

Unit 3 - Interactive Animations and Games

In the Interactive Games and Animations unit, students build on their coding experience as they create programmatic images, animations, interactive art, and games. Starting off with simple, primitive shapes and building up to more sophisticated sprite-based games, students become familiar with the programming concepts and the design process computer scientists use daily. They then learn how these simpler constructs can be combined to create more complex programs. In the final project, students develop a personalized, interactive program. Along the way, they practice design, testing, and iteration, as they come to see that failure and debugging are an expected and valuable part of the programming process.

Chapter 1: Images and Animations

Big Questions

- What is a computer program?
- What are the core features of most programming languages?
- How does programming enable creativity and individual expression?
- What practices and strategies will help me as I write programs?

Week 1

Lesson 1: Programming for Entertainment

Unplugged

The class is asked to consider the "problems" of boredom and self expression, and to reflect on how they approach those problems in their own lives. From there, they will explore how Computer Science in general, and programming specifically, plays a role in either a specific form of entertainment or as a vehicle for self expression.

Lesson 2: Plotting Shapes

Unplugged

This lesson explores the challenges of communicating how to draw with shapes and use a tool that introduces how this problem is approached in Game Lab. The class uses a Game Lab tool to interactively place shapes on Game Lab's 400 by 400 grid. Partners then take turns instructing each other how to draw a hidden image using this tool, accounting for many of the challenges of programming in Game Lab.

Lesson 3: Drawing in Game Lab

Game Lab

The class is introduced to Game Lab, the programming environment for this unit, and begins to use it to position shapes on the screen. The lesson covers the basics of sequencing and debugging, as well as a few simple commands. At the end of the lesson, the class creates an online version of the image they designed in the previous lesson.

Lesson 4: Shapes and Randomization

Game Lab

This lesson extends the drawing skills to include width and height and introduces the concept of random number generation. The class learns to draw with versions of `ellipse()` and `rect()` that include width and height parameters and to use the `background()` block to fill the screen with color. At the end of the progression the class is introduced to the `randomNumber()` block and uses the new blocks to draw a randomized rainbow snake.

Week 2

Lesson 5: Variables

Game Lab

This lesson introduces variables as a way to label a number in a program or save a randomly generated value. The class begins the lesson with a very basic description of the purpose of a variable and practices using the new blocks. Afterwards, the class uses variables to save a random number, allowing the programs to use the same random number multiple times.

Lesson 6: Sprites

Game Lab

In order to create more interesting and detailed images, the class is introduced to the sprite object. Every sprite can be assigned an image to show, and sprites also keep track of multiple values about themselves, which will prove useful down the road when making animations. At the end of the lesson, everyone creates a scene using sprites.

Lesson 7: The Draw Loop

Game Lab

This lesson introduces the draw loop, one of the core programming paradigms in Game Lab. The class combines the draw loop with random numbers to manipulate some simple animations with dots and then with sprites. Afterwards, everyone uses what they learned to update the sprite scene from the previous lesson.

Lesson 8: Counter Pattern Unplugged

Unplugged

This unplugged lesson explores the underlying behavior of variables. Using notecards and string to simulate variables within a program, the class implements a few short programs. Once comfortable with this syntax, the class uses the same process with sprite properties, tracking a sprite's progress across the screen.

Week 3

Lesson 9: Sprite Movement

Game Lab

By combining the Draw Loop and the Counter Pattern, the class writes programs that move sprites across the screen, as well as animate other sprite properties.

Lesson 10: Booleans Unplugged

Unplugged

This lesson introduces boolean values and logic, as well as conditional statements. The class starts by playing a simple game of Stand Up, Sit Down in which the boolean (true/false) statements describe personal properties (hair or eye color, clothing type, age, etc). The class then groups objects based on increasingly complex boolean statements, then looks at how conditionals can impact the flow of a program.

Lesson 11: Booleans and Conditionals

Game Lab

The class starts by using booleans to compare the current value of a sprite property with a target value, using that comparison to determine when a sprite has reached a point on the screen, grown to a given size, or otherwise reached a value using the counter pattern. After using booleans directly to investigate the values or sprite properties, the class adds conditional if statements to write code that responds to those boolean comparisons.

Lesson 12: Conditionals and User Input

Game Lab

Following the introduction to booleans and if statements in the previous lesson, students are introduced to a new block called `keyDown()` which returns a boolean and can be used in conditionals statements to move sprites around the screen. By the end of this lesson students will have written programs that take keyboard input from the user to control sprites on the screen.

Week 4

Lesson 13: Other Forms of Input

Game Lab

The class continues to explore ways to use conditional statements to take user input. In addition to the simple `keyDown()` command learned in the previous lesson, the class learns about several other keyboard input commands as well as ways to take mouse input.

Lesson 14: Project - Interactive Card

Game Lab | Project

In this cumulative project for Chapter 1, the class plans for and develops an interactive greeting card using all of the programming techniques they've learned to this point.

Chapter Commentary

Students build up toward programming interactive animations in the Game Lab environment. They begin with simple shapes and sprite objects, then use loops to create flipbook style animations. Next, they learn to use booleans and conditionals to respond to user input. At the end of the chapter, students design and create an interactive animation that they can share with the world.

Chapter 2: Building Games

Big Questions

- How do software developers manage complexity and scale?
- How can programs be organized so that common problems only need to be solved once?
- How can I build on previous solutions to create even more complex behavior?

Week 5

Lesson 15: Velocity

Game Lab

After a brief review of how the counter pattern is used to move sprites, the class is introduced to the properties that set velocity and rotation speed directly. As they use these new properties in different ways, they build up the skills they need to create a basic side scroller game.

Lesson 16: Collision Detection

Game Lab

The class learns about collision detection on the computer. Pairs explore how a computer could use sprite location and size properties and math to detect whether two sprites are touching. The class then uses the `isTouching()` block to create different effects when sprites collide, including playing sounds. Last, they use their new skills to improve the sidescroller game that they started in the last lesson.

Lesson 17: Complex Sprite Movement

Game Lab

The class learns to combine the velocity properties of sprites with the counter pattern to create more complex sprite movement, such as simulating gravity, making a sprite jump, and allowing a sprite to float left or right. In the final levels the class combine these movements to animate and control a single sprite and build a simple game in which a character flies around and collects coins.

Lesson 18: Collisions

Game Lab

The class programs their sprites to interact in new ways. After a brief review of how they used the `isTouching` block, the class brainstorms other ways that two sprites could interact. They then use `isTouching` to make one sprite push another across the screen before practicing with the four collision blocks (`collide`, `displace`, `bounce`, and `bounceOff`).

Week 6

Lesson 19: Functions

Game Lab

This lesson covers functions as a way to organize their code, make it more readable, and remove repeated blocks of code. The class learns that higher level or more abstract steps make it easier to understand and reason about steps, then begins to create functions in Game Lab. At the end of the lesson the class uses these skills to organize and add functionality to the final version of their side scroller game.

Lesson 20: The Game Design Process

Game Lab

This lesson introduces the process the class will use to design games for the remainder of the unit. The class walks through this process in a series of levels. As part of this lesson the class also briefly learn to use multi-frame animations in Game Lab. At the end of the lesson they have an opportunity to make improvements to the game to make it their own.

Lesson 21: Using the Game Design Process

Game Lab

In this multi-day lesson, the class uses the problem solving process from Unit 1 to create a platform jumper game. After looking at a sample game, the class defines what their games will look like and uses a structured process to build them. Finally, the class reflects on how the games could be improved, and implements those changes.

Week 7

Lesson 22: Project - Design a Game

Game Lab | Project

The class plans and builds original games using the project guide from the previous two lessons. Working individually or in pairs, the class plans, develops, and gives feedback on the games. After incorporating the peer feedback, the class shares out the completed games.

Chapter Commentary

In this chapter students combine the constructs that they learned in the first chapter to program more complex movement and collisions in their sprites. As they create more complex programs, they begin to use functions to organize their code. In the end, students use a design process to create an original game.



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 1: Programming for Entertainment

Unplugged

Overview

Students are asked to consider the "problems" of boredom and self expression, and to reflect on how they approach those problems in their own lives. From there, students will explore how Computer Science in general, and programming specifically, plays a role in either a specific form of entertainment or as a vehicle for self expression.

Purpose

This lesson is intended to kick off this programming unit in a way that engages students of all backgrounds and interests. Though the end point of this unit asks students to develop a game, you should avoid starting out with a strong emphasis on **video** games. Instead, we attempt to broaden students' perspective about how programming is relevant to a form of entertainment or self expression that is personally engaging. This will provide an anchor for students to come back to throughout the unit as they consider the potential applications of the various programming skills that they learn.

