KAREL JUNIOR PROGRAMMING
COURSE

LESSON PLANS

REVISION: AUGUST 17, 2016

OVERVIEW

The Karel Jr course is a set of five units designed to teach students computational thinking and the beginning fundamentals of computer programming. The language itself is a simplified version of Python, which is used extensively in engineering, science and design work. The basic concepts are common to all programming languages. By the end of Karel Jr, students will have learned these skills:

1. Algorithmic thinking
2. Typing single commands
3. Running and debugging programs
4. Using counting (repeating) loops, nested loops
5. Using conditional (if-else) statements
6. Using conditional (while) loops
7. Defining and using custom commands
8. Using functions that return values
9. Using local and global variables
10. Using basic operations with Python lists
11. Designing recursive algorithms

Karel can be learned independently or under the guidance of a teacher. The five courses are each divided into five sections, with seven levels in each section. Each level builds on the previous one. Tutorials, YouTube videos and hints guide the students. In most levels, the programs are partially written, so that students can focus on the skill that they are learning. Students are able to run their programs in their entirety or line-by-line to detect and fix bugs.

Tasks are embedded in a narrative about Sophia and her robot Karel. The graphic interface is colorful and easy to follow, as the robot responds to commands written and executed by the student.

Although the program stands on its own, the value of the lessons is greatly enhanced by classroom discussion and solution sharing. There are multiple ways to solve problems, and by comparing solutions, students will develop logical reasoning, communication skills and creativity.

Once students have learned a few tasks, they will be able to create their own games, tasks and solutions using the Creative Suite. Creating the games cements the learning and develops a love for programming. Students can also flex their narrative writing muscles! A good game has a good story.

Students can save games to their own NCLab folder. They can publish and share links to the games. Games can be submitted to NCLab for display on the Gallery page.

Printable student journals are available for review of concepts and skills, reflection and design.

EQUIPMENT AND ACCOUNTS REQUIRED:

- **Personal computers or tablets with keyboard functions; Internet access:** one per student. Both PC and MAC platforms are supported. Preferred browsers are Google Chrome or Firefox.
- **Projector or Smartboard** (optional but recommended) attached to a computer for demonstration or modeling
- **Accounts:** The Karel Course requires individual accounts for each student. Visit the FAQ page for information on free and paid accounts. https://nclab.com/faq/ Have names and passwords ready on Day 1 to make logging on a smooth process (small cards with this information can be passed out to each student)
- **Progress monitoring:** Students accounts associated with a teacher can be progress monitored from the teacher's NCLab desktop.
- **The teacher textbook** can be downloaded as a .pdf file from the Resources page https://nclab.com/resources/
- **Student Journals***:* available separately as a downloadable .pdf file from NCLab, one per student.
- **YouTube videos:** some schools block YouTube, so the demonstration videos may need to be unblocked by an administrator to make them available to students.
- **Publishing:** Students should have a way to share a link to their games with others, such as a shared folder on a network drive; class or student wikis, web pages, blogs or email accounts; commercial networks such as Google Drives or Edmodo; or public social media network such as Facebook or Twitter.
- **Publishing to the NCLab Gallery:** students can submit their games to https://nclab.com/karel-gallery-submit/
    - **Student work can be viewed at:** https://nclab.com/karel-gallery/

SUGGESTED AGE RANGE FOR STUDENTS

Karel Jr is designed to teach students between ages 10-16. The younger students tend to progress more slowly but can still be successful, especially in Karel 1 and 2. High school students will have more experience in formal reasoning, problem solving and mathematical functions, which is helpful in understanding the commands and algorithms in Karel 3, 4 and 5.

TIME REQUIRED AND SUGGESTIONS FOR COURSE DELIVERY

There are 175 levels or lessons in Karel Jr. Each of the five units is divided into 5 sections of 7 levels each. The following lessons are written for each section, with screenshots and notes on the specific skills in each level within the section. Students will naturally slow down as the coding becomes more complex. In general, the amount of time required for the course is about 15 hours of actual computer time. Here are some suggestions for lesson delivery:

- **As a camp or workshop** to introduce students to the course. This setting allows long stretches of computer time. In a one-day workshop, students may complete the first two units and have time to create some simple games using Creative Suite.

- **As a self-paced course for independent study**, for after school programs, programming clubs, gifted and talented programs or home study. Students are more likely to complete the course if they are encouraged and supported by adults, and if they have the opportunity to publish their own games.

- **As part of an elective computer programming class** at the middle school or high school level. Karel Jr is comprehensive and rigorous. Students who complete all five units will have been introduced to all the basic tools of programming.

- **As mini-lessons** of about 20-30 minutes each, addressing one or two levels at a time. This might be a good option as a supplement to regular math instruction in upper elementary and middle school where time is a premium. At this rate, students may only complete Karel 1 and 2. However, spreading out the lessons may be more successful at reaching students from a broader range of ability and background, because the course is chunked into smaller segments with teacher and peer support.

A separate pacing guide is available for the course.

CROSS-CUTTING CONCEPTS: MATH AND ELA STANDARDS

Math Content Standards: There is no particular math content prerequisite for this course other than a basic understanding of arithmetic and algebraic relationships. Students will learn new concepts as they go through the course, which can be correlated to Common Core content standards as follows:

Unit, Section, Level	Concept or Skill	Content Standard
Karel 1 Section 1-5	Develop fundamentals of writing code, including repeat loops and nested loops.	OA.C Patterns and relationships (3^{rd} grade onward)
Karel 2 Sections 6-10	Use conditions, logical operators	OA. A, EE.A, 1,2,3 Expressions and equations, algebraic relationships (5^{th} grade onward)
Karel 3 Sections 11-15	Define functions and use variables within operations to count, and to increase or decrease a function.	8.F.A.1; Understand functions and variables. HS.F.BF.A.1 Determine, combine and compose functions.
Karel 4 Sections 16-20	Continue developing use of functions, variables. Define, retrieve and output specific data. Use random number generators.	HS.F.BF.A.1 Determine, combine and compose functions. HSS.MD.A.1,7, B6. Define and use random variables; display output.
Karel 5 Sections 21-24	Use recursion.	HS.F.BF.A.1,2,3. Building and interpreting recursive functions.

Math Process Standards: Students will develop good math process skills as they learn to write code. In fact, all of the Common Core Standards for Mathematical Practices apply, so students may very well improve in their regular math studies as a result.

SMP 1: **Make sense of problems and persevere in solving them.**

- Each lesson is presented as a problem or puzzle to be solved. Students can test their programs instantly, as they go. This feedback encourages them to correct errors and continue until the task is completely solved.

SMP 2: **Reason abstractly and quantitatively.**

- Students learn how to write logical command sequences, including conditions and repeated routines (loops and nested loops)

SMP 3: **Construct viable arguments and critique the reasoning of others.**

- In the search for code that meets or beats the criteria, students naturally engage in discussions about the best way to solve a puzzle. They often help each other uncover errors. Class discussions and journals enhance this communication.

SMP 4: **Model with mathematics.**

- Coding, by its very nature, is translating actions, conditions and goals into defined terms and symbols.

SMP 5: **Use appropriate tools strategically.**

- Students have to choose the most effective commands and sequences needed to solve the problem. Subroutines (loops), conditions, and commands are selected to create code that is efficient, robust, readable and flexible.

SMP 6: **Attend to precision.**

- Programs will not run correctly if there are any logical or syntax errors.

SMP 7: **Look for and make use of structure.**

- To solve a puzzle, students must break down a task into logical steps.

SMP 8: **Look for and express regularity in repeated reasoning.**

- Patterns are the key to writing repeated loops, nested loops and conditions.

English Language Arts Student journals, discourse and game creation are all opportunities to practice language skills.

- W.x.1: Argumentative writing: Students evaluate, compare or defend a method of problem solving.
- W.x.2: Informational writing: Students write explanations of reasoning, instructions for their own games.
- W.x.3: Narrative writing: Students write short stories to provide context for their games.
- W.x.6: Students use technology to publish and share writing.
- W.x.10: Students write routinely … for a range of discipline-specific tasks, purposes, and audiences.
- SL.x.1: Students engage in collaborative discussions, building on each other's ideas.
- L.x.1, 2: Students must use precise syntax, grammar, spelling and punctuation in programming, or their programs will not run. Indirectly, students may improve their use of these skills in other academic tasks.

NEXT GENERATION SCIENCE STANDARDS (NGSS)

NGSS is built on three dimensions: Scientific and Engineering Practices (SEP), Disciplinary Core Ideas (DCI), and Cross-Cutting Concepts (CCC).

Learning to write code using Karel develops skills in engineering practices and cross-cutting concepts. By writing their own code in Creative Suite, students develop models that could be applied in data collection, storage and retrieval, measurement, search functions, and other areas.

Scientific and engineering practices exercised in Karel are

SEP 2: Developing and using models. Since Karel programs use functions and variables, students are learning coding habits that will lead them to develop testing models.

SEP 5: Using mathematics and computational thinking. Karel helps students make sense of concepts in algebra. Students create visual representations and output lists of these functions.

Cross-cutting concepts are valuable tools that can be used to link the skills learned in Karel with fields of scientific and engineering. The main cross-cutting concepts in Karel are

CCC 1: *Patterns. Observed patterns of forms and events guide organization and classification, and they prompt questions about relationships and the factors that influence them.* Students design loops based on patterns. Students can use their skills to build models of repeated patterns found in natural and man-made systems and procedures.

CCC 4: *Systems and system models. Defining the system under study—specifying its boundaries and making explicit a model of that system—provides tools for understanding and testing ideas that are applicable throughout science and engineering.* Students are learning how to create models, which can be then be applied to real world problems.

Engineering Design (ETS1, 2, 3): Students apply what they learn in each Section to create a game. In the process, they are learning how to define, design and optimize.

From the NGSS website:

The core idea of engineering design includes three component ideas:

A. Defining and delimiting engineering problems involves stating the problem to be solved as clearly as possible in terms of criteria for success, and constraints or limits.

B. Designing solutions to engineering problems begins with generating a number of different possible solutions, then evaluating potential solutions to see which ones best meet the criteria and constraints of the problem.

C. Optimizing the design solution involves a process in which solutions are systematically tested and refined and the final design is improved by trading off less important features for those that are more important.

How Karel Jr fits this model:

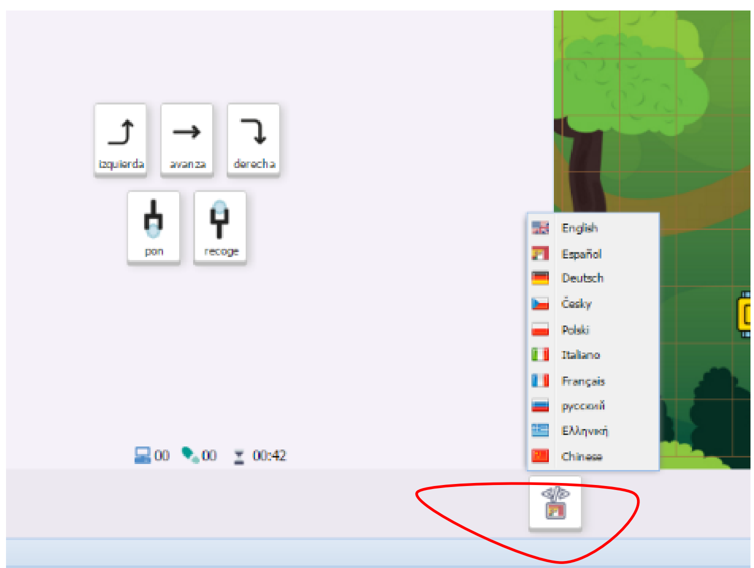
- The course sets **criteria and constraints** (for example, students may need to solve a problem with a limited number of steps or lines).
- Students look for the **best solutions** with the simplest, most efficient and robust code. The program can be tested on different mazes to see if it holds true under all conditions.
- Once students develop some proficiency, they can **design** their own mazes, problems and solutions.

To view the Engineering Design in the NGSS document in detail:

http://www.nextgenscience.org/sites/ngss/files/Appendix%20I%20-%20Engineering%20Design%20in%20NGSS%20-%20FINAL_V2.pdf

VOCABULARY, LANGUAGE AND PROGRAM SUPPORTS

- **Program and Story Line:** Text complexity (Lexile score) is about 620-850L, suitable for 3rd to 4th grade upwards. There is picture support for the story line.
- **YouTube videos:** demonstrate steps learned in the lesson. Links are listed within the lessons.
- **The Settings drop down menu** enables the user to adjust robot speed, choose colors and indentations and turn sound on and off.
- **Text size** can be adjusted for readability.
- **Instant feedback:** Karel's actions in the maze provide instant feedback.
- **Hints:** The user can select hints from the menu to help solve the problem.
- **Textbook:** The textbook is geared to teachers and advanced students. It provides more detailed explanation of functions and terminology.
- **Vocabulary:** many commands are Tier I or Tier II words that have a specific Tier III function. These are noted under each section.
- **Student Journal:** a journal is provided for concept and vocabulary review, and reflections on learning. It includes sketch pages to design programs while offline.
- **Language Options:** A Code Language button is located at the bottom center of the screen and can be toggled to one of several languages.



BACKGROUND-BUILDING AND SUPPORT ACTIVITIES

- **Hour of Code** (https://code.org/learn): As a warm-up to Karel, students can benefit by exploring free Drag and Drop programming games found at Hour of Code.

- **Act It Out**: Students can physically walk out the steps and turns in a program, especially effective in a room with a tiled floor or carpet squares. Students can work with a partner, with one person calling out commands and the other person acting them out.

- **Gameboard:** Using a Lego figure and centimeter or ½ inch square graph paper, draw the pathway and walk the steps and turns.

- **Map and Compass work:** An understanding of compass cardinal points will help students to orient Karel in the maze.

- **Paper and Pencil or Online Mazes** (caution: online mazes use the arrow keys differently than the way they are used in the program).

- **Student Journal Sharing**. Journaling provides an opportunity to reflect on learning and deepen understanding of concepts and procedures. It is a place to imagine new designs and programs. All of this can be shared as partners, small groups or whole class.
- **Failure is an Option.** After students have passed a level, have them change a line in their program that would make it fail. Rotate the students to a different work station. Can they find the error? This is a great team exercise.
- **Robots in Action.**
  - Bring a Roomba to clean the classroom. What "decisions" is the robot making?
  - Play with robotic toys and remote control vehicles. How are these controlled? Visualize the command sequences as lines of programming.
  - On-line videos. Many robotics companies post videos of their **industrial** robots in action, which are great examples of get and put commands. Robots are being developed for the **military and public safety** to navigate a hazardous situation, detect explosives, move supplies, and so forth. **Robotic arms** and other prostheses also use commands similar to those in Karel.
- **Video and Board Games.** Have students describe a scenario in one of their favorite video or board games in terms of commands and functions.

DEPTH OF KNOWLEDGE

Most problems in the lower levels have one solution given the parameters; a few problems can be solved with more than one pathway (Depth of Knowledge 1 and 2). The upper levels provide more opportunities to analyze and choose solutions (DOK 2 to 4). Using Creative Suite, DOK 3 and 4 level problems can be created and solved.

BLOOM'S TAXONOMY

- Application and Analysis: Students must analyze the maze and problem parameters to come up with a solution. Students immediately apply what they are learning at each stage by writing a program.
- Synthesis and Creation: Students can create mazes and their own problems and solutions using the Creativity Suite, bringing together all the skills they have learned.

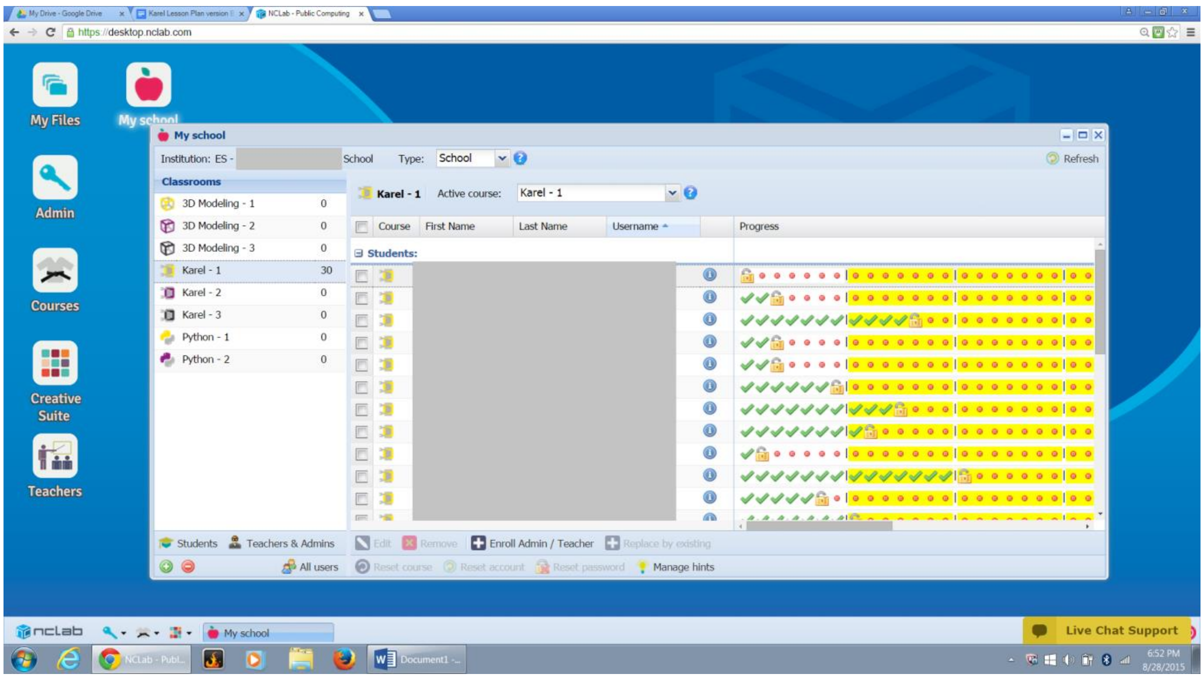
ENRICHMENT, REMEDIATION AND PROGRESS MONITORING

- Since the course is self-paced, students can move through the lessons based on their own rate of learning.
- Students must unlock the next levels, so it is not possible to race through or "cherry pick" the program without successfully completing each stage.
- Steps can be repeated at any time for review and reinforcement.
- Teachers should monitor and provide support as needed. At some point, most students will hit their own personal threshold level in which they aren't immediately successful. Point out the built-in hints and comment line prompts within the program. Follow up with discussions about what they learned from these hints.
- In a camp or workshop setting, it is important to build in physical breaks. Students tend to stay longer than they should in front of the computer.
- In any setting, encourage opportunities to interact and discuss progress.
- In Creative Suite, set challenges for students. For example: "Design a program that requires a repeat loop, at least two turns, and retrieving 4 objects."

ASSESSMENT

Assessment built into the program:

- Within each level, students get immediate feedback by trying out their program in the maze. The program will show what line is causing problems.
- Upon successful completion of a level, students will unlock the next level. Likewise, upon successful completion of each section, students will receive a certificate and unlock the next section.
- Teachers can monitor the progress of their students by clicking on the My School apple. This opens a new window.



Journals: A Student Journal is included in the course materials and can be used as a portfolio artifact.

Quizzes: Students can complete paper and pencil or online quizzes (in development).

Games as assessments: Students can create games using the Creative Suite and save them to their NCLab folders. One game assessment is included in this document for each Section.

Student Feedback: At the end of each level, students are asked to evaluate the level of difficulty by clicking on an EASY, MEDIUM or HARD button. This gives the NCLab designers valuable feedback for improving games.

KAREL JR UNIT 1

Karel 1 Overview: Car companies use robotic arms to spot weld automobiles on an assembly line. The military uses mobile robots to detect explosives. 3D printers print precise models of buildings, ears, and even pizza. What do all of them need? Instructions! The machines need to move, to pick up and place objects, to detect and move around obstacles. Many of their functions have to be repeated over and over again. In Karel 1, students learn how to direct the movements of Karel, how to pick up and place objects, and how to write repeat loops. They also learn the layout of the course, how to create their own programs, and the basic syntax of code writing.

SECTION 1: Students learn to guide Karel using remote control, switch Karel's commands into other languages, and guide Karel using the keyboard. They also know that the left panel: describes your task, shows game goals and limitations, shows the counters of steps and operations, and shows elapsed time.

INTRODUCTION TO CREATIVE SUITE: Students learn how to use the Creative Suite to create, save and publish their own Karel games and mazes.

SECTION 2: Students learn how to write programs using the commands go, right, left, get, put. They also know that to write one command per line, and that each commands start at the beginning of line.

SECTION 3: Students learn how to use the repeat loop. They also know that the repeat command must be followed by a number, the body of the loop is indented, and the loop can repeat one or more commands.

SECTION 4: Students learn how to figure out the body of a loop with certainty, write commands before and after a loop. They also know that to put commands after a loop, their indentation must be canceled.

SECTION 5: Students learn how to write programs that have multiple loops, and how to use nested loops. They also know that indentation increases when loops are nested.

LESSONS

Note: The best way to prepare for these lessons is to do them as a user either ahead of time or alongside the students. When you set up your teacher account, you will have access to the unlocked course, so that you can jump in on any level. You will also receive a link to answer keys for all levels.

INTRODUCTION TO THE COURSE (ABOUT 20 MINUTES)

In the very first session, allow for time to log in the students and show them where the course is located on the desktop. Demonstrate the log in steps and first lesson on a computer (for larger classes, attached to a projector or Smartboard if available).

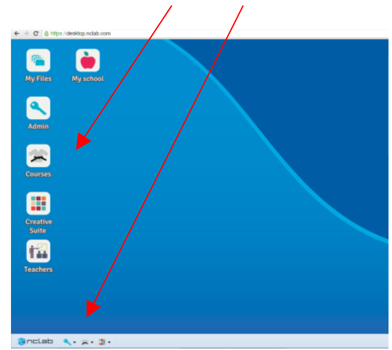
Background knowledge/Introductory Set/Purpose:

- Build background knowledge by showing a video of industrial robots, and discussing how the robot is controlled (movement, actions (welding, painting, picking a part and installing it)
- Do a warm-up activity such as playing with remote control cars or toy robots and discussing how they are controlled.
- Use information from the Preface and Introduction of the built-in textbook to introduce the history and purpose of Karel programming.
- The purpose of the whole Karel course (Karel 1 to 5) is to learn:
    - Algorithmic thinking
    - Typing single commands
    - Running and debugging programs
    - Using counting (repeating) loops, nested loops
    - Using conditional (if-else) statements
    - Using conditional (while) loops
    - Defining and using custom commands
    - Using functions that return values
    - Using local and global variables
    - Using basic operations with Python lists
    - Designing recursive algorithms
    - Solving advanced problems using the skills learned.
- In the first unit, Karel 1, students learn to guide the robot, type simple programs, recognize repeating patterns, and use the repeat loop.
    - Demonstrate how to log on and navigate the desktop. Provide names and passwords to the students.

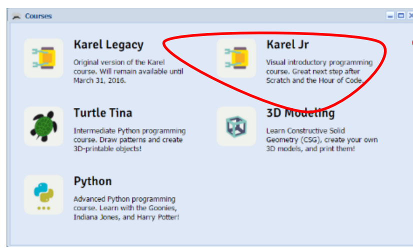
How to Log in and Navigate the Desktop:

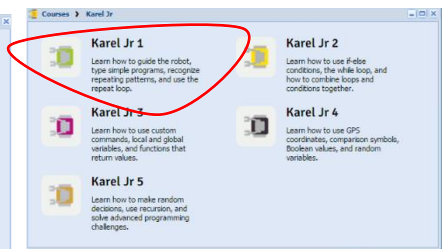
Step 1:

Log in to account https://desktop.nclab.com/. Select "Courses" (click on icon on the left side of the screen, or on the pull-up menu on the bottom bar).



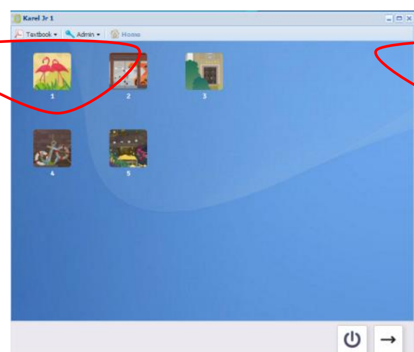
Step 2. Select Karel Jr and then Karel Jr 1.

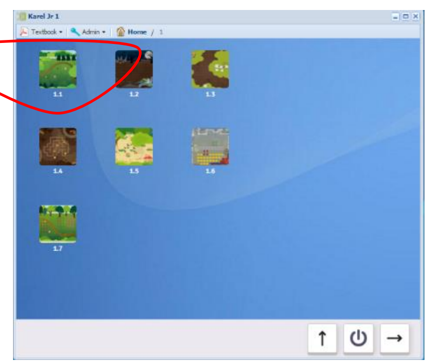




Select the Section 1, then Level 1.1.

Students will only have access to Section 1, Level 1.1 to start. The other levels will unlock as they progress.





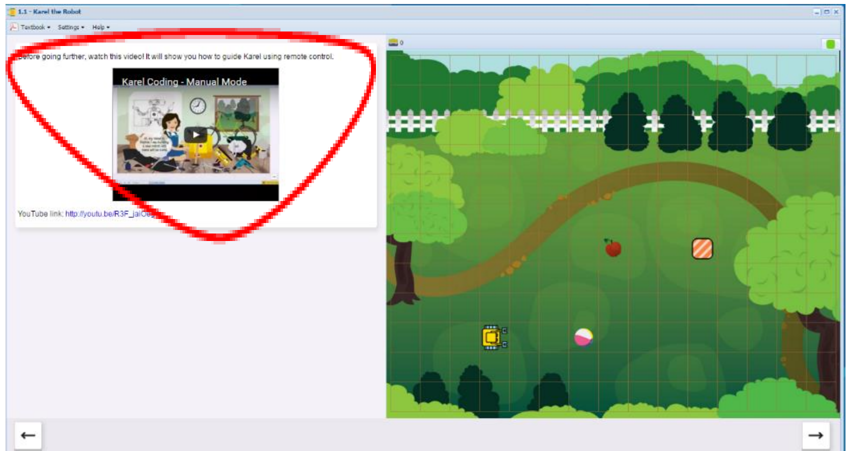
Each section and level starts with a screen that introduces the storyline for that section.

Each section includes videos that demonstrate the concepts and skills needed to complete that section.

The video for Section 1 (3 min. 34 sec.) demonstrates what will be learned in this section.

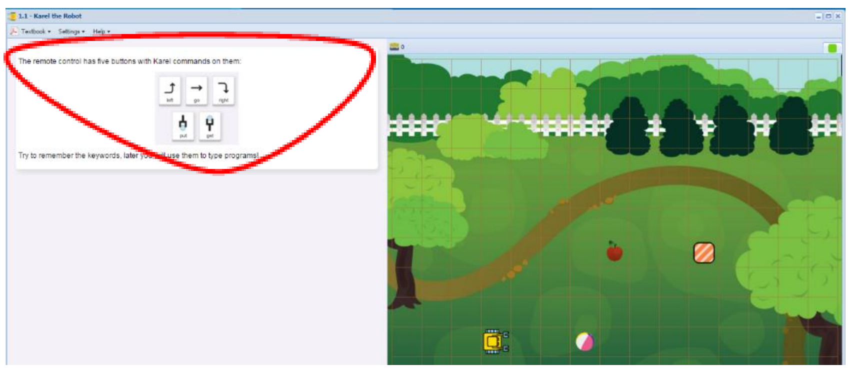
http://youtu.be/R3F_jaiOeg4

Play the video or demonstrate the steps on the following screens.

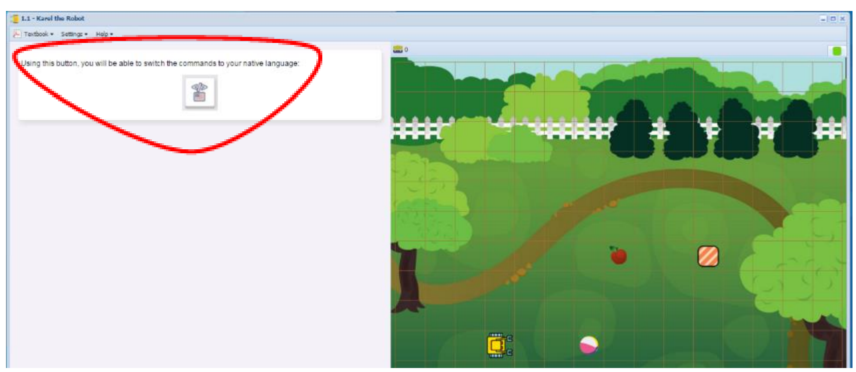


In Manual Mode, Karel can be controlled by pressing the buttons on the screen.

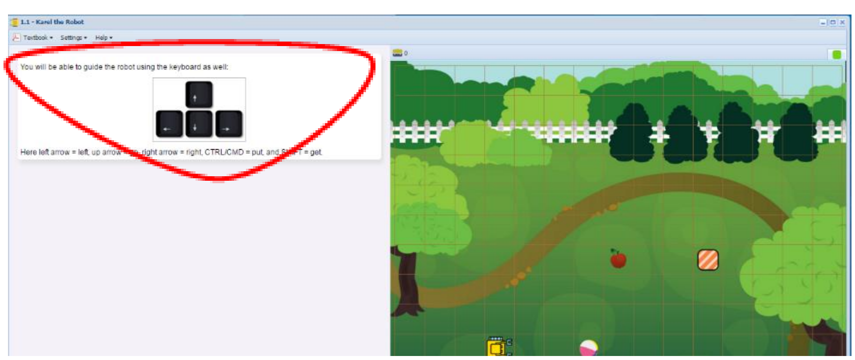
There are buttons for left, go, right, get, and put.



The next screen shows the native language button, which normally appears at the bottom center of the screen. When it is there, students can use the button to select one of several languages for the commands.



The other way to control Karel in Manual Mode is to use the keyboard.



Left arrow = left,

Up arrow = go

Right arrow = right

CTRL/CMD = put

SHIFT = get

This is the first game. Here is a guide to some of the features available on the screen.

A counter that displays the number of objects in Karel's

Colored tab. If there is more than one version of the game each one can be accessed by selecting its tab.)

Drop down menus for help functions, settings, and textbook sections.

Review the video at any time by pressing the red button

The left hand side contains instructions, hints and any restrictions, such as the maximum number of steps, operations, or lines of programming, the number and type of objects to collect or place.

Manual control buttons

Karel

Home square

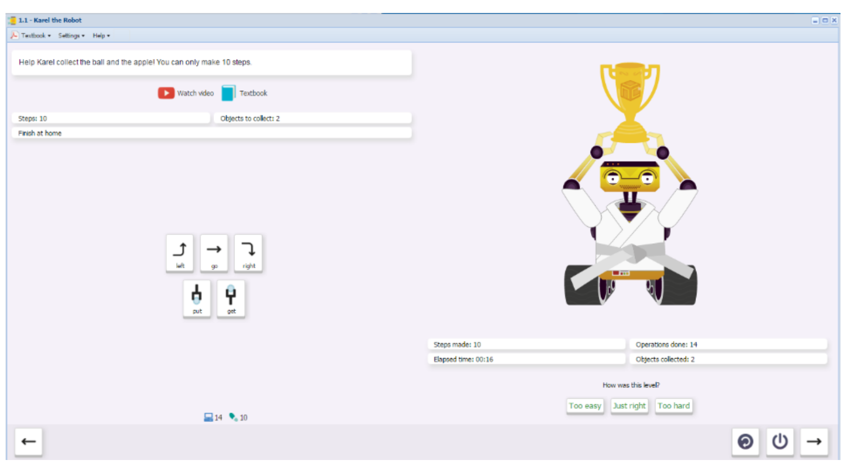
Counters for number of operations, steps, and elapsed time

Language selector

Reset button to reset the game

Students should try this on their own, using the buttons or keystrokes. Check to see if students are viewing the arrows from the robot's point of view.

Once successful, students will see this screen. They can rate the task and see their elapsed time, number of operations done and steps made.



SECTION 1: LEVELS 1.1-1.7

Objectives: Students learn to guide Karel using remote control, switch Karel's commands into other languages, and guide Karel using the keyboard. They also know that the left panel: describes your task, shows game goals and limitations, shows the counters of steps and operations, and shows elapsed time.

Vocabulary:

Command words: `go, left, right, get, put`

Directional commands (`go, left, right`) are always from the robot's point of view.

`go` advances the robot one step

`left` turns the robot to its left.

`right` turns the robot to its right.

Retrieving and placing objects (get, put)

`get` picks up an object

`put` places an object

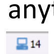
Tier I words used in programming: `home`, max, collect, object, step

Simple words have specific meaning in the context of programming and may need explanation

Home is the destination square, marked by red diagonal stripes which change to green when Karel approaches the square. The word `home` is also used in conjunction with commands.

Max may refer to maximum number of steps, operations, or programming lines.

Steps are the number of squares that Karel moves. The shoe icon  counts the number of steps.

Operations are anything that Karel does: move, turn, pick up or put down objects. The computer icon  counts the number of operations.

Objects are items placed in the maze. (The word "object" can have other connotations in programming that are not used here).

Time required:

Once students have learned how to log into the Desktop and select their course, most will complete Section 1 in about 30 minutes. This section requires no code writing and simply introduces students to the movements of the robot, and simple get and put commands. Most students will already be familiar with mouse and keyboard movement and game-based learning.

Prerequisite skills:

Introduction to the Course

Reading and writing: Students should be able to read text at a 3rd grade level

Computer skills: basic keyboard and mouse skills. From Section 2 onward, commands are typed.

Math: Coding at this level encourages math processing skills such as pattern recognition and problem solving. Numerical calculations are not needed.

Introduction (5 minutes)

Students will have been introduced to the program and the beginning screens of Section 1 during the Introduction to the Course.

Explain to students that they will be learning how to move the robot, pick up and put down objects, first with the mouse or keyboard buttons, then by creating a program with typed commands.

Students will find this section to be fairly simple and should move through it quickly.

Their assessment will be to create their own game once they have completed the section. The directions for using Creative Suite to create games are at the end of Section 1.

Individual/Group practice: (approximately 20 – 30 minutes)

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Levels 1.1-1.7 (Manual Mode)

1.1 Karel goes to the home square, collecting the ball and apple along the way.

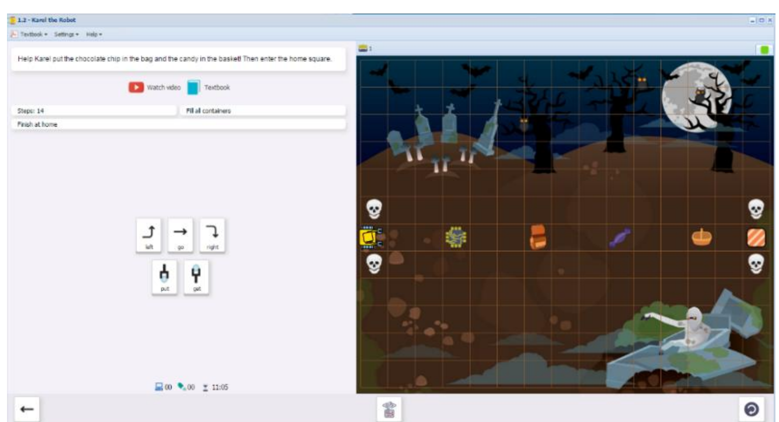
Number of steps: 10

Commands: `go, left, right, get`

1.2 Karel goes to the home square, collecting the chip and placing it in the bag, and collecting the candy and placing it in the basket.

Number of steps: 14

Commands: `go, get, put`

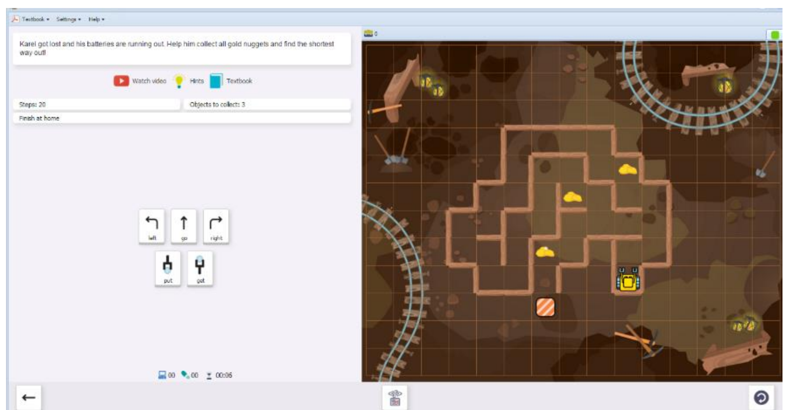


1.3 Karel goes through the corn maze to the home square, collecting one corn and one pumpkin on the way. There are multiple pathways, but only one with 25 steps.

Number of steps: 25

Commands: `go, left, right, get`

1.4 Karel collects all the gold nuggets in the maze and goes home. Students must figure out the most efficient path. They will learn to reverse direction by turning twice.



Number of steps: 20

Commands: `go, left, right, get`

1.5 Karel puts snakes and spiders into boxes and goes home.

Number of steps: 10

Commands: `go, left, right, get, put`

This level includes all 5 commands

1.6 Fire! (30 steps)

Karel moves 4 barrels from one side of the firewall to marks on the other side and goes home.

Number of steps: 30

Commands: `go, left, right, get, put`

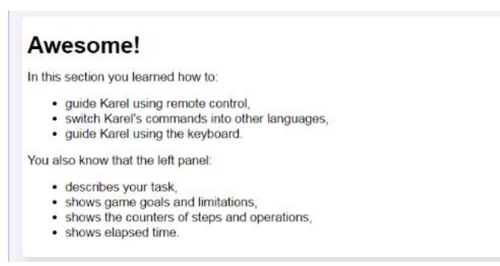
There are multiple solutions (pathways)

1.7 Karel collects 12 flowers for Sophia. He must negotiate the garden wall, and follow a certain pattern to minimize his steps.

Number of steps: 13

Commands: `go, left, right, get`

Upon completion of 1.7, students will see this screen that summarizes the concepts and skills learned in Section 1. They will receive their White Belt certificate on the next screen. Section 2 will be unlocked.



Suggested questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

Describe the movement commands used to move the robot. What are their limitations?

move forward (`go`, only one step at a time)

change directions (`left, right`, only 90 degrees)

move backward (Karel can't move backward, but he can turn around using `right/right` or `left/left`)

How can you plan the number of steps to stay less than or equal to the maximum allowed?

With a partner, discuss at least two different pathways through the maze to complete Fire!

What pattern was needed to complete Flowers within 13 steps? Would you have chosen this pattern without the fences to guide you?

Assessment:

Within the program itself, students receive a printable White Belt certificate upon successful completion of Section 1.

At the end of Section 1, introduce students to the Creative Suite (see below). Having students create a game using the suite is a good way to assess their progress, and it makes programming real! Allow a separate lesson block for teaching and practicing Creative Suite (about 50 minutes)

See the Assessment section for other journal and project ideas.

USING CREATIVE SUITE TO DESIGN KAREL MAZES AND GAMES

Creating mazes and games with the Creative Suite serves several functions and is strongly recommended as a course component.

- By stopping at the end of each section to create a game, students become active programmers.
- Since creating a game is open-ended, students of all abilities are free to make games as simple or complex as they desire within the given parameters.
- The games are an artifact that can be used as part of a portfolio for the course.
- Students have the opportunity to publish their games on the NCLab website.

BASIC INSTRUCTIONS:

- Click on “Creative Suite” from the menu on the left side of the Desktop.
- Click on “Programming”
- Click on “Karel the Robot”
- Programs can be written under the programming tab
- Mazes can be created under the designer tab
- Games can be created with the maze
- All files should be saved to the student/user folder on the NCLab Server
- Files can be edited at any time.

CARDS:

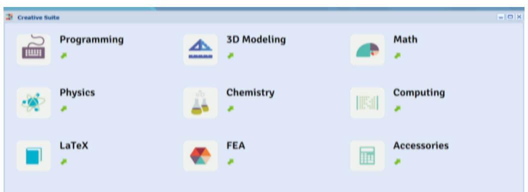
A set of printable instruction cards is included at the end of this lesson.

STEP 1: Start by showing students how to navigate to the folder

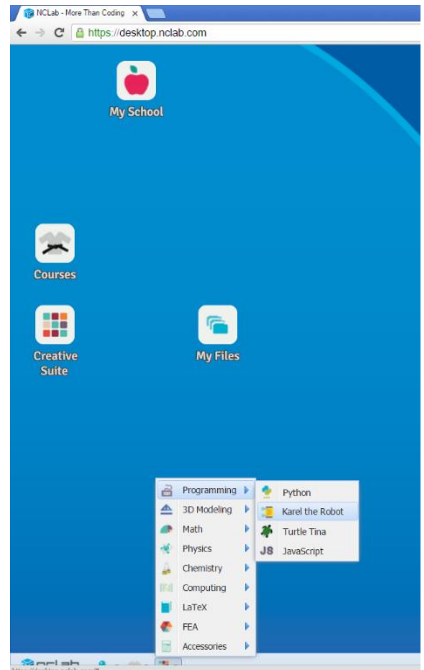
Select “Creative Suite” from the menu on the left side of the Desktop or from the pull-up menu at the bottom of the screen.

From the menu, select “Programming

From the next menu, select “Karel the Robot”



or



STEP 2: Exploring the different tabs

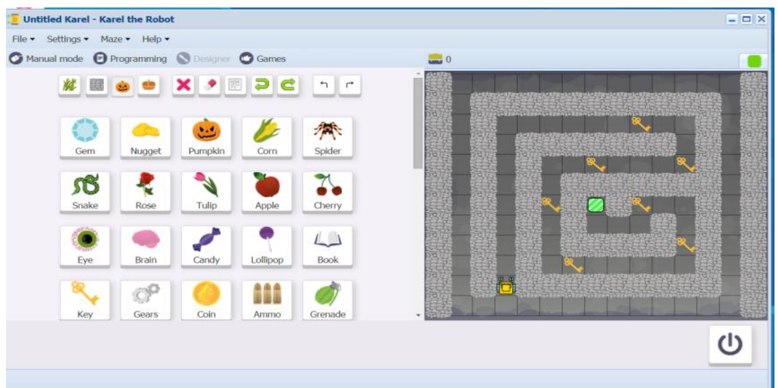
The screen opens in **Programming Mode** with a demo file that can be played. Note that it says Untitled at the top. The user is prompted to save the file before closing the screen.

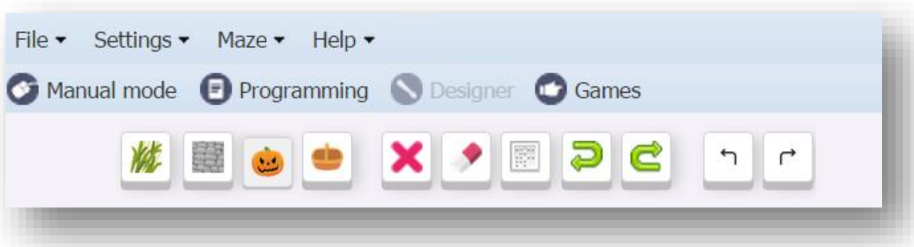
Explore the other heading tabs (Manual Mode, Designer, Games)

Manual mode allows the user to use keystrokes to navigate the maze, like they did in Section 1. Although this may seem like a just a precursor skill to be replaced by typed commands, manual mode simulates the way most machines are controlled: by pressing buttons.

Designer mode is used to create a maze.

Students will enjoy selecting objects, backgrounds and walls to build their own mazes. They should be given some time to play with this screen. **Card 1** explains how to use this screen.





The first four buttons allow the user to change the Theme, Obstacles, Objects and Containers

The next five buttons are editors: “Remove Object”, “Clear”, “Place Elements Randomly”, “Undo”, “Redo”

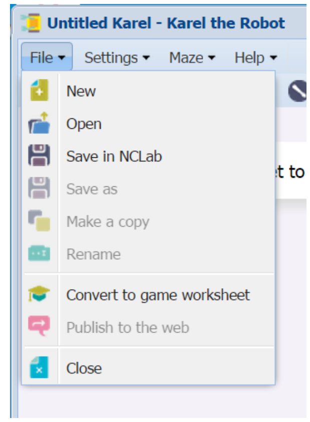
The last two buttons are the commands “Turn Left” Turn Right” which can be used to rotate the starting position of Karel.

