her to the left.
	14	Since the first statement was driving to the left, the second statement (else) has to be driving slightly to the right. To do this, we have to give Ruby's left wheel more speed than the right, forcing her to the right.
	15	Since the if/else loop is inside a **repeat forever loop**, these two statements (drive slightly to the left if you see green, else drive slightly to the right) will drive Ruby gradually forwards in a funny zig-zagging way.
	19	When a line curves to the left, the right edge - the "**outside**" edge - of the line will be slightly longer/wider than the left "inside" edge. We are using high speeds in this mission because of the time limit, so we need as much edge as we can find, so Ruby doesn't drive too far and lose the edge. By using the right color sensor and following the right edge, Ruby will still be mostly "on" the line itself because only her right sensor, on the front right bumper, is on the right edge. If the line was curving to the right, we would work with the left "outside" edge and use the left color sensor.
	20	Clicking the hand icon will show a reminder of the OR logic operator. If the students do not remember logic operators, you can show them Lesson 10 Part 2.



Appendix - Teacher’s Notes

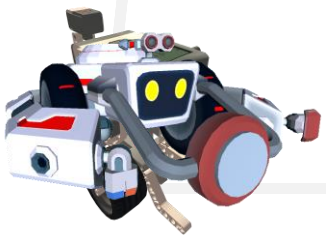
Session 13: Line Following Logic

Part #	Slide #	Notes
2	**4-5**	If the students have a hard time understanding the need for middle ground or the mean formula (35 + 100) : 2 ≈ 70, perform an experiment with them: Ask for 2 volunteers. Mark 2 lines on the floor with electrical tape/erasable marker, several meters apart, one marked 35 and the other marked 100. Have a student stand on each marking, facing each other. Then have both students take a (small) step forward, towards each other. Then another step. Then another - until the two students meet in the middle. (Maybe ask them to shake hands) Explain that they have now reached the middle ground between the two values, 35 and 100. This point, this middle ground, is the same distance from 35 as it is from 100. It is exactly in the middle. This is the “mean” or “average”. Technically the value is 65, but it’s okay to use something close to it, 70 will do as well.
	7	A different way of phrasing the comments in the picture could be: do (comment: reflection value 71 or higher - too close to the white line) else (comment: reflection value 69 or lower - too close to the brown floor)
3	**4-5**	By placing the “set variable to reflection value” block inside the repeat forever loop, we make sure that the “reflection” variable always shows the current value the sensor detects - it might change from second to second! This will make the corrections (later determined by the “correction” variable - slide 5) more precise, since the loop is always checking for the reflection value and acting according to it.



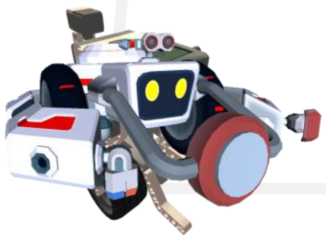
Session 13: Line Following Logic

Part #	Slide #	Notes
4	**3**	If our deviation from the desired value is small, we'll set our correction to a small value as well, resulting in a small, gentle correction. If our deviation from the desired value is LARGE - we'll set our correction to a high value, which will make our corrections fast and quick. Note that the two lines in the graph are aligned - this is because the ratio/proportion is the same for all deviation values. This is the point of proportion.
	5	Note that with a big deviation such as 50 points (a reflection value of 100) Ruby's correction would be quite big; but a minor deviation such as 10 points (a reflection value of 60 or 40) would need a much smaller, gentler correction. A reflection value of 50 - exactly our desired value, the middle ground between 0 and 100 - would require no correction at all, which means Ruby could keep driving straight.
5	**General note**	These missions are quite complex. If students cannot complete these missions, show them solutions or give them hints.



Session 14: A Hard Block Life

Part #	Slide #	Notes
1	**5**	An encoder is an electromechanical device that converts the angular position of the motor axle to an electrical signal that is used for speed and/or position control. Each motor has its own independent encoder, just like every wheel has its own motor.
	6	The encoders' values are in degrees, since they are connected to the rotation of the wheel axles. One full rotation of a wheel when driving forward will show 360 degrees on that wheel's encoder, two rotations will show 720 degrees, etc. One full rotation driving backward will show -360. If Ruby drives straight (either forward or backward) then both wheels/axles are turning the same amount/the same degrees, so both encoders will show the same number.
	7	If Ruby turns on the spot (screw turn) then the two encoders will show the same values, just one positive and one negative, since one wheel is spinning forward and the other one backward. If performing a smooth turn, then one encoder will show a higher value and the other a lower value, since one wheel is spinning at a higher speed than the other.
	11	If students need more reminders about joints and the sensor rotation blocks, go to Lesson 9 Part 1.
2	**3**	This technique of correcting by driving Ruby in gentle zig-zags (where one wheel spins faster than the other) is the same technique we used for line following in Lesson 13.
	13	Note that in this mission, we'll be using the error / deviation from (Y=0) as it is, using the **get gyro y axis** block. In more advanced missions, we will use a variable to represent this error, since we might want to manipulate it (multiply or divide) to aid our code.



Appendix - Teacher's Notes

Session 14: A Hard Block Life

Part #	Slide #	Notes
3	**2**	Explain the strategy for aligning to the white lines: At first both of Ruby's color sensors see the brown floor (a reflection value of 35). Once Ruby starts driving forward, her left color sensor will be the first of the two sensors to detect a higher reflection value (the white line's value is 100) because of the angle the line is drawn at. This is why we will be using the left color sensor throughout this mission to determine whether Ruby has reached the white line's edge.
	3	But since we will be using a **repeat forever loop**, we have to take into consideration the second, third, fourth iterations of the loop, where Ruby is not standing with both sensors over the brown floor. After reaching the first white line and aligning to it, both of Ruby's sensors will see the white line. This is why our first command inside the repeat loop, has to be driving forward until Ruby's sensors no longer see the white line, but the **brown floor** instead. Since the brown floor's reflection value is 35, we'll use $\leq$ 50 (smaller or equal to 50) for this. The next command is exactly the opposite - waiting until we see a reflection value greater than 50, which is just past the edge of the white line.
	4	Working with reflection values will be much easier and more precise for this mission, because we don't want to limit ourselves to the Color Name values - they rely on almost completely white to work. We want to use a wider variety of values for this mission, which is why we'll use reflection values. The real reason why will be shown in slide 5.
	10	In **Adjusting Left** mission, the lines were drawn in a half-circle around the car in the middle, facing "inward", which meant that it would always be Ruby's left ("inner") color sensor which would detect the edge of the white line first. In **Reflection Redirection** mission, the white lines are drawn on the floor in opposite angles, which means that every time Ruby drives off of one line and onto another - it will be a different color sensor which will reach the edge of the line first.