Assessment Opportunities

1. Identify how computer science is used in a field of entertainment

In the activity guide, look at the "Interesting Fact or Use" section of the second page, make sure students have identified a use of computer science in their chosen fields.

Agenda

Warm Up (10 min)

The Entertainment Problem

Activity (45 min)

CS in Entertainment

Wrap Up (5 min)

Looking Forward

View on Code Studio

Objectives

Students will be able to:

- Identify how computer science is used in a field of entertainment

Preparation

- ☐ Review the research resources linked in Code Studio
- ☐ Print a copy of the activity guide for each group of three students

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teacher

- **CS in Entertainment** - Exemplar

For the Students

- **CS in Entertainment** - Activity Guide

Make a Copy ▼


Teaching Guide

Warm Up (10 min)

The Entertainment Problem

Remarks

Why do we seek out entertainment? Whether it's movies, music, art, games, or any number of other forms of entertainment, what problem does entertainment solve for us?

-  **Discuss:** What is your favorite form of entertainment, and what problem does it solve for you?

Discussion

The goal here is to get students to reflect more critically on **why** people seek out entertainment. Students are likely to come up with "boredom" as a common answer, but push them to think more deeply about what their chosen form of entertainment actually does for them - what problem does it solve. Potential answers include:

- Escape from difficult reality
- Connection with others
- Learning or experiencing new things
- Sparking creativity

Activity (45 min)

CS in Entertainment

Prompt: Select a few of the forms of entertainment that students identified in the warm up and ask them whether (and how) Computer Science plays a role in that field.

Group: Place students in groups of three. Consider allowing students to group based on common interest as each group will be exploring a field of entertainment together.


Distribute: Give each group a copy of the activity guide.

CS in Entertainment

During this activity student groups will do some light research into the role that CS and programming play in various fields of entertainment. The primary goal of this activity is to broaden students' perspective about how programming can be used to make fun or entertaining things. Some of the fields that students could research (such as art, animation, and games) can be directly connected to programs they will write later in this unit, while others may serve more as an inspiration for how the skills that they learn here may be applied in different domains.

Entertainment Exploration

- **Introduce the topics:** In the activity guide we've listed a number of potential fields for research. These specific fields were chosen to go along with resources that are provided on Code Studio, but you can allow students to look into other fields if they wish.
- **Explain the exploration task:** The goal of this exploration is twofold:
 - First, develop a deeper understanding of how programming is used in your chosen field. How is computer technology changing this field, what are some of the problems that people are trying to solve with technology?
 - Second, identify some interesting applications of CS or facts to share. What are some cool things that people are doing in this field that make use of CS?

-  • **Share selected research links :** On Code Studio, inside the blue teacher box, we have compiled a handful of useful sites to help students kick off their research. Share the links that you feel are most helpful and appropriate for your class.

Teaching Tip

Because many of these sites change their content daily, we have not shared them directly with students. We suggest that you check these links the morning before class to ensure that everything is still appropriate to be shared in school

Code Studio levels

Lesson Overview

[Teacher Overview](#)[Student Overview](#)

[View on Code Studio to access answer key\(s\)](#)

[View on Code Studio](#)



Exploring CS in Entertainment

[Teacher Overview](#)[Student Overview](#)

Here are a few sites to get students started. These sites are generally appropriate for school, but the content with them changes frequently, so **we strongly suggest that you check each site for inappropriate content before sharing it with students.**

[View on Code Studio](#)



[View on Code Studio to access answer key\(s\)](#)

Sample Programs

[3](#)[4](#)[5](#)[6](#)

(click tabs to see student view)

Interesting Information

Once groups have learned a bit about how CS is used in their chosen field, they can complete the second page of this activity guide, which asks them to do the following:

- **What Problem Does It Solve?:** "Problem" in this context can be fairly broad. It might be simplifying an otherwise laborious or complex task, doing things that wouldn't otherwise be possible, or any number of things that make this form of entertainment more accessible to end users.
- **How is it an Improvement?:** This could be tightly coupled with the answer to the previous question. Push students to consider how their form of entertainment was created before programming was prevalent.
- **An Interesting Fact or Use:** Share something fun or interesting about how CS is used in this field.
- **An Open Question:** It's likely that students come away with more questions than answers after just some quick research. Encourage them to reflect on what they don't yet know, and what questions they'd still like answered.

Share: Give groups a minute each to share their findings.

Wrap Up (5 min)

Looking Forward

- 💡 **Display:** Show either as a whole class, or let students explore independently, the example programs at the end of this lesson's Code Studio progression. These programs are designed to show a variety of different kinds of programs that it's possible to make in Game Lab.

Journal: Based on what you saw today, both in your research and the example apps, what kinds of programs are you most interested in learning to create?

💡 Teaching Tip

While some of these examples are pulled directly from lessons that students will complete later in the unit, a few of them use commands or techniques that aren't explicitly covered in the course. You can view the source for all of these programs in order to support students who may want to incorporate some of these techniques later on.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **IC** - Impacts of Computing



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 2: Plotting Shapes

Unplugged

Overview

Students explore the challenges of communicating how to draw with shapes and use a tool that introduces how this problem is approached in Game Lab. The warm up activity quickly demonstrates the challenges of communicating position without some shared reference point. In the main activity students explore a Game Lab tool that allows students to interactively place shapes on Game Lab's 400 by 400 grid. They then take turns instructing a partner how to draw a hidden image using this tool, accounting for many challenges students will encounter when programming in Game Lab. Students optionally create their own image to communicate before a debrief discussion.

Purpose

The primary purpose of this lesson is to introduce students to the coordinate system they will use in Game Lab. Students may have limited experience with using a coordinate grid or may struggle with the "flipped" y-axis in Game Lab. The drawing tool also forces students to think about other features of Game Lab students will see when they begin programming in the next lesson. These include the need to consider order while drawing, the need to specify color, and the fact that circles are positioned by their center and squares by their top-left corner. By the end of this activity, students should be ready to transfer what they have learned about communicating position to the programming they will do in the next lesson.

Assessment Opportunities

1. Reason about locations on the Game Lab coordinate grid

See the final reflection questions. This objective will also be assessed in the next lesson, within the context of programming in Game Lab.

2. Communicate how to draw an image in Game Lab, accounting for shape position, color, and order

See the final reflection questions.

Agenda

Warm Up (10 min)

Communicating Drawing Information

Activity (35 min)

Drawing with a Computer

[View on Code Studio](#)

Objectives

Students will be able to:

- Reason about locations on the Game Lab coordinate grid
- Communicate how to draw an image in Game Lab, accounting for shape position, color, and order

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teacher

- **Drawing Shapes** - Exemplar
- **Sample Shape Drawing** - Exemplar

For the Students

- **Drawing Shapes (Version B)** - Activity Guide [Make a Copy](#)
- **Drawing Shapes (Version A)** - Activity Guide [Make a Copy](#)

Wrap Up (5 min)
Journaling



Teaching Guide

Warm Up (10 min)

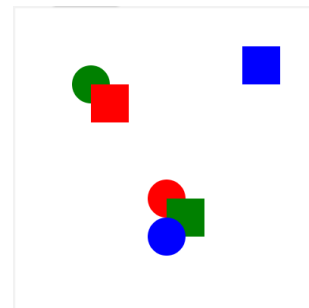
Communicating Drawing Information

Remarks

We saw a lot of different programs yesterday, and you started to think about what types you might want to create. When we create a program, one of the things we need to do is draw everything on the screen. We're going to try that with a "student" computer today.

Ask one or two volunteers to come to the front of the room and act as "computers" for the activity. They should sit with their backs to the board so that they cannot see what is being projected. Give each volunteer a blank sheet of paper.

Display: Project **Sample Shape Drawing - Exemplar** where it can be seen by the class.




Remarks

You will need to explain to our "computer" how to draw the picture. In the end, we'll compare the drawing to the actual picture.

Give the students a minute or two to describe the drawing as the students at the front of the room try to draw it. After one minute, stop them and allow the students to compare both pictures.

Prompt: What are the different "challenges" or problems we're going to need to solve in order to successfully communicate these kinds of drawings.

 **Discuss:** Students should silently write their answers in their journals. Afterwards they should discuss with a partner and finally with the entire class.

Remarks

There were several challenges we needed to solve in this activity. We need to be able to clearly communicate position, color, and order of the shapes. We're going to start exploring how to solve this problem.

Discussion

Goal This discussion is intended to bring up some challenges that students will need to address in the next few lessons, such as how to specify position, order, and color. The students don't necessarily need to decide how they would specify such things, but recognize that they will need a method for doing so. Students who have just come out of the HTML unit may make connections to how HTML helped solve similar problems.