Karel and the Home Square can be moved simply by drag and drop.

STEP 3: Show students how to save the maze to their NCLab folders.

Once students have created a maze in Designer Mode that they want to keep, they can save it to their NCLab folder.

Card 2 explains these steps.



To save the file:

- Pull down the File menu
- Select "Save in NCLab"
- The next screen will display the student's Home Folder.
- Create a file name for the maze and press OK.

STEP 4: Create a working game.

Students may want to create a copy of their maze and save it so that the same maze can be used for different games.

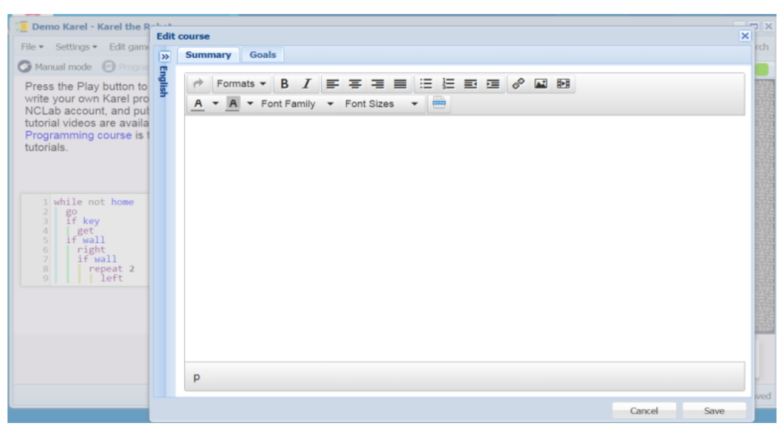
To do this, select "Create a copy" from the File menu and save it to a different file name.

To create a game from a maze, pull down the File menu and select "Convert to game worksheet". The program will prompt "Are you sure?". Select "Yes" to proceed.

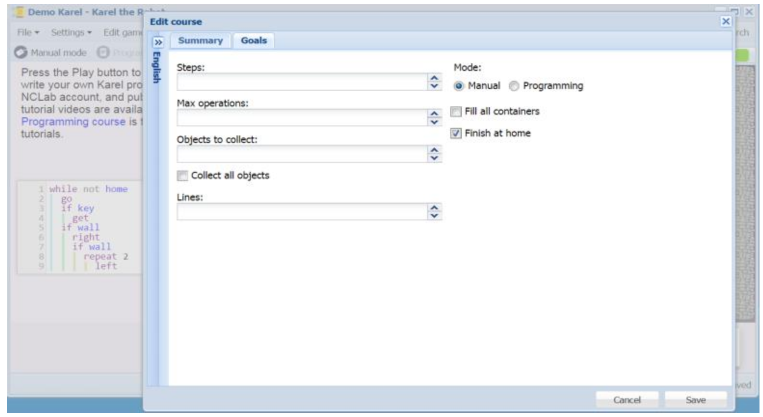
A new screen will pop up.

Select "Edit game" from the top menu to bring up the editing screen.

The **Summary** tab allows the student to create instructions. This is also an opportunity to create a short narrative or story line. It has several word processing features, including inserting a picture or video.



The **Goals** tab allows the student to create the goals of the game.



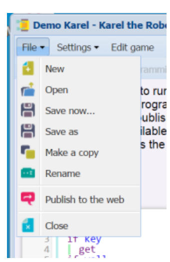
- **Mode:** The first time students create a game (i.e. at the end of Section 1), select **Manual mode**, since the students have not yet learned to write code.
- **Steps:** Set the number of steps (i.e. the number of squares Karel will step on in the maze). You may want to have students leave this blank for now and fill it in after they have run the game and see the results. Or you could specify the number of steps as part of the assignment.
- **Max Operations:** Set the maximum number of operations. This includes not only forward movement, but also turn, get and put commands. This may also be left blank initially.
- **Objects to Collect:** Either set the number of objects to collect, or select the box to collect all objects. At the beginning, it is best to set a limited number of objects.
- **Save:** Once the goals have been created, click the Save button.

Step 5: Running and Testing the Game

For this first game, go to the Manual Screen.

- Run the game.
- If the game is successfully completed, the exit screen will show this. If not, the "Try Again" screen will appear.
- The number of operations, steps, objects collected, and elapsed time will be displayed.
- Students can return to the editing screen to modify goals and summary at any time.

Step 6: Publish the Game (Optional)



- This is done from the main screen. If you do not see the File tab, close the game and reopen it.
- Select "Publish to the Web"
- Choose the Status option. For the first game, select "View and Run", unless students are collaborating on a game. Most students will not appreciate someone else editing their first game. They will however, benefit from feedback if other students have permission to run it.The ability to share will depend on what structures are set up at the school (see page 3 for suggestions)

Suggestions for the First Game Assignment:

To reflect what was learned in Section 1, the game should contain

- Opportunities to use the go, turn left, turn right, get and put buttons.
- A maze with a theme, walls that require Karel to turn left and right, objects to "get" and containers to "put" the objects in.
- A short narrative that describes the objectives of the game.
- A specified number of objects to collect.

If time is limited, keep the number of steps, objects and containers low.

To enhance student independence, keep the instructions to a minimum.

Encourage students to use the Help menu if they are not sure what to do.

For students who need extra support:

- Show them the next step needed.
- Print the help cards with visual support for each stage (Create a Maze, Save to Folder, Convert to Game, Edit Game, Test Game, Publish Game)
- Partner them with another student.
- Decrease the number of steps or objectives. (e.g. "Collect one spider and go home.")

For students who need a challenge:

- Have them create an imaginative narrative to include in their summary.
- Create more than one path to the home square that will meet the objective, or create paths that will meet the number of steps or operations and others that won't.
- Create more than one set of objects and containers, so that the player has to think about and choose which ones to use in order to keep the number of steps or operations under the maximum.

Section 1 Assignment

END OF SECTION 1: CREATE A GAME FOR KAREL (25 POINTS)

Create and publish a game for Karel in **Manual Mode**

.

- The game will require the player to **Go, Turn Left, Turn Right, Get and Put**. (5 points)
- The number of steps should be between __ and __ . (5 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Publish the game to your folder. Inform someone else about the game by providing the link on ____________________________________(5 points)

Support Cards for students who need step-by-step directions

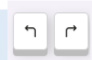
Card 1

1. CREATE YOUR MAZE.

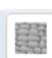
CLEAR THE SCREEN USING THE ERASER IF YOU WANT TO START FROM SCRATCH. 

SELECT A THEME (EXAMPLE GARDEN, FORTRESS) 

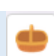
DECIDE WHERE YOU WANT KAREL TO START. DRAG HIM TO THAT SPOT. 

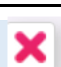
TURN HIM IF YOU NEED TO WITH THE ARROW KEYS. 

DECIDE WHERE THE HOME SQUARE WILL BE. DRAG THE SQUARE TO THAT SPOT. 

SELECT AND PLACE THE WALLS. 

SELECT AND PLACE THE OBJECTS YOU WANT KAREL TO COLLECT. 

SELECT AND PLACE THE CONTAINERS. 

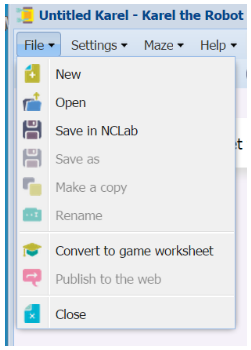
DELETE AN OBJECT 

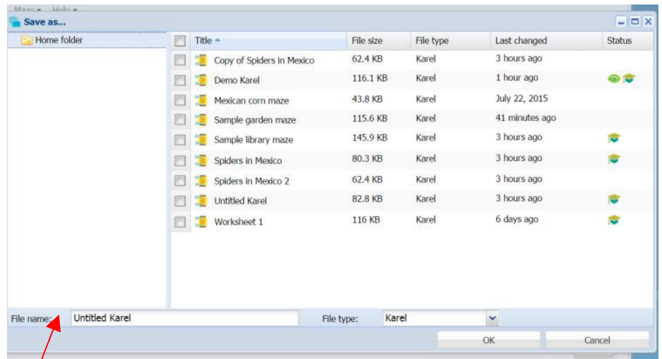
UNDO OR REDO YOUR LAST MOVE 

Card 2

2. SAVE YOUR MAZE TO A FOLDER.

Go to the File Menu.





Select Save in NCLab

Type your Maze name in the Name box.

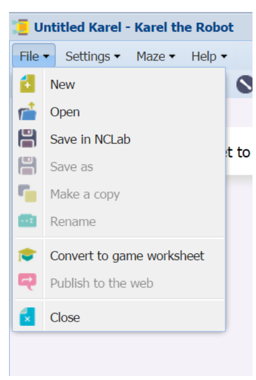
Press OK.

Card 3

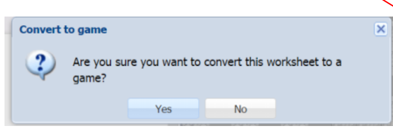
3. CONVERT TO GAME.

Go to File Menu.

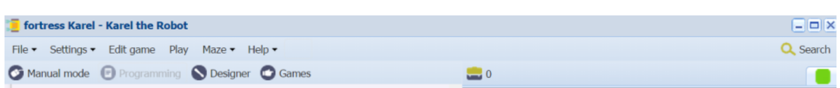
Select "Convert to game worksheet"



Select "Yes"



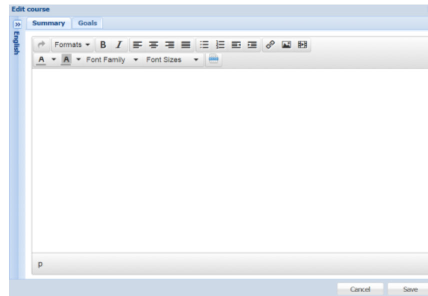
The "**Edit game**" Menu will now appear at the top of the screen.



Card 4

4. EDIT GAME.

Select “Edit game.”

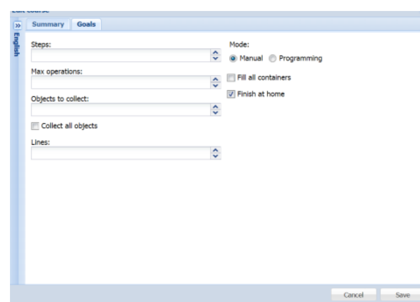


Write instructions for the player on this **Summary** screen.

What does Karel need to do?

Include the maximum number of steps or operations.

You can make this part of a story about Karel. *“Help! Karel needs seven gold keys to unlock the doors from the prison and escape the fortress!”*



Set the Goals on the **Goals** screen.

Select **Manual** or **Programming** (Manual for the Section 1 game).

Select or type in the number of operations and steps. *(if you don’t know these, you can fill them in after you have tested the game. The screen will tell you how many you took.)*

Select or type in the number of **objects** to collect, or check the box to collect all objects. Check the **container** box if the player has to fill all the containers.

Type in the maximum number of lines of programming if the player needs to write and run a program

(this can be done after you test the program)

******Don’t forget to Save!******

Card 5

5. TEST AND EDIT YOUR GAME

After you have saved your game, you can try it out to see if it works the way you want it to.

Press the **Play** button on the top menu to begin. If your game is Manual only, it will go to that screen. Otherwise, choose Programming or Manual to begin. Play the game.

If you win, the screen will show Karel with a trophy. It will list the number of steps, operations and objects collected, as well as how long it took.

If you fail, the screen will tell you if you took too many steps or operations or missed picking up some objects.

You can exit Play mode and go back to **Edit mode** any time to make changes to your Goals and Summary.

You can go to **Designer** to change elements in your Maze.

Always save after you are done

Card 6

6. PUBLISH YOUR GAME.

From the File menu, select “Publish to the web”

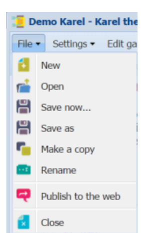
Choose one:

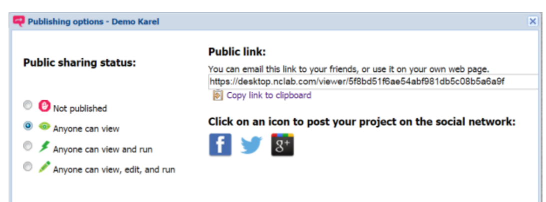
Anyone can **view**.

Anyone can **view and run** *(choose this one for the Section 1 game).*

Anyone can **view, edit, and run**.

Press **OK** when done.





SECTION 2: LEVELS 2.1-2.7

Objectives: Students learn how to write programs using the commands go, right, left, get, put. They also know that to write one command per line, and that each commands start at the beginning of line.

Vocabulary:

Programming terms: command, operation, lines of code

Command words: `go, left, right, get, put`

Directional commands (`go, left, right`) are always from the robot's point of view.

`go` advances the robot one step.

`left` turns the robot to its left.

`right` turns the robot to its right.

Retrieving and placing objects (`get, put`)

`get` picks up an object.

`put` places an object.

Tier I words used in programming: home, max, collect, object, step

Simple words have specific meaning in the context of programming and may need explanation

Prerequisite skills: Completion of Section 1 and familiarity with keyboard.

Time required: Time required will vary based on student ability and experience. Most students will complete this section in one to two hours.

Background knowledge/Introductory Set/Purpose:

Review: Explain the concepts of code and programming language. How do we define `get, go, left, put` and `right` in terms of programming?

Example: Go means "Move forward one step".

How to write code: show video (follow link on second screen of 2.1 or here https://youtu.be/s4EwI1p2wX0) which explains how to type the code, how to run the program either all at once or step by step, and the importance of writing code at the beginning of the line, spelling correctly and only writing one command on each line.

Big Idea: Why do we need to write programs for computers? Basically, computers need instructions for everything they do.

Purpose: Section 2 (Levels 2.1-2.7) introduces writing programs of one-command lines of code using `get, go, left, put,` and `right` to complete a series of tasks.

Direct Instruction and Modeling:

The video models how to type commands and execute the programs. Alternatively, Level 2.1 can be stepped through as a demonstration. Most sections include **step-through demonstration levels**. The program is already written and the **black arrow** at the bottom of the screen is used to step through each line of programming to see how the program works.

Students will always be given the maximum number of lines needed by the program, and the command words that must be included. Most programs can be written with less than the maximum number of lines. Some levels issue a challenge to students: “10 lines is good, 7 lines is awesome!”

Most levels have some of the code already written. Students may only need to type in code on lines marked with three dots . . . . The dotted lines focus students’ attention on the particular skill being taught and should make progress easier. In other cases, students have to insert several lines of code.

Remind students to read the instructional screens in each level.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 2.1-2.7

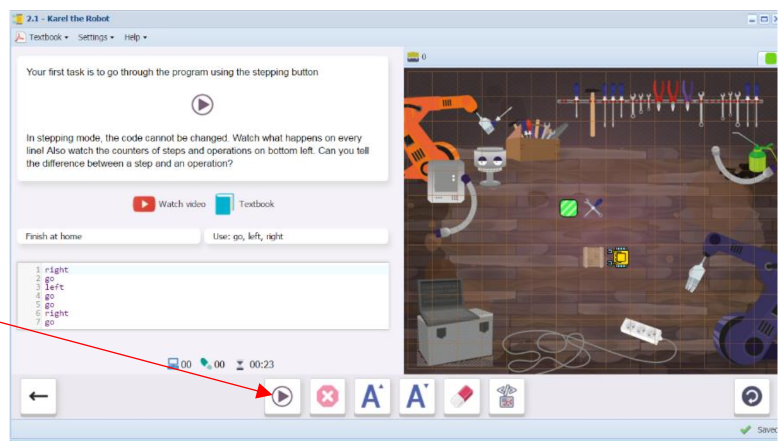
2.1 (Step Through Demonstration)

Karel goes to the home square.

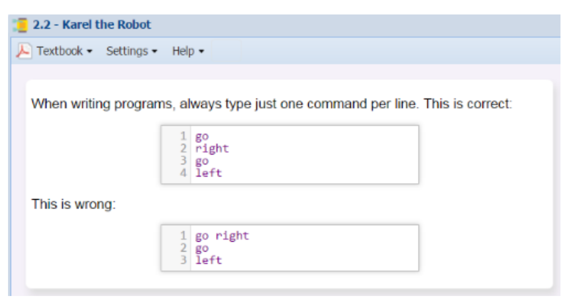
Commands: `go, left, right`

Students step through the code, one line at a time, by pressing the black arrow. They will observe what Karel is doing in response to the commands.

Question: Can you tell the difference between a step and an operation?

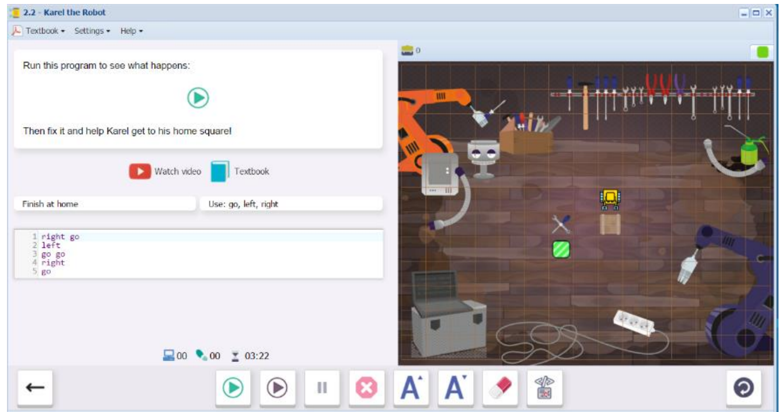


2.2 2.2 opens with an instructional screen explains that only one command can be written on each line and demonstrates the right and wrong way to write the commands.

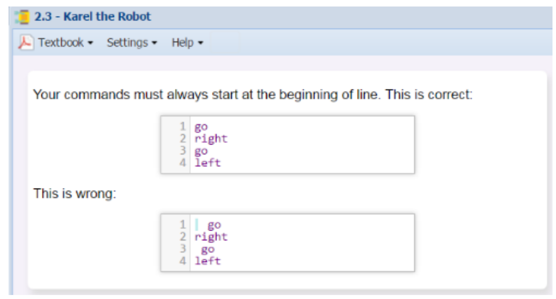


Commands:, `go, left, right`

Students will run the program first to see the effect of the syntax error, then repair the code and run the program again. There are two lines with two commands on them: the second command must be moved to its own line.

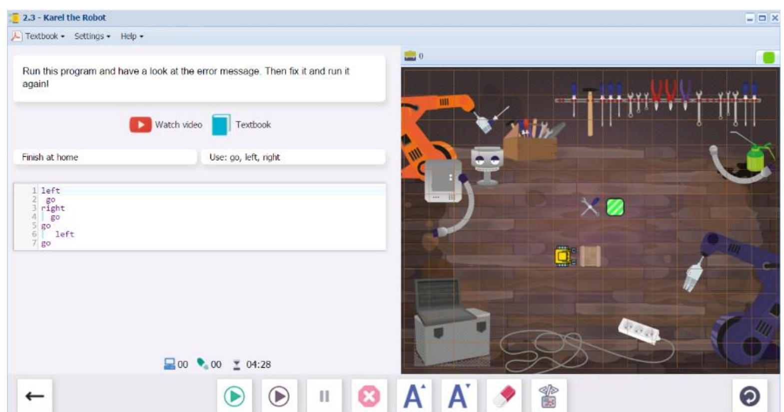


2.3 2.3 opens with an instructional screen explains that commands must always start at the beginning of the line, and demonstrates the right and wrong way to write the commands.



Commands: `go, left, right`

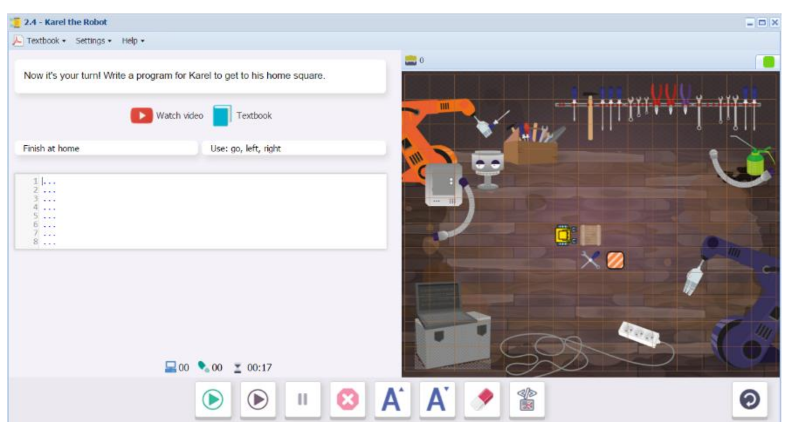
Students will run the program first to see the effect of the syntax error, then repair the code and run the program again. There are several lines with improper indentation: students repair these lines and run the program again.



2.4 Students write code that gets Karel to the home square.

Number of lines: 8

Commands: `go, left, right`

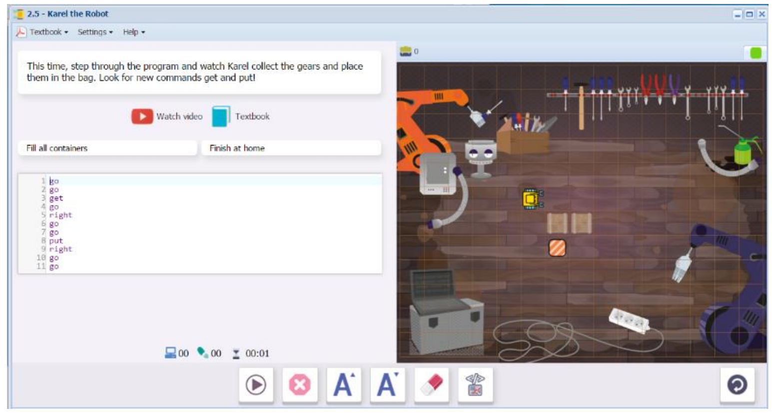


2.5 (Step Through Demonstration)

This level demonstrates the get and put commands.

Commands: `go, left, right, get, put`

Students step through the code, one line at a time, by pressing the black arrow. They will observe what Karel is doing in response to the commands.



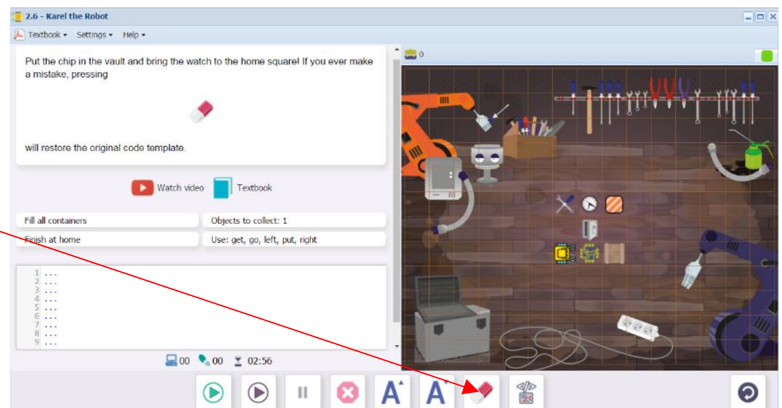
2.6 Karel must pick up the chip and put it in the vault, then pick up the watch and go home.

Lines: not limited, 9 lines are sufficient.

Commands: `get, go, left, put, right`

Students start by replacing all the dots with the eraser (clear code), which erases all the code on the screen below.

Then they write their own program to complete the tasks and send Karel home.

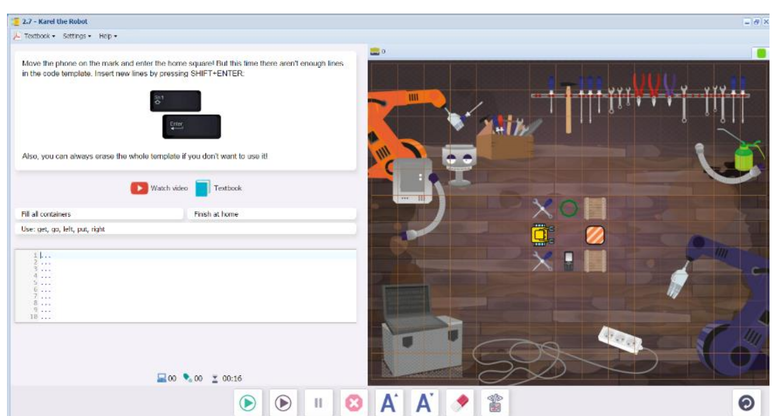


2.7 Karel must move the phone on the mark and enter the home square.

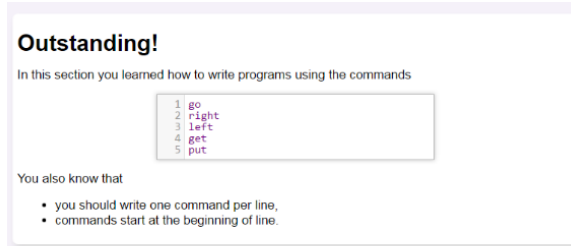
Lines: not limited. 14 lines are sufficient.

Commands: `get, go, left, put, right`

This time there are only 10 lines in the code template, which won't be enough. The instruction screen explains how to insert new lines by pressing SHIFT+ENTER:



Upon successful completion of 2.7, students will see this message, summarizing the skills and concepts learned in Section 2. On the following screen, they will receive their next certificate. Section 3 will now be unlocked.



Possible questions for post-session discussion:

Big Question: Why do we need to write programs for computers?

What parts were easy to do? What was frustrating?

What real life tasks could a robot do with these commands?

What real life problems could be solved by programming a computer?

What is the difference between steps and operations? (Steps are the number of squares that Karel moves. Operations includes all the things he does – (`go, left, right, get, put`).

Assessment:

Students will receive a printable "Yellow Belt" certificate upon completion of Section 2. See Assessment section for journal and project ideas.

Suggested Game Assessment:

Number of programming lines will vary. The number of lines can be specified: for example, between 10 and 25 lines. Inform students where they will share their game. Remind them that the names of the objects must be used as sensor words when writing the program.

END OF SECTION 2: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in **Programming Mode**.

- Create a maze with a theme, walls, objects and containers. (10 points)
- The game must include steps that can be solved by using the commands `get`, `go`, `left`, `put` and `right` in the program. (5 points)
- The number of **programming lines** should be between __ and __ . (5 points)
- When editing the game, write the objectives of the game under the **Summary** tab. Include a storyline that relates to your maze. (8 points)
- Set the goals under the **Goals** tab. (7 points).
- Test the game by running the program. Edit as needed. (10 points)
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 3: LEVELS 3.1-3.7

Objectives: Students learn how to use the repeat loop. They also know that the repeat command must be followed by a number, the body of the loop is indented, and the loop can repeat one or more commands.

Vocabulary:

Programming terms: repeat, loop, nested loop, body, syntax, syntax error

Command words: `get, go, left, put, repeat, right`

Repeat is written on its own line as `repeat x`, where x = the number of times the command is to be repeated.

Body: the body contains the commands to be repeated. The commands are written on the lines following the Repeat command, indented two spaces.

Loop: A set of commands repeated a given number of times.

Nested loop: A loop that is within another loop.

This is a good time to introduce some of the terms used in programming. Refer to the online textbook under Section 5 Programming for details.

Algorithm: a series of logical steps that leads to the solution of a task. Students may be familiar with algorithms used in operations such as subtraction and long division.

Logical error: a mistake in an algorithm. Planning helps reduce the number of errors.

Computer Program: An algorithm written using a programming language.

Syntax: the way a command line is written.

Syntax error: a mistake in spelling, operators, indentations, spaces

Tier I words used in programming: home, max, collect, object, step

Simple words have specific meaning in the context of programming and may need explanation

Time required:

Time required will vary based on student ability and experience. Most students will complete this section in two hours.

Prerequisite skills:

Completion of Section 2.

Background knowledge/Introductory Set/Purpose:

Explain the concepts of repeat loops. Remembering that `go` means "Move forward one step", how many lines of commands would it take to move Karel forward 10 steps? (10 lines) Instead of writing the command “Go” once on ten separate lines, we can use a repeat command and then type the Go command only once.

Warm up activity: practice walking out a set of commands such as:

Go 5 steps.

Turn left.

Go 2 steps.

Pick up the book.

Turn around.

Go 10 steps.

Put the book on the shelf.

Students could write out short routines for each other that include repeated steps. This could be expanded into a mini-treasure hunt (for example, by repeating the steps exactly, they find a wrapped candy).

In real life, we might want our computer or robot to do something over and over again. This is why we write loops in programming. A loop (or cycle) just repeats a command a certain number of times.

Big Idea: What are examples of repeated loops in real life (human, computer, robot or otherwise)?

Review vocabulary.

Direct Instruction and Modeling:

Show video on the third screen of 3.1 or by following this link:

http://youtu.be/GwFT25bHWlg

(7 minutes, 53 seconds). The video explains how to build a repeat loop.

Level 3.1 can also be modeled to the class. It is a step-through demonstration level.

At this stage, programming requires some thought and planning. Emphasize the importance of studying the tasks and the layout before starting to type. What tasks are cyclic and can be written as loops? How many times are these loops repeated? Which way is the robot facing at the beginning and end of each loop?

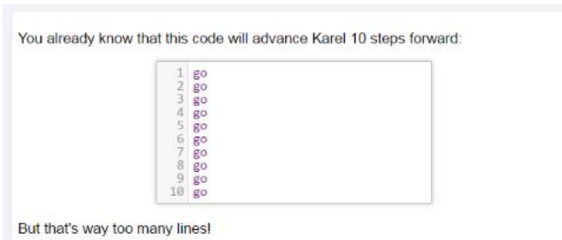
The repeat number has been included in the required command words as a hint for the number of times an action should be repeated.

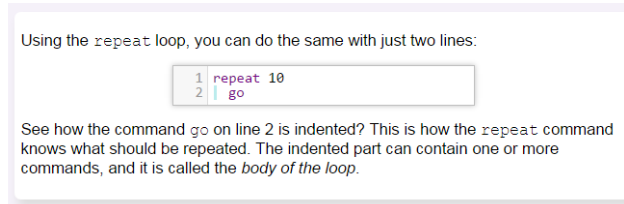
Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 3.1-3.7

3.1 Two instructional screens, followed by a step-through demonstration video.

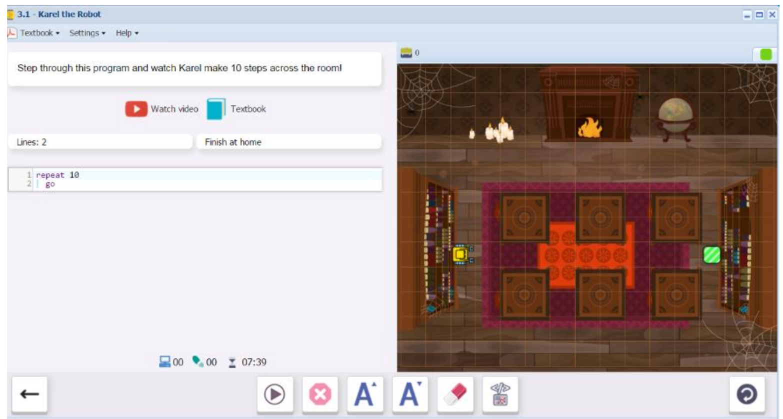




Karel moves 10 steps to go home.

Lines: 2

Commands: go

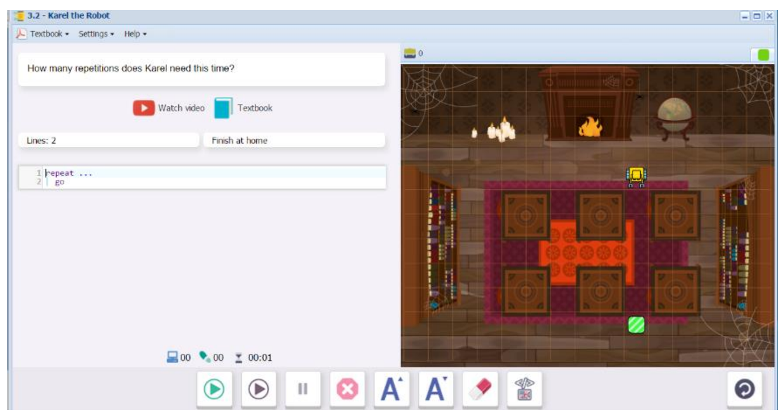


3.2 Karel moves x steps to go home.

Lines: 2

Commands: go

Students fill in the number of steps on the first line of the repeat loop.

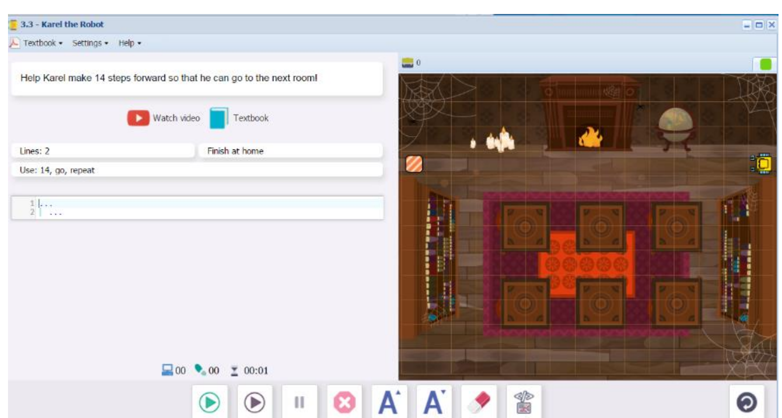


3.3 Karel moves x steps to go home.

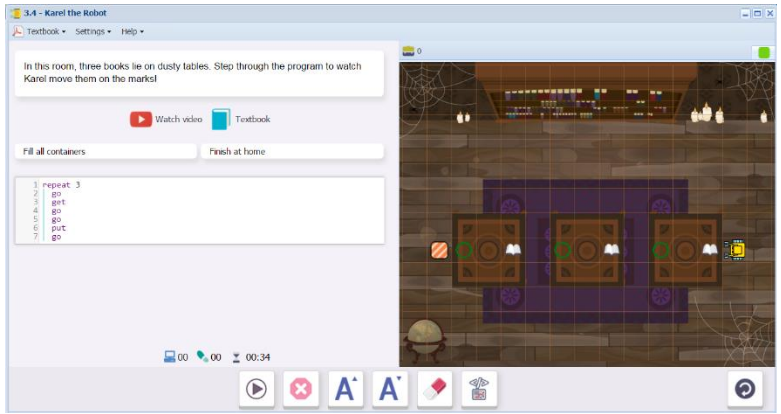
Lines: 2

Commands: go

Students write the complete repeat loop.



3.4 Step-through demonstration level

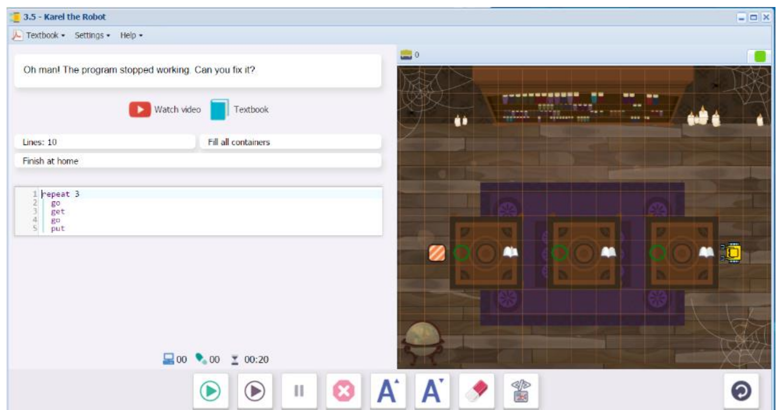


Karel repeats three sets of commands, picking up books, setting them on the marks, and moving forward.

Lines: 7

Commands: `go, get, put`

3.5 Repair the program



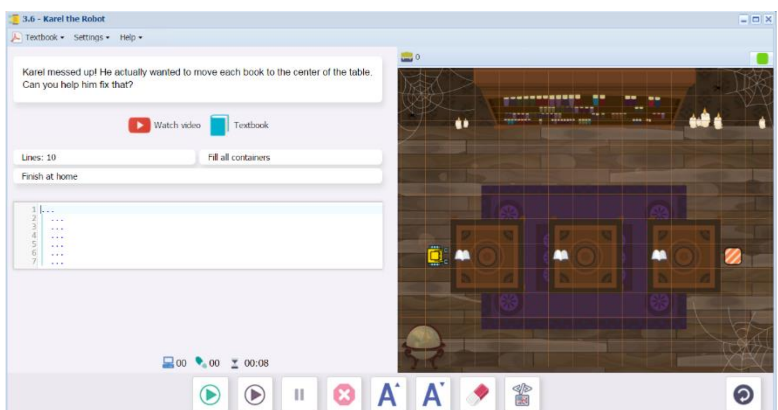
Karel is doing the same set of tasks as in 3.4, but there is an error in the program.

Lines: 7

Commands: `go, get, put`

Students run the program first, then insert lines to correct the error.

3.6 Karel does a similar routine, but places the books on the center of the tables.



Lines: 10

Commands: `go, get, put`

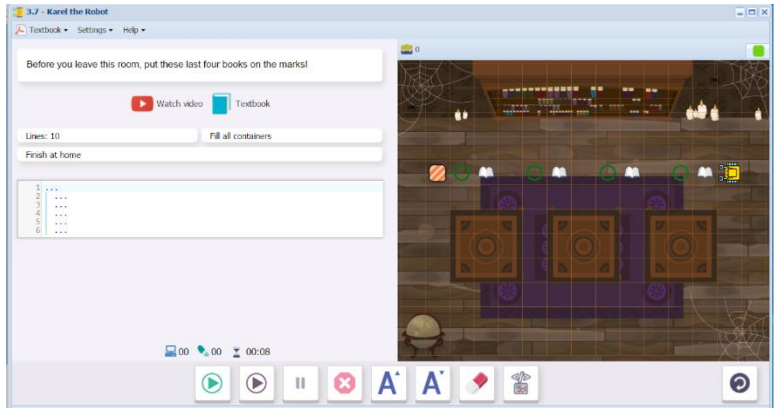
Students write the complete repeat loop.

3.7 Karel does a similar routine, collecting and placing books.

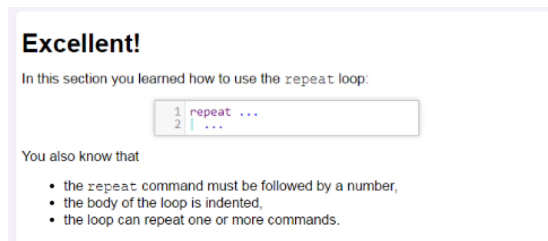
Lines: 10

Commands: `go, get, put`

Students write the complete loop.



Upon successful completion of 3.7, students will see this message, summarizing the skills and concepts learned in Section 2. On the following screen, they will receive their next certificate. Section 4 will now be unlocked.



Questions for post-session discussion:

What are the benefits of writing loops into programs? What are some of the pitfalls?

What real life repeated tasks could a robot or computer do with these commands?

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. Students will receive a printable “Yellow Belt of Second Degree” certificate upon completion of Section 3. See Assessment section for journal and project ideas.

Suggested Game Assessment:

Number of programming lines will vary. A suggestion is between 6 and 15 lines. Inform students where they will share their game.

END OF SECTION 3: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in **Programming Mode**.

- Create a maze with a theme, walls, objects and containers.(10 points)
- The game must include patterns that would be best solved by using a repeat loop. Programming must include the commands `get, go, left, put, repeat and right.` (6 points)
- The number of **programming lines** should be between __ and __ . (5 points)
- When editing the game, write the objectives of the game under the **Summary** tab. Include a storyline that relates to your maze. (7 points)
- Set the goals under the **Goals** tab. (7 points).
- Test the game by running the program. Edit as needed. (10 points)
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 4: LEVELS 4.1-4.7

Objectives: Students learn how to figure out the body of a loop with certainty, write commands before and after a loop. They also know that to put commands after a loop, their indentation must be canceled.

Vocabulary:

Programming terms: repeat

Command words: `go, left, right, get, put, repeat`

Key words:

Time required: Time required will vary based on student ability and experience. Most students will complete this section in about 1 hour.

Prerequisite skills:

Completion of Section 3.

Background knowledge/Introductory Set:

Section 4 builds understanding of the repeat loop that was introduced in Section 3. Students should pay attention to indentations that indicate which lines belong in the body of the repeat loop.

Repeated patterns are usually part of a bigger program. An example might be getting the ingredients together to make bread. You prepare the water, yeast, salt and sugar. Then you measure out 4 cups of flour, one cup at a time. Finally, you mix the ingredients together. The repeat loop of measuring the flour is embedded in the larger procedure of making bread.

Big Idea: Think of other examples that include a repeated set of steps (exercise routines, practicing a set of math problems, planting a row of seeds, clipping a fence to a post in three places, driving several miles between an on-ramp and an exit on the highway, etc.). How could a computer or robot be involved in these routines? What kind of program would it take?

Direct Instruction and Modeling:

Section 4 is a continuation of Section 3, and does not require much prior instruction. The first screen of 4.1 can be discussed as a check on understanding of the beginning and end of a repeated pattern. Special attention should be paid to Karel's orientation. Is he facing the same way at the beginning of each loop?

Review the syntax: the body of the repeat loop must be indented 2 spaces.

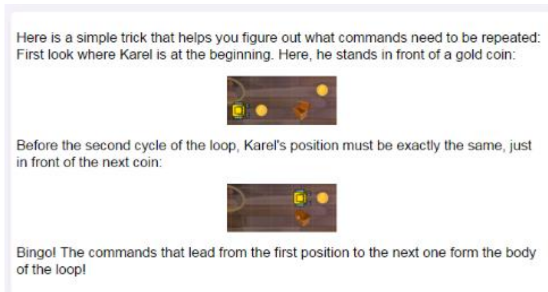
The step-through demonstration levels are 4.3 (writing steps preceding the loop) and 4.5 (writing steps following the loop).

Challenge students to come up with 15, 16 or 17 line solutions to 4.7, even though it can be passed with a longer program.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 4.1-4.7 (All previous commands may be needed but not necessarily listed under each lesson. Defined objects are listed)

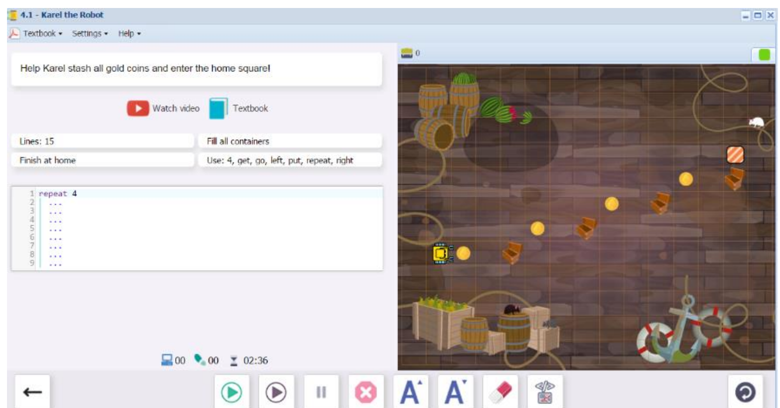


4.1 Karel collects all the coins and places them in containers, ending at the home square.

Commands and keywords: 4, get, go, left, put, repeat, right

Lines: 15

Students practice writing loops that include turns.

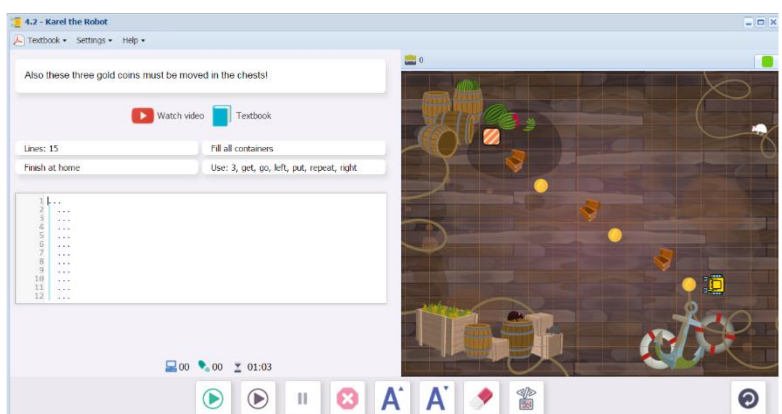


4.2 Karel collects all the coins and places them in containers, ending at the home square.

Commands and keywords: `4, get, go, left, put, repeat, right`

Lines: 15

Students practice writing loops that include turns, this time without clues.