Activity (35 min)

Drawing with a Computer

Group: Place students in pairs.

Transition: Have one member of each group open a laptop and go to the contents for this lesson. There is a single level with a Game Lab tool.

Code Studio levels


Lesson Overview

[Teacher Overview](#)[Student Overview](#)[View on Code Studio to access answer key\(s\)](#)[View on Code Studio](#)

Drawing Shapes

[Student Overview](#)

Prompt: Working with your partner take two or three minutes to figure out how this tool works. Afterwards be ready to share as a class.

 **Discuss:** After pairs have had a chance to work with the tool run a quick share-out where students discuss features they notice.

Drawing Shapes


Distribute activity guides to each pair, ensuring that one student (Student A) receives Version A and the other student (Student B) receives the Version B. Students should not look at each other's papers.

Set Up: In this activity, students will try to recreate images based on a partner's directions. The student who is drawing will use the shape drawing tool on Game Lab to draw the shapes. Students should keep their drawings hidden from one another throughout the activity. While completing a drawing the instruction giver should also not be able to see the computer screen.

Drawing 1: Each member of the pair should complete their first drawing, taking turns giving instructions and using the tool. These drawings do not feature any overlapping shapes, but students may need to grapple with the fact that circles are drawn from the middle and squares from the top left corner. Additionally students may just struggle with the direction of the Y-axis.

Discuss: Give pairs a couple minutes to discuss any common issues they're noticing in trying to complete their drawings.

Drawing 2: Each member of the pair should describe their second drawing to their partner. These drawing feature overlapping shapes and so students will need to consider the order in which shapes are being placed as well as when they should change the color of the pen.

 **Draw Your Own:** If time allows, give students a chance to create their own drawing to communicate to their partner.

Remarks

When we make images, we need a way to communicate exactly where each shape goes. The coordinate plane helps us to do that. Our coordinate plane has two coordinates, x and y . The x -coordinate tells us how far our shape is from the left of the grid. The y -coordinate tells us how far our shape is from the top of the grid. The black dots on the shapes help you be very specific about how the shape is placed on the grid.

Discussion

Goal: Rather than doing a live-demo of the tool, use this strategy to let students explore the tool themselves. Afterwards use the debrief to ensure all students are aware of key features of the tool. The most important components are the grid and the fact that the mouse coordinates are displayed below the drawing space.

Content Corner

Location of the Origin: The origin of this grid, as well as the origin in Game Lab, lies at the top left corner. This reflects the fact that documents tend to start at the top left, and ensures that every point on the plane has positive coordinates.

Teaching Tip

When to Move On: Determine whether this last activity is worth the time in your schedule. Giving students a chance to create and communicate their own drawing can help reinforce their knowledge, but if students are obviously achieving the learning objectives of the lesson without it then you can also just move to the wrap up to synthesize their learning.

Wrap Up (5 min)

Journaling

Journal: Have students reflect on each of the following prompts

- What things were important in communicating about position, color, and order of the shapes in this activity?
- What's a way you have seen similar problems solved in the past?

Discuss: Have pairs share their answers with one another. Then open up the discussion to the whole class.

Remarks

- ✓ At the beginning of class we saw that communicating how to draw even simple shapes can be pretty challenging. The grid we learned about today is one solution to this problem but there are many others that could've worked. In fact a lot of you probably noticed that the grid in Game Lab is "flipped". Computer screens come in all different shapes and sizes, as does the content we show on them. We need to agree on one point where all the content can grow from. Since we read starting at the top left corner, the grid on a computer screen starts at the top left corner as well. There's also the benefit of not having to use any negative numbers to talk about locations on the screen. Don't worry if this flipped grid is a little tricky still. We'll have plenty more time to work on it in coming lessons.

✓ Assessment Opportunity

Students should be able to explain the coordinate system of Game Lab. Make sure the students understand that it was important to note not only where the shape was positioned on the coordinate system, but the exact point of the shape (corner or center) that falls on that point on the grid.

Students should also explain that the code needs to set a color and specify the shape to be drawn to the screen, and that the order of the code determines which shape is on top.

The second prompt may be used to reinforce that the grid system students saw today is different from the one they may have seen in a math class. They should see, however, that both solve the same problem. Finally this discussion can lead into a teacher explanation of why the grid in Game Lab is "flipped" from ones they may have seen previously.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ AP - Algorithms & Programming



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 3: Drawing in Game Lab

Game Lab

Overview

Students are introduced to Game Lab, the programming environment for this unit, and begin to use it to position shapes on the screen. They learn the basics of sequencing and debugging, as well as a few simple commands. At the end of the lesson, students will be able to program images like the ones they made with the drawing tool in the previous lesson.

Purpose

The main purpose of this lesson is to give students a chance to get used to the programming environment, as well as the basic sequencing and debugging that they will use throughout the unit. Students begin with an introduction to the GameLab interactive development environment (IDE), then learn the three commands (`rect` , `ellipse` , and `fill`) that they will need to code the same types of images that they created on paper in the previous lesson. Challenge levels provide a chance for students who have more programming experience to further explore Game Lab.

Assessment Opportunities

1. Use a coordinate system to place elements on the screen.

See level 9 in Code Studio, in particular the placement of the square in the picture.

2. Sequence code correctly to overlay shapes.

See level 9 in Code Studio, in particular that the square is displayed in front of the circles.

Agenda

Warm Up (5 minutes)

Programming Images

Activity (30 minutes)

Simple Drawing in Game Lab

Wrap Up (10 minutes)

Share Drawings

Exit Ticket

View on Code Studio

Objectives

Students will be able to:

- Use a coordinate system to place elements on the screen.
- Sequence code correctly to overlay shapes.

Preparation

☐ Read the Forum

☐ Prepare projector or other means of showing videos if you wish to watch as a class

Vocabulary

- **Bug** - Part of a program that does not work correctly.
- **Debugging** - Finding and fixing problems in an algorithm or program.
- **Program** - An algorithm that has been coded into something that can be run by a machine.

Introduced Code

- `fill(color)`
- `ellipse(x, y, w, h)`
- `rect(x, y, w, h)`

Teaching Guide

Warm Up (5 minutes)


Programming Images

Remarks

In the last lesson we created images on the computer by organizing squares and circles on a grid. For each image you wanted to create, you had to lay those images out manually, and if you wanted to recreate an image it was a lot of work. Today, we're going to **program** the computer to draw those images for us. Based on what you know about computers, what do you think will be different between telling a person about your image and telling a computer about your image?

Discussion

Goal From Unit 2, students may remember that a computer can only understand what to do if you use a particular language in order to communicate with it. For Web Development, that language was HTML. Computers can't "figure out" what you mean in the same way that a human can, and they are usually much more specific in how they follow instructions.

-  Allow students time to think individually and discuss with a partner, then bring the class together and write their ideas on the board.

Remarks

In order to give instructions to a computer, we need to use a language that a computer understands. In the last unit, we used HTML, which is great for making web pages. To make our animations and games, we will use a version of Javascript that uses blocks. The environment that we'll be programming in is called Game Lab.

Activity (30 minutes)

Simple Drawing in Game Lab

Group: Place students in pairs to program together.

Transition: Send students to Code Studio.

Support: As students work on the levels you can help them but encourage them to try to spend some time figuring things out themselves first. If you need help supporting students, see the exemplars in the teacher answer viewer. When students hit the **challenge levels**, they can choose to pursue one or more of the challenges, return to improve upon previous levels, or help a classmate.

Teaching Tip

Pair programming is a great way to increase student confidence and foster the practices of collaboration and communication. You can read more about how to use Code Studio's pair programming feature **here**.

Teaching Tip

Students that are new to programming often have some common misconceptions they run into. In order to prevent those keep reminding students about the following things

- One command per line
- Commands run in order from top to bottom
- Order of inputs into shape commands matter
- Each input into shape commands are separated by commas
- (0,0) is in the upper left corner of the display
- All x and y values on the display are positive

Code Studio levels

Lesson Overview

Student Overview

Tour of Game Lab

[View on Code Studio](#)

Depending on the age and comfort level of your students, you may choose to use this level to tour the environment as a whole class. Make sure that students can find the level instructions, coding area, display area, and block drawers. This is also a good opportunity to point out some of the useful resources like documentation and the blocks to text button.

Video: Drawing in Game Lab - Part 1

Teacher Overview

Student Overview

Discussion Goals

[View on Code Studio](#)

Make sure students understand how to access the block documentation by clicking on the blocks inside the toolbox and clicking "see examples".