4.3 Demonstration Level: Karel collects all the coins and places them in the containers, ending at home.

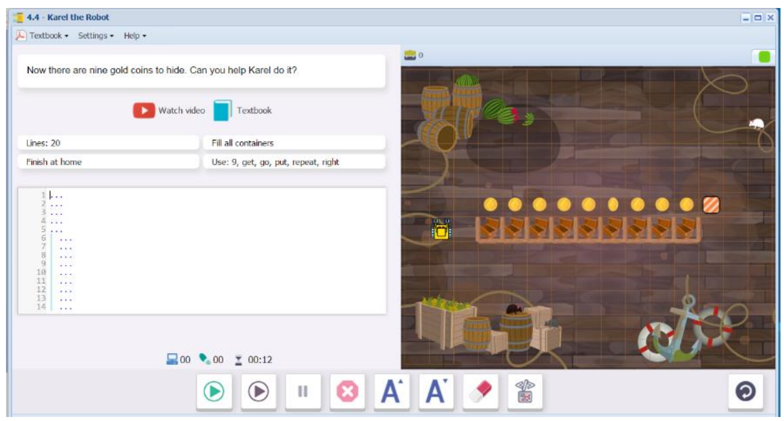
This demonstration shows a repeat loop **preceded** by a set of commands outside of the loop.

4.4 Karel collects all coins and places them in the containers, ending at home.

Commands: `9, get, go, put, repeat, right.`

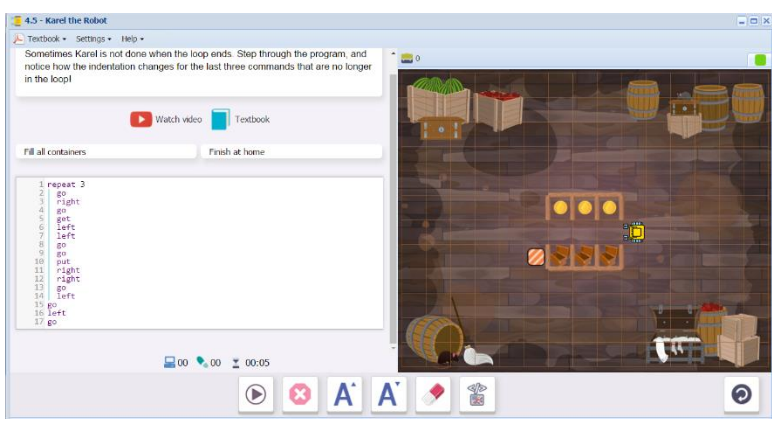
Lines: 20

Students practice writing a loop preceded by a set of commands, similar to 4.3.



4.5 Demonstration level: Karel collects all the coins and places them in the containers, ending at home.

This demonstration shows a repeat loop **followed** by a set of commands outside of the loop.

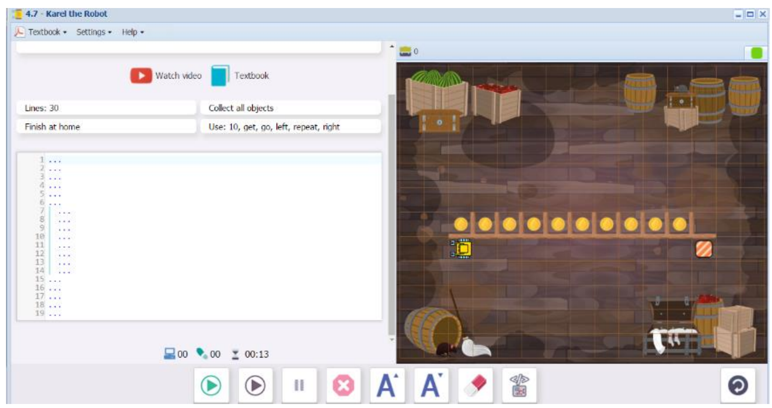


4.6 Karel collects all the coins and places them in containers, ending at home.

Commands and keywords: `4, get, left, put, repeat, right`

Students practice writing a loop followed by commands outside of the loop, similar to 4.5.

4.7 Karel collects all the coins and places them in containers, ending at home.

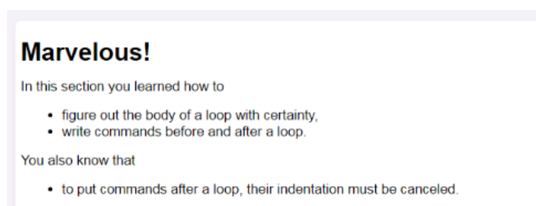


Commands and keywords: `10, get, go, left, repeat, right`

Lines: 30

Student practice writing a loop that is both preceded and followed by commands outside the loop.

NOTE: This level challenges students to solve the puzzle in fewer lines (17, 16, or 15 lines). In order to do this, they will need to look for other patterns that can be written as loops.



Upon successful completion of 4.7, students will see this message, summarizing the skills and concepts learned in Section 4. Section 5 will now be unlocked.

Possible questions for post-session discussion:

In 4.7, there were several possible solutions. Compare your solutions.

What indentation rules did you learn regarding repeat loops and commands that precede or follow the loops?

Assessment: Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. Students will receive a printable “Yellow Belt of Third Degree” certificate upon completion of Section 4. See Assessment section for journal and project ideas.

Suggested Game Assessment: Number of programming lines will vary. A suggestion is between 6 and 20 lines. Inform students where they will share their game.

END OF SECTION 4: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in **Programming Mode**.

- Create a maze with a theme, walls, objects and containers that includes repeated sections. (10 points)
- Programming must include the commands `get, go, left, put, repeat and right.` (6 points)
- The number of **programming lines** should be between __ and __ . (5 points)
- When editing the game, write the objectives of the game under the **Summary** tab. Include a storyline that relates to your maze. (7 points)
- Set the goals under the **Goals** tab. (7 points).
- Test the game by running the program. Edit as needed. (10 points)
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 5: LEVELS 5.1-5.7

Objectives: Students learn how to write programs that have multiple loops, and how to use nested loops. They also know that indentation increases when loops are nested.

Vocabulary:

Programming terms: `repeat`

Command words: all previous words

Sensor words: `pearl` (Karel has an extensive library of sensors)

Time required: Time required will vary based on student ability and experience. Most students will complete this section in about 1 hour.

Prerequisite skills: Completion of Section 4.

Background knowledge/Introductory Set:

Karel uses the repeat loop to repeat operations a certain number of times. These operations are often made up of other repeated operations. We call these **nested** loops. Think of how you may have modeled multiplication and division: equal groups or equal shares, arrays, areas, repeated addition and subtraction. Look for these patterns in Karel's tasks.

A gardener plants 5 rows of tomato plants (main loop) with 4 plants in each row (nested loop).

A robot fastens screws in 4 places along the edge of a piece of sheet metal. Each screw is turned 6 times.

Big Idea: Repeated patterns or sequences often contain smaller repeated sequences within them.

Direct Instruction and Modeling:

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 5.1-5.7

5.1 Step-through demonstration level: Multiple loops.

Karel collect 5 pearls, ending at home.

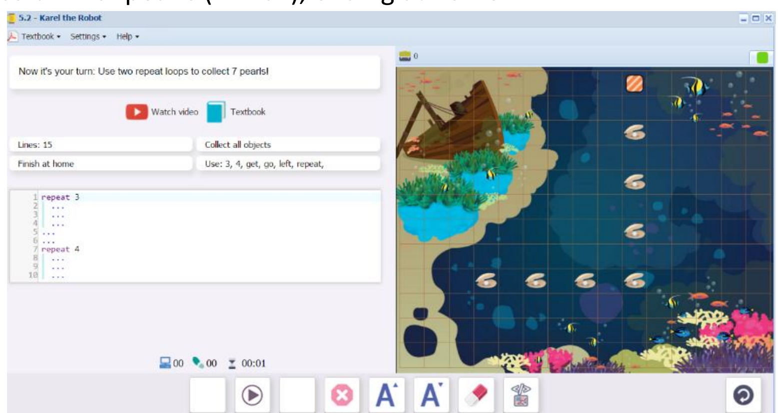
This level demonstrates the use of two repeat loops: one to go to the location of the pearls, and the other to collect the pearls.

5.2 Karel collects one row and one column of pearls (7 in all), ending at home.

Lines: 15

Commands and keywords: `3, 4, get, go, left, repeat.`

Students create 2 loops: one for the row, and the other for the column.

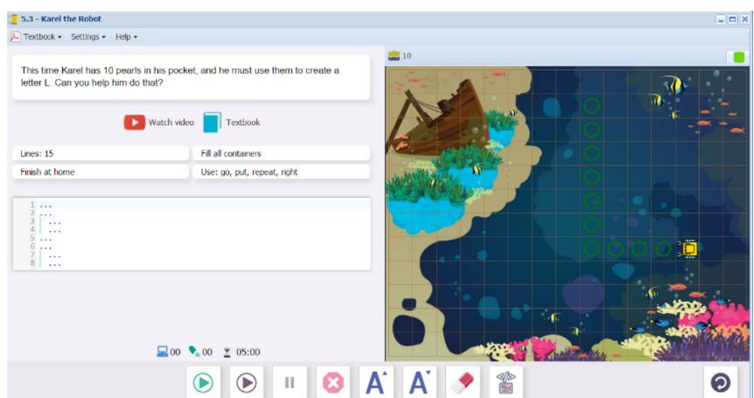


5.3 Karel places the 10 pearls that are in his pocket, ending at home.

Lines: 15

Commands and keywords: `go, put, repeat, right`

Students create two repeat loops, similar to 5.2.



5.4 Karel picks up 5 pearls and places them in the fishing nets, ending at home.

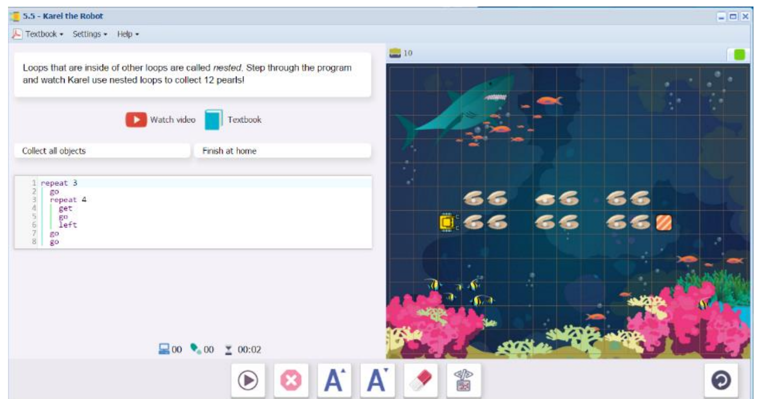
Lines: 20

Commands and keywords: `get, go, left, put, repeat, right`

Students create two repeat loops, preceded by a set of commands.

5.5 Step-through demonstration level: Nested loops

The pearls are in 3 groups of 4. The inside loop collects the 4 pearls; the outside loop repeats this procedure 3 times.



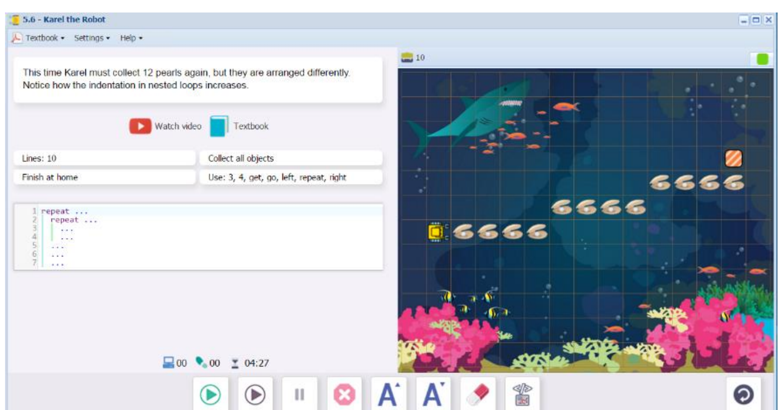
5.6 Karel picks up 3 lines of 4 pearls each, ending at home.

Lines: 10

Commands and keywords: `3, 4, put, go, left, repeat, right`

Students create an inner loop to collect the pearls, and an outer loop to do this 3 times.

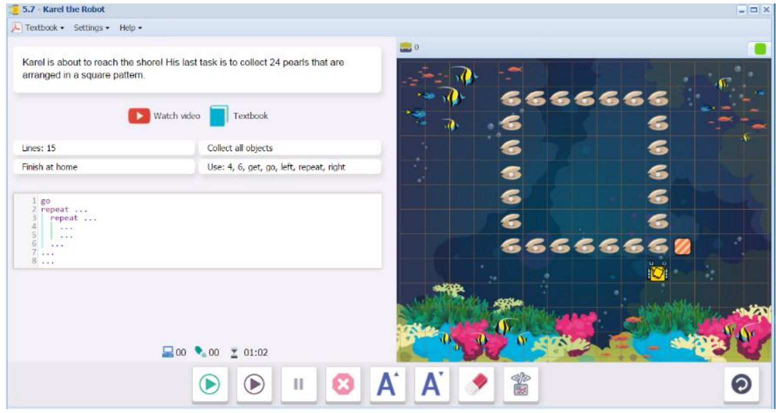
The indentation increase by 2 for each loop.



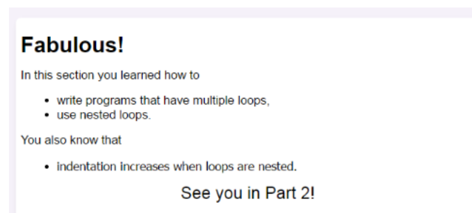
5.7 Karel collects 24 pearls in a square pattern, ending at home.

Lines: 15

Commands and keywords: `4, 6, get, go, left, repeat, right`



Upon successful completion of 5.7, students will see this message, summarizing the skills and concepts learned in Section 5. On the following screen, they will receive their next certificate. Karel 2 (Unit 2 of the Karel Jr Course) is now unlocked.



Questions for post-session discussion:

What numerical operations are similar to these nested loops? (multiplication/division)

What indentation rules must be followed with nested loops?

Think of some real-life scenarios that operate like nested loops. (Any repeated sets of operations, such as planting several rows with the same number of plants in each row.)

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. Students will receive a printable "Yellow Belt of Fourth Degree" certificate upon completion of Section 5. See Assessment section for journal and project ideas.

After completing Karel 1, students will be ready to start Karel 2 and learn more advanced programming skills.

Suggested Game Assessment:

Number of programming lines will vary. A suggestion is between 6 and 20 lines. Inform students where they will share their game.

The maze must include nested loops, similar to those in the instructional levels (groups of objects in clusters, rows of objects, etc).

END OF SECTION 5: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in **Programming Mode**.

- Create a maze with a theme, walls, objects and containers that includes repeated groups of items. (10 points)
- Programming must include the commands `get, go, left, put, repeat and right,` and included nested repeat loops. (6 points)
- The number of **programming lines** should be between __ and __ . (5 points)
- When editing the game, write the objectives of the game under the **Summary** tab. Include a storyline that relates to your maze. (7 points)
- Set the goals under the **Goals** tab. (7 points).
- Test the game by running the program. Edit as needed. (10 points)
- Publish the game to your folder. Inform someone else about the game by providing the link on ____(5 points)

KAREL JR UNIT 2

Karel 2 Overview: In real life, tasks are do not always follow the same path. We make decisions based on what we observe, and act accordingly. Machines need that same capability. We equip them with the ability to detect certain parameters and decide how to act. In Karel 2, students learn how to write conditional loops, using "if/else", "while", logical operators "and", "or", not", and various sensors.

SECTION 6: Students learn how to use if-conditions to check for collectible objects, to check for obstacles, and how to use if-conditions inside of loops. They also know that the body of conditions is indented the same as the body of loops. Karel can only detect collectible objects which are in his square, and obstacles which are in the adjacent square.

SECTION 7: Students learn how to use the else-branch with if-conditions, and how to use Karel's north sensor. They also know that the body of the else-branch is indented, the north sensor can be used to make Karel point North, and the north sensor can be used to make Karel point East, West or South as well. Conditions may contain other conditions or loops, and loops may contain other loops or conditions.

SECTION 8: Students learn how to use the empty sensor to check if Karel's pocket is empty, use keyword not to reverses the outcome of conditions, use keyword and to make sure that two or more conditions are satisfied at the same time, and use keyword or to ensure that at least one of multiple conditions is satisfied. They also know that it is a good idea to use parentheses in more complex logical expressions.

SECTION 9: Students learn how to use the while loop. They also know that the while loop is used when the number of repetitions is not known in advance. With while loops you can use the same sensors as with if-conditions. The body of while loops is indented same as the body of repeat loops.

SECTION 10: Students learn how to navigate a maze where the path goes either forward, to the left, or to the right. They continue practicing the while loop and combine it with other loops and conditions.

SECTION 6: LEVELS 6.1-6.7

Objectives: Students learn how to use if-conditions to check for collectible objects, to check for obstacles, and how to use if-conditions inside of loops. They also know that the body of conditions is indented the same as the body of loops. Karel can only detect collectible objects which are in his square, and obstacles which are in the adjacent square.

Vocabulary:

Programming terms: if, condition

Command words: all previous words

Key words: `if`

Sensor words: items from the Karel library, which can include collectible items (such as `orchid`), containers (such as `basket`), and obstacles (such as `wall, plant`). A word that is both in the library and correctly spelled will be blue-colored. Collectible and container items are sensed in the square that Karel occupies. Obstacles are sensed in the square in front of Karel.

If is written on its own line as If x, where x = a defined condition. In these lessons, predefined objects from the library are used as sensor words for the condition.

Just like the repeat loop, the body contains the commands to be followed if the "If" condition is met. The commands are written on the lines following the If command, indented two spaces.

Condition (Section 8 in the textbook): tells the program what to look for and how to act. Conditions make decisions while the program is running and handle unexpected situations. The program may need to collect all the coins it finds, but may not know where the coins will be located. The if condition says: "Is there a coin? If there is a coin, get it." Conditions work like a switch.

Satisfy: in programming, satisfy means to meet the condition - the condition exists.

Aisle: a row or column with objects on either side

Sensor: the presence of something, such as a coin, used to create a condition.

Time required: Time required will vary based on student ability and experience. Most students will complete this section in about 1 hour of programming time.

Prerequisite skills:

Completion of Karel 1.

Background knowledge/Introductory Set/Purpose: Explain the concepts of conditions. We want the robot to assess his situation. What task does he need to do? What objects does he need to avoid? How can we control where he goes no matter what the maze looks like? Does he have a choice of what to do?

In real life, we might want our computer or robot to look for conditions and act in a certain way under those conditions. This is why we write **if** and **else** conditions.

If sets the condition and the following line tells the robot what to do.

How to write conditions: show video (follow link on first screen of 6.1 or here:

http://youtu.be/Mk8JDkaZhsA

The video explains how to build a condition loop.

Big Idea: What are examples of conditions in real life (human, computer, robot or otherwise)?

Direct Instruction and Modeling:

The video and step-through demonstration in Level 6.1 model how to write conditions. 6.3 can be used to model the procedure as well.

Point out that Karel might be running more than one maze. Click on the colored tabs on the upper right corner of the maze to view the different versions. This is a good way to test whether or not the program will work under different conditions. Multiple mazes are noted and practiced in 6.2.

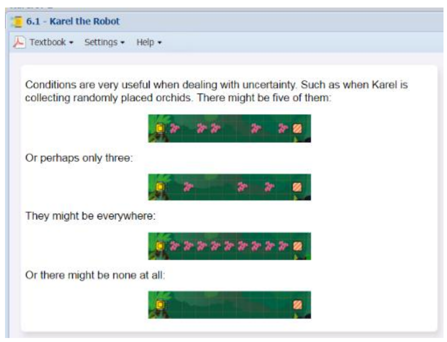
Individual/Group practice:

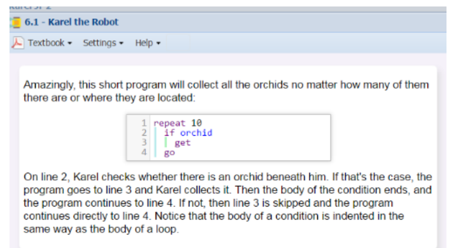
The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Levels 6.1-6.7: Self-paced instruction

6.1 Level 6.1 begins with an instructional video on building condition loops.

The video is followed by two screens explaining the value of writing conditions. In this case, Karel does not know how many orchids are there in advance, but he can test to see if an orchid is in the square and then pick it up. The condition is indented the same way as a loop. This condition starts with "if".

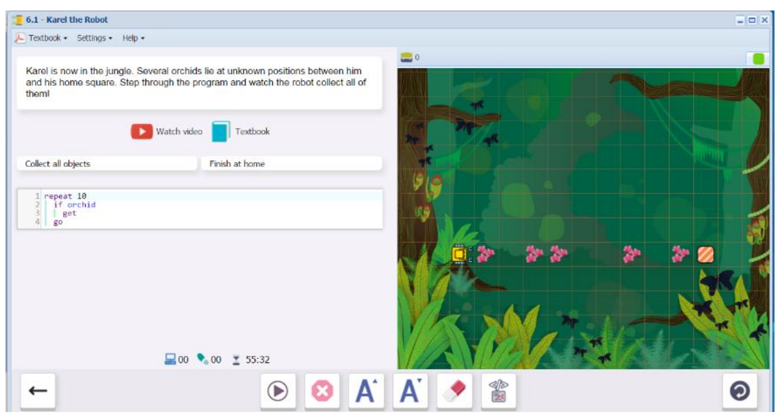




Step-through demonstration: Conditions. Karel collects all the orchids he can find.

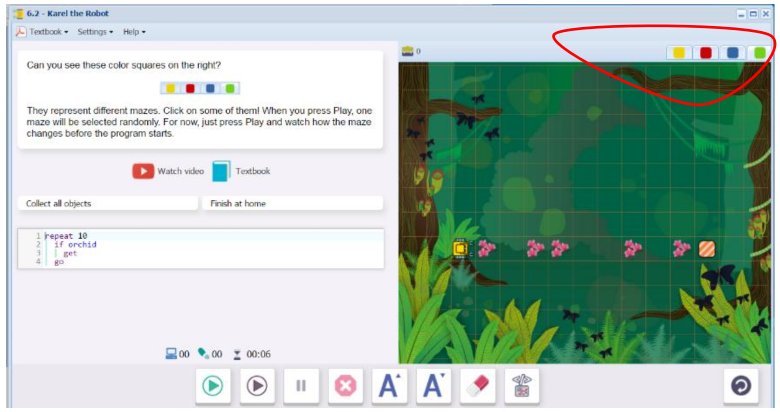
This level demonstrates the use of a condition (if orchid/get), within a repeat loop (repeat 10/go).

Students should note how the program pauses at "get" when an orchid is detected.



6.2 Karel moves through the jungle, checking to see if there are orchids, and collecting them when he does.

A condition can be tested on several mazes to see if it works for all of them. The colored tabs open up different versions of the maze. Students can run the program in each of them to see if it works in all cases (refresh the screen to run the program again, press on a different tab, then press the green play button).



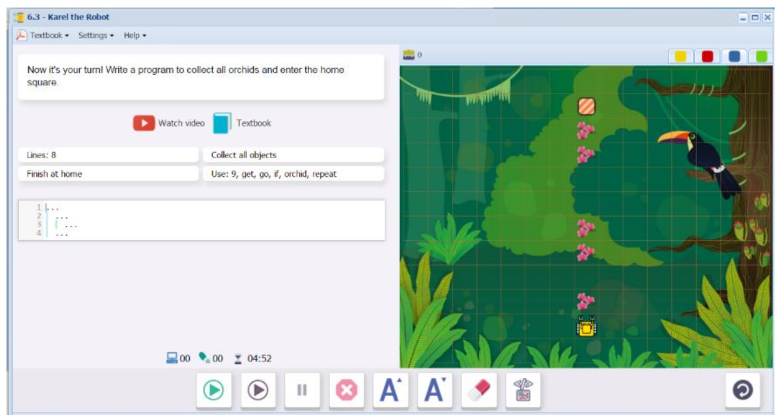
6.3 Karel moves through the jungle, checking to see if there are orchids, and collecting them when he does.

Lines: 8

Commands and keywords: `9, get, go, if, orchid, repeat`

Students write the repeat loop and the condition using the commands and keywords.

Orchid is a sensor word. Sensor words are blue in color if they exist in the library and if they are spelled correctly.

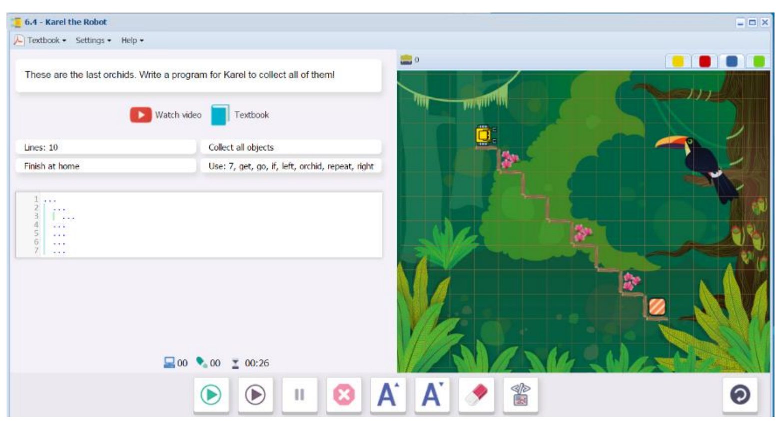


6.4 Karel moves through the jungle, checking to see if there are orchids, and collecting them when he does.

Lines: 10

Commands and keywords: `7, get, go, if, left, orchid, repeat, right`

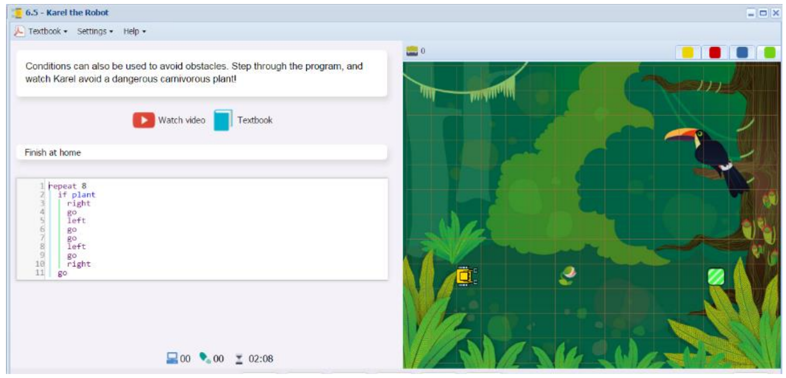
Students write the repeat loop and the condition using the commands and keywords. This time, the path is diagonal and requires left and right commands. . Notice the introduction of a wall. Objects to be avoided will become another set of condition for Karel to observe in the next levels.



6.5 Step-through demonstration level. Karel moves through the jungle, checking to see if there are dangerous, carnivorous plants to be avoided.

Note that Karel is checking the square ahead of him, rather than the one he is in. He must move around the obstacle.

When stepping through, watch how the program skips over the whole body of the condition if the condition is not met.

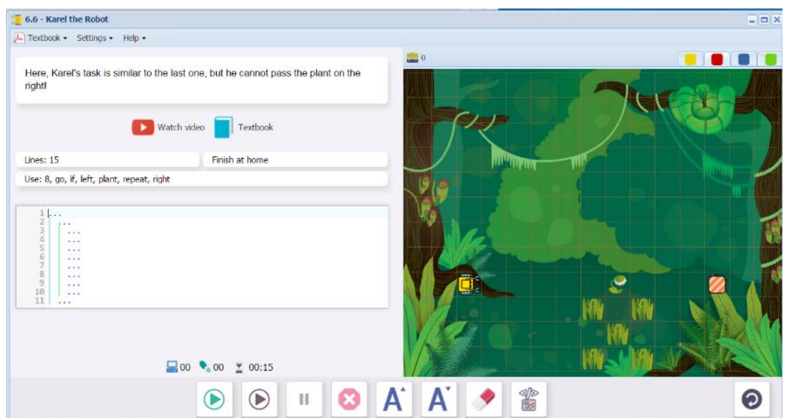


6.6 Karel moves through the jungle, checking to see if there are dangerous, carnivorous plants to be avoided. He can only move to the left because of obstacles to the right of the path.

Lines: 15

Commands and keywords: `8, go, if, left, plant, repeat, right`

Students build a program similar to 6.5, except that this time, Karel can only move to his left to avoid the plant.

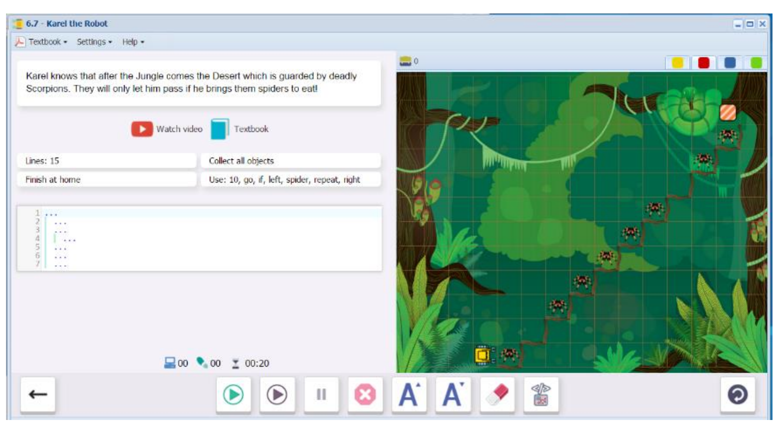


6.7 Karel moves throught the jungle, collecting spiders that he will need to feed the scorpions.

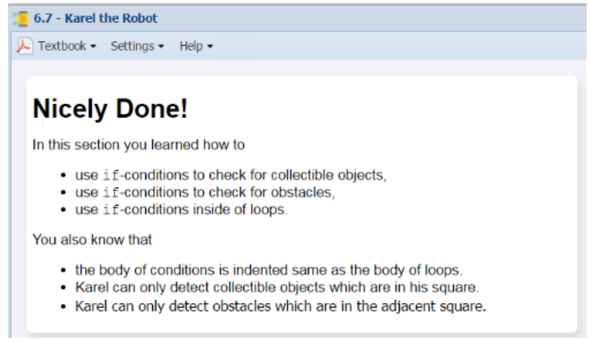
Lines: 15

Commands and keywords: `10, go, if, left, spider, get, repeat, right`

Students build a program similar to 6.5, except that this time, Karel is collecting spiders.



Upon successful completion of 6.7, students will see this message, summarizing the skills and concepts learned in Section 6. Section 7 is now unlocked.



Possible questions for post-session discussion:

What are the benefits of writing conditions into your program?

Give a couple of examples of how conditions were used in this section. (to collect orchids, to avoid carnivorous plants when the location of either one was not known in advance)

How could you use conditions in the real world?

What indentation rules did you learn regarding conditions? (The body of a condition is indented the same way as a repeat loop.)

For a sensor word to be blue-colored it must ________________ (exist in the library for Karel) and _________________ (be spelled correctly).

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment:

Number of programming lines will vary. A suggestion is between 6 and 20 lines. Inform students where they will share their game.

END OF SECTION 6: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in **Programming Mode**.

- Create a maze with a theme, walls, objects and obstacles. (10 points)
- Programming should include the commands and keywords `get, go, left, repeat and right` as needed. Programming <u>must</u> include conditions using **If,** and sensor words. Use sensor words to match the objects which you have selected for your maze. (6 points)
- The number of **programming lines** should be between __ and __ . (5 points)
- When editing the game, write the objectives of the game under the **Summary** tab. Include a storyline that relates to your maze. (7 points)
- Set the goals under the **Goals** tab. (7 points).
- Test the game by running the program. Edit as needed. (10 points)
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 7: LEVELS 7.1-7.7

Objectives: Students learn how to use the else-branch with if-conditions, and how to use Karel's north sensor. They also know that the body of the else-branch is indented, the north sensor can be used to make Karel point North, and the north sensor can be used to make Karel point East, West or South as well. Conditions may contain other conditions or loops, and loops may contain other loops or conditions.

Vocabulary:

Programming terms: if, condition

Command words: all previous words

Key words: `not`

Sensor words: `north, wall, mark, spider`

If is written on its own line as If x, where x = a defined condition. In these lessons, predefined objects such as “coin” are used as sensor words for the condition.

Just like the repeat loop, the body contains the commands to be followed if the “If” condition is met. The commands are written on the lines following the If command, indented two spaces.

Condition (Section 8 in the textbook): tells the program what to look for and how to act. Conditions make decisions while the program is running and handle unexpected situations. The program may need to collect all the coins it finds, but may not know where the coins will be located. The `if` condition says: “Is there a coin? If there is a coin, get it.” Conditions work like a switch.

Not is a logical operators for the condition. In order to execute the command,

Not means that condition must not be met.

Else provides an alternate set of commands if the condition is <u>not</u> satisfied.

Satisfy: in programming, satisfy means to meet the condition - the condition exists.

Aisle: a row or column with objects on either side

Sensor: the presence of something, such as a coin, used to create a condition.

Time required: Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills:

Completion of Section 6.

Background knowledge/Introductory Set:

We have already learned how to create a set of if conditions, so that Karel can do his tasks in variety of settings. We used the **if** condition to form a decision. If there was a spider, Karel picked it up. If there was a wall, Karel went around it. But what if we want Karel to make two choices: do one set of commands if the condition is met, and another set of commands if the condition is not met?

In real life applications, we make such branching decisions.

> If I am sick, I will stay at home.
>
> Otherwise (else), I will go to school.
>
> If I get 4 out of 5 answers wrong on a test, it will start asking easier questions.
>
> Else, it will continue with questions at the same level.

We use the keyword **else** to indicate the second choice, the one that is made if the condition is not met.

We can also use logical operators to refine our condition. In Section 7, the logical operator **not** is used. **Not** indicates the absence of a condition.

> If I am **not** home, please leave the parcel in the box by the garage.
>
> If you do **not** have checked baggage, please proceed to the exit.

Big Idea: What are some other examples of **if/else** and **if not** conditions in real life (human, computer, robot or otherwise)?

Direct Instruction and Modeling:

The program models the else condition as a step-through demonstration in Level 7.3.

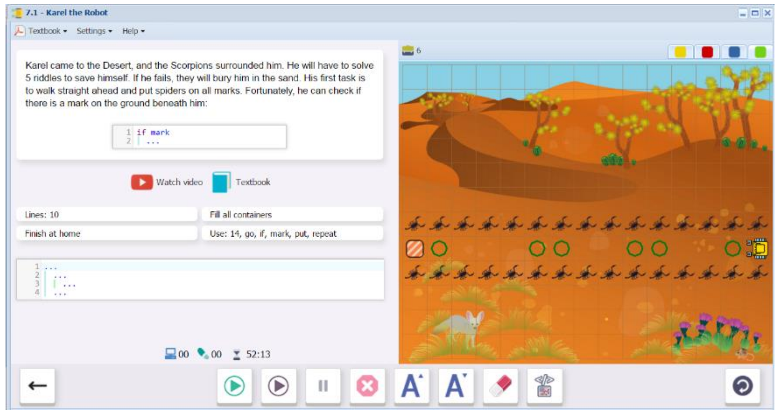
The north sensor is modeled in Level 7.6.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced instruction: Levels 7.1-7.7

7.1 Karel must place the spiders he collected for the scorpions on a row of random marks.

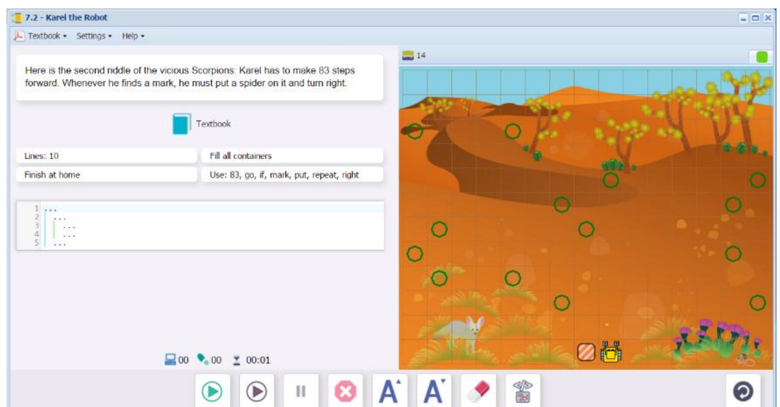


Line: 10

Commands and keywords: `14, go, if, mark, put, repeat`

Students write a repeat loop for the number of steps. The repeat loop contains a condition to put the spiders on the marks. Note that "spiders" are not named in the program. It merely takes the items out of Karel's pocket. The pocket counter is located on the upper left corner of the maze.

7.2 Karel must place the spiders on "random" marks throughout the maze.

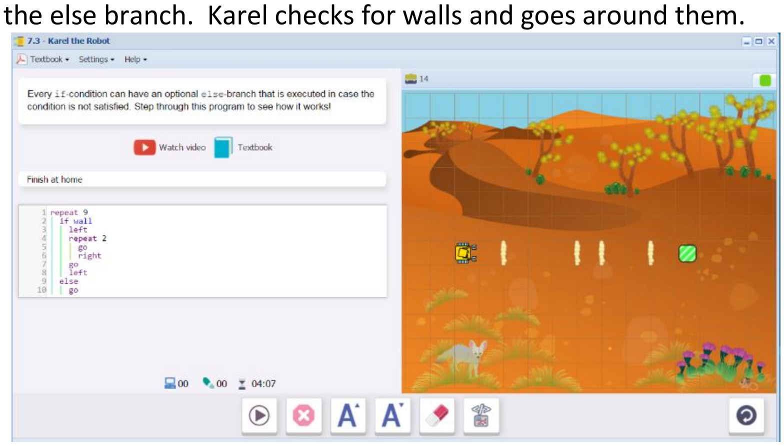


Lines: 10

Commands and keywords: `83, go, if, mark, put, repeat, right`

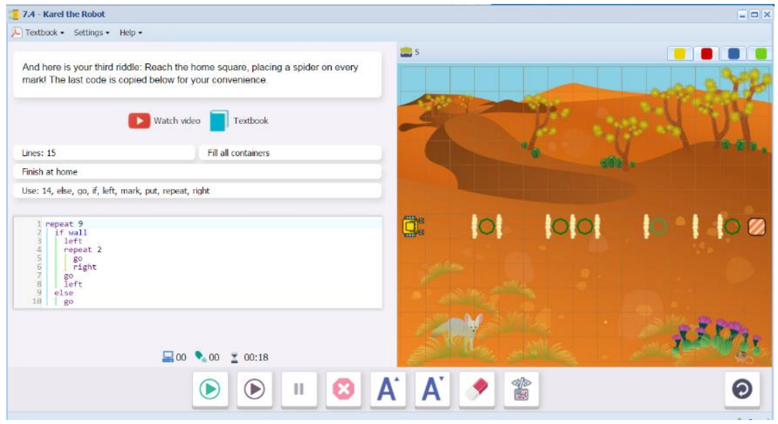
How do we search the entire maze? Karel turns right every time he puts a spider on the mark. This keeps him moving in a rectangular pattern until he is done. The instructions call for the loop to be repeated 83 times. The marks are not truly random: they are strategically placed so that the right turn solves the puzzle.

7.3 Step-through demonstration of the else branch. Karel checks for walls and goes around them.



The if condition has Karel go around the wall if he detects one. The else branch tells him to go forward if he does not detect a wall.

7.4 Karel places a spider on every mark and goes around every wall.

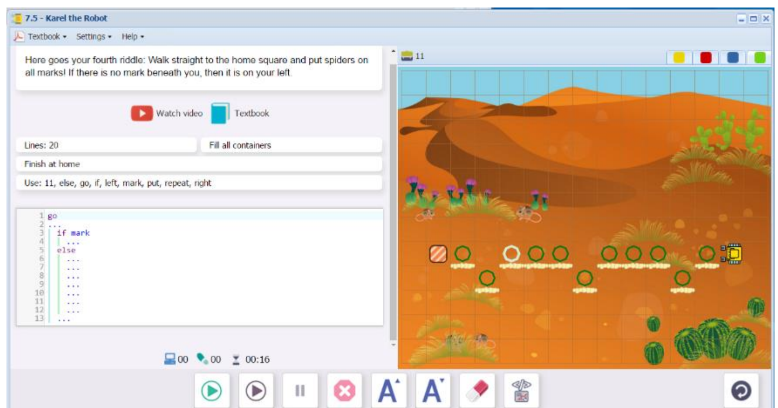


Lines: 15

Commands and keywords: `14, else, go, if, left, mark, put, repeat, right`

Students insert the lines needed to place the spiders on the marks, and adjust the number of repetitions.

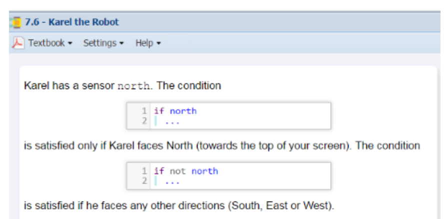
7.5 Karel places a spider on every mark and goes around every wall.



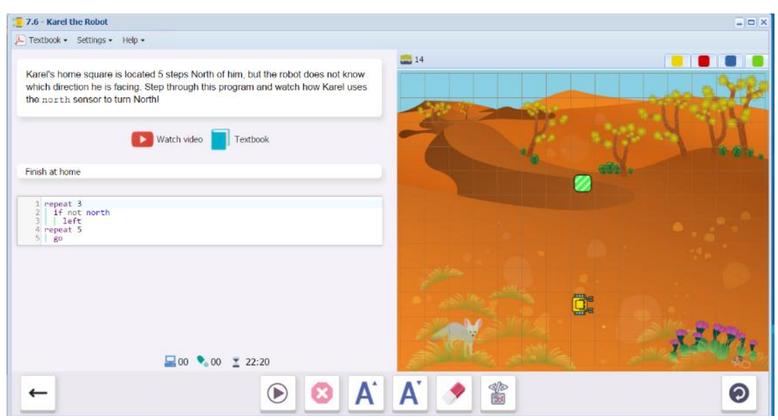
Lines: 20

Commands and keywords: `11, else, go, if, left, mark, put, repeat, right`

Students are prompted to start the condition with " if mark ". The else portion is triggered when there is no mark in front of Karel. Else will be a set of commands that turns Karel left, places the spider on the mark and return to his position on the main path, facing forward for the next step.



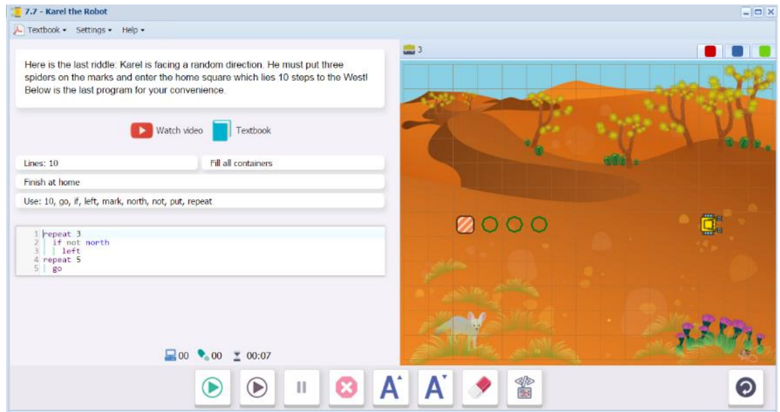
7.6 Students learn about the directional sensor north. "If not north" can be used to detect if Karel is facing North (the top of the maze). The condition " if not north" tests to see if Karel is facing any other direction (East, South, or West). Notice that the "not" operator is used.



Step-through demonstration: Karel's home is only 10 steps away, but he does not know which direction he is facing.

The program uses the "if not north"condition to re-orient Karel and send him home. He would need to make three right turns at most to face north.

7.7 Karel orients himself facing north, then turns and follows a path home, placing spiders on marks as he goes.

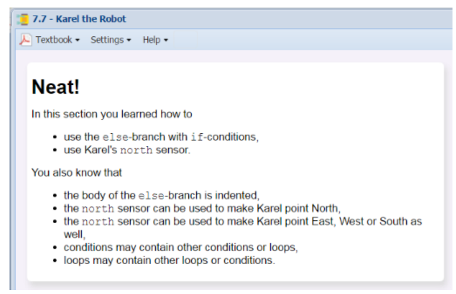


Lines: 10

Commands and keywords: 10, go, if, left, mark, north, not, put, repeat

Students complete the program by orienting Karel on the path, adding lines to place the spiders and adjusting the number of repetitions. The "if north" portion is already written.

Upon successful completion of 7.7, students will see this message, summarizing the skills and concepts learned in Section 7. Section 6 is now unlocked.



Possible questions for post-session discussion:

When do you need an "else" branch? (when the absence of the condition requires its own set of commands)

What does the operator "not" mean? (the absence of the condition)

How does the north sensor help orient Karel? (Once Karel faces north, he can be oriented in a new direction with certainty) Why do you have to repeat the "if not" condition 3 times? (He needs to turn left once if he is facing east, 2 times if he is facing south, and 3 times if he is facing west)

How could you use conditions in the real world?

What indentation rules did you learn regarding conditions? (The body of a condition is indented the same way as a repeat loop.)

For a sensor word to be blue-colored it must ________________ (exist in the library for Karel) and _________________ (be spelled correctly).

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment:

Number of programming lines will vary. A suggestion is between 6 and 20 lines. Inform students where they will share their game.

END OF SECTION 7: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in **Programming Mode**.

- Create a maze with a theme, walls, objects and obstacles. (10 points)
- Programming should include the commands and keywords **Get, Go, Left, Put, Repeat and Right** as needed. (6 points)
- Programming must include conditions using **If, else, and not** and sensor words. Use sensor words to match the objects which you have selected for your maze. (6 points)
- The number of **programming lines** should be between __ and __ . (5 points)
- When editing the game, write the objectives of the game under the **Summary** tab. Include a storyline that relates to your maze. (7 points)
- Set the goals under the **Goals** tab. (7 points).
- Test the game by running the program. Edit as needed. (10 points)
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 8: LEVELS 8.1- 8.7

Objectives: Students learn how to use the **empty** sensor to check if Karel's pocket is empty, use keyword **not** to reverses the outcome of conditions, use keyword **and** to make sure that two or more conditions are satisfied at the same time, and use keyword **or** to ensure that at least one of multiple conditions is satisfied. They also know that it is a good idea to use parentheses in more complex logical expressions.

Vocabulary: (new words: empty, or, and)

Programming terms: if, condition

Command words: all previous words

Key words: or, and, not

Sensor words: empty, wall, coin, nugget, cart, snake

Or, **and**, **not** are logical operators for the condition. In order to execute the command,

Or means that one (or a set of conditions within parentheses) of two or more conditions must be met,

And means both or all of the conditions must be met,

Not means that condition must not be met.

Empty tells whether or not the robot has an object in its pocket. This creates a condition, either **if empty**, or **if not empty**

Time required: Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills:

Completion of Section 7. The level of difficulty in both concept and skill is increasing and you may find a divergence in rate of success among your students, especially in the younger grades.

Background knowledge/Introductory Set:

Karel knows how to check for conditions one at a time. Now, we can create more complex conditions for his decisions. Think of how you choose your lunch from a menu:

You will have soup or salad.

If you have salad, you will have no dressing or house dressing.

You will have spaghetti, which is composed of noodles and meatballs and sauce.

If you are full (not empty), you will not have dessert.

In this section, we learn the logical operators **and**, **or** and **not**. These help Karel make more complex decisions. We will also keep track of the number of objects in his pocket by using **empty** or **not empty**.

Direct Instruction and Modeling:

Step through demonstrations are located in Level 8.3 (empty, not empty), and 8.5 (and, or, use of parentheses). These demonstrations can be discussed as a class.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

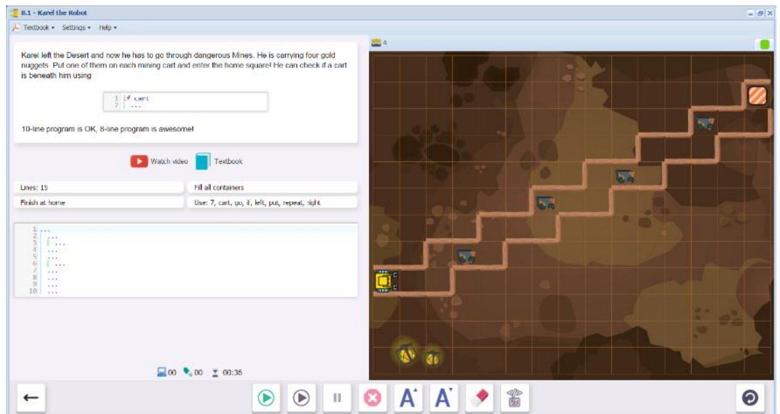
Levels 8.1-8.7: Self-paced instruction

8.1 Karel must place 4 gold nuggets in carts in the tunnel.

Line: 15 (Challenge students to write the program in 8 or 10 lines. The 8-line program requires a nested repeat loop)..

Commands and keywords: 7, cart, go, if, left, put, repeat, right

Students write a repeat loop that contains "if cart".

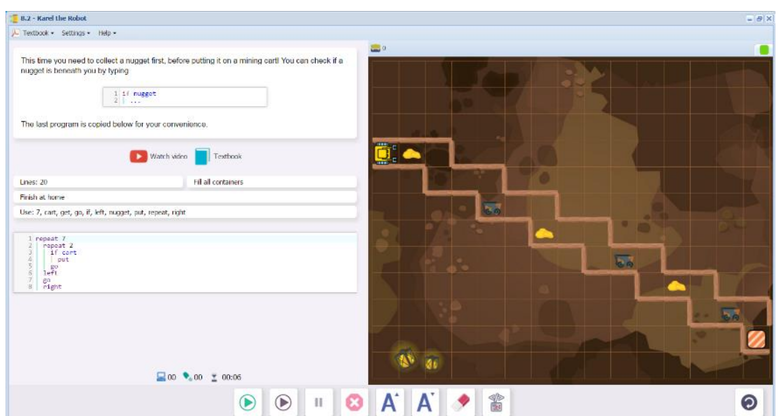


8.2 Karel must collect the nuggets and place them in the carts in the mine tunnel.

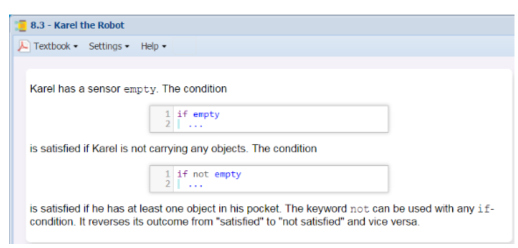
Lines: 20

Commands and keywords: 7, cart, get, go, if, left, nugget, put, repeat, right

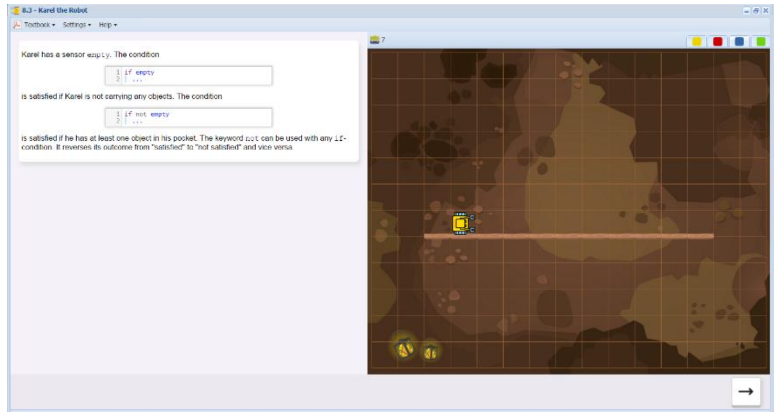
Students modify the program in 8.1 to adjust the direction, and include retrieving the nuggets before placing them in carts.



8.3 Step-through demonstration of the empty sensor. Karel checks for nuggets in his pocket.



In the demonstration, Karel checks his pocket for nuggets. If he has one, he will put it on the square and move forward. If he doesn't have one, he will stop. When he stops, the program ends. Until now, the program has ended when Karel reaches the home square.

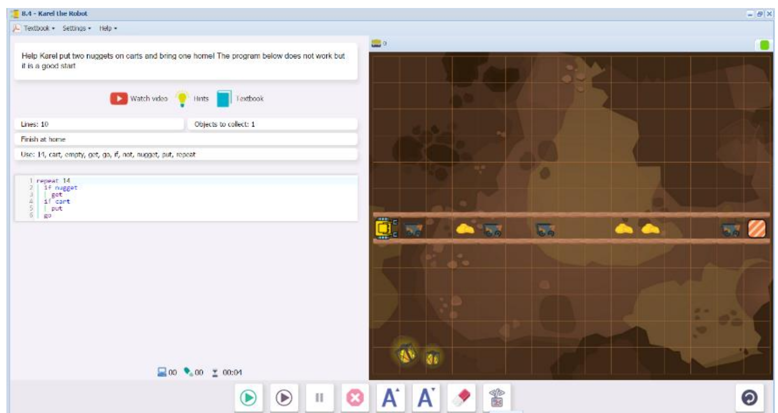


8.4 Karel needs to collect nuggets, put 2 on the carts and bring 1 home.

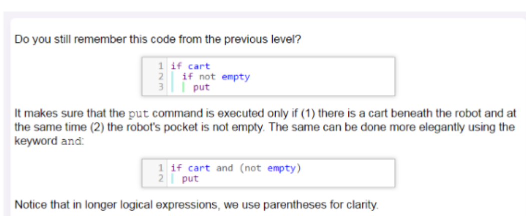
Lines: 10

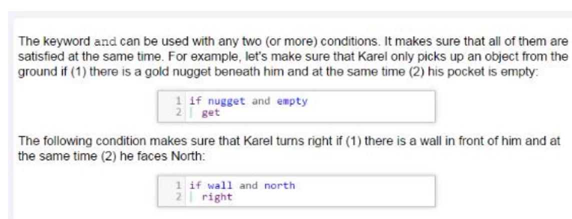
Commands and keywords: 14, cart, empty, get, go, if, not, nugget, put, repeat

Students need to repair the program that is already written, by adding an if not empty condition.



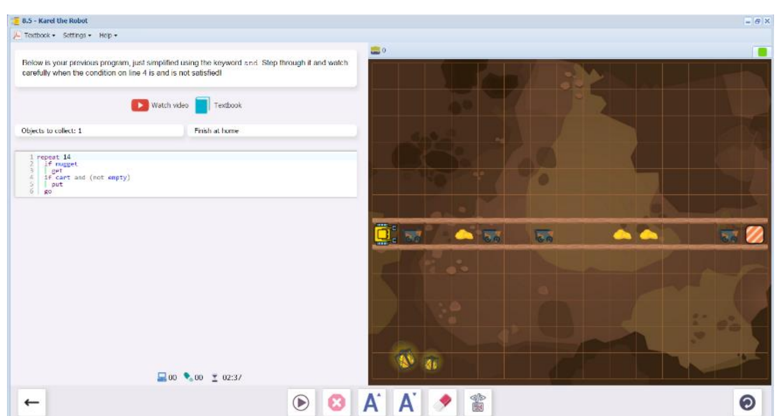
8.5 Step-through demonstration on combining logical operations on one line.





The demonstration uses the program from 8.4, combining the **if not empty** and the **if cart** lines into one line. The parentheses are used to clarify that **not** applies only to **empty**

If cart and (not empty)



8.6 Step-through demonstration on the logical operator (keyword) **or**.

Karel must collect nuggets and jewels.

Or ensures that at least one out of the two or more conditions are met in order for Karel to pick up the object. In this case, if Karel finds a nugget or a jewel, he will collect it. He will not collect the other objects because they are not specified.

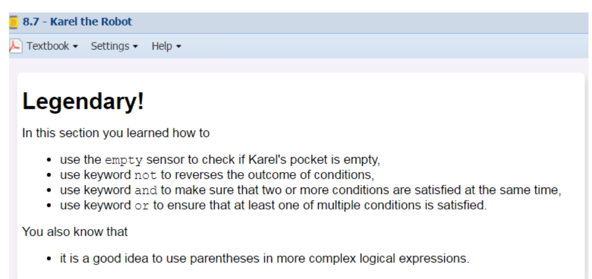
8.7 Karel must get through the maze to the home square, picking up nuggets and coins and avoiding traps. If he encounters a wall, he goes to the right. However, if there is a snake, he must turn left.

Lines: 15

Commands and keywords: 18, get, go, if, left, or, repeat, right, snake

Students complete the program by orienting Karel on the path, adding lines to place the spiders and adjusting the number of repetitions. The "if north" portion is already written.

Upon successful completion of 8.7, students will receive the Yellow Belt of Third Degree and see this message, summarizing the skills and concepts learned in Section 8. Section 9 is now unlocked.



Possible questions for post-session discussion:

Explain when you use the operator **or** and when you use **and.** (**Or** is used when one of the conditions needs to be satisfied. **And** is used when both (or all) of the conditions must be satisfied.)

When would you use parentheses? (When you want the operator to apply to specific keywords)

Think of real world situations that require **and/or** conditions.

Think of real world situations that require **empty** or **not empty** conditions.

Assessment: Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment:

Number of programming lines will vary. A suggestion is between 6 and 20 lines. Inform students where they will share their game.

END OF SECTION 8: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in **Programming Mode**.

- Create a maze with a theme, walls, objects and obstacles. (10 points)
- Programming should include the commands and keywords **Get, Go, If, Left, Repeat and Right** as needed.
- Programming must include conditions using **If,** operators **and, or, not,** and sensors, including **empty**. Use sensor words to match the objects which you have selected for your maze. (6 points)
- The number of **programming lines** should be between __ and __ . (5 points)
- When editing the game, write the objectives of the game under the **Summary** tab. Include a storyline that relates to your maze. (7 points)
- Set the goals under the **Goals** tab. (7 points).
- Test the game by running the program. Edit as needed. (10 points)
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 9: LEVELS 9.1-9.7

Objectives: Students learn how to use the while loop. They also know that the while loop is used when the number of repetitions is not known in advance. With while loops you can use the same sensors as with if-conditions. The body of while loops is indented same as the body of repeat loops.

Vocabulary:

While: A while loop is a repeated set of commands that will continue as long as the condition being sensed is present. The number of repetitions is not known in advance. The while loop continues until the condition is no longer sensed. While loops use the same sensors as if conditions. They differ because they continue the loop until the condition is no longer sensed, whereas the **if** condition senses each square as a separate test.

Infinite loop: If a loop never senses when to end (the stopping condition), it can continue infinitely. Fortunately, most programs will time out if this happens. In Karel, programs can always be stopped manually if this happens.

Time required: Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills:

Completion of Section 8.

Background knowledge/Introductory Set:

If conditions test for the presence of a sensor. Else can provide an alternative action if the sensor is not there. Karel tests every square as long as the **if** condition is in play. However, what if we don't know where the sensors are, and we want Karel to keep checking for them while he is doing other tasks? The while loop is used for this purpose. For example, Karel could keep looking for an item as long as he hasn't reached the home square. We would start such a loop with **while not home**. On the other hand, Karel might have to repeat a function several times until he no longer senses a condition. For example, a **while wall** loop would continue until Karel no longer senses a wall.

In this section, we learn how to write **while** conditional loops. We are still using if conditions. Examine these carefully to understand the difference.

Direct Instruction and Modeling:

There are several instructional screens in 9.1 that describe the purpose and usefulness of the **while** command. The teacher can go through these screens with the class prior to individual instruction, and model how to type the while loop in the Level 9.1 lesson. The video in 9.1 describes and demonstrates while loops.

http://youtu.be/9YpKSfwJCTs

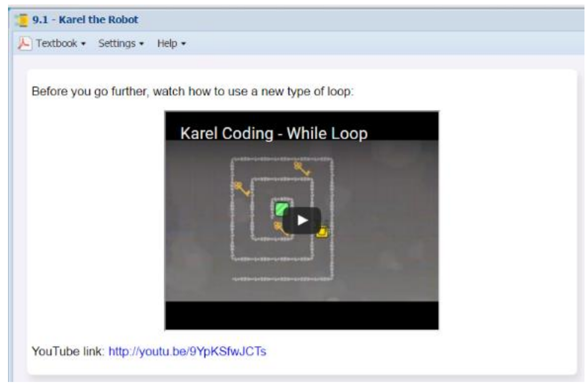
Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion. At this stage, programming requires some thought and planning. Students now have all the basic tools: when are the repeat loops, if conditions, and while loops best used? Emphasize the importance of studying the tasks and the layout before starting to type. Even small syntax errors can cause failure. Why did a program work or not work?

Levels 9.1-9.7: Self-paced instruction

Level 9.1 begins with a YouTube video that teaches While Loops, located at

http://youtu.be/9YpKSfwJCTs



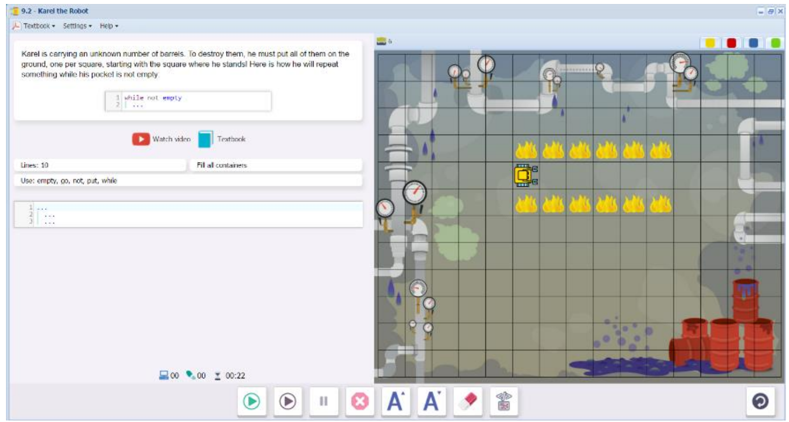
The step-through demonstration shows how the **while** loop works. Karel will continue to collect barrels **while** there are barrels. When there are no more barrels, he will stop.

9.2 Karel must place an unknown number of barrels in a row (the unknown is the number in his pocket).

Lines: 10

Commands and keywords: empty, go, not, put, while

Students write a program using the while loop. While not empty means that Karel will continue to perform the task until his pocket is empty. Then he will stop.



9.3 Karel collects barrels until he reaches home.

Lines: 10

Commands and keywords: barrel, get, go, home, if, not, while

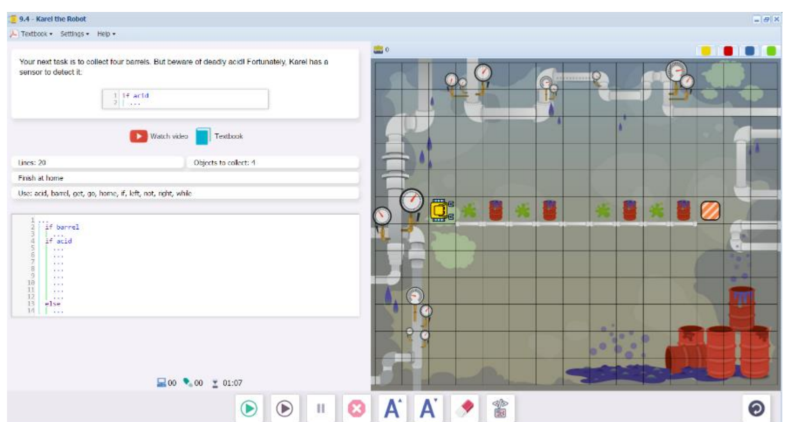
On this level, the “while not home” condition is introduced. This condition allows Karel to continue performing a task until he reaches the home square. This way, the number of steps does not need to be specified.

9.4 Karel collects barrels but must avoid the puddles of acid.

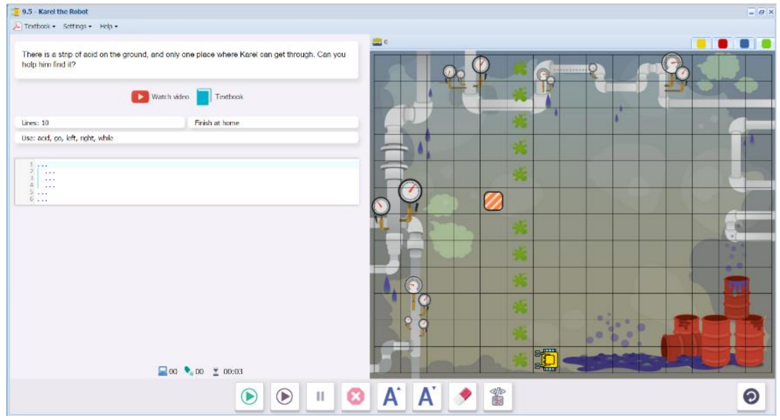
Lines: 20

Commands and keywords: acid, barrel, get, go, home, if, left, not, right, while

Students write a program that incorporates “while not home”, creates an if condition for collecting the barrels and an if/else condition for avoiding the acid (if results in moving around the acid, else results in moving forward).



9.5 Karel needs to check a strip of acid for an opening to go through.

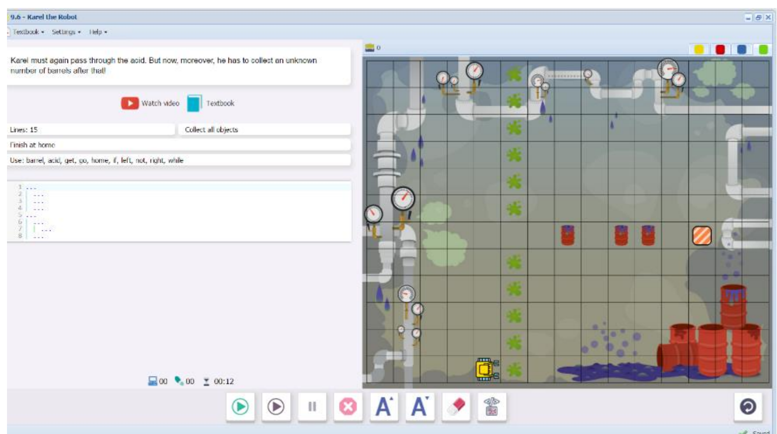


Lines: 10

Commands and keywords: acid, go, left, right, while

Karel tests the strip of acid by turning to face it after ever step, and using the “while” command to check for acid. The first time that there is no acid, he can safely go through. This is an example of using “while” instead of “if/else”: we don’t know when Karel will find the opening, but when he does, the condition will stop. Until then, the presence of acid is consistent.

9.6 Karel again looks for an opening in a strip of acid, and he must collect an unknown number of barrels.

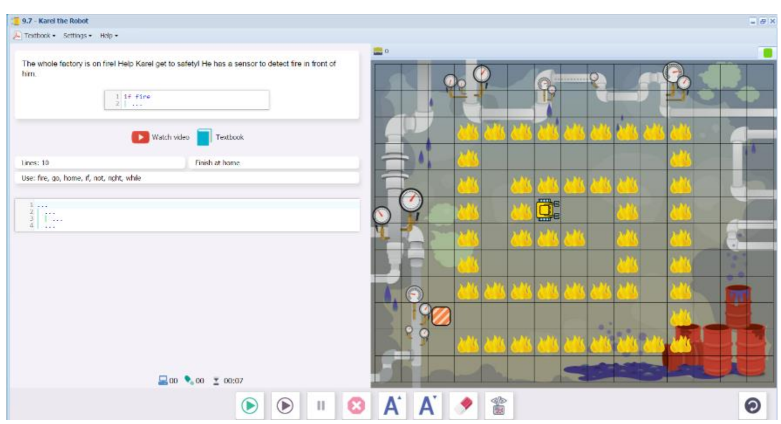


Lines: 15

Commands and keywords: barrel, acid, get, go, home, if, left, not, right, while

Students write two while loops: one for testing the strip of acid for an opening, as in 9.5 (“while acid”), and one for setting the stopping condition of reaching home (“while not home”). The “if” condition is used to check for barrels, since it must check every square as a separate test.

9.7 Karel finds a safe passage home by testing for fire.

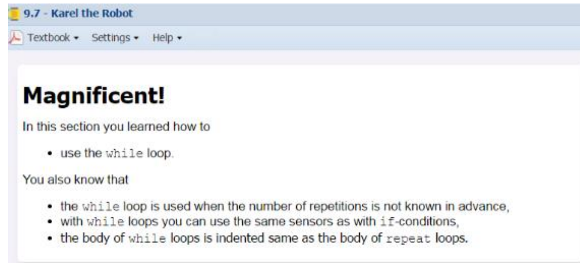


Lines: 10

Commands and keywords: fire, go, home, if, not, right, while

This program is very simple. Because the path is a spiral, Karel only needs to turn right to avoid the fire. “While not home” sets the stopping condition.

Upon successful completion of 9.7, students will see this message, summarizing the skills and concepts learned in Section 9. Section 10 is now unlocked.



Possible questions for post-session discussion:

Compare while and if loops. How are they similar? (They use the same sensors. They both test a condition or conditions. If the condition(s) is met, then the commands in the body of the loop are executed)

How are they different? (A while loop keeps going until the condition is not met. An if condition tests each step individually. It can branch to another command using else)

Assessment: Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment:

Number of programming lines will vary. A suggestion is between 6 and 20 lines. Inform students where they will share their game.

END OF SECTION 9: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in **Programming Mode**.

- Create a maze with a theme, walls, objects and containers. (10 points)
- Programming should include several basic commands `get, go, left, put, repeat and right`.
- Programming must include conditional loops using `while, home, and, not, or`. It may include `empty`. (6 points)
- The number of **programming lines** should be between __ and __ . (5 points)
- When editing the game, write the objectives of the game under the **Summary** tab. Include a storyline that relates to your maze. (7 points)
- Set the goals under the **Goals** tab. (7 points).
- Test the game by running the program. Edit as needed. (20 points)
- Publish the game to your folder. Inform someone else about the game by providing the link on _______________________________________(5 points)

SECTION 10: LEVELS 10.1-10.7

Objectives: Students learn how to navigate a maze where the path goes either forward, to the left, or to the right. They continue practicing the while loop and combine it with other loops and conditions.

Vocabulary:

No new vocabulary in this Section

Time required: Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills:

Completion of Section 9.

Background knowledge/Introductory Set:

We have learned how to create different kinds of loops, including repeat (counting) loops, and loops based on if/else conditions and while conditions. We use logical operators and, or, not to customize the sensors.

In past sections, the mazes have followed predictable patterns. In this section, we will start by practicing loop combinations that are useful for spirals, squares, and steps. Then we will learn how to navigate more complex mazes.

Direct Instruction and Modeling:

The first five levels practice while loops, if conditions and repeat loops in different combinations to solve spiral, square and step mazes. 10.6 is a step-through level that shows how to make choices on which way to turn to avoid running into a wall. This level can be demonstrated and discussed as a class as needed, or reviewed as a follow-up discussion after students complete Section 10.

Note: this is the final level for Karel 2. For the final project, students can create multiple mazes to test their program.

To do this, use the **"Add a Copy" tool on the Maze menu to create additional mazes.** To save time and also to take advantage of the versatility of the While loop, the tool **"Place Elements Randomly"** can be used to place objects.

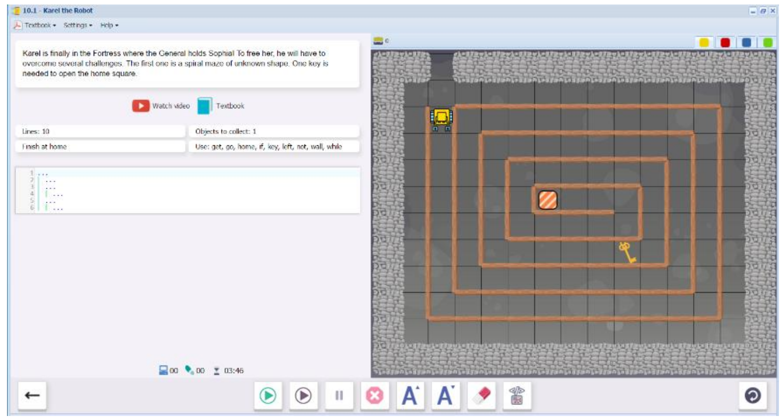
Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Levels 10.1-10.7: Self-paced instruction

10.1 Karel goes through a maze to reach the home square, collecting one key along the way.

Line: 10

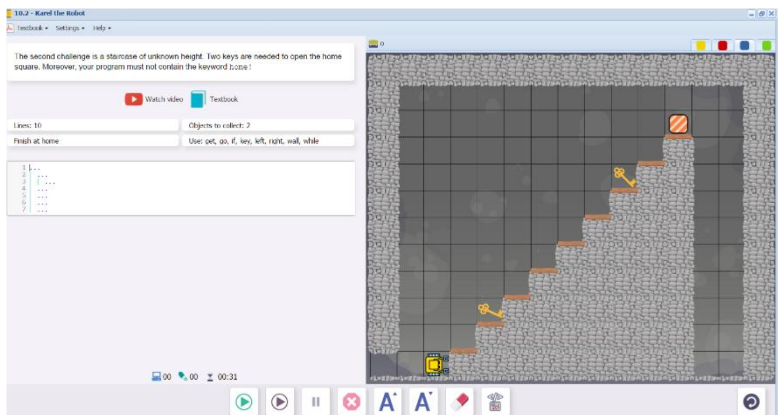


Commands and keywords: `get, go, home, if, key, left, not, wall, while`

As in 9.7, Karel is following a spiral. To avoid crashing into a wall, he turns left if he detects a wall. Student program a while loop that contains two if conditions.

10.2 Next, Karel climbs a set of stairs, collecting two more keys.

Lines: 10

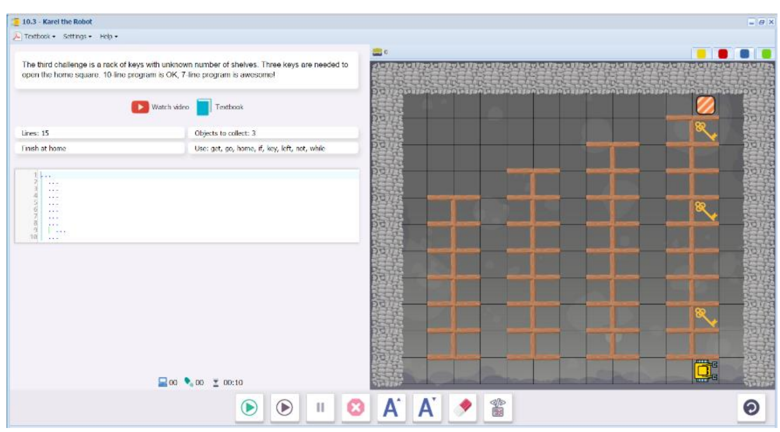


Commands and keywords: `get, go, if, key, left, right, wall, while`

Students practice writing a while loop with embedded if conditions. Similar to the program for the strip of acid in 9.5 and 9.6, the "while wall" loop will continue the pattern of climbing until there is no more wall (stopping condition).

10.3 Karel

Lines: 10

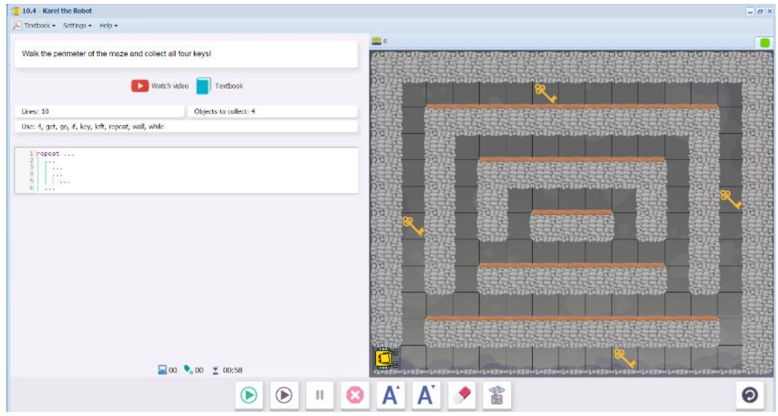


Commands and keywords: `get, go, home, if, key, left, not, while`

Students write a "while not home" loop that has Karel search each shelf for a key. The number of shelves is unknown.

Challenge: a 10-line program is OK, a 7-line program is awesome! (Students should look for repeated patterns to shorten the program)

10.4 Karel walks the perimeter wall and collects keys.

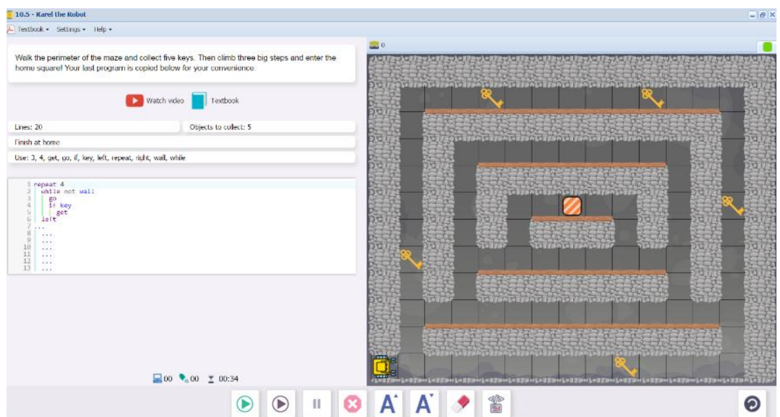


Lines: 10

Commands and keywords: `4, get, go, if, key, left, repeat, wall, while`

10.4 is part one of 10.5. The while loop is test for the presence of the wall. Karel keeps moving as long as there is no wall (while not wall). The while condition is embedded in a repeat 4 loop: one repetition for each wall.

10.5 Karel completes the perimeter walk from 10.4, then finds his way to the center of the maze.



Lines: 20

Commands and keywords: `3, 4, get, go, if, key, left, repeat, right, wall, while`

The program starts with the code from 10.4, which collects the keys along the perimeter wall. Students write the rest of the program, which will get Karel home in the center of the maze. Look for repeated patterns. A simple repeat loop will work.

10.6 **Step-through instructional level.** Karel must check for walls more than once to know whether he should proceed left or right.

The program demonstrates how to write a nested set of two if conditions.

First, Karel checks for a wall. If it is there, he turns left. If he senses a wall again, he needs to turn around so that he faces the opposite direction (right, right or left, left). This will prevent him from crashing into a corner.

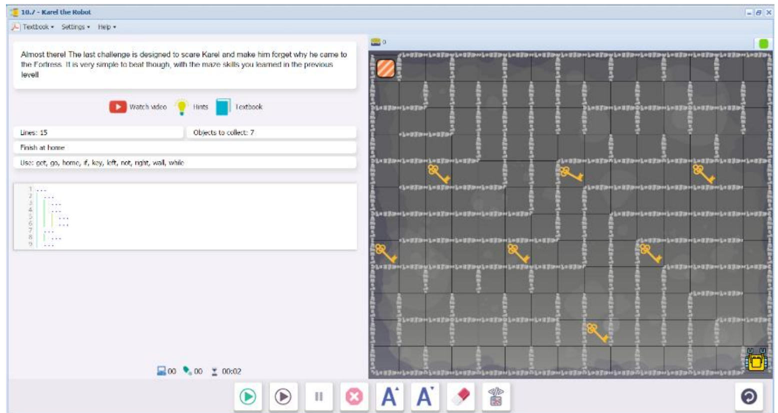
This nested set of if conditions is very useful for navigating mazes in any direction, not just a set spiral or step pattern.

10.7 Karel navigates a maze and collects all the keys in his path.

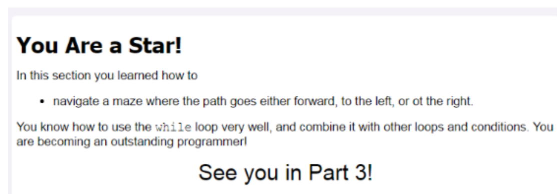
Lines: 15

Commands and keywords: get, go, home, if, key, left, not, right, wall, while

This maze looks complicated, but it will respond to the same set of commands as 10.6. This is a good example of how a simple, elegant program can work in a complex setting.



Upon successful completion of 10.7, students will receive the Yellow Belt of Fourth Degree and see this message, summarizing the skills and concepts learned in Section 10. Karel 3 is now unlocked.



Possible questions for post-session discussion:

Compare the different kinds of mazes (spiral, square, step, complex). What types of loops or conditions were best for each one?

Review how to make multiple mazes in Creative Suite (see Direct Instruction).

Assessment: Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment:

Number of programming lines will vary. The game creation will take longer if students create multiple mazes. Inform students where they will share their game.

END OF SECTION 10: CREATE A GAME FOR KAREL (100 POINTS)

Create and publish a game for Karel in **Programming Mode**.

- Create **four** complex mazes with a theme, walls, objects and containers. Use the "Add a Copy" tool on the Maze menu to create additional mazes.
  Using the "Place Elements Randomly tool may help. (40 points)

- Programming should include the commands `get, go, left, put, repeat and right` as needed (6 points)

- Programming must include conditional loops using `while, if/else, home, and, not, or, north` as needed. It may include `empty`. (10 points)

- The number of **programming lines** should be between __ and __ . (5 points)

- When editing the game, write the objectives of the game under the **Summary** tab. Include a storyline that relates to your maze. (7 points)

- Set the goals under the **Goals** tab. (7 points).

- Test the game on all four mazes by writing the lines of code and running the program. Edit as needed. (20 points)

- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

KAREL JR UNIT 3

Karel 3 Overview: Whether human or robot, we often follow routines in our daily lives. Such a routine can be defined once, then used whenever it is needed. In Karel, these are referred to as defined commands, which combine a set of commands that can be called upon whenever needed in the main program. In programming, as in real life, a defined command should be tested on a simple situation before using it in a more complex program. We also use variables to count and report events or items. At this stage, we are also learning to optimize programs, not just finding the simplest or shortest way to complete our tasks.

SECTION 11: Students learn how to define a custom command using the keyword def and call it in the main program whenever it is needed. They know that the body of a new command must be indented.

SECTION 12: Students learn that a new command should always be tested on a simple task first, and then it can be safely used as part of a larger program. They also learn advanced maze skills: how to follow a line that is on Karel's left, or one that is on Karel's right.

SECTION 13: Students learn that the shortest program may not always be the best. A slightly longer program that is much faster, is better than a slightly shorter program that takes a lot of time. Students know to break a complex problem into smaller tasks which are solved first.

SECTION 14: Students learn how to create new variables and initialize them with numbers. They use the function inc() to increase the value of a variable by one, the function dec() to decrease the value of a variable by one, and the print command to display results. The print command can be used to display the values of variables while the program is running.

SECTION 15: Students learn how to define new functions and return values using the keyword return, use functions inc() and dec() to increase / decrease the value of a variable by more than one. They know that the value returned from a function can be stored in a variable, and if the returned value is not used, it will be automatically printed. Any code typed after the return command is dead. Variables defined inside commands and functions are local, and local variables cannot be used outside of the command or function where they were defined. Variables created in the main program are global, and global variables should not be used inside commands and functions.

SECTION 11: LEVELS 11.1-11.7

Objectives: Students learn how to define a custom command using the keyword `def` and call it in the main program whenever it is needed. They know that the body of a new command must be indented.

A defined command has two advantages:

- The program requires less lines of code, once the definition has been created.
- It is easier to fix problems within the defined command, rather than searching through the program and fixing several lines.

Note that the program makes use of comment lines to explain what is happening in each section. These lines begin with #, which indicates a text string rather than a programming line. These comment lines will assist students in writing the next step of code.

Vocabulary: Students should already be familiar with:

Command words: `go, left, right, get, put`

Repeat (counting) loops

If/Else Conditions and While Conditional loops

Logical Operators `and, or, not`

Keyword/ Sensor Words home, defined objects and obstacles

New Vocabulary:

`def` `def` begins a defined command, which is a set of commands that will be called in the main program.

Text string: words included in the program that are descriptive and not part of a command. Text strings are enclosed in quotation marks and are separated from command words by a comma.

Comment lines: lines of text strings, always starting with the # sign that describe what is happening in the program. Quotation marks are not needed in this case. Students will already be familiar with comment lines viewed in previous Karel levels, but they may want to start writing them into their own programs at this point.

Time Required

The presentation to the class takes about 10 to 15 minutes. Since the course is self-paced, the amount of time to complete this Section will vary from student to student. Most students will finish the Section in about two hours.

Prerequisite Skills

The Karel 2 unit must be completed in order to unlock Karel 3.

Background knowledge/Introductory Set/Purpose:

In Karel 1 and 2, students learned to create code that can:

- Control forward and turn movement
- Control picking up and putting down objects
- Simplify repeated patterns into repeat loops
- Make Karel responsive to unknowns by writing conditional loops.

In Karel 3, students will learn new tools that will make their programs more manageable, flexible and powerful. Section 11 starts by introducing defined commands.

Explain to students that they will be learning how to define a command in Karel using the reserved word `def`. A defined command creates a mini-program that can take care of a whole routine with one command.

For example, let's say that a teacher wants her students to get ready for the next subject. She could create a command "Ready". Student would know that when they heard "Ready", they would put away their books and supplies, and get out a fresh pencil, textbook, and journal for the next subject.

In a computer program, code can get very lengthy. By defining commands for different routines, the code becomes more manageable.

A defined command can be called when it is needed.

It can be edited separately without disrupting the flow of the main program.

It must be defined for each program. In other words, if you create a `def` command in one program, it will not be recognized in another one. You would have to recreate the command in the second program as well.

Look for repeated sets of commands that could be turned into defined commands.

Direct Instruction/Guided Practice

You may choose to watch Levels 11.1 and 11.2 together as a class. Level 11.1 demonstrates an example of "bad" programming, and Level 11.2 how to clean it up using defined commands. Level 11.3 begins with a video that teaches how to use defined commands. Here is the link for that video:

https://www.youtube.com/watch?v=Kj_LTtyFYZA

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion. At this stage, programming requires some thought and planning. Students now have all the basic tools: when are the repeat, if conditions, and while loops best used? Emphasize the importance of studying the tasks and the layout before starting to type. Even small syntax errors can cause failure. Why did a program work or not work?

Levels 11.1-11.7: Self-paced instruction

11.1 Demonstration level. Karel is collecting several groups of computer chips (codes).

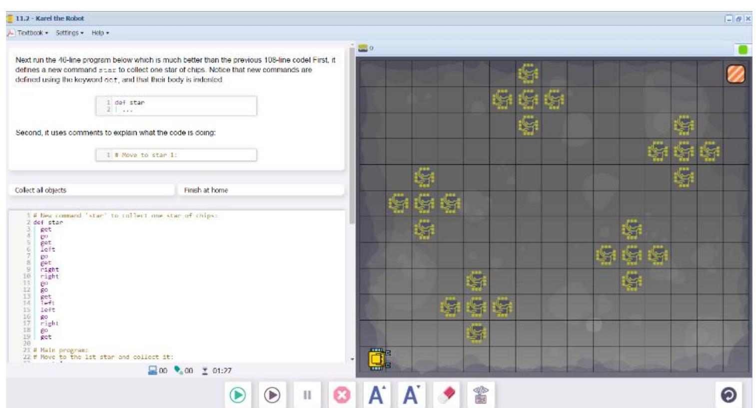
This program repeats the same set of commands to search one group over and over again. The program works, but is difficult to manage because there are 108 lines of code.

What parts of the program can be grouped together to make a set of commands?

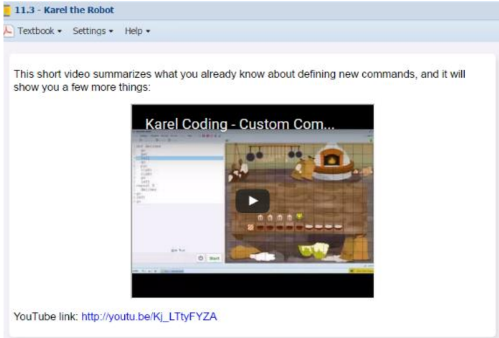
11.2 Demonstration level. Karel completes the same task as 11.1. This time, we use `def star` to create a set of commands to collect the chips, and call it within the main program.