As students think of ideas of why they might prefer to use block mode, make sure that they understand that the block-based version of the programming language is just as legitimate as the text-based version. Students may offer that blocks make it easier to remember the exact commands or that they don't have to worry about the details of the parentheses or semicolons.

(Alternatively, advantages to text might be that it's easier to edit or that the text takes up less space.)

Using the Grid

Student Overview

Video: Drawing in Game Lab - Part 2

Teacher Overview

Student Overview

Discussion Goals

[View on Code Studio](#)

Stroke controls the border color of the shape, and fill controls the color inside of it.

Drawing

 6 7 8  9*(click tabs to see student view)*

Plotting Shapes in Game Lab

Teacher Overview

Student Overview

Simplified Shapes

[View on Code Studio](#)



The `rect` and `ellipse` commands that students were introduced to in this lessons are simplified versions of the full commands that they will see later. This allows the class to focus solely on the placement of shapes on the coordinate plane before also worrying about the width and height of those shapes.

Challenges



Extra



Extra



Extra

(click tabs to see student view)

Levels



Extra

(click tabs to see student view)

Wrap Up (10 minutes)

Share Drawings

Goal: Students get to see the variety of different things you can create with just simple shape drawings.

Share: Once students have completed their drawings have them share their drawings with the class. One way to do this is with a gallery walk.

Exit Ticket

Goal : Students share tricks they learned as they went through levels.

Prompt: Today you learned how to draw in Game Lab for the first time. What type of advice would you share with a friend who was going to learn about drawing in Game Lab to make it easier for them? Write it down on a piece of paper.

Collect: Collect answers from students and pick out a few that might be helpful for all students to hear. Share those at the beginning of the next class.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

► **AP** - Algorithms & Programming



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 4: Shapes and Randomization

Game Lab

Overview

In this lesson students continue to develop their familiarity with Game Lab by manipulating the width and height of the shapes they use to draw. The lesson kicks off with a discussion that connects expanded block functionality (e.g. different sized shapes) with the need for more block inputs, or "parameters". Students learn to draw with versions of `ellipse()` and `rect()` that include width and height parameters. They also learn to use the `background()` block. At the end of the progression students are introduced to the `randomNumber()` block. Combining all of these skills students will draw a randomized rainbow snake at the end of the lesson.

Purpose

This lesson gives students a chance to slightly expand their drawing skills while continuing to develop general purpose programming skills. They will need to reason about the x-y coordinate plane, consider the order of their code, and slightly increase their programs' complexity. The `randomNumber()` block is important for the next lesson where students learn to store values using variables. This lesson should be focused primarily on skill-building. If students are able to complete the rainbow snake independently, then they have the skills they'll need for the coming lessons.

Assessment Opportunities

1. Use and reason about drawing commands with multiple parameters

See level 7 in Code Studio.

2. Generate and use random numbers in a program

See level 11 in Code Studio.

Agenda

Warm Up (5 min)

Shapes of Different Sizes

Activity (40 min)

Programming Images

Wrap Up (5 min)

Journal

View on Code Studio

Objectives

Students will be able to:

- Use and reason about drawing commands with multiple parameters
- Generate and use random numbers in a program

Preparation

 Review the level sequence in Code Studio

Vocabulary

- **Parameter** - An extra piece of information passed to a function to customize it for a specific need

Introduced Code

- `background(color)`
- `ellipse(x, y, w, h)`
- `rect(x, y, w, h)`
- `randomNumber()`

Teaching Guide

Warm Up (5 min)

1. `ellipse()`
2. `randomNumber(5, 10)`
3. `rect`

Shapes of Different Sizes

Prompt: Our `ellipse` and `rect` blocks each have two inputs that control where they're drawn - the x and y position. If you wanted these commands to draw a wider variety of rectangles and ellipses, what additional inputs might you need to give these blocks? What would each additional input control?

- 🗣️ **Discuss:** Students should brainstorm ideas silently, then share with a neighbor, then discuss share out with the whole class. Record ideas as students share them on the board.

🗣️ Remarks

This was a good list of ideas. If we want our blocks to draw shapes in different ways they'll need more inputs that let us tell them how to draw. The inputs or openings in our blocks have a formal name, parameters, and today we're going to be learning more about how to use them.

Discussion

Goal: This discussion introduces the vocabulary word "parameter" and also helps motivate their use. Students will be seeing versions of the `ellipse()` and `rect()` block in this lesson that have additional parameters, as well as the `randomNumber()` block which has two parameters. Students may say they want inputs for the size of the shapes, their color, etc. During this conversation tie the behaviors the students want to the inputs the block would need. For example, if you want circles to be a different size the block will need an input that lets the programmer decide how large to make it.

Activity (40 min)

Programming Images

Transition: Move students onto Code Studio

🖥️ Code Studio levels

Lesson Overview 🖥️

Student Overview

New Shapes



(click tabs to see student view)

More Parameters



(click tabs to see student view)

Shapes and Parameters 📖

Student Overview

More Parameters



(click tabs to see student view)

Using Random Numbers



(click tabs to see student view)

Levels



Extra

(click tabs to see student view)

Share: If some students have taken extra time to work on their projects, give them a chance to share their more complex rainbow snakes. Focus conversation on which parameters students are manipulating or randomizing to create their drawings.

Wrap Up (5 min)

Journal

Prompt: Have students reflect on their development of the **five practices of CS Discoveries** (Problem Solving, Persistence, Creativity, Collaboration, Communication). Choose one of the following prompts as you deem appropriate.

- Choose one of the five practices in which you believe you demonstrated growth in this lesson. Write something you did that exemplified this practice.
- Choose one practice you think you can continue to grow in. What's one thing you'd like to do better?
- Choose one practice you thought was especially important for the activity we completed today. What made it so important?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 5: Variables

Game Lab

Overview

In this lesson students learn how to use variables to label a number in their program or save a randomly generated value. Students begin the lesson with a very basic description of the purpose of a variable. Students then complete a level progression that reinforces the model of a variable as a way to label or name a number. Students use variables to save a random number to see that variables actually store or save their values, allowing them to use the same random number multiple times in their programs.

Purpose

This lesson is the first time students will see variables in the course, and they are not expected to fully understand how variables work by its conclusion. Students should therefore leave this lesson knowing that variables are a way to label a value in their programs so that they can be reused or referenced later. In the following lesson students will be introduced to sprites, which need to be referenced by a variable.

Using variables to manipulate drawings is a surprisingly challenging skill that requires a great deal of forethought and planning. While students will use or modify many programs in this lesson, they are not expected to compose programs that use variables to modify the features of a drawing. In later lessons, students will expand their understanding of variables and more advanced ways they can be used.

Assessment Opportunities

1. **Identify a variable as a way to label and reference a value in a program**

See the reflection prompt in the Wrap Up.

2. **Use variables in a program to store a piece of information that is used multiple times**

See Level 9 in Code Studio.

Agenda

Warm Up (10 Mins)

Labels and Values

Activity (30 Mins)

Programming with Variables

Wrap Up (5 Mins)


View on Code Studio

Objectives

Students will be able to:

- Identify a variable as a way to label and reference a value in a program
- Use variables in a program to store a piece of information that is used multiple times

Preparation

 Review the level progression in Code Studio

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Introduction to Variables** - Video ([download](#))

Vocabulary

- **Variable** - A placeholder for a piece of information that can change.

Introduced Code

- Declare and assign a value to a variable
- Declare a variable

Teaching Guide

Warm Up (10 Mins)

Labels and Values

💡 **Video:** As a class watch the video introducing variables.

Review: Students can review the most important points from the video on the map level in Level 4. Students don't need to understand all of these ideas right now, but they can use this level as a reference throughout the lesson.

🎤 **Remarks**

There's a lot to learn about variables and we'll be seeing them throughout this unit. Today we're going to see them used to help us draw pictures. The most important thing we care about today, though, is just seeing how giving a label to a value helps us write programs.

💡 **Teaching Tip**

Avoiding Frontloading: While this lesson begins with two resources that explain how variables work, they'll likely be more meaningful to students once they have used variables in the drawing programs. Make sure students know these resources are available, and then return to them if you like at the end of the lesson to help in sense-making.

Activity (30 Mins)

Programming with Variables

💻 **Code Studio levels**

Lesson Overview

Student Overview

Introduction to Variables

Teacher Overview

Student Overview

Discussion Goals

[View on Code Studio](#)



Students should understand that variables hold information and can be accessed using their labels. With simple drawings, students may not see the power of variables, so you may want them to think of some different apps that they use and what information needs to be stored for the app to work, or think about a more complex program that they want to use variables for.

Numbers, text, and colors can all go into variables, as well as more complicated data structures that students will see later in the course.