Observe how the defined command star is called in the program. This time, the program is only 46 lines long.

Comment lines are also used to head different sections of the program. These are preceded by the # symbol, so that the computer can ignore them.



11.3 This level starts with a YouTube video, which explains how to create and use defined commands.



Here is the link to the YouTube video:

https://www.youtube.com/watch?v=Kj_LTtyFYZA

Karel collects chips in a star pattern.

Lines: 30

Commands and keywords: (previously learned commands are assumed to be available), `def`

Part of the program is already written. Students complete the set of commands defined by `star`, and call it in the main program.

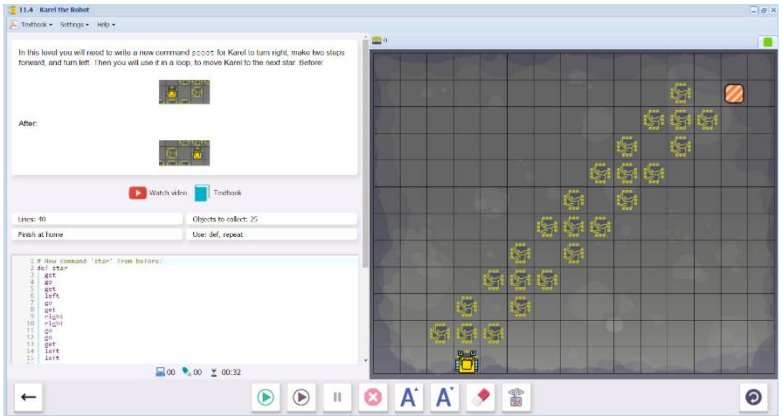
11.4 Karel collects 5 groups of chips, moving from group to group.

The previously defined command `star` is used to collect the chips. A new command `scoot` is defined for the steps needed to move from group to group.

Lines: 40

Commands/keywords: `def, scoot`

Most of the program is already written, including the `star` command. Students write the commands needed for `scoot`, and call both `star` and `scoot` in the main program.

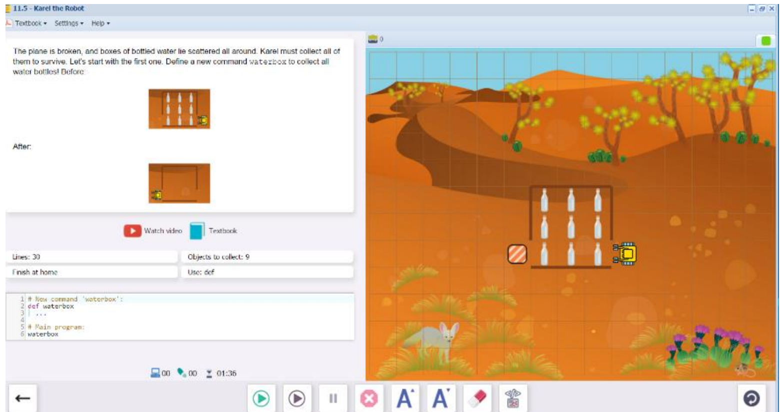


11.5 Karel collects a box of water bottles.

Lines: 40

Commands/keywords: `def`, `waterbox`

There is more than one way to write the code for `waterbox`. Have students compare their solutions. Did they use repeat loops?



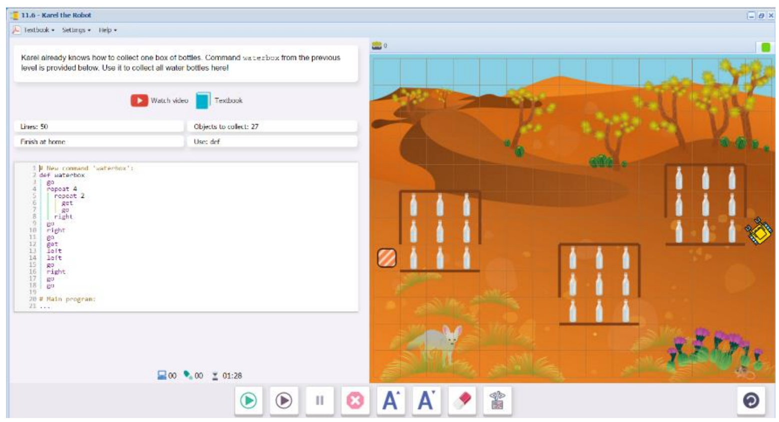
11.6 Karel collects several boxes of water bottles.

Lines: 50

Objects to collect: 27

Commands and keywords: `def`, `waterbox`

Most of the program is already written. Students call `waterbox` and create the steps in between the waterboxes within the main program.



11.7 Karel collects rows of individual water bottles.

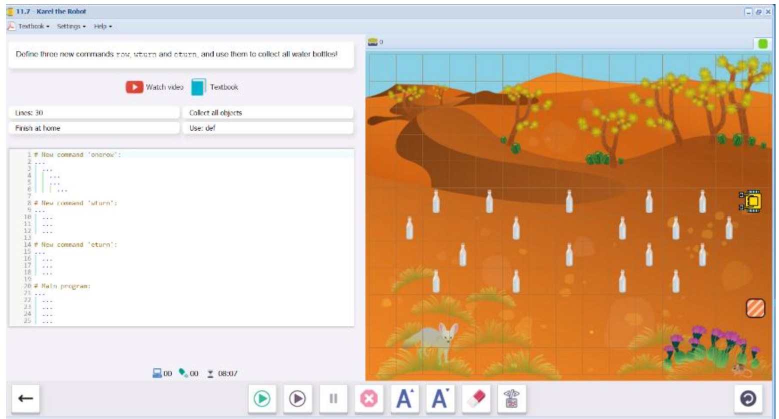
The first three screens explain the defined commands needed.

One command called `onerow` will collect all the bottles in one row.

Two commands are needed to turn Karel from one row onto the next:

`wturn` and `eturn`

Using these three defined commands, students write a program to collect all the bottles.



Lines: 50

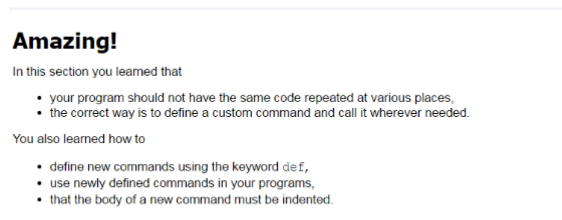
Objects collected: all

Commands: `def`

The structure of the program is laid out, with comment lines used as headings for each part.

Remind students to look for repeated patterns. They will need to use if conditions to pick up the bottles.

Upon completion of 11.7, students will see this message, summarizing what the skills and concepts learned in Section 11. Section 12 is now unlocked.



Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

What are the advantages to writing defined commands?

- Can be used many times in the program by simply putting in the defined command name
- Easier to edit as a separate set of code

Compare your solutions with a partner. Did you come up with the same code, or were there different ways to solve the maze?

Think of a real life scenario where a robot would have to repeat a set of commands over and over again.

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment:

The best way to cement a concept is to use it. As in Karel 1 and 2, students can use Creative Suite to create a game for Karel that creates a defined command and then uses it in a program. Have them picture a set of tasks that Karel would have to repeat. The student instructions are on the following page.

END OF SECTION 11: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game that requires some repeated action that can be used to define a command (15 points)
- The game will include at least one defined command `def` (10 points)
- The game will include a program section that calls the defined command (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 12 (LEVELS 12.1-12.7)

Objectives: Students learn that a new command should always be tested on a simple task first, and then it can be safely used as part of a larger program. They also learn advanced maze skills: how to follow a line that is on Karel's left, or one that is on Karel's right.

Vocabulary: no new terms. Section 12 continues to develop skills learned in Section 11.

Prerequisite Skills

Section 11 must be completed in order to unlock Section 12.

Background knowledge/Introductory Set/Purpose:

In Section 11, we built a defined command for waterbox before using it to collect water bottles from three waterboxes. This is a sensible practice: test a component before including in a larger program. This would be true for any type of assembly. Think of a car. The average car is made up of about 2,000 parts. Many of these parts form assemblies. We would want to test and troubleshoot the parts themselves, the parts assemblies, and finally, the whole car, **before** setting up our assembly line to produce thousands of cars.

In this section, you will build on your understanding of defined commands, testing them before creating a complex program. You will also learn more about navigating an arbitrary maze (one with no pattern).

Direct Instruction/Guided Practice:

Since this level builds on Section 11, there are no new videos. Demonstration levels 12.4 and 12.6 show how to follow an arbitrary path, by either following a wall to left, or to the right. These can be viewed and discussed as a class.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Levels 12.1-12.7: Self-paced instruction

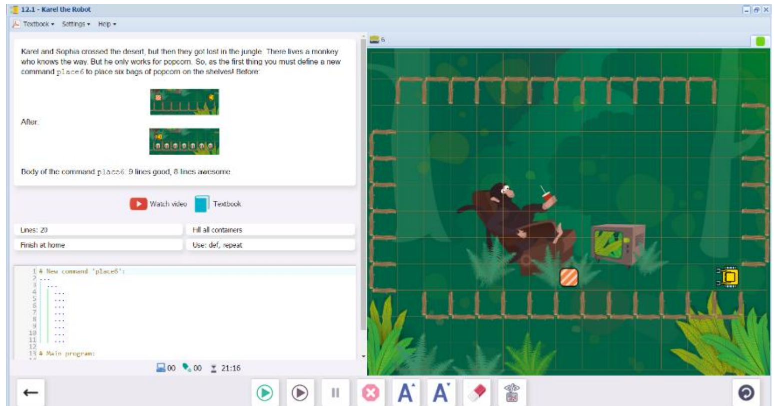
12.1 Karel places 6 bags of popcorn on the shelves (bottom row, starting from the right).

Lines: 20

Fill all containers

Commands: `def, repeat`

This is an example of a defined command `place6` that will be used in a larger program in 12.2

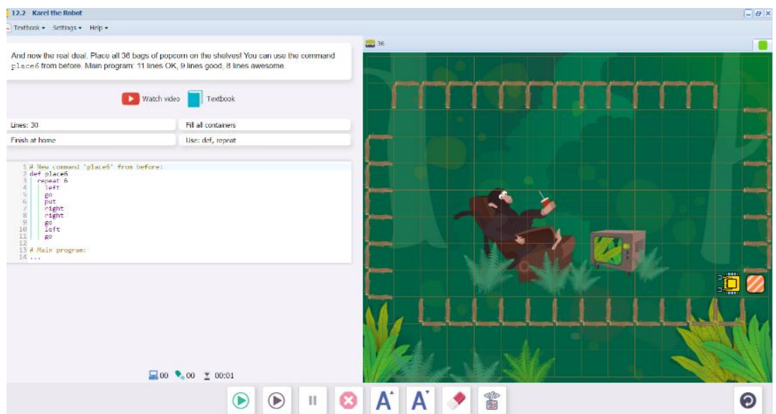


12.2 Karel uses place6 to place bags of popcorn on all 36 shelves.

Lines: 30

Fill all containers

Commands: `def, repeat`



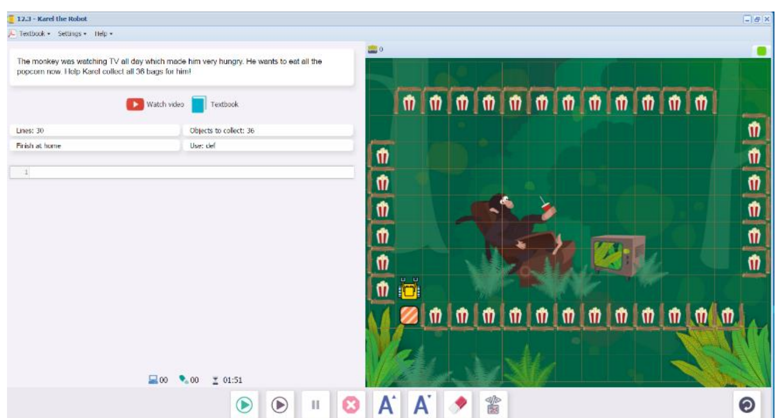
12.3 Karel then collects all 36 bags of popcorn to feed the monkey.

Lines: 30

Collect all objects

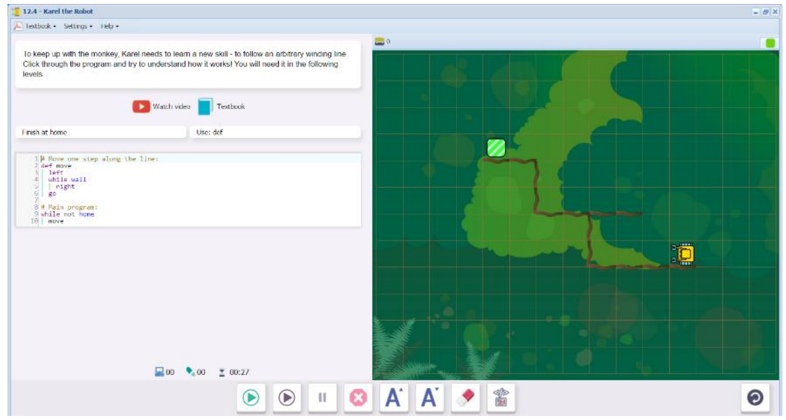
Commands: `def, repeat`

Students practice writing a similar program to 12.2 with some changes. This time Karel is getting instead of putting.



12.4 Step-through demonstration level. Karel follows a winding (arbitrary) path.

Ask students, what is controlling Karel's movements?



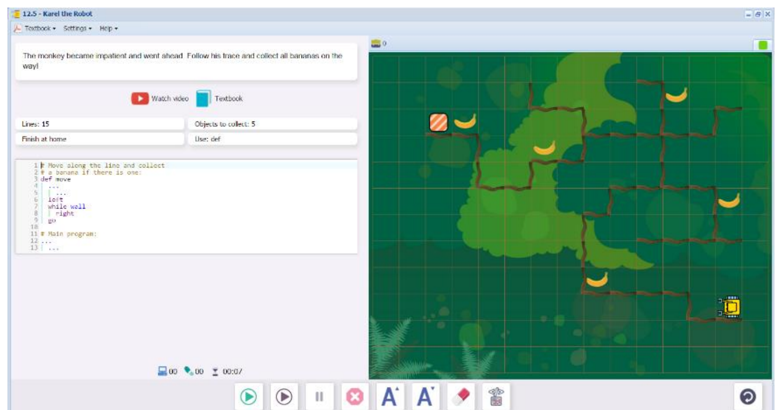
To go along the pathway, Karel uses the wall to guide him. As long as there is a wall, he keeps moving and testing for turns. Therefore, we can use the `while wall` condition as part of the defined command `move`.

12.5 Karel follows a path to find the monkey, collecting any bananas along the way.

Lines: 15

Objects to collect: 5

Commands: `def`



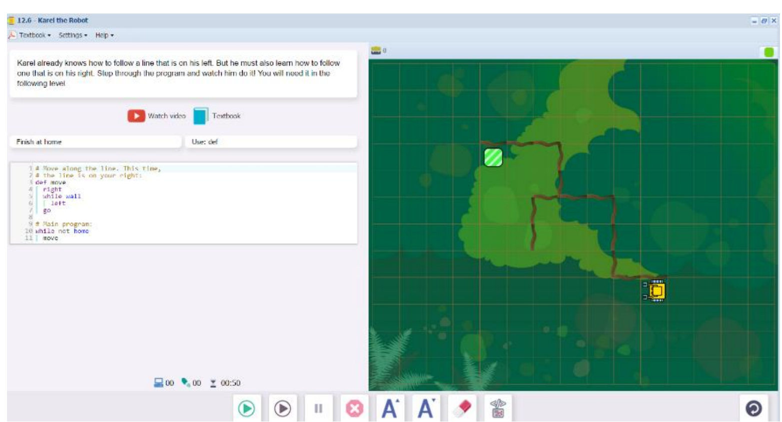
Students write a defined command `move` similar to Level 12.6, which includes an `if` condition to collect bananas.

The program is partially written.

Students might notice that Karel has to check every square, even though we can see many empty sections from our birds-eye view.

12.6 Step-through demonstration level.

This time, Karel follows a wall to his right, instead of his left, as in 12.4 and 12.5.



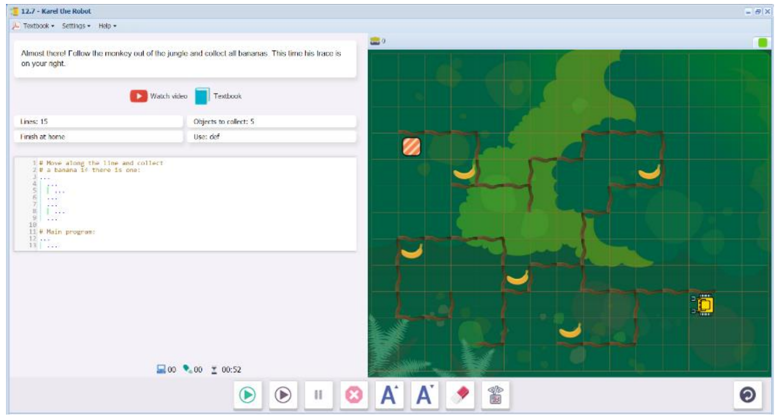
12.7 Karel follows the monkey along a path that is to his right, collecting all the bananas.

Lines: 15

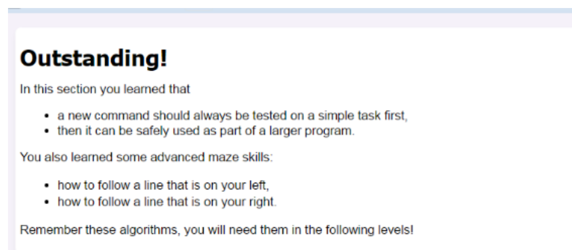
Objects to collect: 5

Use: `def`

This time, students write the entire defined command `move` themselves and call it in the main program.



Upon completion of 12.7, students will see this message, summarizing what the skills and concepts learned in Section 12. Section 13 is now unlocked.



Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

Why is the command `right` written before the conditional loop `while wall`? Why is the command `go` written after the loop? (Karel turns right each time to check for a wall, but he has to keep moving whether or not there is a wall. The only command within the loop is `left`. This command ensures that Karel keeps moving along the wall, and not heading off into the jungle)

Programs and defined commands can modified slightly to fit a new set of conditions:

What changes did you make from 12.2 to 12.3? (The start position is different, so the repeated sections come first, and the single section comes second instead of vice versa. Within the defined command `place6`, the command `put` was replaced by `get`.)

What changes did you make from 12.5 to 12.7? (The path was to the right of Karel, so the `left` and `right` commands had to be switched within the defined command `move`.)

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using defined commands, and movement along a path or wall. A possible assessment is on the following page.

END OF SECTION 12: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game that requires some repeated action that can be used to define a command (15 points)
- This time, create a wall of some kind for Karel to move along.
- The game will include at least one defined command `def` (10 points)
- The game will include a program section that calls the defined command (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 13: LEVELS 13.1-13.7

Objectives: Students learn that the shortest program may not always be the best. A slightly longer program that is much faster, is better than a slightly shorter program that takes a lot of time. Students know to break a complex problem into smaller tasks which are solved first.

Vocabulary: no new terms. Section 13 continues to develop skills learned in Section 12.

Prerequisite Skills

Section 12 must be completed in order to unlock Section 13.

Background knowledge/Introductory Set/Purpose:

In Section 12, we started breaking down large tasks into smaller ones, making sure those worked first before using them as part of a larger program. In this Section, we are still testing small components, then applying them, but we will also see how the same task can be solved in different ways. How do we choose which is best for the situation?

For example, when you first learned to multiply, it was probably easier to skip count: 3, 6, 9, 12, 15, 18. As you learned your multiplication facts it became easier to simply multiply 3 x 6 = 18. However, maybe you have to get to 18 using nickels and pennies. Then, (3 x 5) + 3 makes more sense.

When we write code, we evaluate the conditions of the problem and decide the best way to solve it. We look for some optimal combination of

> **Reliability:** does the code work correctly every time? Try all the mazes. Does it work in each one?
>
> **Speed:** does the program work quickly?
>
> **Ease of use:** is the code easy to understand and repair?
>
> **Limitations:** is the code limited to certain conditions? For example, does the path have to be straight in order for the code to work?

Make notes on the different ways of solving the same problem. It's handy to have a library of algorithms that you can draw from depending on the application.

Direct Instruction/Guided Practice:

There are no videos or step-through demonstrations in this Section, so direct instruction is not necessary. The levels are grouped like this:

13.1 solves a problem using a defined command `move` learned in Section 12, where Karel turns and faces the wall every step, then makes a decision. 13.2 shows a basic set of steps to solve the same problem but more quickly (defined command `column`). 13.3 runs `column` from 13.2 as part of a larger program. Students should compare 13.1 and 13.3.

13.4 uses a combination of commands similar to `move` and `column`. This will be used again in 13.5.

13.6 creates a defined command `cabin` that will be used in 13.7

Individual/Group practice:

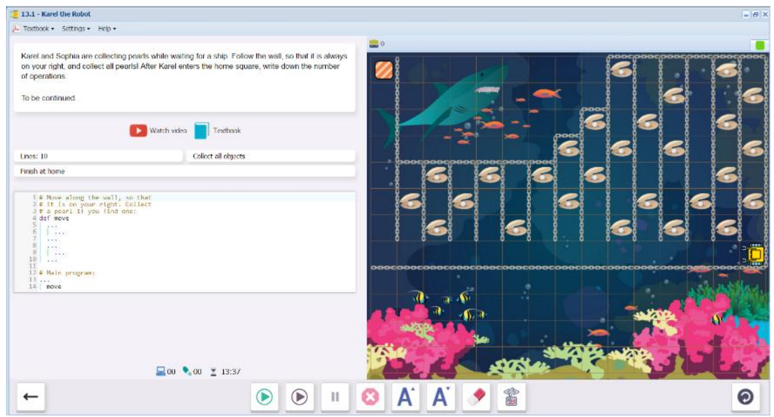
The program is designed to be used individually by students. Encourage peer support, sharing and discussion. At this stage, programming requires some thought and planning.

13.1 Karel follows a wall (as in Section 12), and collects all the pearls.

Lines: 10

Collect all objects

The program is written in the same way as Level 12.5. The defined command `move` is used to move along the wall, collecting pearls. Students may note how long it takes to get through the maze. Karel's progress is slow because he checks for a wall every step.



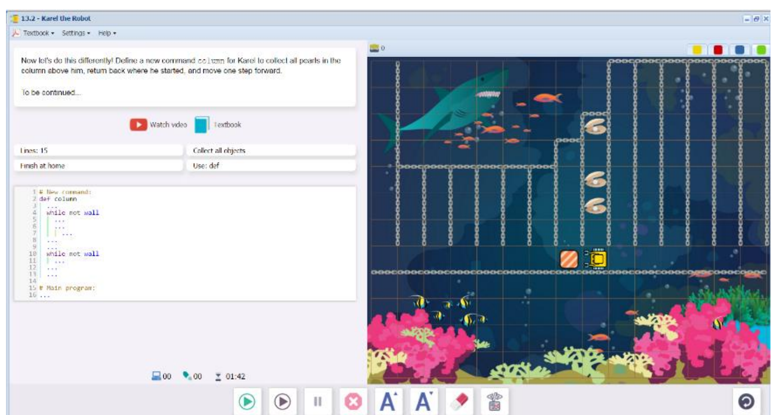
13.2 Karel collects all the pearls in one column.

Lines: 15
Collect all objects
Use: `def`
This time, a defined command `column` is used to:

Travel straight to the north wall, collecting pearls along the way.

Turn around.

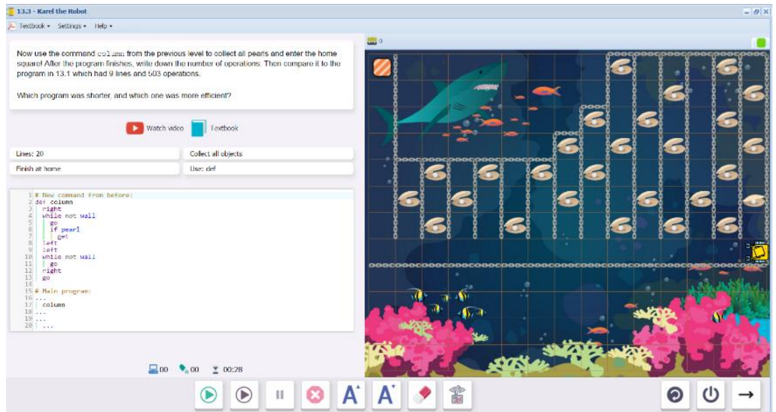
Travel straight to the south wall.



Each loop is written as `while not wall`. In other words, Karel can keep going until he reaches either the north wall on the first loop, or the south wall on the second loop.

This is a much **faster** algorithm. The **limitation** is that the path must be straight.

13.3 Karel collects all the pearls in several columns, using an algorithm similar to 13.2.



Lines: 20

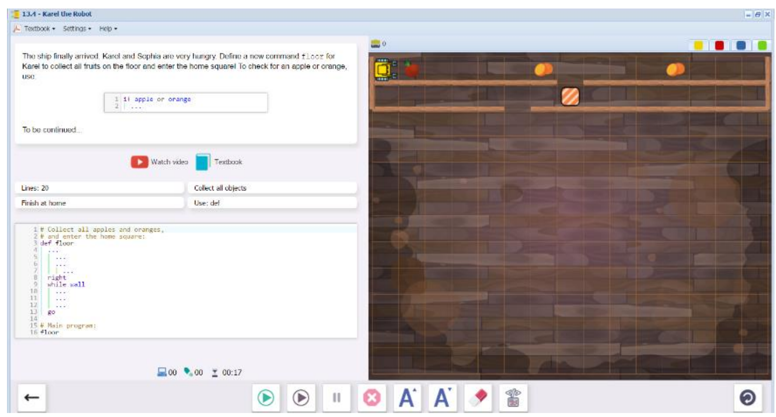
Collect all objects

Use: `def`

The code from 13.2 is already written. Students write the code for the main program. Two repeat loops are needed, one for the 14 columns containing pearls, and one to go home in the last column.

Students are asked to compare the number of operations in 13.3 to 13.1, which had only 9 lines, but 503 operations.

13.4 Karel collects apples and oranges along a row, then moves to the home square on the next row.



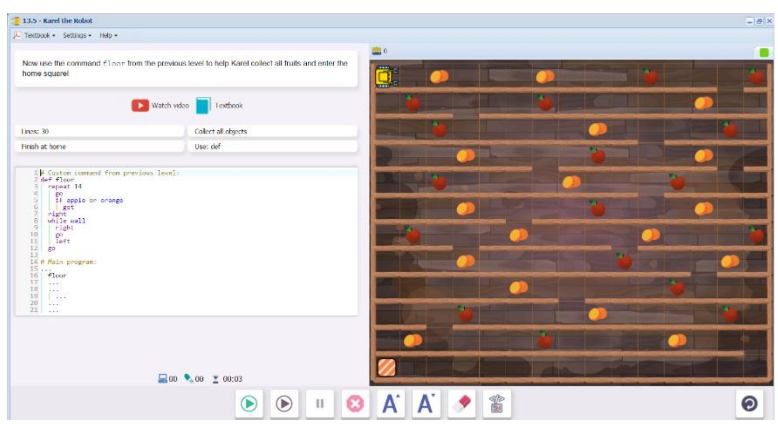
Lines: 20

Collect all objects

Use: `def`

This level uses a defined command `floor` that combines walking straight along the row to collect the fruit (similar to `column` in 13.3), then traveling back along the wall testing for an opening (similar to 13.1).

13.5 Karel collects fruit along several rows, finding openings to the next row until he gets home.



Lines: 30

Collect all objects

Use: `def`

13.5 uses the defined command from 13.4, along with another `column` type set of commands to return Karel to the west end of each row after going through the opening.

13.6 Karel collects all the gold coins in a room.

Lines: 30

Collect all objects

Use: `def, repeat`

Students create a defined command `cabin` to collect all the coins. This will be used again in 13.7. The program uses a nested `if` condition to test for walls (see 10.6 for an explanation).

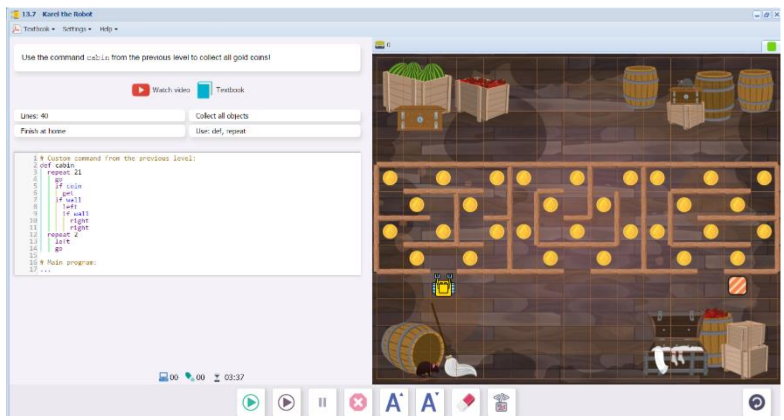
13.7 Karel collects coins from three rooms using the defined command from 13.6.

Lines: 40

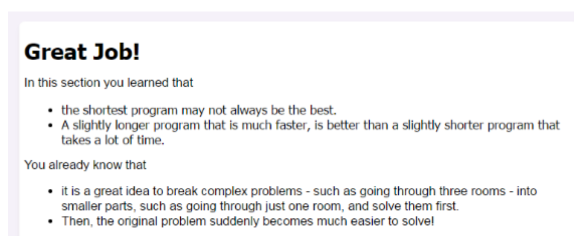
Collect all objects

Use: def, repeat

Students use cabin to collect within the cabins. This level can be solved in different ways, but encourage students to look for repeated patterns. (The number of squares to the second cabin is the same as to the third cabin)



Upon completion of 13.7, students will see this message, summarizing what the skills and concepts learned in Section 13. Section 14 is now unlocked. Students also earn the Purple Belt of First Degree certificate.



Great Job!

In this section you learned that

- the shortest program may not always be the best.
- A slightly longer program that is much faster, is better than a slightly shorter program that takes a lot of time.

You already know that

- it is a great idea to break complex problems - such as going through three rooms - into smaller parts, such as going through just one room, and solve them first.
- Then, the original problem suddenly becomes much easier to solve!

Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

In this level, we practiced several ways to navigate a maze and collect objects. Compare the commands used to build `move`, `column`, and `cabin`. What are advantages and disadvantages to each algorithm? (`move` is flexible because it can follow irregular paths, but it is slow and requires a lot of operations.

`column` is fast but needs a straight path, `cabin` used a simple procedure to check for turns, but a fixed number of repetitions, so it require knowing the number of objects to collect.)

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using defined commands, and movement along a path or wall. A possible assessment is on the following page.

END OF SECTION 13: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game that uses more than one way to move in a maze and collect objects.(15 points)
- The game will include at least one defined command def (10 points)
- The game will include a program section that calls the defined command (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ___________________________________(5 points)

SECTION 14: LEVELS 14.1-14.7

Objectives: Students learn how to create new variables and initialize them with numbers. They use the function inc() to increase the value of a variable by one, the function dec() to decrease the value of a variable by one, and the print command to display results. The print command can be used to display the values of variables while the program is running.

Vocabulary:

Programming terms

Variable: in terms of programming, variable is the **name** and **value** of something that will be recorded in **memory**. The **counting variable** will be used in the lessons in Section 14.

When used in a program, the initial value of the counting variable is set. For example, `n=0` sets the initial value of n to zero.

`inc(n)` tells the program to increase the value of n. The default increment is 1.

`dec(n)` tells the program to decrease the value of n. The default is -1.

`print(n)` tells the program to print the final value of n after the program has ended. Text strings can be printed out on their own or as part of a command. The text is always enclosed in quotation marks.

Example:

`Print "Placed one bottle."` will print "Placed one bottle."

`Print (n) "bottles remain."` will print "36 bottles remain." (if n=36)

Time required

Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills

Completion of Section 13 and a basic understanding of defined commands.

Background knowledge/Introductory Set/Purpose

In Section 11, we learned how to create defined commands to streamline our programming. We then learned algorithms that can be used to create effective defined commands. We will now explore the power of the **variable**. A variable can be used to collect data that can be useful in analysis or application to a task.

There are many types of variables used in programming, but we will only focus on the counting variable in Section 14. There are many situations when a computer or robot might need to count and record items. It could be the number of heartbeats in a minute, the number of items in a store inventory, the number of cloudy days in a month, and so forth.

Big Idea: What kind of variables might be needed in a program? (answers to 5W questions: who, what, where, **when, why, how many, how much, etc.)**

Purpose:

- Section 14 (Levels 14.1-714.7) introduces writing variables, specifically counting variables, and printing statements about the results.
-

Direct Instruction and Modeling:

The main type of variable you will be using is a **counting variable**, for example, n. We set an initial value for n, which could be any number, depending on what we are starting with. Karel uses the if condition to check for whatever it is we have chosen to count, and either increases or decreases the value of the variable, depending on what we need. For example, in the first levels, Karel is counting the maps that he picks up. He starts with no maps (`n=0`). Each time he finds a map (`if map`), he picks it up (`get`) and increases the value of n (`inc(n)`). The program adds these up, one by one. The total can be printed at the end (`print(n)`).

Watch the introductory video in Level 14.1 together as a class, or have students watch it on their own.

http://youtu.be/RNEhx1iz_k4

The programs uses the `print` command to write a statement. A statement that makes sense will need words that are not just commands. These words, called a **text string,** are always enclosed in quotation marks, so that they aren't mistaken for commands. You will see the printed line in an orange box after you run Karel through the program (you may need to scroll down to the end).

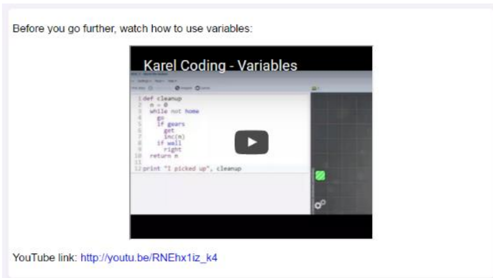
You can also go through the code in the demonstration level 14.1 together, so that students can see how the defined command increments the counting variable, and how the program prints the results. (

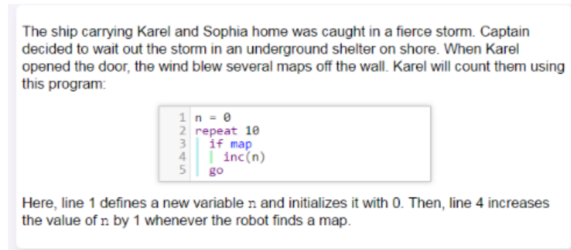
The programs are starting to become more complex. We have been using comment lines as headings. Here, comment lines can include more explanation about what the program is doing. Text strings can be included as descriptors in the lines of code as well. These descriptions will always start with the # sign. This tells the computer to ignore them when it reads these # signs. Get into the habit of describing the function of each custom command just before the code, and describing what will be printed out just before the print line.

Individual/Group practice: The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced instruction 14.1-14.7

14.1 14.1 begins with a video that introduces variables, followed by a screen that explains how to write a counting variable. The next screen is a step-through demonstration. This shows how the counting variable works and the results, which are printed out using the `print` command.

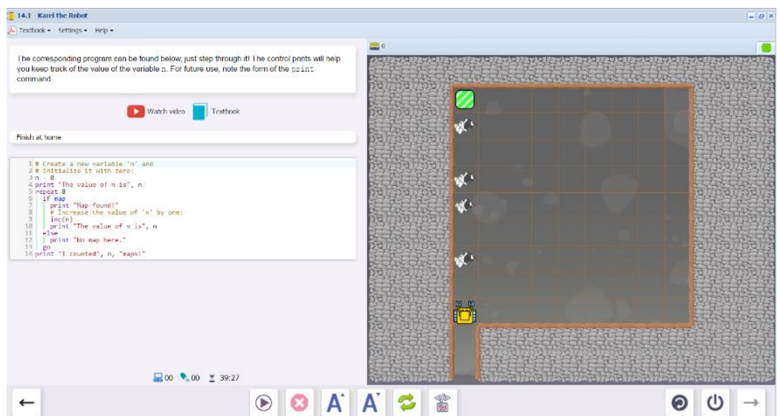




http://youtu.be/RNEhx1iz_k4

Karel collects an unknown number of maps, counting each one as he picks them up.

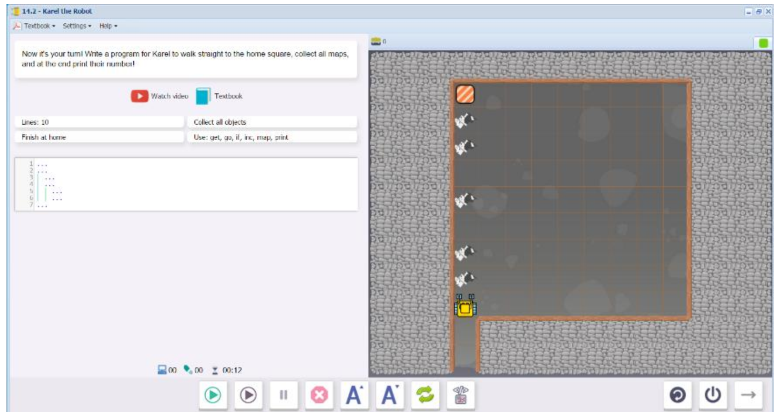
A log is kept of every step and the total number of maps found is printed at the end of the program.



This is what the print log looks like:

```
The value of n is 0
No map here.
No map here.
Map found!
The value of n is 1
No map here.
Map found!
The value of n is 2
Map found!
The value of n is 3
No map here.
Map found!
The value of n is 4
I counted 4 maps!
```

14.2 Karel walks to the home square, collecting and counting maps.



Lines: 10

Collect all objects

Use: `get, go, if, inc, map, print`

Students practice writing the same program, using variables, setting the initial value of the variable to 0 (n=0), and printing out the results. This can be done without text strings, in which case the printout will simply "5". The text strings make a better statement.

14.3 This time, Karel goes around the room, collecting and counting maps.

Lines: 15

Collect all objects

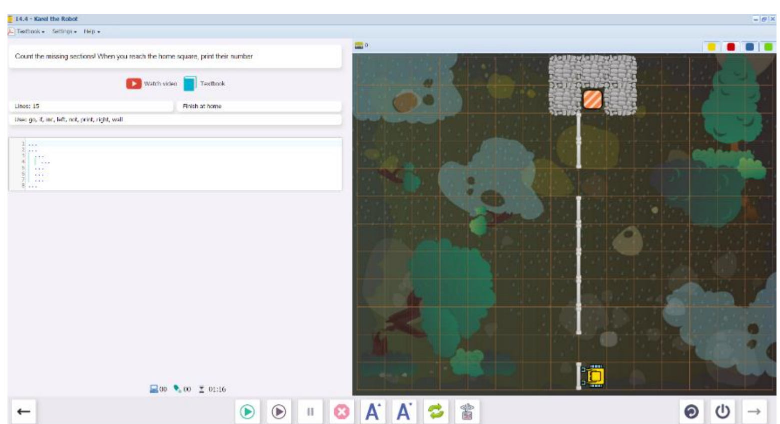
Use: `get, go, if, inc, map, print`

This program is similar to 14.2, but is repeated 4 times with turns. The suggested repeat lines (4 sides, 8 steps for each side) are already written.

Challenge: solve this level in 13 steps (great!), in 10 steps (awesome!) (both are in the solution manual)

14.4 Karel tests for breaks in a pipeline (wall), reporting the number of breaks at the end.

Lines: 15

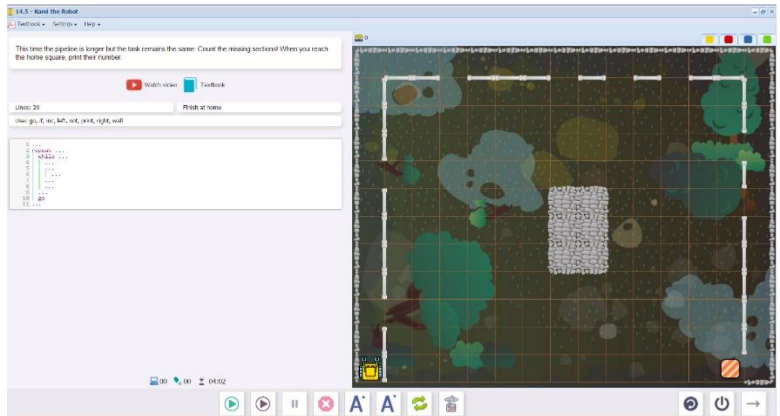


Use: `go, if, inc, left, not, print, right, wall`

Students write a program to count missing sections until they reach home, printing the results at the end.

This level can be solved with `while not home` and the algorithm that turns Karel towards the wall each time, counting if the wall is not present.

14.5 Karel performs the same task as 14.4, but this time on three sides.



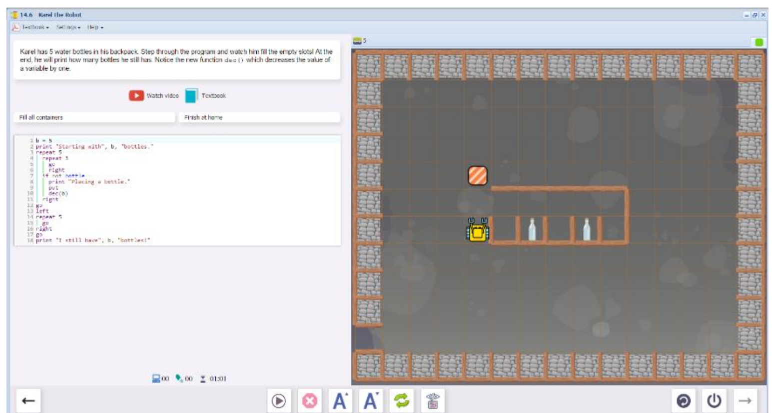
Lines: 20

Use: `go, if, inc, left, not, print, right, wall`

Students write the same program within a `while not wall` loop.

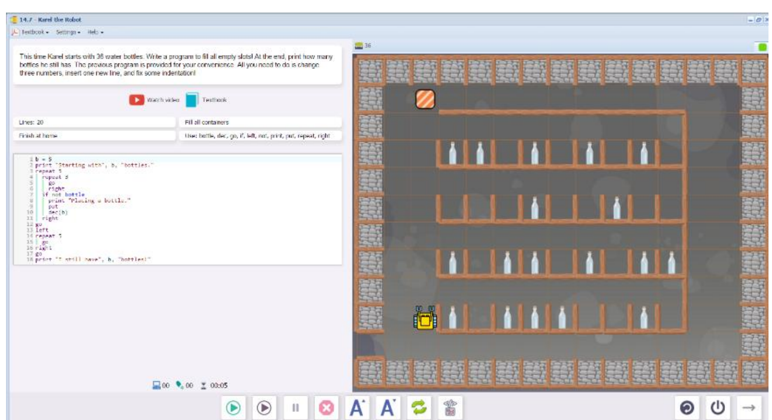
Even though wall refers to both the stone wall and the pipeline, the sensor wall still works because it is used in different parts of the program. Why can't we use repeat loops? (the walls are different lengths)

14.6 Step-through demonstration level, showing how to use the `dec` command to decrease a count.



Karel had 5 water bottles. He fills the empty shelves and then counts how many bottles he still has in his pocket.

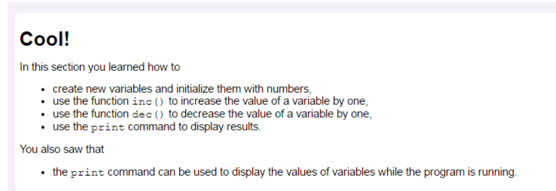
14.7 Karel is carrying out a similar task in a room full of shelves. The program is already written but needs a few repairs.



Students need to change 3 numbers, correct indents, and insert a line.

See Solution manual for the corrected version. This is a good "trial and error" level: students will see the effects of errors that haven't been corrected.

Upon completion of 14.7, students will see this message, summarizing what the skills and concepts learned in Section 14. Section 15 is now unlocked.



Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

Write out the main steps needed to use a variable in a program. (create the variable; initialize it by assigning a value to the variable; increase or decrease the value of the variable based on a condition using inc() or dec(); print the value of the variable after the program is completed)

Describe two situations in real life where a counting program could be useful.

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using counting variables. A possible assessment is on the following page.

END OF SECTION 14: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game that uses items that need to be counted.(15 points)
- The game will include a counting variable. (10 points)
- The game will print out a log of what was being counted, and a statement of the total at the end. Use text strings so that the log and statement make sense. (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ____________________________________(5 points)

SECTION 15: LEVELS 15.1-15.7

Objectives: Students learn how to define new functions and return values using the keyword return, use functions inc() and dec() to increase / decrease the value of a variable by more than one. They know that the value returned from a function can be stored in a variable, and if the returned value is not used, it will be automatically printed. Any code typed after the return command is dead. Variables defined inside commands and functions are local, and local variables cannot be used outside of the command or function where they were defined. Variables created in the main program are global, and global variables should not be used inside commands and functions.

Vocabulary:

Programming terms

Function: a defined command or set of commands based on a variable that returns a value. In Section 15, functions inc() and dec() are used to increase or decrease a variable by more than one (in Section 14, the program only counted up or down by 1).

Local variable: a variable created within a command or function. A local variable cannot be used outside of that particular command or function.

Global variable: a variable created in the main program. A global variable cannot be used inside of a command or function.

Return: the return command ends the function, returning a final value for the variable.

Time required: Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills: Completion of Section 14.

Background knowledge/Introductory Set/Purpose

In Section 14, we learned how to use variables to count items or events. However, we were restricted to counting by one. In this section, we write functions based on variables, which can be used for more complex relationships.

Let's say I'm running a futuristic household where robots comply with our every wish. My robot's job is to count how many guests are present today and where they sit. Then the robot will order three appetizers and a beverage for every guest and deliver them to the correct locations. We can actually write a program that will do this, using functions and variables. Later on, we will learn how to use coordinates that could be added to this scenario to deliver refreshments to the correct locations.

In Section 14, we increased and decreased values of the variables by one. Now, we will learn how to change the values by any number. So getting three appetizers for every guest will not be a problem.

Big Idea: What kind of variables might be needed in a program? (Students might think of answers to 5W questions: **who, what, where, when, why, how many, how much, etc.)**

Purpose: Learn how to define and use functions.

Direct Instruction and Modeling:

Explain definition of a **function**. Students are already familiar with defined commands and variables. Now we combine the two: a function is a **defined command** that includes a **variable**.

Step-through Level 15.1 can be viewed and discussed as a whole class to introduce functions.

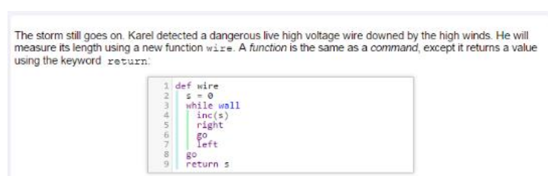
Level 15.4 and **15.5** explains the difference between **a local variable** (one that can only be used within the function) and a **global variable** (one that is used in the main program), and how to transfer values from the local to the global variable. This could also be viewed and discussed as a class after students have worked through 15.2 and 15.3, or reviewed after the Section.

Individual/Group practice:

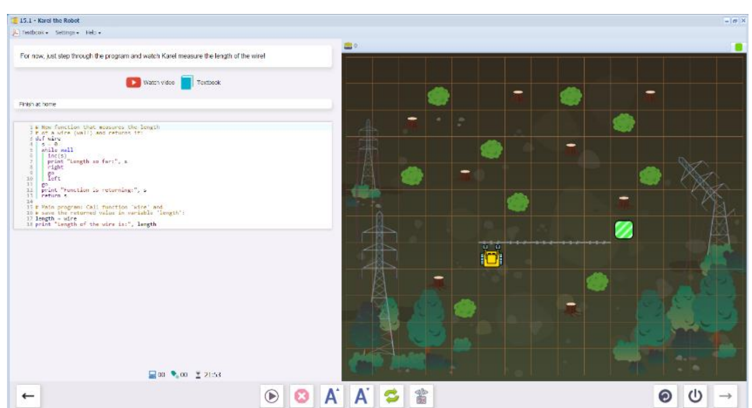
The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 15.1-15.7

15.1 This demonstration level shows how to define a function that includes a variable. In this case the function `wire` is created to find the length of a powerline in `s` units. `s` is initially set to zero. As Karel walks along the powerline (`while wall`), `s` is incremented each time he faces the powerline (`wall`). At the end, the total number of `s` units is returned and can be printed.



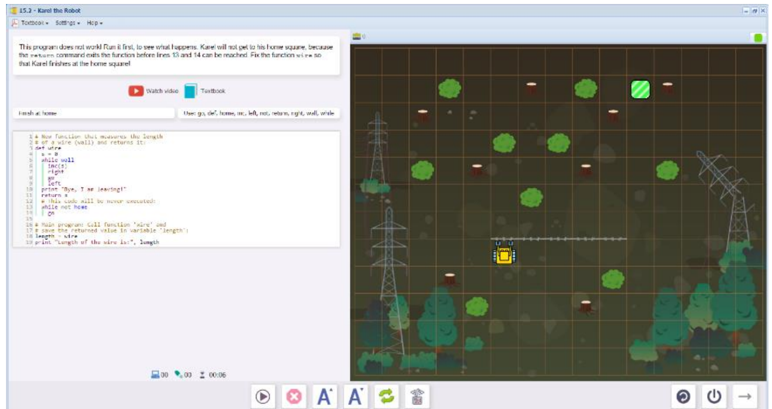
Stepping through the program will show how the function works.



15.2 Karel is again measuring a length of wire. The program is already written, but contains an error.

Use: `go, def, home, inc, left, not, return, right, wall, while`

Students must discover the error and repair the program (all commands must be written before `return` or they will not be executed)

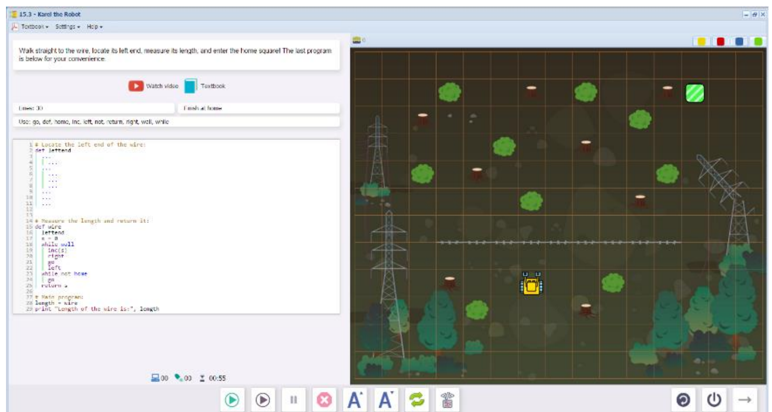


15.3 15.3 builds on 15.2. Karel must first walk to the wire, find its end, and then start counting units.

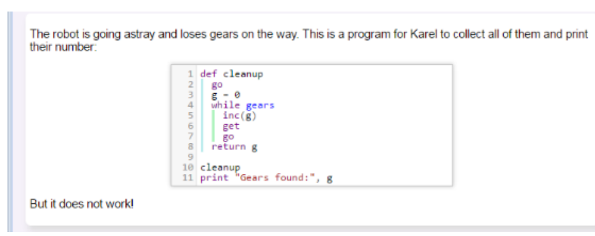
Lines: 30

Use: `go, def, home, inc, left, not, return, right, wall, while`

Students define a command `leftend` to find the left end of the wall. This is another useful algorithm. Move along the wall until the `not wall` condition is met, then move back one square to start measuring.

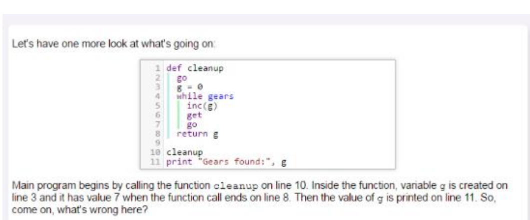


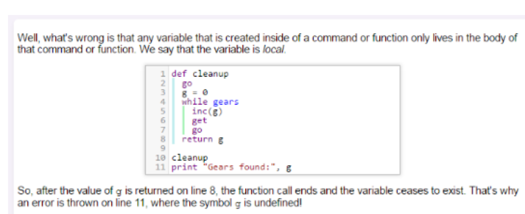
15.4 15.4 is a demonstration on why a local variable must be renamed to be used within the main program. The first three screens explain the problem. Then, students rewrite one line in the program.

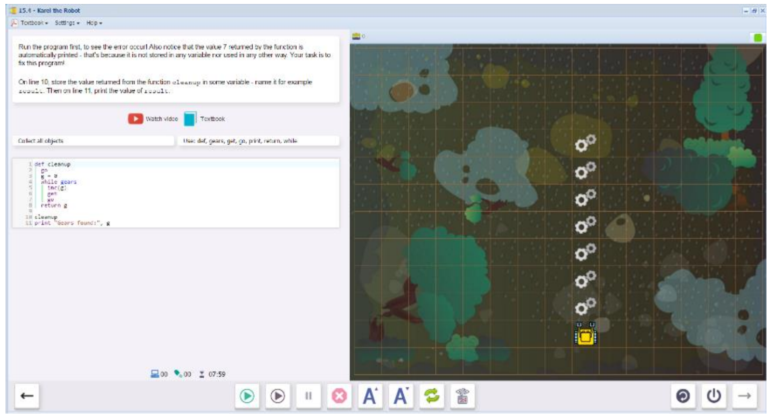


The variable g is created within the function.

The function ends on line 8 with the return command. But we try to use g in the print statement on line 11. If we run the program, we will get an error message stating that g is undefined.







On the program screen, students learn how to rewrite a line to create a global variable `result`, so that the value of `g`, now `result`, can be called in a print statement.

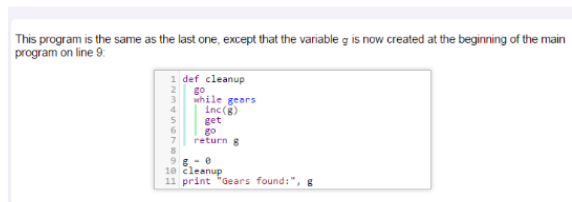
Here's how:

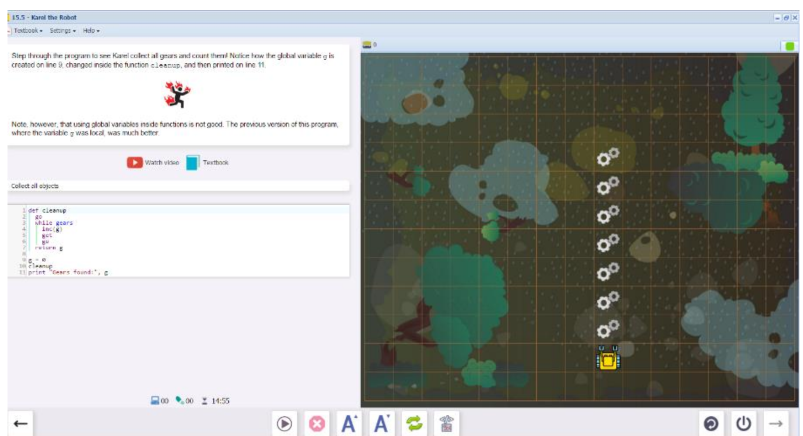
Replace Line 10 `cleanup` with

```
result = cleanup
```

Note that the numerical value for `g` will still print if the function is called without renaming the results. However, the `print` command will not be able to use `g` as part of a statement.

15.5 15.5 builds on 15.4, showing how the variable can be set to a certain value in the main program **before** calling the function. This now makes it a global variable, which will be recognized in the main program.



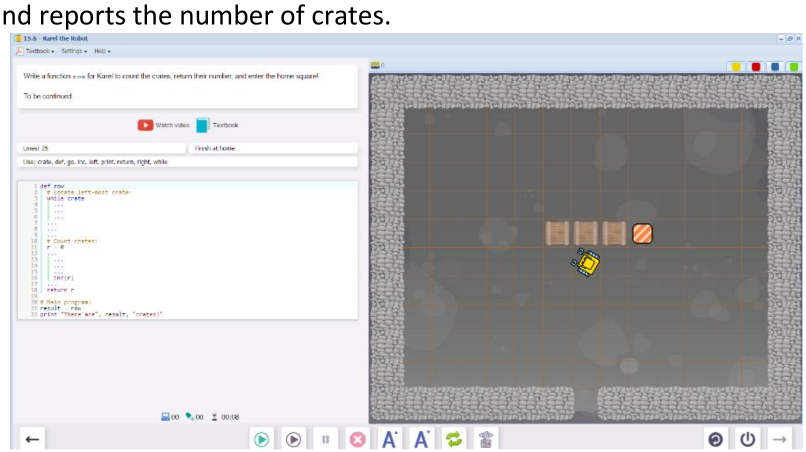


The step-through demonstration shows how this works.

However, the screen also warns that it is not good practice to use a global variable within a function.

It is still preferable to use a local variable, then rename the function in the main program.

15.6 Karel counts crates in a row and reports the number of crates.



Lines: 25

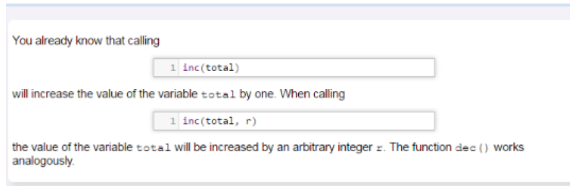
Use: `crate, def, go, inc, left, print, return, right, while`

Students write a function `row` to count the crates and return their number. The program is partially written. The program is similar to 15.3

15.7 Karel counts all the crates in a rectangular array.

The first screen describes how the value returned from counting row can be used as an integer.

This integer increase the count along the column by the row value rather than by 1.

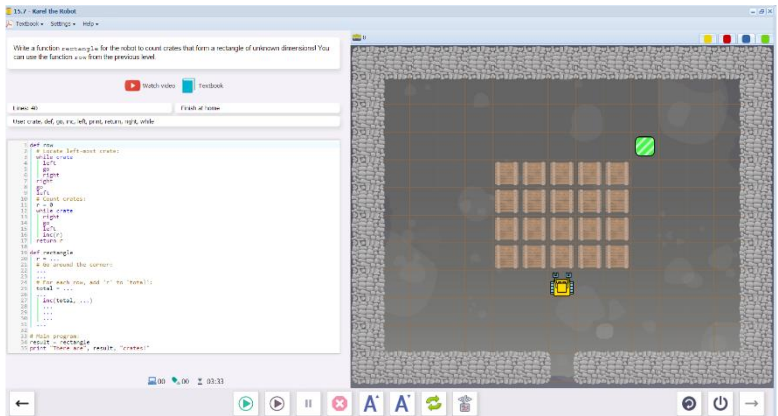


Lines: 40

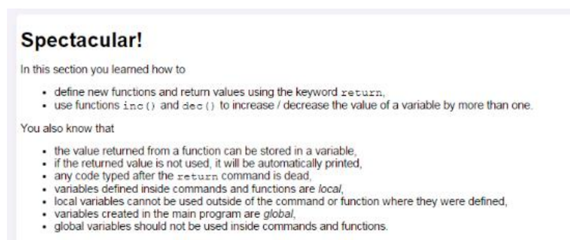
Use: `crate, def, go, inc, left, print, return, right, while`

The code from 15.6, which returns a value for `row`, is already written.

Students complete the second function, which increments total by r (the value returned from row), for each crate.



Upon completion of 15.7, students will see this message, summarizing what the skills and concepts learned in Section 15. Karel 4 is now unlocked. Students also receive a Purple Belt of Second Degree diploma.



Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

Describe functions as defined commands (functions are a type of defined command that contains a variable. It can be used within a program, but not elsewhere.)

What is the difference between local and global variables? How do we export the value from a local variable to a global variable? (Local variables only work within a function. We export the value by defining a variable equal to the function, such as `result` = `row`. Then we can call `result` within a command such as `print`.

What other lengths and areas in real life situations could be measured using a function program?

Assessment: Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using counting variables (see following page).

END OF SECTION 15: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game that uses items that need to be counted.(15 points)
- The game will include a function that uses a counting variable. (10 points)
- The game will print out a statement of the total at the end. Use text strings so that the log and statement make sense. (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

KAREL JR UNIT 4

Karel 4 Overview:

SECTION 16: Students learn how to use the gpsx sensor to determine Karel's horizontal position in the maze, and use the gpsy sensor to determine Karel's elevation in the maze. They also use the symbols ==, !=, < and >. They know that gpsx is 0 in the left-most column and 14 on the right-most one, gpsy is 0 in the bottom row and 11 in the top one. The keyword and ensures that conditions are satisfied at the same time, and the keyword or makes sure that at least one condition is satisfied. Parentheses should be used for expressions such as (gpsx == 7), (gpsy < 3).

SECTION 17: Students learn how to use Boolean (logical) values True and False, store them in Boolean or logical variables), return Boolean values from Boolean functions, and use Boolean variables in conditions and while loops. Students know that Karel's sensors such as wall, nugget, mark, empty, north etc. are Boolean functions. With Boolean variables they can do logical operations such as and or or. The symbol = is used to assign a value to a variable, and for mathematical equality ("is equal to") the symbol == is used. The result of a comparison such as a == b is either True or False.

SECTION 18: Students learn how to generate random integers using the function randint(), make Karel repeat something a random number of times, calculate the maximum and the minimum of a given set of numbers. They know that the function randint(6) can be used to simulate rolling dice.

SECTION 19: Students learn how to create empty and non-empty lists, append items to a list using append(), go through list items one at a time, and get the length of a list L using len(L). They know that lists are like variables, but they can hold multiple values.

SECTION 20: Students learn how to remove and return the last item of a list using pop(), remove and return the first item of a list using pop(0), get the length of a list using len(), use the for loop to go through lists one item at a time, and merge lists. They know that list items can be numbers, Boolean variables, and even text strings. Lists can contain other lists, such as for example [gpsx, gpsy] pairs.

SECTION 16: LEVELS 16.1 – 16.7

Objectives: Students learn how to use the gpsx sensor to determine Karel's horizontal position in the maze, and use the gpsy sensor to determine Karel's elevation in the maze. They also use the symbols ==, !=, < and >. They know that gpsx is 0 in the left-most column and 14 on the right-most one, gpsy is 0 in the bottom row and 11 in the top one. The keyword and ensures that conditions are satisfied at the same time, and the keyword or makes sure that at least one condition is satisfied. Parentheses should be used for expressions such as (gpsx == 7), (gpsy < 3).

Vocabulary:

Sensor: gpsx, gpsy use the grid coordinates to locate Karel (gps is "Global Positioning System"). gpsx = 0, gpsy = 0 is the southeast corner square of the maze.

gpsx indicates the point along the horizontal x axis, measured in grid squares starting on the west (left) side.

gpsy indicates the point along the vertical y axis, measured in grid squares starting on the south (bottom) side.

== means "is equal to". For example, "gpsx == 8" means "The x coordinate position equals 8."

!= is a symbol that means "is not equal to". For example, "gpsx != 7" means "The x coordinate position is not equal to 7." This is useful when you want to carry out a task on every square except the ones flagged with !=. Make sure the two symbols are together with no spaces in between.

< and > serve the same function as in math. gpsx < 4 would mean "All gpsx locations less than 4." gpsy > 6 would mean "All gpsy locations greater than 6."

Expressions can be combined with all these symbols. For example: (gpsx > 9) and (gpsy < 5)

Time required

Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills

Completion of Karel 3 (Section 15).

Background knowledge/Introductory Set/Purpose

Location, location, location! One of the most important parts of programming a robot is pinpointing the location before it carries out a task. This is true for plotters (pen position), for spot welders on a car assembly line, for playing chess, for a farmer using GPS to plant his corn, even for our robot serving beverages and appetizers in our deluxe household.

In 2 dimensions, we use an x and a y axis. In 3 dimensions, we add a z axis. In Karel's simple, 2-dimensional maze, we just count the number of squares from the southwest corner.

The sensors have names: gpsx counts squares along the x axis from left to right. Gpsy counts squares along the y axis from bottom to top.

If students need concrete experiences with this idea, walk out coordinates on floor tiles or tiled game boards, such as chess or checkers.

Big Idea: why do we need to know a robot's position? (think of examples similar to those above)

Purpose: Section 16 (Levels 16.1-16.7) introduces using gps sensors.

Direct Instruction and Modeling: If students aren't familiar with xy coordinate systems, practice naming squares on a chessboard. Name the southwest corner square (0,0), or gpsx = 0, gpsy = 0. Practice locating and naming different squares on the chessboard.

Another way to demonstrate coordinates is to show a real Global Positioning device (if you can't get signal near a window, you may need to take the device outside). The display will show coordinates changing as the unit is moved from one location to another.

The demonstration levels 16.1 (gpsx), 16.3 (gpsy) and 16.5 (both gpsx and gpsy) can be viewed and discussed as a whole class. There are no videos on this level.

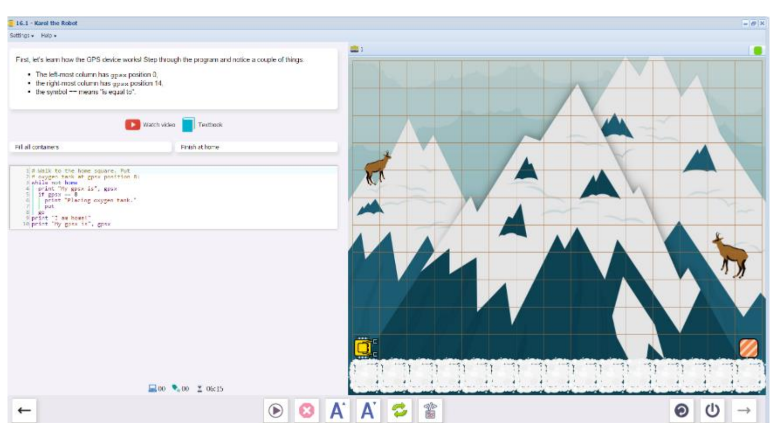
Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 16.1-16.7

16.1 Step-through demonstration level. Karel places an oxygen tank on gpsx == 8.

Note that each square can be reported. The print out shows Karel's progress as he moves from square to square: "My gpsx is 3".

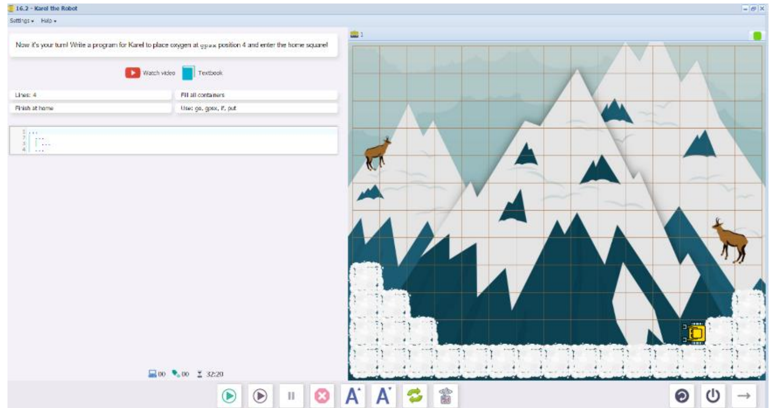


16.2 Karel places an oxygen tank at gpsx location 4 and goes home.

Lines: 4

Use: `go, gpsx, if, put`

Students write a program to carry out Karel's task, using gpsx.

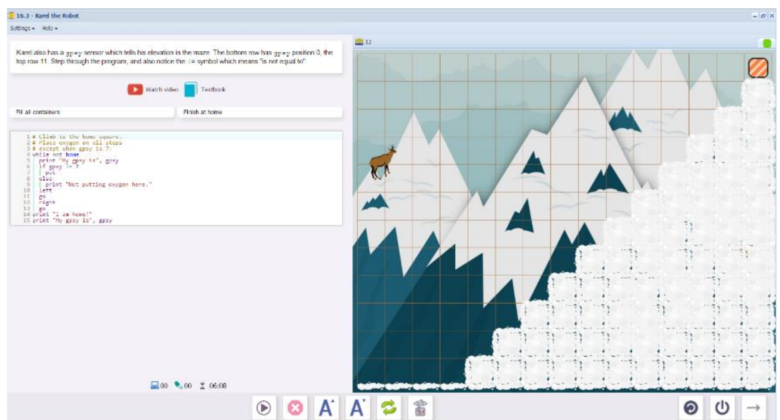


16.3 Step-through demonstration level. Karel uses gpsy sensor to put oxygen tanks on all steps except step 7.

The print log shows the gpsy location for each step.

Notice that the program uses

If gpsy!=7, meaning put the tanks on all squares except for gpsy7.



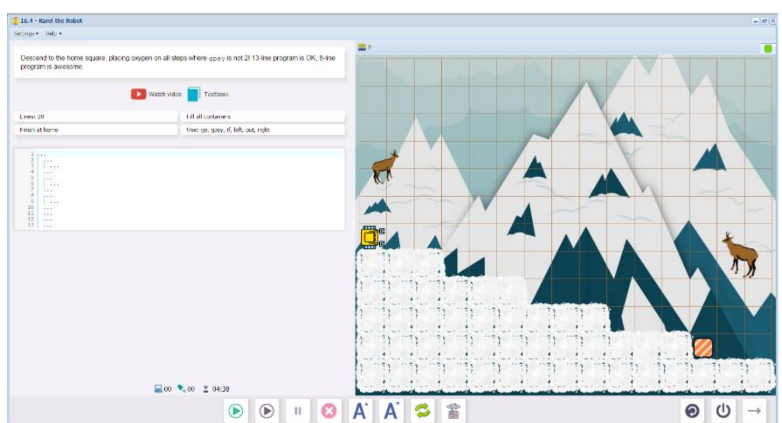
16.4 Karel descends to the home square, placing oxygen tanks on all squares except gpsy2.

Lines: 20

Use: `go, gpsy, if, left, put, right`

Students write the entire program, using `if gpsy!=2` to place the tanks.

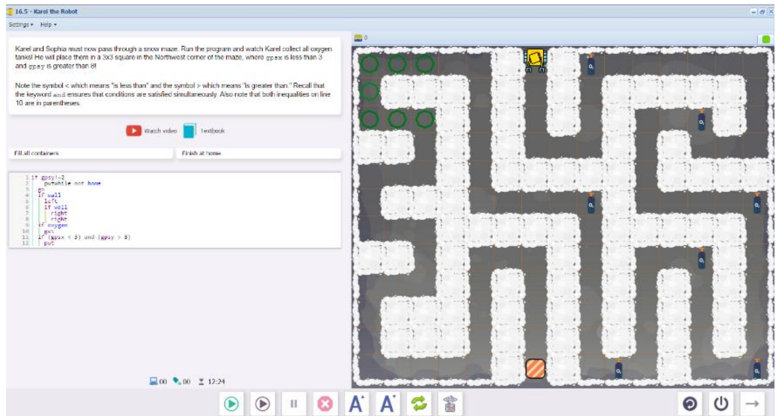
Challenge: solve the puzzle in 13 lines (OK), 8 lines (Awesome!). Both solutions are in the Solution Manual. A simple repeat loop will do the trick.



16.5 Demonstration level showing the use of less than, greater than.

Karel moves through the maze, finding oxygen bottles and then placing them in a specific area defined by gpsx < 3, gpsy > 8

Note the green marks in the northwest corner where Karel will place the tanks.



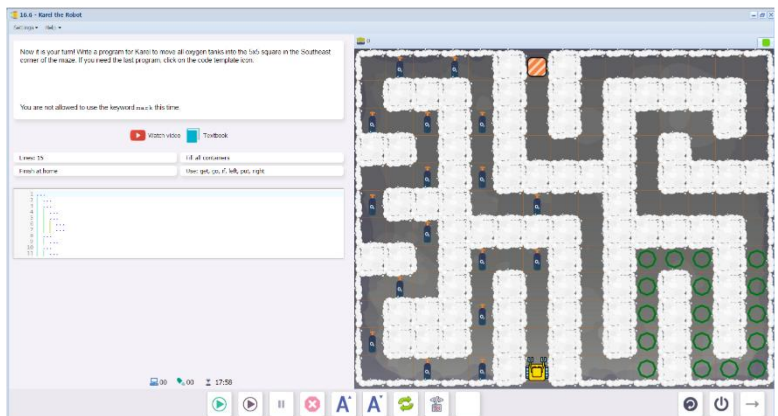
16.6 Karel collects oxygen tanks and places them in a designated area.

Lines: 15

Use: `get, go, if, left, put, right`

Note: do not use the sensor `mark`.

If students are stuck, they can click on the scroll icon (which will appear in the upper left screen) to load the code from the previous screen. This can be edited to fit the new maze conditions.

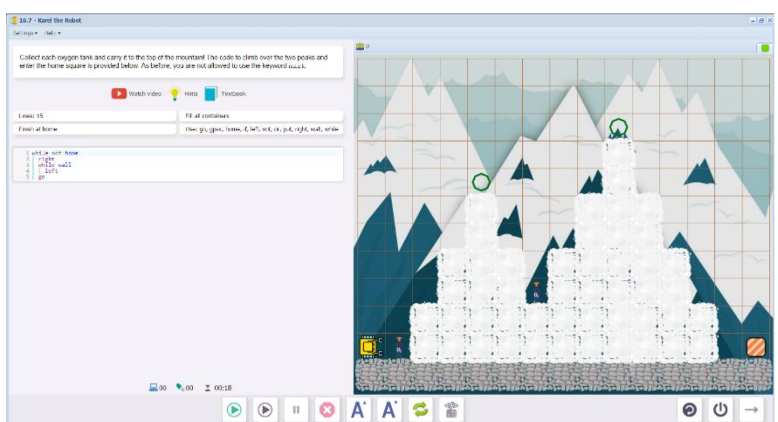


16.7 Karel carries oxygen tanks up to the peak of two mountains, and places one tank on the each peak.

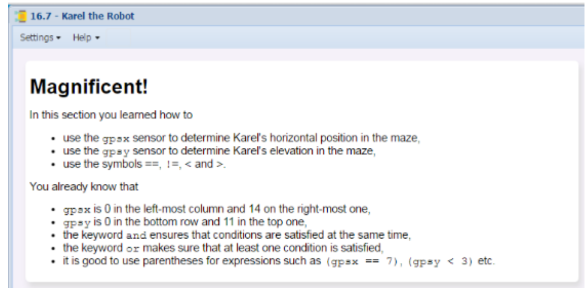
Lines: 15

Use: `go, gpsx, home, if, left, not, or, put, right, wall, while`

The code to climb over the two peaks is already written. Students add the code to get and place the tanks, using gpsx and gpsy coordinates.



Upon completion of 16.7, students will see this message, summarizing what the skills and concepts learned in Section 16. Section 17 is now unlocked.



Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

Identify the gpsx = 0 row and the gpsy = 0 column.

Write an equation for the top (northernmost) row.

Write an equation for the column farthest to the right (easternmost side).

Explain the difference between using `and` and `or` in a condition. (`and` requires that both conditions must be met; `or` requires that either one or the other condition is met)

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using counting variables. A possible assessment is on the following page.

END OF SECTION 16: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game made up of a complex maze (15 points)
- The game will need gpsx and gpsy to solve the puzzle (for example, to pick up or place objects in specific places). (10 points)
- The game will include at least one feature from previous levels, such as repeat loops, conditional loops, defined commands, variables or functions. (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 17: LEVELS 17.1 – 17.7

Objectives: Students learn how to use Boolean (logical) values True and False, store them in Boolean or logical variables), return Boolean values from Boolean functions, and use Boolean variables in conditions and while loops. Students know that Karel's sensors such as wall, nugget, mark, empty, north etc. are Boolean functions. With Boolean variables they can do logical operations such as and or or. The symbol = is used to assign a value to a variable, and for mathematical equality ("is equal to") the symbol == is used. The result of a comparison such as a == b is either True or False.

Vocabulary:

Boolean operator: a logical operator True or False.

True indicates that a condition is true.

False indicates that a condition is false (does not exist, for example).

Time required

Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills

Completion of Section 16

Background knowledge/Introductory Set/Purpose

True/False statements are basic logic. A computer uses true/false statements at the machine level, and at the programming level.

At the machine level, a computer works on the presence or absence of electrical impulses: either something is on, or it is off. From there, we can build all kinds of logic gates. For example, we can compare information from two sources: on/on, on/off, off/on, off/off, and so forth. We can build some pretty complex pathways using these simple gates. The most basic machine level programming uses binary code to represent the on/off condition: 1 = on; 0 = off.

At any programming level, we can use true/false sensors to make decisions. We can also use the sensors to map out the location of objects: for example, which squares contain coins? This is how the built-in sensors in Karel work: a function checks to see if an object or condition is there (true) or not (false), then outputs a decision about what action to take. This is a powerful programming tool. We can combine true and false conditions using `and`, `or`, `not` to make more complex decisions.

Boolean algebra is named after George Boole, a 19th century English mathematician who developed the idea of logical operators.

Big Idea: What kinds of decisions are made in electronics, by computers, and by robots? Do humans make decisions this way?

Purpose: Section 17 (Levels 17.1-17.7) introduces using Boolean operators.

Direct Instruction and Modeling: The demonstration levels 17.1 (using True/False), 17.4 (True/and) and 17.6 (False/or) can be viewed and discussed as a whole class. There are no videos on this level.

Individual/Group practice: The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 17.1-17.7

17.1 Step-through demonstration level. Karel reports on the presence (true) or absence (false) of specific sensors.

As students step through the first screen, they should ask themselves: "Which steps are logical functions that return a True or False result?"

They will see the results printed in a print log. It should be evident that Karel is looking for a specific sensor in each spot. If it is present, the return will be true. If it is not present, the return will be false.

Here is the print log for 17.1:

```
cactus: True
cactus: False
wall: True
wall: False
north: True
north: False
gem: True
gem: False
nugget: True
nugget: False
empty: True
empty: False
home: False
home: True
```

17.2 Karel picks up either a snake or a spider on his way home. He will report one way if he found a snake (True), and another if he did not (False).

Lines: 15

Use: `False, get, go, snake, spider, True`

The program is partially written. Students fill in the logical operators.

17.3 Karel is exploring a ruin. If he finds an object he will pick it up. If it is a nugget, the results are True; if a gem, the results are False.

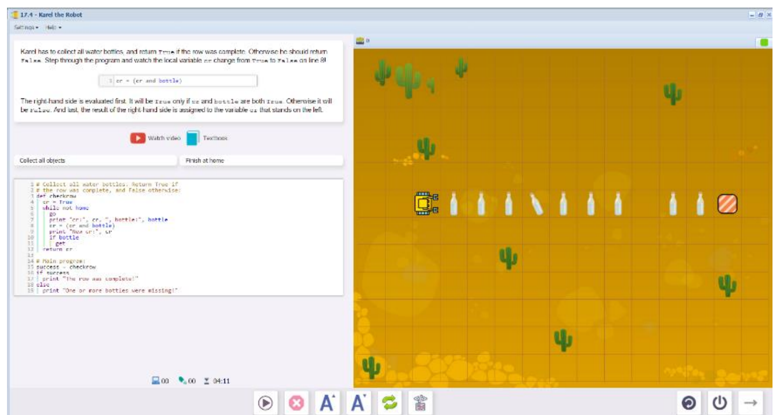
Lines: 20

Use: `False, gem, get, nugget, True`

The program is partially written. Students complete code needed in the function ruin, which tests whether the object is a nugget or not.

The program prints out a statement based on the results. ("I found a gold nugget!" if true, "I found a gem!" if false)

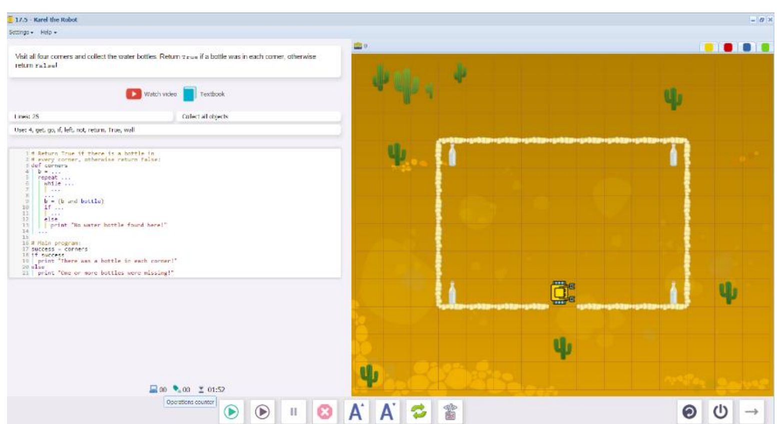
17.4 Step-through demonstration level. Karel checks to see if the row is completely filled with bottles. He collects them and makes a statement about the row at the end.



The function `checkrow` sets the logical statement `cr=true` at the beginning. If a bottle is found, the statement is reset as true; if not, it changes to false. This is an example if a statement that starts out true.

The `and` operator is used as part of the testing statement.

17.5 Karel checks all four corners of a ruin for water bottles, returning different statements depending on whether or not there are water bottles.



Lines: 25

Collect all objects

Use: `4, get, go, if, left, not, return, True, wall`

The statement `b=true` starts as true. If any water bottles are missing when Karel turns left corners, then the statement will change to `b=false`.

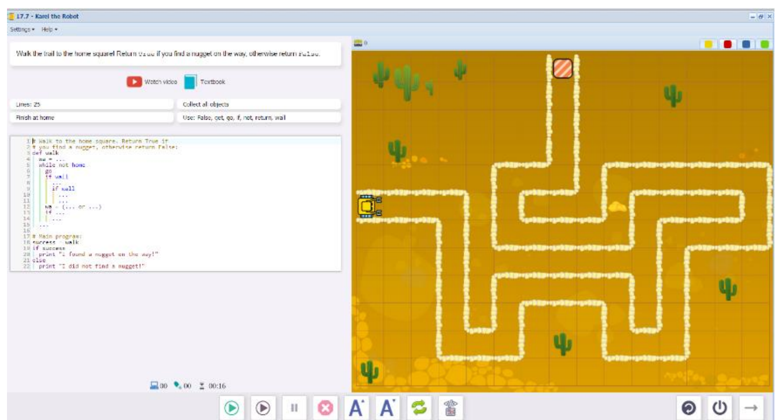
17.6 Step-through demonstration level Karel is looking for a lost map.

When he finds it, the statement will become true and the printed statement will be "I found the map!"

This is a case where the logical statement starts out false, which is the opposite of 17.5

Notice that an `or` condition is being used as a test this time.

17.7 Karel walks through the maze on his way home. If he finds a nugget, the logical statement will change from false to true.



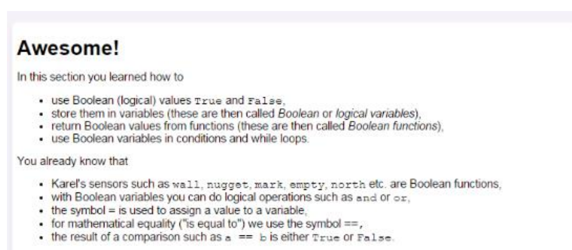
Lines: 25

Collect all objects

Use: `go, gpsx, home, if, left, not, or, put, right, wall, while`

The code is partially written. Students add missing code, including the logical operators.

Upon completion of 17.7, students will see this message, summarizing what the skills and concepts learned in Section 17. Section 18 is now unlocked.



Awesome!

In this section you learned how to

- use Boolean (logical) values `True` and `False`,
- store them in variables (these are then called *Boolean* or *logical variables*),
- return Boolean values from functions (these are then called *Boolean functions*),
- use Boolean variables in conditions and while loops.

You already know that

- Karel's sensors such as `wall`, `nugget`, `mark`, `empty`, `north` etc. are Boolean functions,
- with Boolean variables you can do logical operations such as `and` or `or`,
- the symbol = is used to assign a value to a variable,
- for mathematical equality ("is equal to") we use the symbol `==`,
- the result of a comparison such as `a == b` is either `True` or `False`.

Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

Explain how true/false values can be written into functions (assign a value to a variable).

What is the purpose of True/False values? (True/False values can map an occurrence, collect data, change or keep the condition of a larger set such as a row, make a decision on what action to take.)

Think of examples of True/False values in the real world. How do computers, robots and humans use them?

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using True/False values as part of a Boolean function. A possible assessment is on the following page.

END OF SECTION 17: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game made up of a complex maze (15 points)
- The game will use True or False operators to report back findings. (10 points)
- The game will include at least one print statement based on the logical results. (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ____(5 points)

SECTION 18: LEVELS 18.1 – 18.7

Objectives: Students learn how to generate random integers using the function randint(), make Karel repeat something a random number of times, calculate the maximum and the minimum of a given set of numbers. They know that the function randint(6) can be used to simulate rolling dice.

Vocabulary:

> **Random:** a random value is selected without regard to pattern, order, or reason. Each value within the set has an equal chance of being selected. A coin has an equal chance of landing heads or tails. A die has an equal chance of landing with 1, 2, 3, 4, 5 or 6 face up.
>
> **Randint:** a command that selects a random integer. The command is written `randint(n)`, where n is an integer between 1 and n.
>
> **Maximum:** the greatest value out of a set of values. The maximum is determined by a function that compares values.
>
> **Minimum:** the least value out of a set of results. The minimum is also determined by a function.

Time required:

Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills: Completion of Section 17

Background knowledge/Introductory Set/Purpose

People are familiar with the idea of randomness from playing games that involve chance. If you have ever rolled a die, then you have generated a random number. The die randomly selects how many steps you will take in the next turn. Why? Random selection ensures fairness among the players, and it also makes the game less predictable and therefore more exciting.

Randomness also occurs to some extent in nature, although what appears to be random often contains poorly defined patterns, or some form of bias that favors one result over another.

Random numbers are useful in many computer applications. Here are some examples:

> **Testing** has many applications for random numbers. In a medical trial, patients can be randomly selected as to whether they take the active medication or the placebo. Students taking a test may be randomly assigned test questions from a pool.

Art and design, especially computer **animation**, use random placements to make a surface look more natural, such as a pebble beach, or an animal's fur.

Lotteries and games of chance depend on random numbers, although the probabilities are very carefully calculated to favor the house. Lotteries can also be used to fairly distribute scare items, such as tickets to an event, hunting tags, and so forth.

Purpose: Section 18 (Levels 18.1-18.7) introduces the use of random numbers in functions.

Direct Instruction and Modeling: The demonstration levels 18.1 (using the roll of a die to advance Karel), 18.5 (determining the maximum height of a randomly generated set of columns) can be viewed and discussed as a whole class. There are no videos on this level.

If students are unfamiliar with random selection, they can run tally mark trials on:

Coin tosses (heads/tails)

Four tiles, each a different color (such as the plastic or wooden square tiles used as math manipulatives) drawn from a bag.

Die rolls

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 18.1-18.7

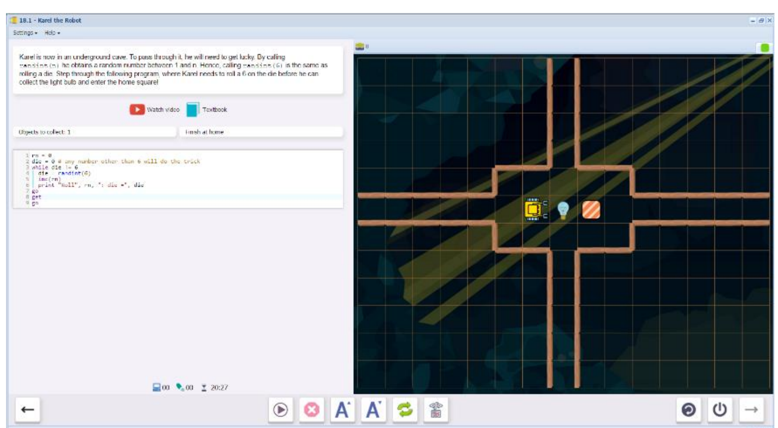
18.1 Step-through demonstration level. Karel must "roll a 6" in order to pass.

In this case `randint(6)` is going to generate a number between 1 and 6. Once a 6 is rolled, then Karel will be able to proceed to home.

A `while` loop tells the program to keep rolling until 6 is the result.

```
while die != 6
```

Students will see the results of each roll on the print log.