Intro to Variables

 3

 4

 5

 6

(click tabs to see student view)

Variables

Student Overview

Intro to Variables

 8

  9

(click tabs to see student view)

Levels

 Extra

 Extra

 Extra

(click tabs to see student view)

Wrap Up (5 Mins)

Reflection

✔ **Prompt:** Give students the following prompts

- What is your own definition of a variable?
- Why are variables useful in programs?

Discuss: Have students silently write their ideas before sharing in pairs and then as a whole group.

💡 **Journal:** What connections do you see between variables and what you learned about the Input-Output-Store-Process model of a computer?

✔ Assessment Opportunity

Use this discussion to assess students' mental models of a variable. You may wish to have students write their responses so you can collect them to review later. You should be looking to see primarily that they understand that variables can label or name a number so that it can be used later in their programs. While there are other properties of a variable students may have learned, this is the most important before moving on to the next lesson.

💡 Teaching Tip

Input-Output-Store-Process: The model students learned in Unit 1 might be nice to reference here. Students are storing information in a variable which means they are saving that information in memory.

CSS Classes: If students studied CSS classes in Unit 2 then they may be familiar with the act of creating a name for something in their programs in order to be able to reference it. While the context here is different, the idea of naming remains the same.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

► **AP** - Algorithms & Programming



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 6: Sprites

Game Lab

Overview

In order to create more interesting and detailed images, students are introduced to the sprite object. Every sprite can be assigned an image to show, and sprites also keep track of multiple values about themselves, which will prove useful down the road when making animations.

Purpose

Keeping track of many shapes and the different variables that control aspects of those shapes can get very complex. There will be lots of variables with different variable names. Instead computer scientists created something called an **object** which allows for one variable name to control both the shape and all its aspects. In Game Lab we use a certain type of object called a **sprite**. A sprite is just a rectangle with **properties** for controlling its look. Properties are the variables that are attached to a sprite. You can access them through **dot notation**.

Using the Animation Tab, students can create or import images to be used with their sprites. Later on, these sprites will become a useful tool for creating animations, as their properties can be changed and updated throughout the course of a program.

Assessment Opportunities

1. Create and use a sprite

See levels 12 and 17 on Code Studio.

2. Use dot notation to update a sprite's properties

See level 17 on Code Studio.

Agenda

Warm Up (5 minutes)

How Much Information?

Activity

Introduction to Sprites

Wrap Up (5-10 min)

Share Out

Assessment

Assessing Sprite Scenes

View on Code Studio

Objectives

Students will be able to:

- Create and use a sprite
- Use dot notation to update a sprite's properties

Preparation

☐ (Optional) Print a copy of the activity guide for each student

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teacher

- **Sprite Scene Planning** - Exemplar
- **Sprite Scene Planning** - Exemplar

For the Students

- **Sprite Scene Planning** - Activity Guide

[Make a Copy](#)

Vocabulary

- **Property** - Attributes that describe an object's characteristics
- **Sprite** - A graphic character on the screen with properties that describe its location, movement, and look.

Introduced Code

- `drawSprites(group)`
- `var sprite = createSprite(x, y, width, height)`
- `sprite.scale`

Teaching Guide

Warm Up (5 minutes)

How Much Information?

🗣️ **Think, Pair, Share:** So far we've only written programs that put simple shapes on the screen. Come up with a list of all of the different pieces of information that you have used to control **how** these shapes are drawn.

Prompt: What if we wanted to create programs with more detailed images, maybe even characters that you could interact with? What other pieces of information might you need in your code?

🗣️ Remarks

Today we'll learn how to create characters in our animations called **sprites**. These sprites will be stored in variables, just like you've stored numbers in the past, but sprites can hold lots of pieces of data, which will allow you to create much more interesting (and eventually animated!) programs.

Activity

Introduction to Sprites

Distribute: (Optional) pass out copies of the activity guide. Students can use this sheet to plan out the Sprite Scene they create at the end of this lesson, but the planning can also be completed on scratch paper.

Transition: Send students to Code Studio

🖥️ Code Studio levels

Lesson Overview 🖥️

[Teacher Overview](#)[Student Overview](#)[View on Code Studio to access answer key\(s\)](#)[View on Code Studio](#)

Video: Introduction to Sprites 🎥

[Teacher Overview](#)[Student Overview](#)

💬 Discussion

The goal here is to get students thinking about all of the different values that go into drawing a single shape on the screen, and how many more values they may need to control a more detailed character in a program. If students are struggling to come up with ideas, you might use some of the following prompts: **How do you tell a shape where to go on the screen?** How do you tell a shape what size it needs to be? **How do you tell a shape what color it should be? What about its outline?** What if you wanted to change any of those values during your program, or control other things like rotation?

🎓 Content Corner

The sprite is a type of data called an **object**. While we aren't yet explicitly introducing the concept of objects, students do need to understand that a sprite is a different type of value from the ones we've seen before, one that can hold references to many more values. For students who are curious about whether there are other objects in our programs, ask them to see if there are more blocks in the toolbox that follow the same **dot** notation (such as `World.width` and `World.height`)

[View on Code Studio](#)

Discussion Goals

Sprites are a very complex concept, and students may have difficulty understanding exactly what they are. The most important aspect for students to understand is that sprites allow them to organize a lot of information about something that they want to draw to the screen.

Students should make the connection between properties and variables, that both hold information that their program needs to run. Properties are accessed through their sprites, and Game Lab sprites already have specific properties that are automatically created when students create each sprite, such as x position, height, and rotation.

Sprites solve the problem of organizing a lot of information about how something should be drawn to a screen. Rather than creating new variables to hold all of that information, sprites use properties to hold all of the information about one thing that is drawn to the screen.

Creating Sprites

[Student Overview](#)

Intro to Sprites

[4](#)[5](#)[6](#)*(click tabs to see student view)*

Video: The Animation Tab

[Teacher Overview](#)[Student Overview](#)[View on Code Studio](#)

Discussion Goals

Make sure students understand that they will need to both create the image (or animation) in the Animation Tab and then add the animation to the sprite using the `setAnimation` block. Students may be confused by the use of the word "animation" for single images, but in Game Lab, still images are considered "animations" with only one frame.

Students can use an animation already in the animation tab library, upload one from their computer, or create their own using the provided drawing tools. Additionally, students can use the drawing tools to modify the images they have chosen from the library or uploaded.

Misconception Alert!

Many students may now confuse the concepts of a sprite, its animation, and the image that it draws to the screen. In the next few lessons, watch out for this misconception, and reinforce the idea that a sprite's animation is just one of its properties, the one that controls what image is drawn to the screen. Remind students that a single sprite may have different animations throughout the course of the program, just as other properties can change, and that two or more sprites might share the same animation.

The Animation Tab

[Student Overview](#)

Adding Images

[9](#)[10](#)[11](#)[12](#)*(click tabs to see student view)*

Making Scenes

[13](#)[14](#)*(click tabs to see student view)*

Sprite Scenes

Student Overview

Making Scenes

 16

  17

 18

(click tabs to see student view)

Levels

 Extra

 Extra

(click tabs to see student view)

Wrap Up (5-10 min)

Share Out

Share: Allow students to share their Sprite Scenes. Encourage students to reflect on their scenes and identify ways in which they'd like to improve.

Assessment

Assessing Sprite Scenes

To assess Sprite Scenes, ask students to do a "talk through" of their code with you. Check to ensure that students know **why** they sequenced their code the way they did, and in particular look for "dead code", or code that doesn't impact the final scene. At this point it's likely that students are still drawing shapes before they draw the background (which then won't be seen) or that they are calling `drawSprites()` multiple times (it only needs to be called once).

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 7: The Draw Loop

Game Lab

Overview

In this lesson students are introduced to the draw loop, one of the core programming paradigms in Game Lab. To begin the lesson students look at some physical flipbooks to see that having many frames with different images creates the impression of motion. Students then watch a video explaining how the draw loop in Game Lab helps to create this same impression in their programs. Students combine the draw loop with random numbers to manipulate some simple animations with dots and then with sprites. At the end of the lesson students use what they learned to update their sprite scene from the previous lesson.

Purpose

The draw loop is a core component of Game Lab. The fact that the Game Lab environment repeatedly calls this function many times a second (by default 30) is what allows the tool to create animations. This lesson has two goals therefore. The first is for students to see how animation in general depends on showing many slightly different images in a sequence. To help students have physical flipbooks they can use, a video, and a map level. The second goal is for students to understand how the draw loop allows them to create this behavior in Game Lab. Students should leave the lesson understanding that the commands in the draw loop are called after all other code but are then called repeatedly at a frame rate. Students will have a chance to continue to develop an understanding of this behavior in the next two lessons, but laying a strong conceptual foundation in this lesson will serve them well for the rest of the unit.