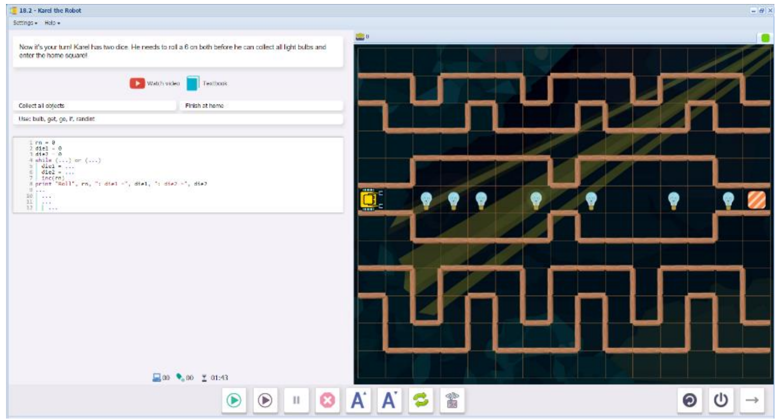


18.2 Karel must roll a 6 on each of two die to pass

Collect all objects.

Use: `bulb, get, go, if, randint`

The program is partially written. Students fill in blanks, including the function and loops for the random number generators.

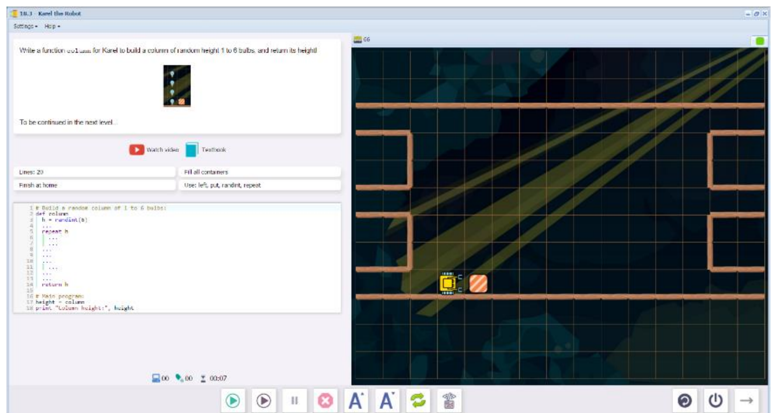


18.3 Karel builds a column with a random height.

Lines: 20

Use: `left, put, randint, repeat`

The program is partially written. Students complete code needed to build the column, return back to the base, then turn and go home.

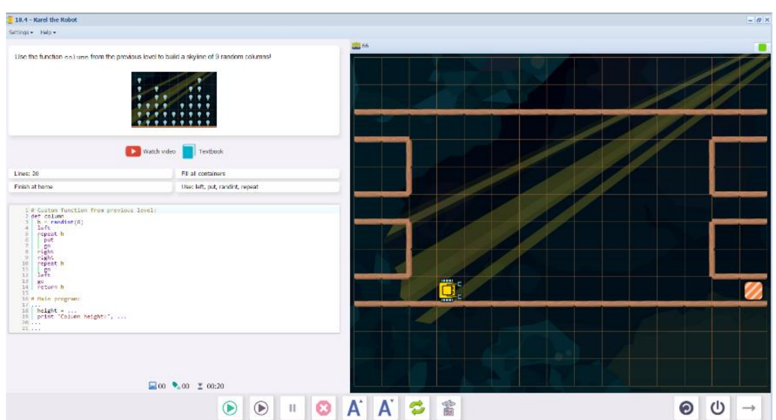


18.4 Karel builds a skyline, using the function `column` from 18.3.

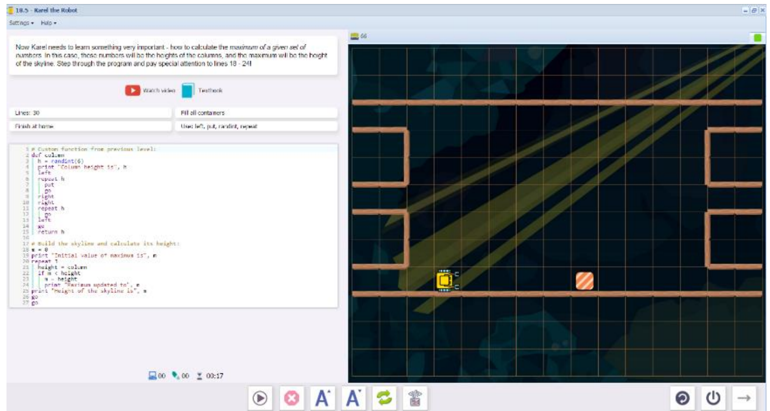
Lines: 20

Use: `left, put, randint, repeat`

The function `column` is already written. Students complete the program by running a repeat loop.



18.5 Step through demonstration level. Karel builds a skyline as in 18.4. This time, he also calculates the maximum height of the skyline.

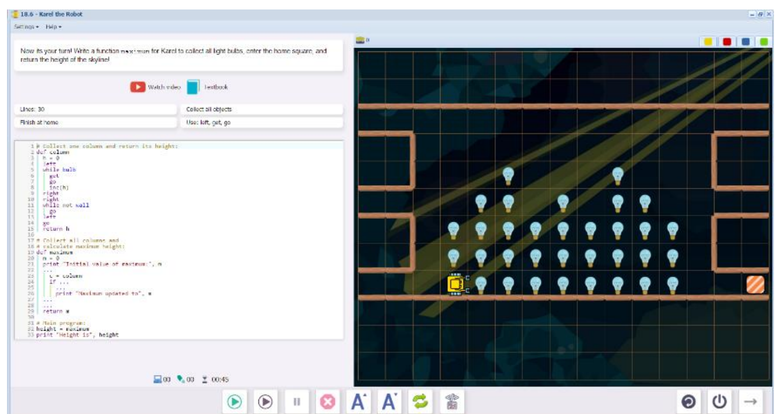


Lines: 30

Use: `left, put, randint, repeat`

A function is used to determine the maximum height. At first, the maximum is set to zero (m=0). Each time a new height is calculated, it is compared to m. If it is greater than m, the value of m is changed to the value of h.

18.6 Karel collects all the lightbulbs and calculated the maximum height of the columns.

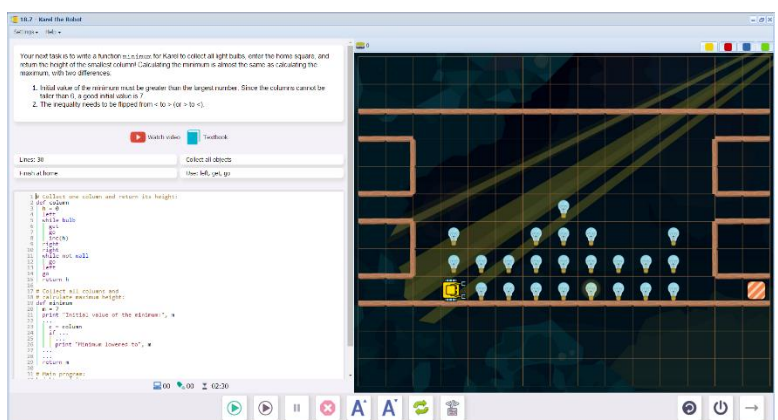


Lines: 30

Use: `left, get, go, repeat`

Students practice writing the function used to calculate the maximum, as in 18.5. Most of the program is already written.

18.7 Karel collects all the lightbulbs and calculates the minimum height of each column.



Lines: 30

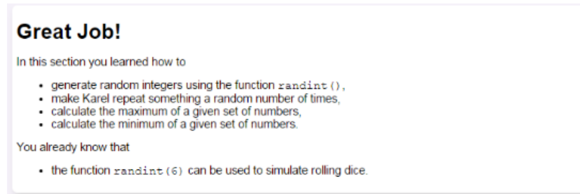
Collect all objects

Use: `left, go, get, repeat`

The code is similar to 18.6, except that we start by comparing to a maximum (set by our randint range value), and only decrease the variable when it compared to a column height that is shorter.

The code is partially written.

Upon completion of 18.7, students will see this message, summarizing what the skills and concepts learned in Section 18. Section 19 is now unlocked.



Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

How do you generate a random number in a program (initialize a variable, use the command randint(), make the variable equal to the random number)?

How do you repeat the rolls of a die until a certain value is reached (see 18.1)?

Explain the process of finding a maximum or minimum (see 18.5, 18.7).

Think of two real life scenarios where a robot or a computer could use random numbers. (Post ideas on a common board).

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using random integers. A possible assessment is on the following page.

END OF SECTION 18: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game that will accommodate a randomly generated pattern (such as the skyline in 18.5). (15 points)
- The game will use `randint` to generate the pattern or make the choice. (10 points)
- The game will include at least one print statement based on a maximum or minimum. (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 19: LEVELS 19.1 – 19.7

Objectives: Students learn how to create empty and non-empty lists, append items to a list using append(), go through list items one at a time, and get the length of a list L using len(L). They know that lists are like variables, but they can hold multiple values.

Vocabulary:

List: A list is a set of items, enclosed in square brackets and separated by commas. For example: L = [2,2,8,3,4]

Empty List: A list that does not contain any items, shown by empty square brackets. For example: L = [ ]

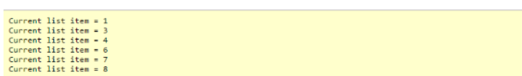
Non-empty List: A list that contains items. For example: L = [1,6,8,3]

Append: Add items to a list. For example: L.append (x) , L.append ([gpsx, gpsy]). Notice that two or more items must be enclosed in one set of parentheses.

Parse: Examine the items in a list. The items can be printed out as a line-by-line log of the list, using a For loop.

For loop: A `for` loop is able to iterate (repeat a function) for items in a list. It is indented the same way as other loops. For example, a `for` loop can print out a log of these items:

```
for x in L
  print "current list item:", x
```

resulting in 

Length of a list: `len(L)` is the number of items in the list.

Time required

Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills: Completion of Section 18

Background knowledge/Introductory Set/Purpose

We have learned how to find items and count them, sense conditions (True/False), generate a group of items using random numbers, use gps coordinates. All of these functions can generate useful data that we may want to store and use. This is where lists come into play. We can make lists of anything from

True/False determinations to gps locations. In this section, we will learn how to add items to a list using `append`, and to analyze the contents of a list. In the next section, we will learn how to extract items from a list using `pop`.

If we want to make an exact replica of a map or an item, we can record all the details in a list, then copy each item in the list. Each item in the replica will be in exactly the same position as the original. One of the greatest advances in assembly lines is using robotics to build exact copies of an original design with great precision

Another common use of lists is in inventory. We buy or create items for inventory, and then use or sell items from inventory. The inventory list not only tells us how many items are in this list, but also the order of when the items are added to or taken from the list (more about this in Section 20).

Purpose: In Section 19 (Levels 19.1-19.7) students learn how to store values in a list, print out some or all the items in a list, retrieve them from a list, and append one list to another list.

Direct Instruction and Modeling: The video and demonstrations in 19.1, 19.4, and 19.6 can be viewed and discussed as a class. Here is the link:

https://www.youtube.com/watch?v=POLH6ouTtrM

The video is followed by three screens that explain empty and non-empty lists, and how to append to a list. 19.1 finishes with a step through demonstration.

19.4 steps through the process of parsing and printing out a list.

19.6 steps through how to obtain and use the length of a list (number of items in the list).

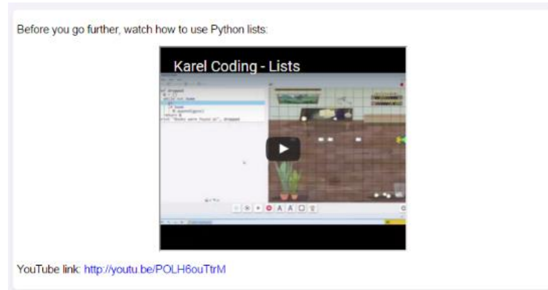
Individual/Group practice:

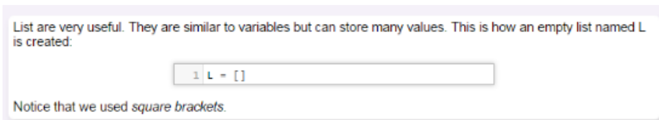
The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

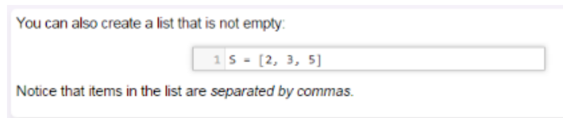
Self-paced Instruction: Levels 19.1-19.7

19.1 19.1 begins with a video that explains how lists work.

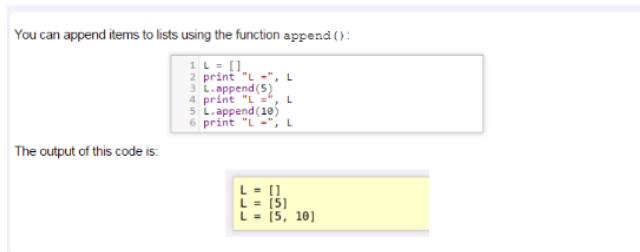
https://www.youtube.com/watch?v=POLH6ouTtrM





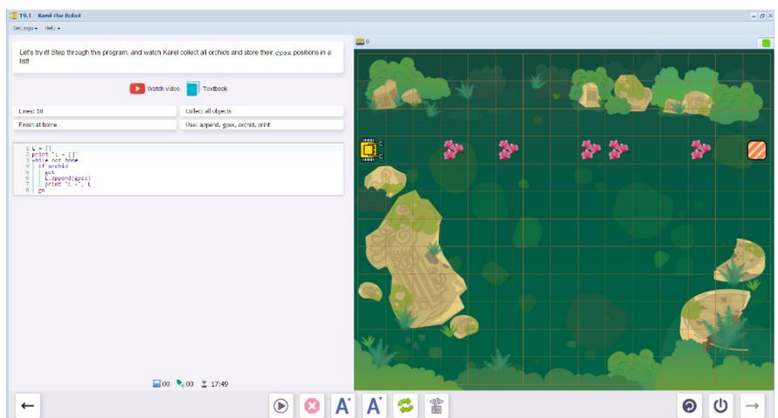


It then moves on to three screens that introduce the concept of lists (empty lists, non-empty lists, how to append to a list).



In the step-through demonstration, the gpsx locations of the orchids are stored in a list.

Each time Karel locates an orchid, the gpsx value is appended to the list.



19.2 Karel collects orchids on a diagonal path, recording the gpsy location of each one in a list.

Lines: 20

Collect all objects

Use: `append, gpsy, orchid, print, repeat`

Students write a program to create a list, collect the orchids and append their gpsy locations to the list.

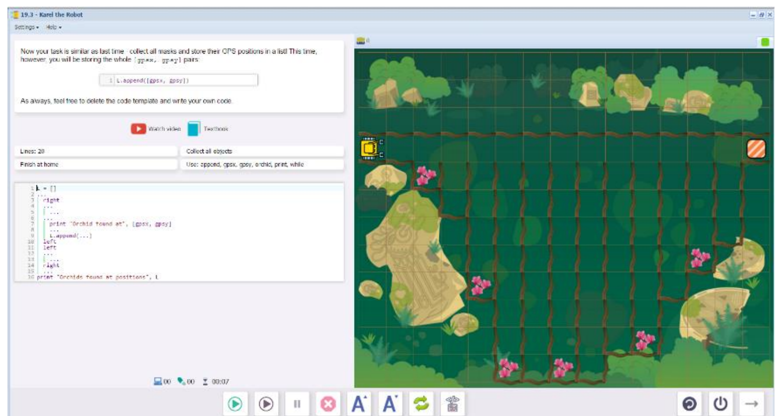
19.3 Karel collects orchids and makes a list of both the gpsx and gpsy coordinates.

Lines: 20

Collect all objects

Use: `append, gpsx, gpsy, orchid, print, while`

Students complete the code, using algorithms for checking columns as in previous levels, creating a list and appending the locations to it.

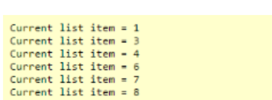


19.4 Step through demonstration level. Parse lists (analyze the list, one item at a time) and print out a log of all the items in the list.

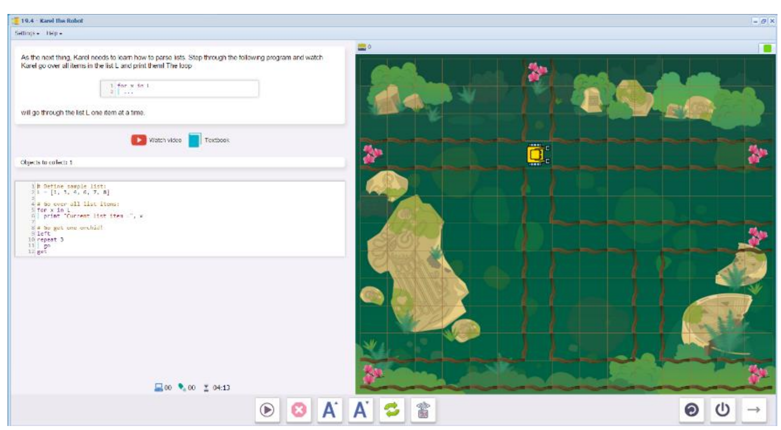
To do this, we use a `for` loop and a variable.

```
for x in L
  print "current list item:", x
```

which results in:



```
Current list item = 1
Current list item = 3
Current list item = 4
Current list item = 6
Current list item = 7
Current list item = 8
```

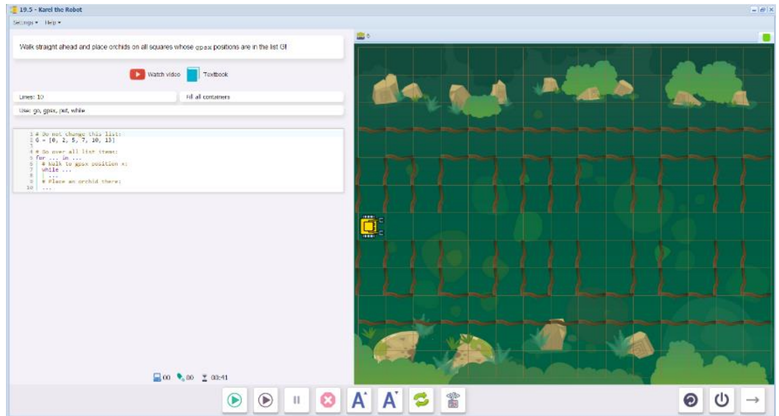


19.5 Karel uses a list of gpsx locations to place the orchids in his pocket.

Lines: 10

Use: `go, gpsx, put, while`

Think about what causes Karel to stop walking: he reaches one of the gpsx coordinate on the list. The while condition for `go` will be `while gpsx != x`.



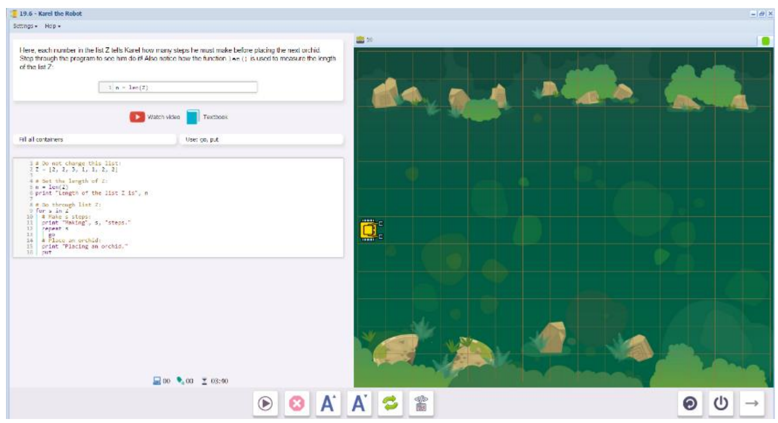
19.6 Step-through demonstration level. Learn how to use the length of a list.

We define a variable as the length of the list:

`n = len(Z)`

The length tells us how many items there are in the list.

Note that the variable `s` is specifically described as the number of steps before placing the next orchid.



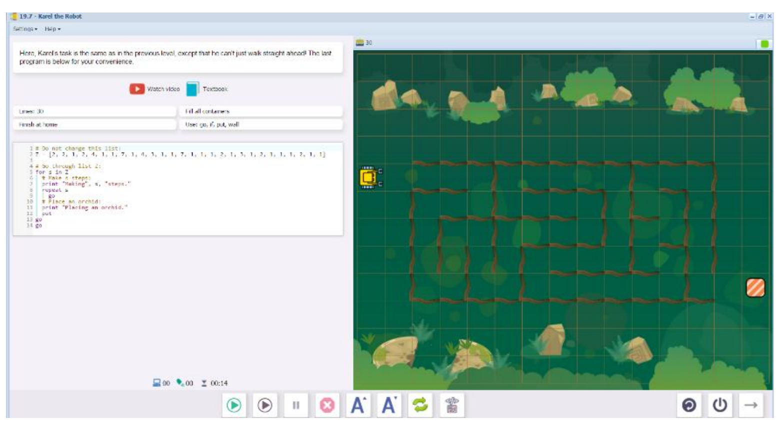
19.7 Karel uses a list to tell him how many steps to take before placing an orchid.

Lines: 30

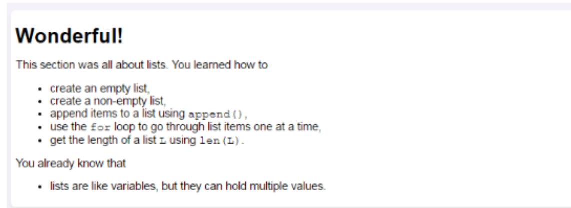
Use: `go, if, put, wall`

The program from 19.6 is already written. Students just need to direct Karel's actions, using the familiar wall testing algorithm, and making a couple of other small changes.

Note: the orchids print out a word.



Upon completion of 19.7, students will see this message, summarizing what the skills and concepts learned in Section 19. Section 20 is now unlocked.



Wonderful!

This section was all about lists. You learned how to

- create an empty list,
- create a non-empty list,
- append items to a list using `append()`,
- use the `for` loop to go through list items one at a time,
- get the length of a list `L` using `len(L)`.

You already know that

- lists are like variables, but they can hold multiple values.

Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

Show examples of the following: empty list, non-empty list (examples: Z=[]; Y=[1, 8, 3, 7, 7, 2, 5])

What does the length of a list tell us? (how many items are in the list)

How does a `for` loop work? (A `for` loop repeats a function for items in a list. In this Section, `for` loops are used to print a list of the items)

Think of a profession. How could a person in that profession use lists in a program? (one example could be a farmer using gps equipment to plant and monitor his field)

Asssessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using counting variables. A possible assessment is on the following page.

END OF SECTION 19: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game made up of a maze that contains several objects in different locations. (15 points)
- Generate a list by appending coordinates of objects and retrieving those objects. (10)
- Print out a list of these coordinates using a For loop (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 20: LEVELS 20.1 – 20.7

Objectives: Students learn how to remove and return the last item of a list using pop(), remove and return the first item of a list using pop(0), get the length of a list using len(), use the for loop to go through lists one item at a time, and merge lists. They know that list items can be numbers, Boolean variables, and even text strings. Lists can contain other lists, such as for example [gpsx, gpsy] pairs.

Vocabulary:

`pop`**:** removes an item from a list and assigns it to a variable. Either the last item or the first item is removed. For example

`la = L.pop()` removes the last item and assigns it to variable `la`

`fi = L.pop(0)` removes the first item and assigns it to variable `fi`

Empty List: A list that does not contain any items, shown by empty square brackets. For example: L = [ ]

Non-empty List: A list that contains items. For example: L = [1,6,8,3]

Append: Add items to a list. For example: L.append (x) , L.append ([gpsx, gpsy]). Notice that two or more items must be enclosed in one set of parentheses.

Parse: Examine the items in a list. The items can be printed out as a line-by-line log of the list, using a For loop.

For loop: A for loop is able to iterate (repeat a function) for items in a list. It is indented the same way as other loops. For example, a for loop can print out a log of these items:

```
for x in L
  print "current list item:", x
```

resulting in

```
Current list item = 1
Current list item = 3
Current list item = 4
Current list item = 6
Current list item = 7
Current list item = 8
```

Length of a list: `len`

Time required

Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills: Completion of Section 19

Background knowledge/Introductory Set/Purpose

In Section 19, we learned how to create and analyze lists. Now we will learn how to remove items from a list so that we can use them elsewhere. We've already talked about a couple of important applications for lists: inventory and assembly line work. Lists are also used in research, and even video games.

Inventory accounting uses a couple of methods for calculating the cost of taking items out for sale or use. One is FIFO, which stands for First In First Out. This method uses the cost of the oldest items first The other method is LIFO, or Last In First Out. LIFO uses the newest cost first, which makes sense if cost have gone up, you want to keep the value of your inventory low, and your expenses higher against your income. Lists can do the same functions: we can extract the first items off the list, or the last items of the list depending on our purpose.

In assembly line work, we want to build each units exactly the same as the original. A list can map out all the components of the original, then copy those components for each unit that is manufactured.

We also want to be able to combine information from various sources, or merge lists. Perhaps we want to take a population census. We collect information from each household to produce a list for one town. This list is combined with lists from other towns, cities and rural areas to create data for the whole county. Each county's lists is combined to create data for the state, and so forth.

Video games keep track of your progress in many ways: the types of items you collect, your gear (for example, armor and weapons), the levels and achievements, the success of yourself and your group. This data is stored and used just like the real life.

Purpose: In Section 20 (Levels 20.1-20.7), students learn how to extract items from a list to use separately, to build other lists, and merge to form larger lists.

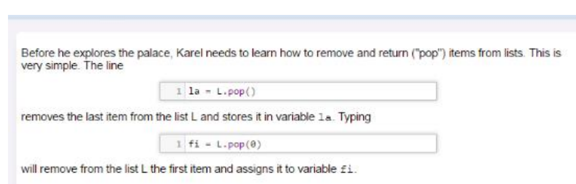
Direct Instruction and Modeling: The demonstration levels 20.1 (using L.pop() and L.pop(0)), and 20.6 (merging lists) can be viewed and discussed as a whole class. There are no videos on this level.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 20.1-20.7

20.1 20.1 begins with instructional screens showing how to use the pop function on lists. Items can be removed one at a time and assigned to a variable. We can either remove the last item on the list, or the first item. The examples on the screen do this:



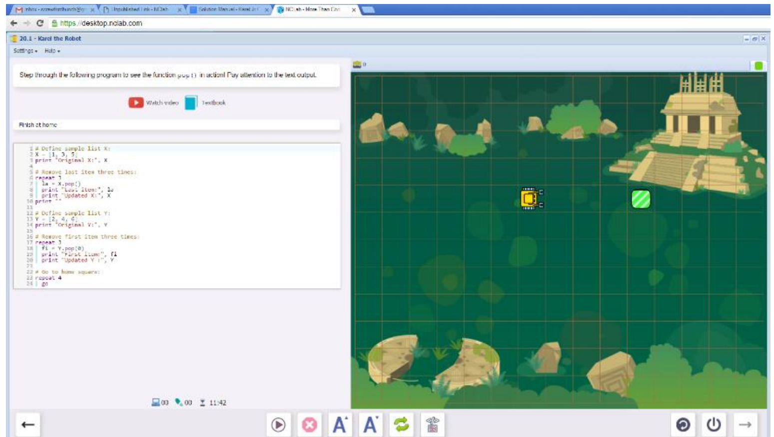
la = L.pop() removes the last item and assigns it to variable la

fi = L.pop(0)   removes the first item and assigns it to variable fi

Step-through demonstration: Two lists are created. The first one has items removed starting with the last item. The second list has items removed starting with the first item.

```
Original X: [1, 3, 5]
Last item: 5
Updated X: [1, 3]
Last item: 3
Updated X: [1]
Last item: 1
Updated X: []

Original Y: [2, 4, 6]
First item: 2
Updated Y : [4, 6]
First item: 4
Updated Y : [6]
First item: 6
Updated Y : []
```



20.2 (part 1 of 2) Karel collects all the masks in one room, saving their locations to a list.

Lines: 20

Collect all objects

Use: `append, get, go, False, home, if, left, not, right, True, wall, while`

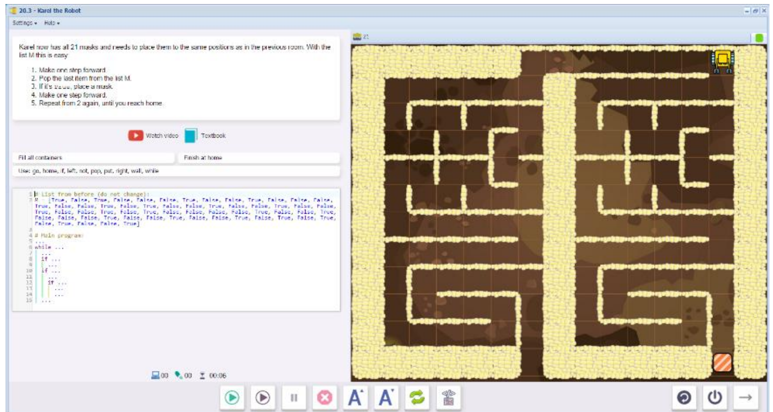
Students complete missing lines of code in the program. (Note: if they don't precede While not home with a go command to enter the room, the total count will be 61 instead of 60. The 60 count is significant because there is one list entry for every square unit of the 10x6 room.

Instead of using gpsx and gpsy coordinates, we use True (if a mask is present) and False (if a mask is not present).

20.3 (part 2 of 2) Karel places the masks in the second room in exactly the same locations.

Place all objects

Use: `go, home, if, left, not, pop, put, right, wall, while`

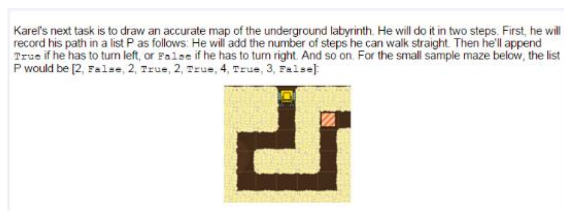


Students write a program to remove the masks from the list using the `pop` command, starting with the **last** mask because they are entering the room from the rear.

Step by step instructions are listed in the upper left screen.

Reminder: one equal sign assigns the value to the variable (la = M.pop()). Two equal signs mean equal in a relationship (if la == True).

20.4 Karel maps out an underground labyrinth by making a list of all the steps and turns. The number of steps (go) are recorded as numbers, left turns are recorded as True, and right turns are recorded as False.



Lines: 20

Use: `and, append, else, if, inc, left, not, right, wall, while`

Students complete the program. Step-by-step instructions are included in the upper left screen. A variable is needed to count the steps (the solution manual names the variable `counter`, but any name will do).

20.5 Karel uses the map of the underground labyrinth to place masks. He will place one mask for every step.

Lines: 20

Use: `go, left, len, pop, put, right, while`

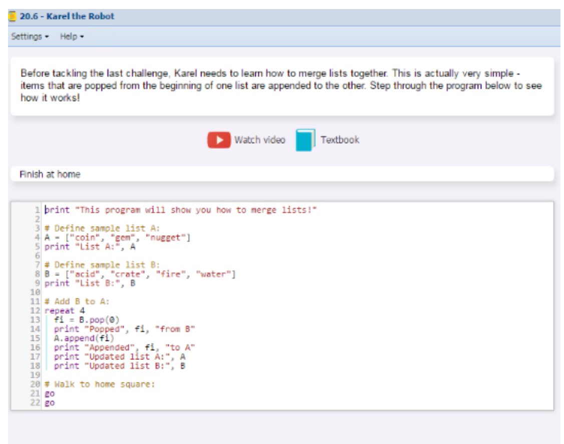
Students write a program, following the step-by-step instructions in the upper left screen.

For example, the program will continue until the list is empty, so that is written as a condition (`if P.len(0)`).

Notice that the `if P.pop(0)` statement checking for True does not use the word True. It is assumed.

20.6 Step-through demonstration level. Learn how to merge lists.

Merging is done by removing items from one list (`fi = A.pop(0)`) and adding them to the end of the other list (`B.append(fi)`).



Here is the print log of each step in this example:

```
This program will show you how to merge lists!
List A: ['coin', 'gem', 'nugget']
List B: ['acid', 'crate', 'fire', 'water']
Popped acid from B
Appended acid to A
Updated list A: ['coin', 'gem', 'nugget', 'acid']
Updated list B: ['crate', 'fire', 'water']
Popped crate from B
Appended crate to A
Updated list A: ['coin', 'gem', 'nugget', 'acid', 'crate']
Updated list B: ['fire', 'water']
Popped fire from B
Appended fire to A
Updated list A: ['coin', 'gem', 'nugget', 'acid', 'crate', 'fire']
Updated list B: ['water']
Popped water from B
Appended water to A
Updated list A: ['coin', 'gem', 'nugget', 'acid', 'crate', 'fire', 'water']
Updated list B: []
```

20.7 Karel collects all the masks in each room, recording their gpsx and gpsy coordinates in a list.

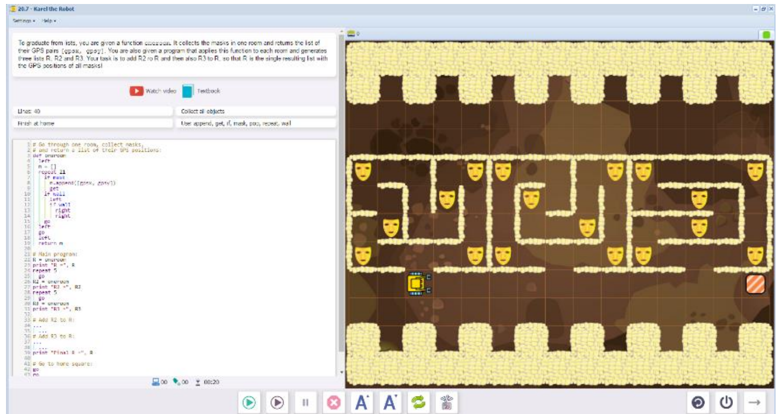
Lines: 40

Collect all objects

Use: `append, get, if, mask, pop, repeat, wall`

The program uses `oneroom` as a defined function. This function is applied to each of the three rooms.

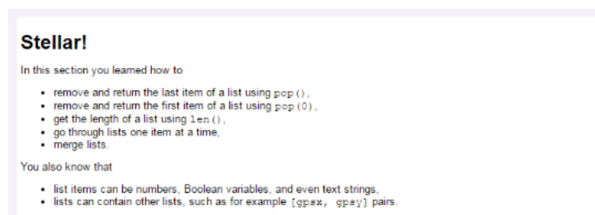
Most of the program is written.



Students practice appending the results of room 2 (R2) and room 3 (R3) to the list for room 1 (R1).

This could be done by popping the items out of a list into a variable, then appending that variable to the other list. A simpler way to merge the lists is to combined the two tasks into one line:
`R.append(R2.pop(0))`

Upon completion of 20.7, students will see this message, summarizing what the skills and concepts learned in Section 20. Karel 5 (Section 21) is now unlocked. Students also receive a Purple Belt of the Third Degree certificate.



Stellar!

In this section you learned how to

- remove and return the last item of a list using `pop()`,
- remove and return the first item of a list using `pop(0)`,
- get the length of a list using `len()`,
- go through lists one item at a time,
- merge lists.

You also know that

- list items can be numbers, Boolean variables, and even text strings,
- lists can contain other lists, such as for example `[gpsx, gpsy]` pairs.

Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

Review the commands for removing first and last items off a list.(Level 20.1)

Explain how to merge lists. (Level 20.6)

At the beginning of Section 20, we discussed how lists can be used for inventory, assembly line work, research and video games. Think of other applications you might have for lists.

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using lists. A possible assessment is on the following page.

END OF SECTION 20: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game made up of a maze. Copy the maze to a second game. (15 points)
- The first game will generate a list by appending coordinates of objects and retrieving those objects.
- The second game will use this list to distribute the objects in Karel's pocket. Note: it is easy to fill Karel's pocket with objects. Just click on the pocket icon in Designer Mode, then drag items to fill the pocket. You will be prompted for the quantity. The items can be changed or erased as needed.(10 points)
- The game will include at least one log of the items making up the list. (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

KAREL JR UNIT 5

Karel 5 Overview:

SECTION 21: Students learn how to use the function rand to create True or False with 50-50 probability. They use the function rand in conditions and while loops, and in in maze algorithms. They know that 50-50 probability means that the two events are equally probable, and that rand and rand yields 25-75 probability, which means that the former event is three times less probable than the latter.

SECTION 22: Students learn how to use recursion, which is a command or function that calls itself. They know that recursion is suitable for tasks that can easily be reduced in size, that the recursive call must be placed in a stopping condition, and that failure to use a stopping condition easily turns recursion into an infinite loop.

SECTION 23: Students review and practice previous sections: how to use stopping conditions in recursion, how to make the recursive call from inside a stopping condition, how to split complex tasks into simpler ones, how to use inequalities, how to get the length of a list, how to increase and decrease values, and how to pop items from lists.

SECTION 24: Students practice all their skills from previous sections in more complex tasks.

SECTION 25: More practice with complex tasks (optional)

SECTION 21: LEVELS 21.1 – 21.7

Objectives: Students learn how to use the function rand to create True or False with 50-50 probability. They use the function rand in conditions and while loops, and in in maze algorithms. They know that 50-50 probability means that the two events are equally probable, and that rand and rand yields 25-75 probability, which means that the former event is three times less probable than the latter.

Vocabulary:

`rand`**:** a function that creates True or False with 50-50 probability. Calling a rand function is like tossing a coin.

`rand` **and** `rand`**:** the combined function creates a 25/75 probability

Time required

Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills

Completion of Karel 4 (Section 20).

Background knowledge/Introductory Set/Purpose

In Karel 4, we learned about True/False operators and random number generators. We wrote functions to simulate the roll of a die. Here, we learn a new function `rand` that simulates a 50/50 coin toss, with an equal probability of returning True or False. We are giving the computer or robot the ability to guess, which is useful in situations which are not fixed and reliable.

Think of moving blindly around a space. You don't know the location of targets and obstacles. You still have the power to decide to go left or right by tossing a coin. Even if you are not successful on that particular coin toss, you can repeat the procedure as many times as you need to eventually find your target.

To visualize how this works, play the coin toss as a game on the floor, using floor tiles as a maze and a target object, or on a chessboard with a one chess piece moving and the other as a target.

You may have situations where you must make decisions fairly, without bias, not favoring one outcome or the other. Tossing a coin is often used for this purpose.

What if we have information that suggests one outcome is more likely to occur than the other? We can modify our coin toss to favor that outcome. `rand and rand` combines the results of two coin tosses, so that only one combination out of four will be true, and the other three will be false (**true and true**, true and false, false and true, false and false).

Purpose: Section 21 (Levels 21.1-21.7) introduces using probability to solve problems.

Direct Instruction and Modeling: There are no videos on this level. Any or all of the following levels can be viewed and discussed as a class, or reviewed afterwards:

> 21.1 demonstrates `rand` in three different settings: as a `repeat` loop, as an `if/else` conditional loop, and as a `while` loop.
>
> 21.3 demonstrates distributing based on `rand`.
>
> 21.4 explains how `rand and rand` can be used to generate a 25/75 probability.
>
> 21.5 and 21.6 demonstrate the problem with trying to locate an object in open space using a column type algorithm (21.5) and a follow-the-wall type algorithm (21.6). In both levels, students stop the program and click on the code template icon to re-run the scenario with `rand`.

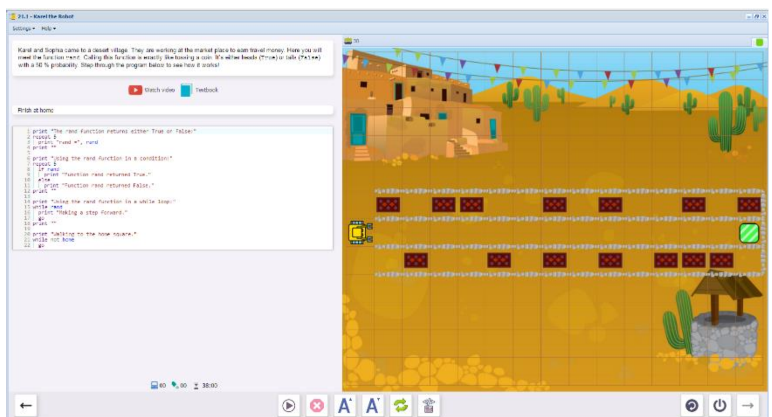
Individual/Group practice: The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 21.1-21.7

21.1 Step-through demonstration level. Learn how the `rand` function works.

Karel and Sophia come to a marketplace. They are looking for a chance to earn money.

This level demonstrates rand in a repeat loop, and if/else conditional loop, and a while conditional loop.



21.2 Karel walks to his home square. In each square, he randomly chooses whether or not to buy a rug.

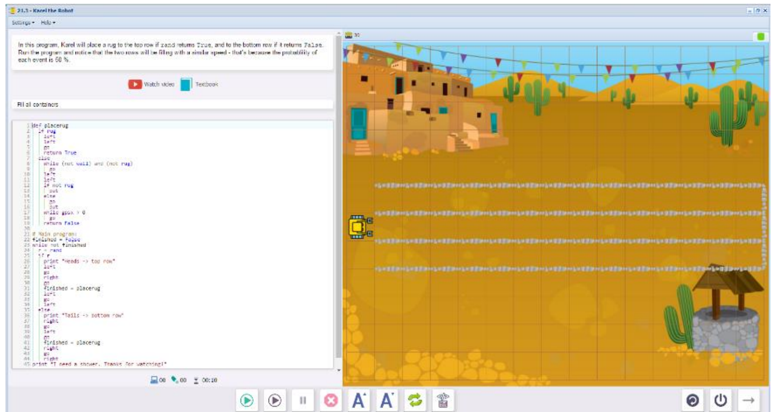
Lines: 4

Use: `13, get, go, if, rand, repeat`.

Students write a repeat loop that contains an `if rand` condition to help Karel decide whether or not to pick up a rug.

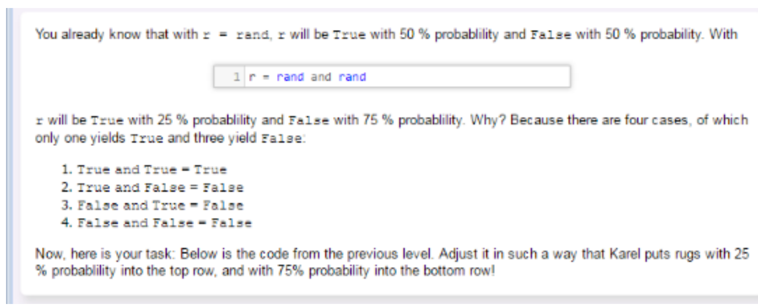
21.3 Demonstration Level, showing an example of 50/50 distribution based on `rand`. Karel places a rug on the top row if rand returns true, or on the bottom row if rand returns false. The rugs should be about evenly distributed.

This program takes a while to run. It will print out a log at the end, showing all the outcomes. Students could tally up all the heads and tails outcomes in their running of the program and compare results with each other (perhaps post notes on a common board).



21.4 Karel again places rugs, this time with a 25/75 distribution.

The upper left panel explains how rand and rand produces a 25//75 distribution, and showing an example.



You already know that with `r = rand`, `r` will be `True` with 50 % probablility and `False` with 50 % probability. With

```
1  r = rand and rand
```

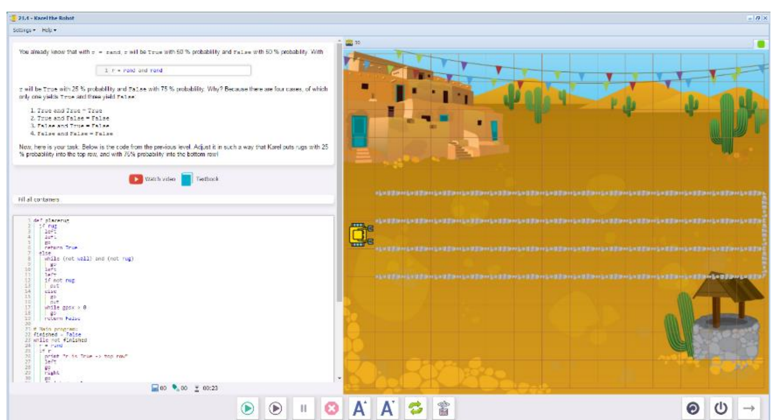
`r` will be `True` with 25 % probablility and `False` with 75 % probablility. Why? Because there are four cases, of which only one yields `True` and three yield `False`:

1. `True and True = True`
2. `True and False = False`
3. `False and True = False`
4. `False and False = False`

Now, here is your task: Below is the code from the previous level. Adjust it in such a way that Karel puts rugs with 25 % probablility into the top row, and with 75% probability into the bottom row!