Assessment Opportunities

1. Explain how the draw loop allows for the creation of animations in Game Lab

In the wrap up discussion, check that students mention that the code in the draw loop is run over and over, whereas code outside the draw loop is just run once, at the beginning of the program.

2. Use the draw loop in combination with the `randomNumber()` command, shapes, and sprites to make simple animations

See level 12 in Code Studio.

Agenda

[View on Code Studio](#)

Objectives

Students will be able to:

- Explain how the draw loop allows for the creation of animations in Game Lab
- Use the draw loop in combination with the `randomNumber()` command, shapes, and sprites to make simple animations

Preparation

- ☐ Print and assemble the manipulatives.
- ☐ Prepare the video.

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teacher

- **Random Dot Flipbook** - Manipulative
- **Random Sprite Flipbook** - Manipulative
- **Flipbook Example** - Video

Vocabulary

- **Animation** - a series of images that create the illusion of motion by being shown rapidly one after the other
- **Frame** - a single image within an animation
- **Frame Rate** - the rate at which frames in an animation are shown, typically measured in frames per second

Introduced Code

- `function draw() { }`
- `World.frameRate`

Warm Up (5 mins)

Activity (60 min)

Wrap Up (10 mins)



Teaching Guide

Warm Up (5 mins)

Video: Show **Flipbook Example - Video**

Prompt: This video shows a flipbook to make animation. In your own words how is it working? Why does it "trick our eyes" into thinking something is moving?

Discuss: Have students write their ideas independently, then share with partners, then as a whole group.

Vocabulary:

- **Frame:** a single image within an animation

Remarks

We're going to start learning how to make animations, not just still images. In order to do this we need a way to make our programs draw many pictures a second.

Our eyes will blur them together to make it look like smooth motion. To do this, though, we're going to need to learn an important new tool

Discussion

Goal: This discussion should introduce some key understandings about animation. Students should understand that the key is seeing many pictures in a row that are slightly different. Introduce the vocabulary word "frame" as one of those pictures. Then transition to the fact that soon students will be creating animations of their own.

Activity (60 min)

Video: Watch the video introducing the draw loop

Map Level: Briefly point students to the map level following the video. Ask students to use it as a reference as they complete the challenges in today's activity.

Demo: If you choose, use the provided manipulatives to provide a hands-on demo for exploring how the draw loop and animations are connected.

Teaching Tip

Pair Programming: This lesson introduces a challenging new paradigm. Until students are working to improve their projects consider having them pair program. Remember to give students clear instructions on when to switch driver and navigator.

Code Studio levels

Lesson Overview

Student Overview

Video: Introduction to the Draw Loop

Teacher Overview

Student Overview



Discussion Goal

The draw function runs the same code over and over in a loop, which is why it's often called "the draw loop". This creates an animation effect by changing what is drawn by a small amount each time the draw function runs. This can be compared to a physical flip book, which can animate an image by changing it slightly on each page.

Misconception Alert!

When the draw loop runs, it does not "clear" out previous drawing, so it will continue to show anything that has not been covered up by the new draw loop. If students do not add a "background" at the beginning of the draw loop, they will still see all the images drawn to the screen previously.

Shapes and the Draw Loop

3

(click tabs to see student view)

The Draw Loop

Student Overview

Shapes and the Draw Loop

5

6

(click tabs to see student view)

Sprites and the Draw Loop

7

(click tabs to see student view)

Sprite Properties

Student Overview

Sprites and the Draw Loop

9

10

11

(click tabs to see student view)

Animate Your Scene

12

(click tabs to see student view)

Levels

 Extra*(click tabs to see student view)*

Wrap Up (10 mins)

Share: Students only need to make small changes to their projects (e.g. shaking a single sprite) but ask them to share with a neighbor or as a full class

✔ **Prompt:** Have students respond to the following prompts

- What is an animation?
- Why does the draw loop help us make animations?
- What are some common errors or mistakes we should look out for as we keep programming with the draw loop?

Review: Return to the resources students saw at the beginning of the lesson (Map Level, Video, physical flip books) and address misconceptions that have arisen in the lesson.

Key Understandings: There are many common misconceptions with the draw loop. Make sure students understand the following

- The draw loop is run after all other code in your program. It does not actually matter where it is located in your program
- The draw loop is run by Game Lab at a constant frame rate of 30 frames per second. You do not actually need to call the function yourself.
- The "frames" in Game Lab can be thought of as transparency sheets. Unless you draw a background then all your new shapes or sprites will simply appear on top of your old ones
- You should only have one draw loop in your program

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

► **AP** - Algorithms & Programming



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 8: Counter Pattern Unplugged

Unplugged

Overview

Students explore the underlying behavior of variables through an unplugged activity. Using notecards and string to simulate variables within a program, students implement a few short programs. Once comfortable with this syntax, students use the same process with sprite properties, tracking a sprite's progress across the screen.

Purpose

Reasoning about variables can be tricky, especially for new programmers. In this lesson students complete an unplugged activity using physical manipulatives (cards and string) to build a mental model of how information can be stored in a variable and manipulated by a program. This model is then extended to sprite properties, which hold values in a similar way. This lesson introduces syntax and concepts that students will be able to “plug-in” in the following lesson. The mental model presented in this lesson will continue to be useful throughout the unit, even as students begin to write larger and increasingly complex programs.

Assessment Opportunities

1. **Describe the connection between updating a sprite's location properties and sprite movement on the screen.**

In the final wrap up discussions, make sure students are identifying updating sprite properties in a pattern as a general solution to the problem of movement.

2. **Read and follow the steps of a short program written in pseudocode that manipulates variable values.**

On pages 4 and 5 of the activity guide, check the “ending state” that students have written at the bottom of each program.

Agenda

Warm Up (15 mins)

Activity (20 mins)

Variables Unplugged Activity

Activity 2 (30 mins)

Sprite Properties

Wrap Up (10 mins)

Discussion

View on Code Studio

Objectives

Students will be able to:

- Describe the connection between updating a sprite's location properties and sprite movement on the screen.
- Read and follow the steps of a short program written in pseudocode that manipulates variable values.

Preparation

☐ Prepare materials for Labels and Values: index cards, post-its, or scraps of paper (2 in. by 2 in.) etc. (~ 50 per pair)

☐ Prepare materials for Connectors: pieces of string, pens, or pipe-cleaners, etc. (~ 4 per pair)

☐ Print copies of the manipulatives for each group or gather paper for students to use to make their boards.

☐ Review the rules of the Variables Unplugged Activity to ensure you understand them and are prepared to answer questions, especially if you will be demonstrating them yourself.

☐ Printed a copy of the activity guide for each student.

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teacher

- **Variables Unplugged** - Exemplar

For the Students

- **Variables Unplugged** - Activity Guide

[Make a Copy](#)

- **Variables Unplugged Board** - Manipulative [Make a Copy](#)

Vocabulary

- **Expression** - Any valid unit of code that resolves to a value.
- **Variable** - A placeholder for a piece of information that can change.

Teaching Guide

Warm Up (15 mins)

Prompt: In the last lesson, we used the draw loop to make our sprites move around on the screen. What are some other ways we might want to make our sprites move?

Put student ideas up on the board.

Remarks

In the next few lessons, we're going to look at lots of ways to have our sprites move around. In order to do that, we need to learn a little bit more about variables and how they work. Today, we're going to do an activity with variables and sprite properties that will help us to make these types of movement. As we go through the activity, think about how what you're learning might help you to make your sprites move in the way you want.

Activity (20 mins)

Variables Unplugged Activity

Group: Put students in pairs.

Distribute: Give each pair:

- 1) A set of labels/values and connectors
- 2) A single sheet of paper to create their board
- 3) 2 copies of the activity guide (One for each student.)

Display: Display the rules from the front page of the activity guide and write the first two programs from the second page.

Remarks

Today we are going to be working in a world of labels, values, and the connectors between them. To simulate this world you'll be using the scraps of paper and string that I've given you. To begin with we'll need to set up our boards, and afterwards we'll go over how the commands for this world work.

Demonstrate: Show the class how to divide their boards into 3 sections and label them accordingly, as shown on the first page of the activity guide.

Support: Students should work through the first two programs as a group, as you reference the steps in the activity guide. As groups feel increasingly comfortable with completing commands themselves encourage groups to independently run each command before comparing their boards with a neighboring pair. The goal is just to make sure everyone has an opportunity to understand the steps of the activity.