Students modify the previous program to make it work as a 25/75 distribution.

They only need to modify one line in the main program, redefining the variable r as shown in the upper left panel.



As in 21.3, have students tally and compare their results.

21.5 Demonstration: Karel needs to collect a rug from an enclosure.

A program is written using a simple column style maze algorithm learned in previous sections. Because of the location of the rugs, Karel will never find them using that algorithm. In addition, this program produces an infinite loop.

Students are encouraged to run the program and see this for themselves.

However, if the program is modified to use the `rand` function, Karel will eventually find a rug by random choice.

By stopping the program, clicking on the code template on the bottom of the screen and resetting the program, students will be able to view the rand program instead.

Since this program uses a random function, run times will vary. Have students compare run times.

Can this level be solved using a systematic approach that does not depend on random choices?

(Answers vary)

21.6 Demonstration: Karel needs to collect a rug from an enclosure.

This time, the first program is written using a "follow the wall" algorithm. Again, it will fail to locate the rugs.

As in 21.5, students should first run this program, reset and click on the code template, then run the second program.

It can be amusing to watch Karel's random movements. He acts as though he has no idea what he is doing!

Students should make note of the program. 21.7 is similar. The program starts with `while not empty`. Once Karel finds a rug, the program will end. Notice that within the `if rand` condition, `if not wall` tests first; `else` is used to test the `if wall` condition.

21.7 Karel is once again looking for rugs in an enclosure. Use rand to complete the program.

Lines: 20

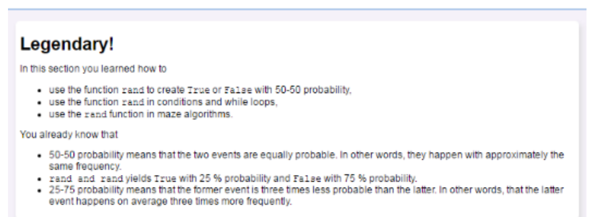
Use: `get, go, if, left, rand, right, rug`

See notes under 21.6 for how to write the loops.

Again, students could write down their run times and compare results.

Is there a way to search this maze systematically rather than relying on a random function?

Upon completion of 21.7, students will see this message, summarizing what the skills and concepts learned in Section 21. Section 22 is now unlocked.



Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

Explain how `rand and rand` works. (see explanation in the background section).

When would you use `rand` in a program? (examples: making fair choices, distributing fairly, finding objects in uncertain locations)

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using counting variables. A possible assessment is on the following page.

END OF SECTION 21: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game made up of a complex maze (15 points)
- Use `rand` to guide Karel's choices. This could govern where to place objects, or where to find them. (10 points)
- The game will include at least one feature from previous levels, such as repeat loops, conditional loops, defined commands, variables or functions. (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on _____________________________________(5 points)

SECTION 22: LEVELS 22.1 – 22.7

Objectives: Students learn how to use recursion, which is a command or function that calls itself. They know that recursion is suitable for tasks that can easily be reduced in size, that the recursive call must be placed in a stopping condition, and that failure to use a stopping condition easily turns recursion into an infinite loop.

Vocabulary:

> **Recursion:** a command or function that calls itself.
>
> > The recursion occurs within the body of the loop.
> >
> > It must have a stopping condition. If not, it can turn into an infinite loop.
>
> **Stopping condition:** a condition that ends a loop.
>
> **Infinite loop:** a loop that theoretically could continue operating infinitely. Most programs have a timer that would eventually time out the loop.

Time required

Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills

Completion of Section 21

Background knowledge/Introductory Set/Purpose

We have already learned how to build loops and know which ones to use depending on what we need to do.

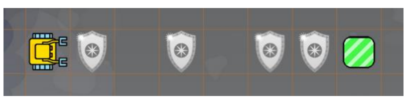
> A repeat loop is used when we know exactly how many times we must repeat a command.
>
> A conditional while loop is used when we can inquire about a condition, act on it, and stop when the condition ends.
>
> A for loop is used when we are working from a list. When we have used all the input data from the list, the loop ends.

Recursion is another kind of loop. The word “recursion” comes from a Latin word that means to run back. In a sense, that is exactly what the program is doing: running back and repeating the command until it is no longer needed. Recursive loops are memory intensive and not suitable for large repetitions. A stopping condition has to be written into the loop, or it becomes infinite. In other words, it will keep calling itself forever. However, most programs will time out or report a stack overflow and stop the loop. In Karel, the recursion is written as a defined function. Once the recursion has finished, `return` is written outside the body of the loop to return to the main program.

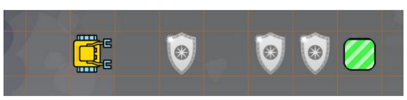
Purpose: Section 22 (Levels 22.1-22.7) Students will learn how to write recursive loops with correct syntax and stopping conditions.

Instruction and Modeling: Section 22 begins with a video on Recursion. The demonstration levels 22.1 shows an example of a recursion: Karel repeats a set of commands (moving forward and picking up shields) until he reaches the home square (the stopping condition). The video and step-through demonstrations can be watched and discussed as a class. One of the introductory screens in 22.1 explains recursion as follows:

“Recursion is an advanced programming technique. It can be used to solve problems which, by doing just a few operations, can be reduced to the same problem which is just smaller in size. Like this one, where Karel needs to walk to his home square and collect all shields:



After making one step and collecting one shield, Karel still needs to *walk to his home square and collect all shields!*


“

22.3 steps through the same recursion, with the stopping condition missing. Students must repair the program. 22.4 steps through the same recursion with the recursion outside of the stopping condition loop. Again, after watching the demonstration, students repair the program.   These levels may be reviewed and discussed at the end of the Section.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 22.1-22.7

22.1 22.1 begins with a video on recursion. Students can press play or follow the link on the screen. Here is the link:

https://www.youtube.com/watch?v=zPkig_0dpNM&feature=youtu.be

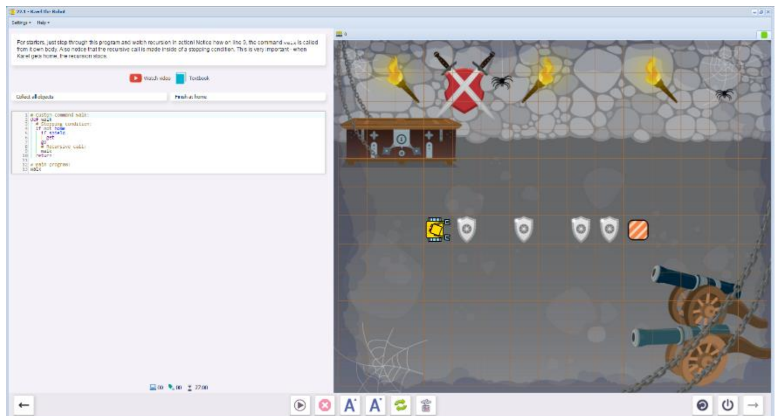
The next screen explains recursion.

After Karel moves forward and picks up a shield, he still needs to move forward and pick up shields.

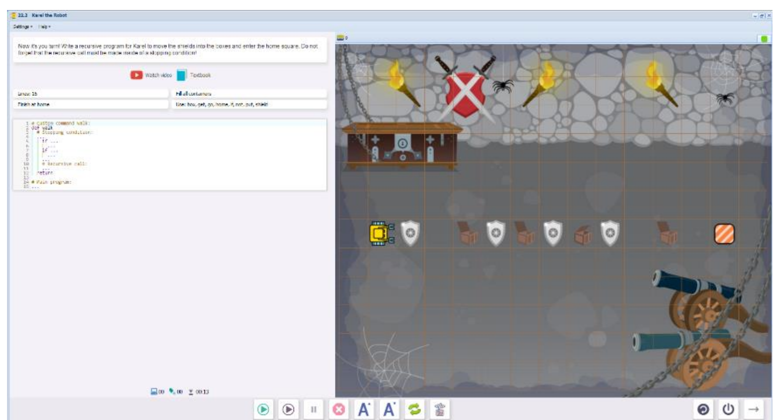
He continues to move forward and pick up shields until he has collected all the shields and reached the home square.

The command set "move forward and pick up a shield" is called over and over again. This is a recursion.

The next screen is a step-through demonstration, so students can see how the recursion works.



22.2 Karel picks all the shields in his path and puts them in boxes on the way home.



Lines: 15

Use: `box, get, go, home, if, not, put, shield`

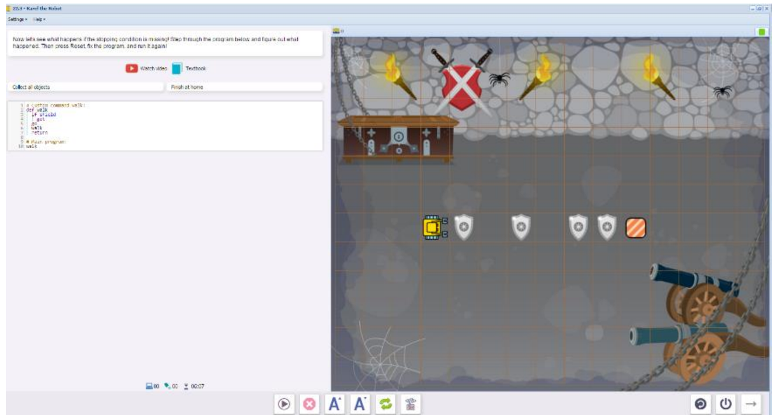
The program is partially written. Students complete the code.

Notice that `if not home` is used instead of `while not home`. That is because each square must be tested independently before calling the recursion. `If not home` is the stopping condition.

(Watch for indent errors: the recursive `walk` must be in the body of the `if not home` condition. The recursive `walk` is followed by a `return` line, which is <u>not</u> part of the `if not home` condition.)

22.3 Begin with a step-through demonstration of the same task: what happens if the stopping condition is missing? (Karel keeps going past home and crashes into the wall)

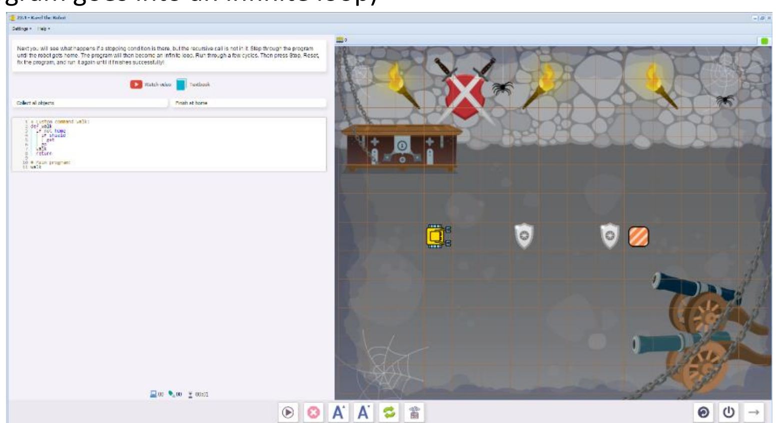
Step through the program and observe the error. Then reset, repair, and rerun the program.



22.4 Step-through the demonstration again. This time, the recursion is not inside the stopping condition loop. What happens? (the program goes into an infinite loop)

Reset, repair, and rerun the program.

.



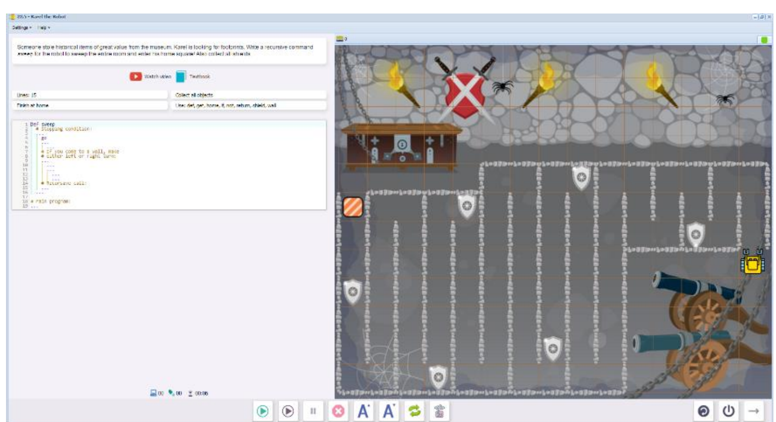
22.5 Karel must sweep the room for footprints, and pick up all the shields.

Lines: 155

Collect all objects

Use: `def, get, home, if, not, return, shield, wall`

Students write a recursive function named `sweep`. The program is partially written.



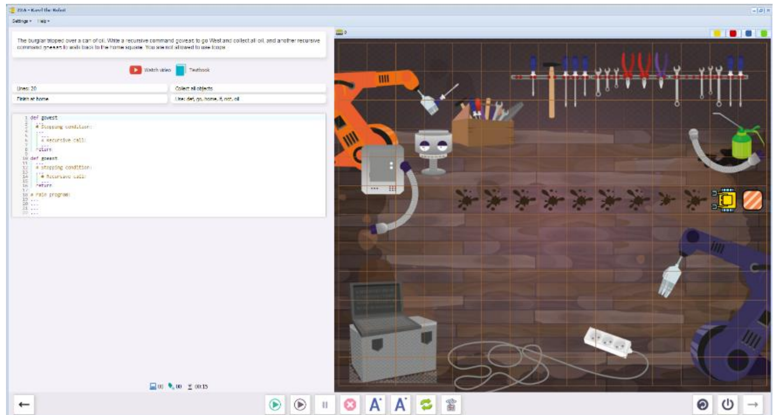
22.6 Karel cleans up the oil trail left by the burglar, then goes home.

Lines: 20

Collect all objects

Use: `def, go, home, if, not, oil`

Students write two functions: one called `gowest` to travel west and pick up the oil, and the other called `goeast` to return back to the home square.



22.7 Karel needs to find his hat and pipe on the way to the exit.

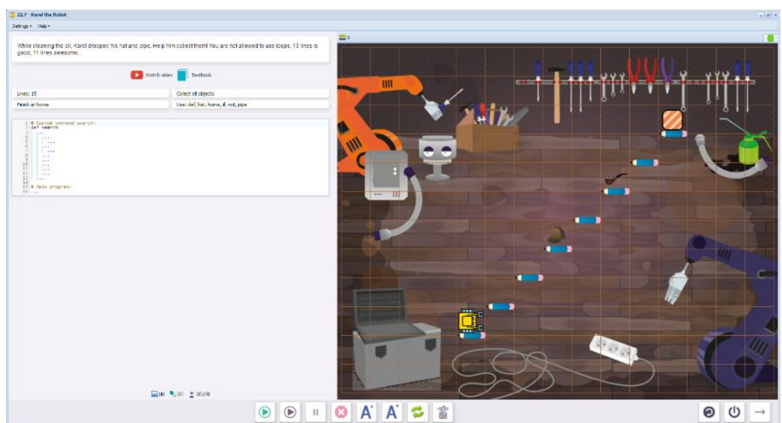
Lines: 15

Collect all objects

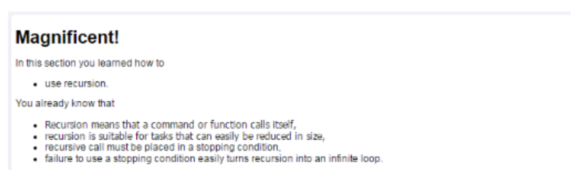
Use: `def, hat, home, if, not, pipe`

Students write the recursive function `search` without using loops.

Challenge: a 13 line program is good, an 11 line program is awesome.



Upon completion of 22.7, students will see this message, summarizing what the skills and concepts learned in Section 22. Section 23 is now unlocked.



Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

What are the key elements of recursion? (The command calls itself. The recursion must be embedded in a loop that includes a stopping condition.)

Try rewriting a while loop as a recursive loop, or vice versa. How do the programs compare? (Number of lines, number of operations, run time)

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using counting variables. A possible assessment is on the following page.

END OF SECTION 22: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game made up of a maze that requires a simple, repetitive task. (15 points)
- The game will include at least one defined, recursive function to perform this task. (20 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 23: LEVELS 23.1 – 23.7

Objectives: Students review and practice previous sections: how to use stopping conditions in recursion, how to make the recursive call from inside a stopping condition, how to split complex tasks into simpler ones, how to use inequalities, how to get the length of a list, how to increase and decrease values, and how to pop items from lists.

Vocabulary: No new vocabulary

Time required

Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills

Completion of Section 22

Background knowledge/Introductory Set/Purpose

Writing recursive loops is a great way to learn more flexible programming. In Section 23, we continue to explore recursion in more complex situations. What is the stopping condition? Can we write a program that ends at home without using the keyword `home`? How can we clear an array in a way that is not row by row (or column by column)? How do we use recursion with lists and inequalities?

Purpose: Section 23 (Levels 23.1-23.7) Students practice and refine their understanding of recursion.

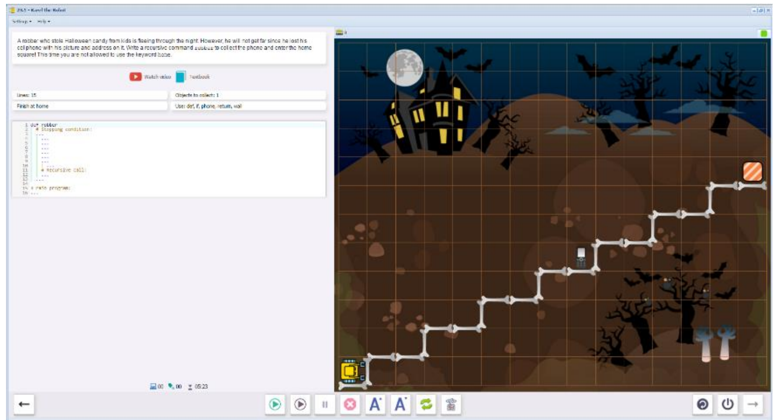
Direct Instruction and Modeling: There are no demonstrations or videos on this level. There are paired levels: 23.2 is used to solve 23.3; 23.6 is used to solve 23.7. Instruction is embedded in each level.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 23.1-23.7

23.1 Karel locates the robber's phone and ends at home.



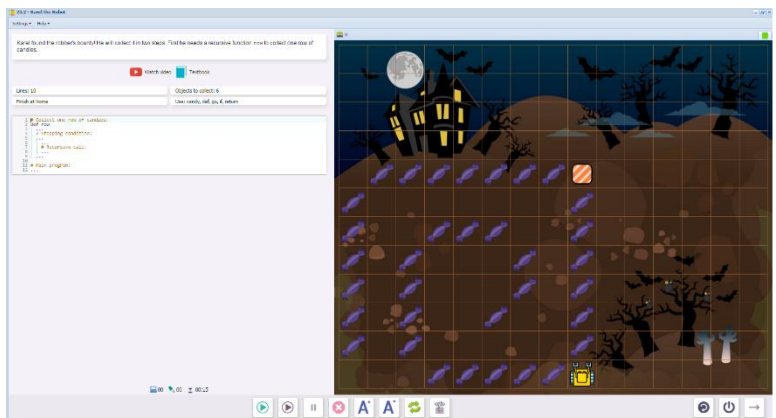
Lines: 15

Use: `def, if, phone, return, wall, randint`

Students write a program to perform the tasks using a recursive function. They cannot use the word `home`.

Think of a condition that is present during the procedure, but goes away at the end.

23.2 (Part 1 of 2) Karel finds the robber's candy. He collects candy from one row.

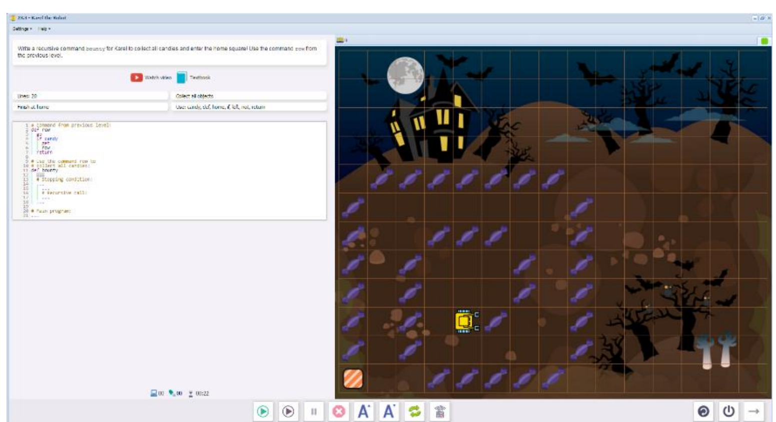


Collect 6 objects.

Use: `candy, def, go, if, return`

Students define a recursive function `row` and use it to collect one row of candy. Students should start thinking of stopping conditions that are based on the situation. This stretches their ability beyond fixed repeats and while not home conditions. Have them think about the task, and what might end it.

23.3 (Part 2 of 2) Karel collects all the candy in the robber's bounty.



Lines: 20

Collect all objects

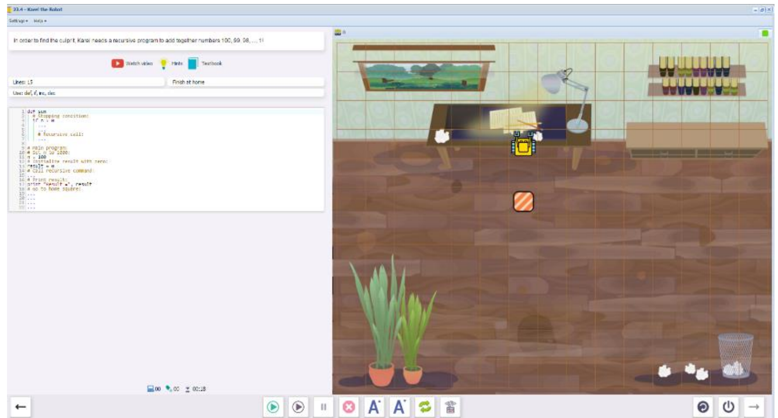
Use: `candy, def, home, if, left, not, return`

Students use the recursive function `row` from 23.2 within the recursive function `bounty` to collect all the candy. This time, they can use home as a stopping condition.

23.4 As part of the investigation, Karel adds up a series of numbers.

Lines: 15

Use: `def, if, inc, dec`



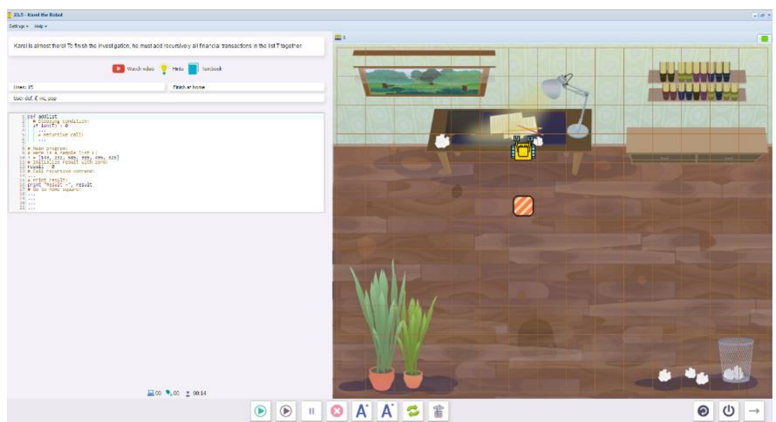
Students complete the program by filling in missing code. This recursive function decreases n and increases result by n (see Section 15 regarding counting variables).

The return line is not needed.

23.5 Now Karel must add up transactions in a list.

Lines: 15

Use: `def, if, inc, pop`

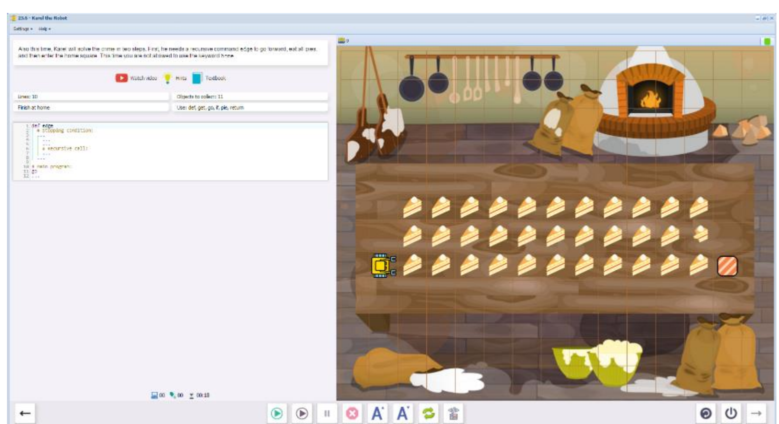


Students complete the program by filling in missing code. Use `pop` to add items to `result` (see Section 20 for an explanation on how to pop items from a list).

23.6 (Part 1 of 2) Karel eats all the pies in one row and goes home.

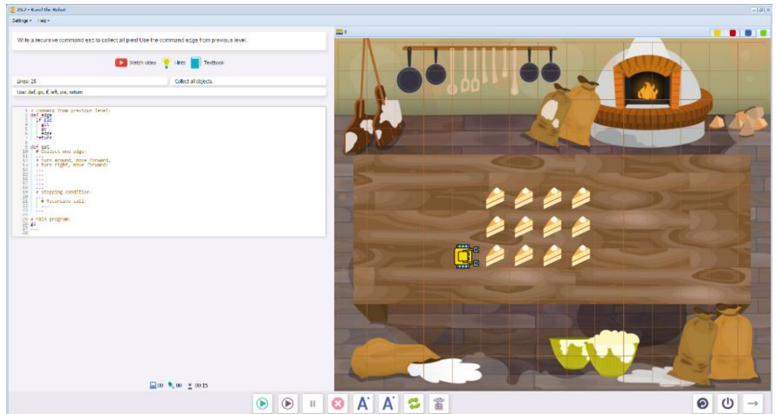
Lines: 10

Use: `def, get, go, if, pie, return`



Students write the program, including the recursive function `edge`. This function is written the same way as 23.2. `home` is not allowed.

23.7 (Part 2 of 2) Karel collects all the lightbulbs and calculates the minimum height of each column.



Lines: 25

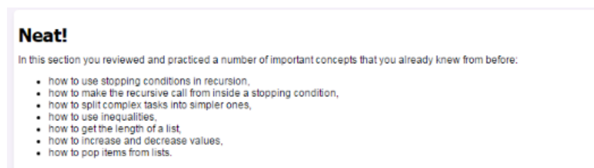
Collect all objects

Use: `def, go, if, left, pie, return`

Students complete the program by filling in missing code. This is another recursion within a recursion. Think of where Karel needs to be to start eating the next edge. The program spirals inward until all the pies are eaten.

How does this compare to the defined commands used in 11.7 (onerow, wturn, eturn)?

Upon completion of 23.7, students will see this message, summarizing what the skills and concepts learned in Section 23. Students also earn a Black Belt in the First Degree certificate. Section 24 is now unlocked.



Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

What were the stopping conditions in each level?

Compare programs for spiral pathways to programs for row-by-row pathways in arrays. Do you see any advantages to using one over the other?

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a game using recursions with one of the features learned in Section 23, such as recursions within recursions (splitting complex tasks into simpler ones), or recursions based on lists, or recursions based on unusual stopping conditions (such as inequalities). A possible assessment is on the following page.

END OF SECTION 23: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game made up of a complex maze (15 points)
- The game will use recursions. (10 points)
- The game will include at least one skill learned in Section 23, such as breaking complex tasks into simpler ones (recursions within recursions), or recursions based on lists, or unusual stopping conditions (for example, inequalities). (10 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

SECTION 24: LEVELS 24.1 – 24.7

Objectives: Students practice all their skills from previous sections in more complex tasks.

Vocabulary: no new terms.

Time required

Time required will vary based on student ability and experience. Most students will complete this section in about 2 hours.

Prerequisite skills

Completion of Section 23

Background knowledge/Introductory Set/Purpose

Section 24 contains puzzles that require different skills to solve, including:

Nested repeat loops, if/else conditions, and while conditional loops

Defined functions with counting variables

Logical operations and, or

Complex patterns reduced to simpler tasks or patterns

Lists and gpsx, gpsy coordinates

Information from one part of a puzzle used to solve another part.

Append and pop on lists

Purpose: Section 24 (Levels 24.1-24.7) Students practice previously learned skills in more complex settings.

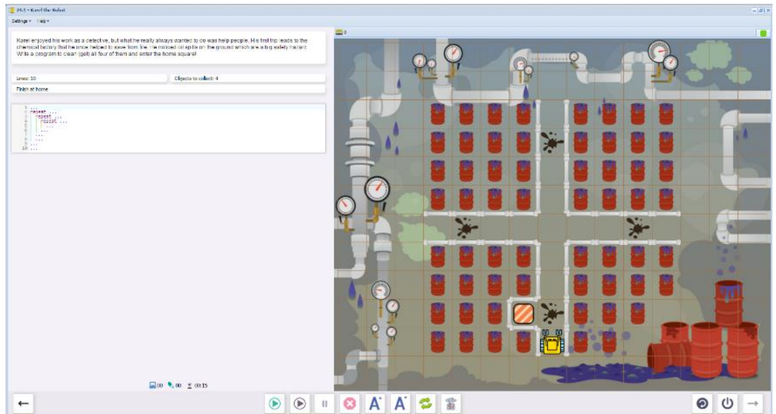
Direct Instruction and Modeling: There are no instructional step-through demonstrations and videos. However, the programs are partially written and include comment lines as guides.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 24.1-24.7

24.1 Karel cleans up oil spills on the floor of the factory.

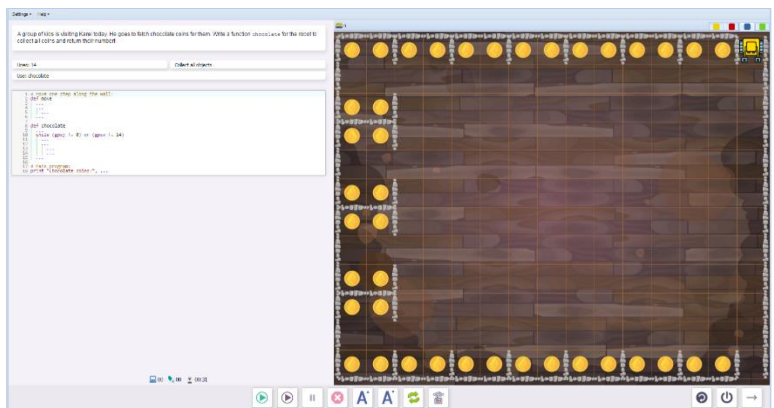


Lines: 10

Students write a program to collect the four oil spills. The challenge is to solve the level in 10 lines.

This level practices nested repeat loops. Remind students to look for patterns that can be broken down into simpler patterns.

24.2 Karel collects all the chocolate coins, and counts the total.



Lines: 14

Collect all objects

Use: `chocolate`

Students write a program to collect all the chocolate coins. The challenge is to solve the level in 14 lines.

This level practices the wall-following defined command learned in Section 12, and functions to increment variables learned in Section 14.

Gps coordinates are used as a stopping condition: these are already written into the program. Students should make a note of this for their own programs.

24.3 Karel follows the marks home, placing a chip on each mark as he goes.

Lines: 40

Fill all containers

Students write a program, following the prompts on the comment lines.

`While wall` keeps Karel on the path when it hugs the wall. Students should think about what Karel should do when he goes off the path without using a wall as a reference. Look for a consistent procedure Karel can follow to get back on the path.

24.4 Karel uses a list of gpsx locations to place the orchids in his pocket.

Lines: 28

Collect all objects

This level again makes use of the wall-following defined command `move`, this time following the wall to the right. This is an example of a simple program that can be used in a complex, arbitrary maze.

The program takes a while to run.

24.5 Karel checks the wall perimeter for escape routes and logs their locations. He collects water bottles along the way.

Lines: 50

Collect all objects.

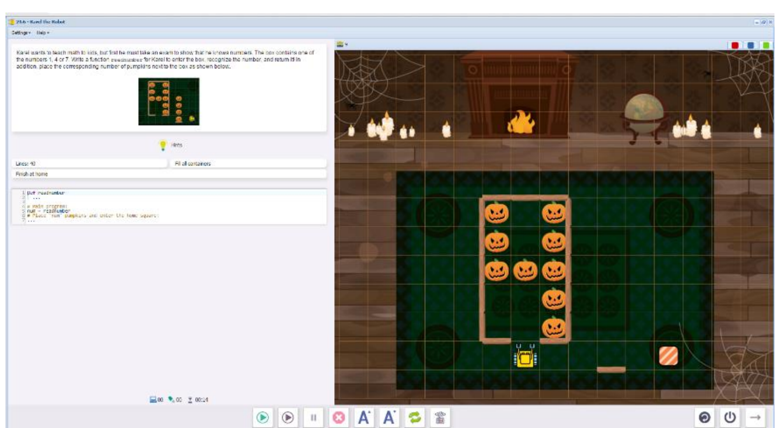
It is fairly simple to check whether not wall is a hole in the wall or a corner. Turning right in the space will either show a wall or open space.

If there is a wall, the gps coordinates can be written to the list; otherwise, nothing is written to the list. The solution manual explains how to use the lists.

24.6 Karel uses a list to tell him how many steps to take before placing an orchid.

Lines: 40

The key to this program is finding what condition is unique to each number. What is unique about a 1, a 4, and a 7? The program should check for those conditions. See the solution manuals for details.

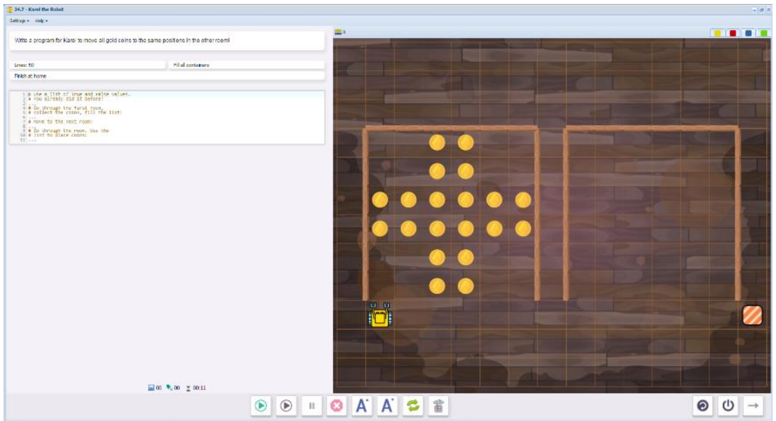


24.7 Karel locates and collects all the coins and moves them to exactly the same positions in the second room.

Lines: 50

Students write a program using True/False criteria to map out the rooms, similar to Section 17

The comments contain hints.



Upon completion of 24.7, students will receive a certificate for a Black Belt of 2nd degree and congratulations. Section 25 is now unlocked and contains advanced puzzles to test and review coding skills.

Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

This is an opportunity to review the course as a whole. Review notebooks, or use a notetaker to include concepts, examples, questions, and summaries. What will students do with this knowledge?

Assessment:

Assessment is built into the program. Students must complete a level successfully in order to unlock the next level. See Assessment section for journal and project ideas.

Suggested Game Assessment: Students create a complex game of their own design.

END OF SECTION 24: CREATE A GAME FOR KAREL (50 POINTS)

Create and publish a game for Karel in programming mode.

- Create a game made up of a maze. Think of what skills you would like to test and design the maze accordingly. (15 points)
- Test at least two advanced skills that you have learned in the course (Defined functions and variables, lists, Boolean functions, random functions, etc.) (20 points)
- When editing the game, write the objectives of the game under the **Summary** tab. (5 points)
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ____________________________________(5 points)

SECTION 25: LEVELS 25.1 – 25.7

Objectives: Students practice skills that they have learned in Karel Jr on complex tasks.

Vocabulary: no new vocabulary

Time required

Time required will vary based on student ability and experience. Some of the puzzles are simple to solve, while others require lengthy solutions.

Prerequisite skills

Completion of Section 24

Background knowledge/Introductory Set/Purpose

This is an enrichment level for students who like challenges. The level of difficulty is several steps above the other levels. Many of the problems are classic logic challenges.

Individual/Group practice:

The program is designed to be used individually by students. Encourage peer support, sharing and discussion.

Self-paced Instruction: Levels 25.1-25.7

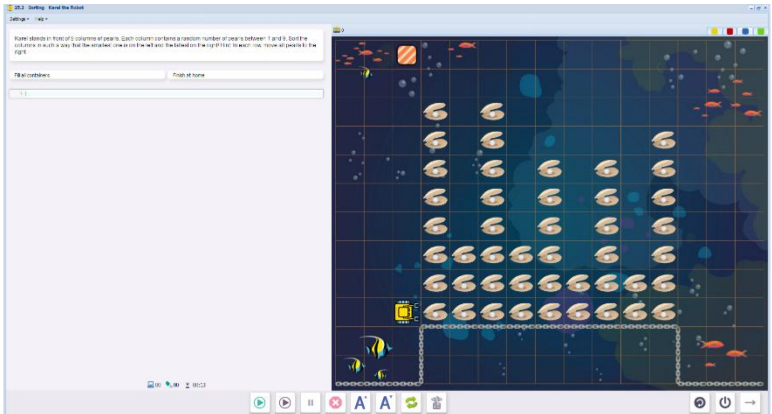
25.1 Karel measures the length of a fence, and collects all the corn along the way. He reports the total length of the fence.

Lines: 12

Collect all objects

Students write a program that increments a variable while moving along the fence.

25.2 Karel orders a set of columns of random heights, arranging them from shortest on the left, to tallest on the right.



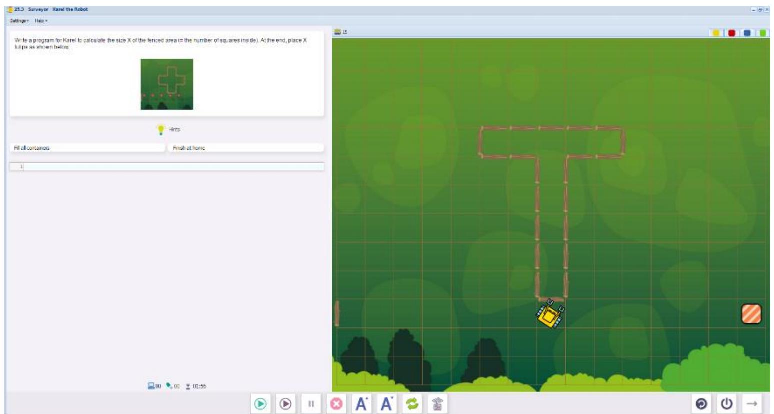
Students write a program to move the pearls to create the ordered columns. The key is working on each row of pearls, rather than each column.

25.3 Karel determines the area of the enclosure, then plants a row of tulips, one tulip for every square unit.

This program uses gps coordinates to compare locations of the fence and calculate the area.

This level is similar to 24.6, in that one task is used to compute a value used in a second task.

How the program works:

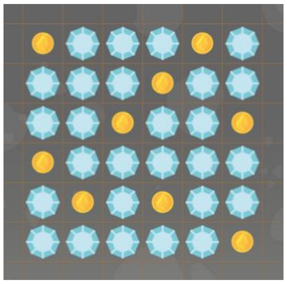


Every time the robot faces the wall to the south, it increases the square count by its gpsy position minus 1. It is actually counting that entire column, which would be too many squares. The count is lessened by the number of times the robot faces the wall to the north, which subtracts out the count by its gpsy position from the total.

25.4 25.4 opens with an explanation of the Cardin Grille.

“Cardan Grille, invented around 1550, belongs to the oldest encryption methods. The grille is a square piece of paper or leather that is subdivided into smaller squares. Some of them are cut out. When the grille is placed on a square table of letters, it reveals some of them. That's the beginning of the secret message. When rotated by 90 degrees, the grille reveals the second part of the message, and so on. The 6x6 sample grille shown below can encode a message of 36 letters. For longer messages, a larger grille such as 8x8 or 10x10 can be used.”

The second screen continues: "It is easy to make a mistake while creating a Cardan Grille. In that case, more than one hole uncovers the same position in the table of letters during the rotation.

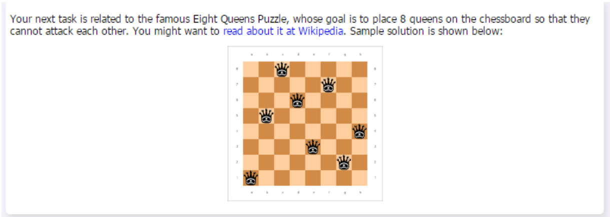


In the grille shown here, holes are represent by coins. The grille is invalid since after two rotations, one of the corner holes ends up taking the position of the other corner hole.

This grille has one more flaw besides the corners. Can you find it?"

Students write a function that tests the Cardan grille for faults and prints the result. Faults occur when the grill is rotated and coins occupy the same coordinates.

25.5 Eight Queens. This puzzle is based on a chessboard pattern: eight queens are placed on the chessboard in positions where they cannot attack each other. Reminder: queens can move horizontally, vertically and diagonally for any number of open squares.



Students write a program to check the pattern for flaws. Several mazes are available for testing.

This requires a lengthy analysis of each queen's position: can she attack another queen in one of 8 directions?

With a human's birds-eye view, it is easy to see whether or not a queen can attack another queen. However, the computer does not have this advantage. It must analyze the problem, square by square.

A number of defined commands can be created and called to test each queen.

25.6 This level is similar to 24.5, in which Karel looks for holes in the wall and collects water bottles along the way. This time the wall is irregular in shape.

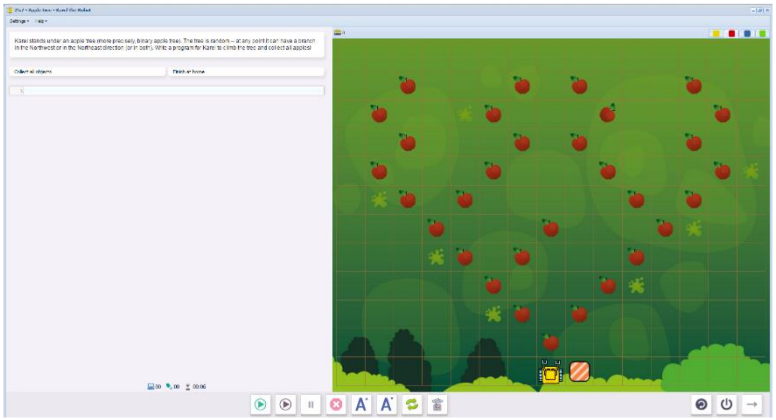
Students write a function desert to count all the holes and report the total. The instructions do not ask for gps coordinates this time.

The upper left panel gives a hint about checking for adjacent walls.

25.7 Karel collects apples off a random binary apple tree. At any point, it can branch in the northwest or northeast direction.

Note: this level cannot be cleared row by row. Beware of the leafy wall squares!

Follow each branch instead, testing for branches by the presence of apples.



Questions for post-session discussion (students can use their journals to write down their ideas and responses) (10-20 minutes):

Undoubtedly, students who attempt this level will have their own questions and ideas. Have them post these on a common chart to generate discussions.

Suggested Game Challenge: Students could research a mathematical puzzle and reproduce it using Karel.

SECTION 25 CHALLENGE

Create and publish a game for Karel based on a classic math, logic or game challenge. Investigate puzzles in books or on line. Can you make a game in Karel to reproduce one of these puzzles?

- As before, create a game made up of a maze of your choosing. Create different versions if applicable.
- Design a solution
- When editing the game, write the objectives of the game under the **Summary** tab.
- Set the goals under the **Goals** tab. (5 points).
- Test the game and edit as needed.
- Copy the program into a document to make an answer key. Save this document to the NCLab folder or a folder specified by your teacher.
- Publish the game to your folder. Inform someone else about the game by providing the link on ______________________________________(5 points)

NOTES