Remarks

Now it's your turn to try running some of these programs on your own. On the bottom of the page there are two more programs that you can run. For each one you should run the program to find out the **ending state** of the program. In other words, you want to know what labels are connected to what values. Once you reach the end of each program you can compare your results with a neighboring group. If you don't agree, then go back though and see if you can find where you lost track.

Support: Students should work in pairs through the two programs on the activity guide. Have a check-in to make sure everyone agrees on the ending state of the program (what labels are connected to what values). If students disagree, reinforce the need to debug when reading code by going back and tracing each step.

Activity 2 (30 mins)

Sprite Properties

Distribute: Sprite Properties in the activity guide if you have not included it on the original activity guide.

Demonstrate: Show students the rules of activity on the activity guide. Work through program 5 as a group, and demonstrate how to make a new sprite card and connect it to both its variable label card and the sprite property cards. Make sure students understand that they should be creating a new sprite card every time they see the `createSprite` command and should draw their sprites on the grid every time they see the `drawSprites` command.

Support: Students should work through the last two programs using their manipulatives.

Prompt: How did the sprite move across the grid in Program 3? How did the sprite move across the grid in Program 4?

After students have filled out the reflection questions, they should compare with another pair, then discuss as a class.

Wrap Up (10 mins)

Discussion

Prompt: We saw some clues today of how we might program the types of movement that we want for our sprites. What are some different ideas for how to program movement that you have after this activity? What are some problems that we still need to solve to make the sprite look like it's moving in the way that you want?

Allow students to brainstorm problems and list them on the board.

Prompt: Choose one or two of these problems and start to think of some ways you could solve this problem.

Allow students time to brainstorm individually before sharing out their solutions.

Remarks

These are great ideas. In the next lesson, we're going to look at how we can use some of the things we've learned today to make our sprites move in lots of different ways.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

► AP - Algorithms & Programming

Content Corner

This activity is designed to address many **common misconceptions** with variables and memory.

Common Misconceptions

- Variables can have multiple values (They cannot, variables hold at most one value)
- Variables “remember” old values (They do not, hence old values being removed to the Trash)
- Variables are connected after a command like “ $x = y$ ” (This may arise later in the course when students use sprites and will be addressed in great detail. For now students are being forced to create new value cards every time because even for a statement like $x = y$ there is no “connection” between those variables formed)
- Variables hold expressions (e.g. $1 + 5$) (Variables only hold values, expressions are computed beforehand. This is why value cards are only created once students have a single value. Enforce this rule closely)

Discussion

Goal: Students should see that commands such as $x = x + 1$ will move a sprite in a deliberate way across the screen, as opposed to the random movement they saw in the previous lesson.

Assessment Opportunity

Prompt 1: The goal of the discussion is not to have students think of solutions to all the problems, but for them to identify them, priming them for the counter pattern lesson. Their solutions may be vague, but make sure that they understand that continually updating the properties of a sprite in a pattern is essential to simulating motion on the screen.

Prompt 2: Some problems students may see are that the smiley faces remained on the screen, rather than moving, that they could only move left and right or up or down, or that the images stopped after a certain amount of time.



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 9: Sprite Movement

Game Lab

Overview

By combining the Draw Loop and the Counter Pattern, students write programs that move sprites across the screen, as well as animate other sprite properties.

Purpose

This lesson combines the Draw Loop that students first saw in Lesson 7 and the Counter Pattern that they learned in Lesson 8 to create programs with **purposeful** motion. By either incrementing or decrementing sprite properties, such as `sprite.x`, you can write programs that move sprites in expected patterns, instead of the randomization that we used in the past. The animations that students learn to create in this lesson lay the foundation for all of the animations and games that they will make throughout the rest of the unit.

Assessment Opportunities

1. **Use the counter pattern to increment or decrement sprite properties**

See Level 15 in Code Studio.

2. **Identify which sprite properties need to be changed, and in what way, to achieve a specific movement**

See Levels 6 and 15 in Code Studio.

Agenda

Warm Up (5 minutes)

Reviewing Sprite Properties

Activity (40 minutes)

Levels: Sprites and Images

Wrap Up (5 minutes)

Journal

[View on Code Studio](#)

Objectives

Students will be able to:

- Use the counter pattern to increment or decrement sprite properties
- Identify which sprite properties need to be changed, and in what way, to achieve a specific movement

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Animating Sprites** - Video ([download](#))

Teaching Guide

Warm Up (5 minutes)

Reviewing Sprite Properties

Prompt: On a piece of scratch paper, list out all of the sprite properties you can think of and what aspect of a sprite they affect.

- **Discuss:** What kinds of animations could you make by combining sprite properties with the counter pattern? Consider both adding and subtracting from properties, or even updating multiple properties at the same time. Record ideas as students share them on the board.

Activity (40 minutes)

Levels: Sprites and Images

Transition: Send students to Code Studio.

Support: Students should be progressing through the levels without stopping in this class. When they have finished the skill building levels they are given the option of which project they wish to extend.

Code Studio levels

Lesson Overview

Student Overview

Video: Sprite Movement

Teacher Overview

Student Overview

[View on Code Studio](#)



Discussion Goals

Students may describe the counter pattern in various ways. Make sure that they go beyond just stating the blocks or code that the counter pattern uses. They should understand that the counter pattern allows the programmer to update the value of a variable in a pattern that counts up (or down) on every iteration of the draw loop. This can be used for many different things, such as spinning, growing, shrinking, or timers, but it's most often used to move sprites across the screen.

When a sprite's x or y property is updated in a counter pattern, its position changes in a consistent way over time, causing it to move across the screen. Students should be able to explain that movement in an animation is just a change in position, and that changing a sprite's x position will cause it to move horizontally, and changing a sprite's y position will cause it to move vertically.

Movement with the Counter Pattern

 3

 4

 5

 6

 7

(click tabs to see student view)

Discussion

The purpose of this discussion is to start students thinking about how they might use the various sprite properties they've seen so far to make animations with purposeful motion. If students struggle to come up with ideas, you can narrow down the question to specific properties. For example:

- What would happen to a sprite if you constantly increased its x property?
- What would happen to a sprite if you constantly increased its y property?
- What about other properties, or combining multiple properties?

Animating Sprites

[9](#)
[10](#)
[11](#)
[12](#)
[\(click tabs to see student view\)](#)

Debugging with Watchers

[Student Overview](#)

Create Your Own Animation

[14](#)
[15](#)
[\(click tabs to see student view\)](#)

Levels

[Extra](#)
[Extra](#)
[Extra](#)
[\(click tabs to see student view\)](#)

Wrap Up (5 minutes)

Journal

Prompt: Have students reflect on their development of the **five practices of CS Discoveries** (Problem Solving, Persistence, Creativity, Collaboration, Communication). Choose one of the following prompts as you deem appropriate.

- Choose one of the five practices in which you believe you demonstrated growth in this lesson. Write something you did that exemplified this practice.
- Choose one practice you think you can continue to grow in. What's one thing you'd like to do better?
- Choose one practice you thought was especially important for the activity we completed today. What made it so important?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 10: Booleans Unplugged

Unplugged

Overview

In this lesson, students are introduced to boolean values and logic, as well as conditional statements. The class starts by playing a simple game of Stand Up, Sit Down in which the boolean (true/false) statements describe personal properties (hair or eye color, clothing type, age, etc). This gets students thinking about how they can frame a property with multiple potential values (such as age) with a binary question.

From there students are provided a group of objects with similar, yet varying, physical properties. With a partner they group those objects based on increasingly complex boolean statements, including compound booleans with AND and OR.

Finally we reveal Conditionals as a tool to make decisions or impact the flow of a program using boolean statements as input.

Assessment Opportunities

1. Evaluate simple and compound boolean statements

As students organize their objects into "true" and "false" piles, circulate the room to check on their answers and listen to their discussions.

2. Use the value of a Boolean statement to determine whether a command should be executed.

In the "Conditionals" part of the main activity, informally check student responses as you ask them to perform different actions based on the properties of their objects.

Agenda

Warm Up (10 min)

Stand Up, Sit Down
Example Translation

Activity (30 min)

Asking the Right Questions
Sorting with Booleans
Conditionals

Wrap Up (5 min)

Explicit Conditionals

[View on Code Studio](#)

Objectives

Students will be able to:

- Evaluate simple and compound boolean statements
- Use the value of a Boolean statement to determine whether a command should be executed.

Preparation

☐ Read the Forum

☐ Print a copy of the activity guide for each student

☐ (Optional) Gather objects with similar but varying features to use instead of the worksheet (LEGO bricks work well, a mixed bag of candy can be fun as well)

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Boolean Properties** - Activity Guide

[Make a Copy](#)

Vocabulary

- **Boolean** - A single value of either TRUE or FALSE
- **Conditionals** - Statements that only run under certain conditions.
- **Expression** - Any valid unit of code that resolves to a value.

Teaching Guide

Warm Up (10 min)

Stand Up, Sit Down

Distribute: Give each student a card and have them answer the following questions on it (feel free to add some of your own)

1. What is your hair color?
2. Do you wear glasses or contacts?
3. What is your favorite number?
4. What is your favorite color?
5. What month were you born?
6. Do you have any siblings?
7. What is the last digit of your phone number?
8. What is something about you that people here don't know and can't tell by looking at you?

Then collect the cards and shuffle them. To play the game, follow these steps:

For Each Card:

1. Select a card
2. Say: I'm going to read the answer to #8 but if it is you, don't say anything.
3. Read the answer to #8
4. Say: Now everyone stand up and we are going to ask some questions with Boolean answers to help determine who this person is. I'm going to say a bunch of statements. If they are true about you stay standing. If they are false sit down.
5. Translate the answers from #1 to #7 into statements that can either be true or false (See below). The person whose card it is should always answer true.
6. Because of how numbers 3, 4, 5, and 7 were asked it is likely that some people will still be standing. You will need to revisit these and ask them again in a more narrow fashion such as "My favorite color is purple".

Example Translation

| Question | Answer | True/False Statement |
|---|--------|---|
| 1) What is your hair color? | brown | "My hair color is brown." |
| 2) Do you wear glasses or contacts? | yes | "I wear glasses or contacts." |
| 3) What is your favorite number? | 12 | "My favorite number is greater than 10 and less than 20." |
| 4) What is your favorite color? | purple | "My favorite color appears in a sunset." |
| 5) What month were you born | May | "I was born in the spring." |
| 6) Do you have any siblings? | yes | "I have siblings." |
| 7) What is the last digit of your phone number? | 5 | "The last digit of my phone number is prime." |

🎓 Play this several times changing the true/false statements you use. Be creative with using **or** and **and**. Remind students that the OR means that either part of the statement being true will result in the entire statement being true.

Discuss: the Stand up Sit Down game with students:

- What kinds of questions did the teacher ask?
- Were you ever confused about whether you should be standing or sitting? Why?
- At any point in the game, how many different states could you be in?

Introduce the vocabulary boolean as a description for the kinds of questions we were asking. The defining feature of a Boolean is that it can have only two states - in our game those states were True and False, or Standing and Sitting

🎓 Content Corner

In English, an “or” is often an “exclusive or” such as “You can have chicken or fish.” In English, you only get to pick one, but with Boolean logic you could have chicken, fish, or both!! For the example person, “I was born in May OR my favorite number is 12” is true. Note that “I was born in May OR my favorite number is 13” is also true.

Activity (30 min)

Asking the Right Questions

Brainstorm

Prompt: Brainstorm places where they've seen Boolean values before, either in the class or in the world.

Discuss: Have students share out their answers. Potential answers could include:

- Binary
- Flow charts
- Light switches (and other devices that can be on or off)

Sorting with Booleans

🎤 Remarks

In the game we played, the boolean questions I asked were all based on your properties. Your properties didn't have to exist in only two states (how many different hair colors are in the room?), but the questions I asked had to split them into two states (how many people in the room have red hair?). We're going to do similar sorting using the properties of various images.

Group: Organize students into pairs

Set Up: Assign each member of the pair as either True or False.

- ✔ **Distribute:** Hand out the cut out images from the worksheet, or provide students with some objects to sort (such as LEGO bricks or candies).

🎤 Remarks

I'm going to read a bunch of binary statements in the form of `shape is equal to square or sides is greater than 4`, and you are going to sort through their objects to organize them into TRUE and FALSE piles. If students disagree about which pile an object should go into they should first discuss what the property is, what the two outcomes of the binary question are, and then if they still cannot agree they should bring it to the class for a vote.

If you are using the provided cutouts, you can start with the following questions:

- sides is equal to 3
- fill is equal to black
- corners is less than 1
- width is equal to height
- fill is equal to grey AND sides is greater than 4
- sides is greater than 4 AND less than 7

✔ Assessment Opportunity

As students organize their objects into "true" and "false" piles, circulate the room to check on their answers and listen to their discussions.

- 💡 • sides is greater than or equal to 5

Conditionals

Goal: After getting used to sorting objects into TRUE and FALSE, we need to introduce students to the concept that Booleans can also be used to control the flow of a program.

🎤 Remarks

A conditional allows us to make a decision based on the outcome of a boolean question (or condition). We actually were implicitly using conditionals in the Stand Up, Sit Down activity because there was an action related to each potential outcome of the boolean. We could have rephrased the instructions as

if statement is true: remain standing else: sit down

Prompt: Select one object from your pile and hold it up.

🎤 Remarks

I'm going to ask you a boolean question about your object and give you an action related to the potential outcome of the boolean. Figure out what your response should be for your shape and do the correct response.

✔ **Prompt:** Ask students some boolean questions about that single object AND give students something to do if that question is true. For example:

- If sides is equal to 4, do a dance
- If pattern is equal to striped, sit down
- If width is equal to height, hop on one foot

💡 Teaching Tip

Students often struggle with the idea of **greater-than** or **equals to** ever being FALSE - they tend to think of those as statements of truth rather than questions of relation.

In this activity we use the language **something** is equal to **something else** in order to mirror the code they'll see later (eg `something == something_else`), but for students who are struggling with seeing this as a question rather than a statement, you can encourage them to reword the statement by moving the 'is' to the beginning, so:

something is equal to something else

becomes

is something equal to something else?

✔ Assessment Opportunity

Informally check student responses as you go through these activities. Have students continue to hold their object up to make it easier for you to assess and allow you to use prompts that test any misconceptions you may have noticed earlier in the activity.

Wrap Up (5 min)

Explicit Conditionals

Goal: Booleans and conditionals are actually something that we use in our everyday lives - we just aren't usually explicit about it.

Model: As a way to practice thinking explicitly about conditionals, consider dismissing your students using compound booleans and conditionals. For example.

- If you sit at table four and your hair is brown, you may leave.
- If your first name starts with A, you may leave.
- If your shoes are black, you may leave.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

► **AP** - Algorithms & Programming



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 11: Booleans and Conditionals

Game Lab

Overview

Students start by using booleans to compare the current value of a sprite property with a target value, using that comparison to determine when a sprite has reached a point on the screen, grown to a given size, or otherwise reached a value using the counter pattern. After using booleans directly to investigate the values or sprite properties, students add conditional **if** statements to write code that responds to those boolean comparisons.

Purpose

This lesson follows closely the booleans model that students first experienced in the Booleans Unplugged lesson. As before, we start with using booleans directly before using booleans to trigger **if** statements. In the following lesson we will introduce some boolean producing blocks, such as `keyDown()`, which can be used in place of simple boolean comparisons to write programs that respond to user input.

Assessment Opportunities

1. **Use conditionals to react to changes in variables and sprite properties**

See Level 11 in Code Studio.

Agenda

Warm Up (5 min)

Answering Boolean Questions

Activity (40 min)

Booleans Plugged

Wrap Up (5 min)

Adding Conditionals

[View on Code Studio](#)

Objectives

Students will be able to:

- Use conditionals to react to changes in variables and sprite properties

Vocabulary

- **Boolean Expression** - in programming, an expression that evaluates to True or False.
- **If-Statement** - The common programming structure that implements "conditional statements".

Introduced Code

- If statement
- Equality operator
- Inequality operator
- Greater than operator
- Greater than or equal operator
- Less than operator
- Less than or equal operator

Teaching Guide

Warm Up (5 min)

Answering Boolean Questions

Goal: At the end of the Boolean Question game from the previous lesson, students began adding **conditions** to their boolean questions - meaning that **if** the answer to the question is true, something should happen. Before programming with conditionals, we want to make sure that students have a solid understanding of what booleans really are.

Prompt:

- How many different numbers are there in the world?
- How many different words or combination of letters and other characters are there?
- How many different boolean values are there?

Discuss: Students should realize that the first two questions (numbers and strings), are essentially infinite, but that booleans are limited to two states.

 **Remarks**

As you begin programming today, you'll be using booleans to make programs that change their behavior depending on the answer to those boolean questions.

Activity (40 min)

Booleans Plugged

Transition: Send students to Code Studio.

 **Content Corner**

Though seemingly simple, understanding how a boolean statement will evaluate can be difficult given that different programming languages have differing opinions on 'truthiness' and 'falsiness'. In fact, JavaScript (the language used in this course) has two different operators to test boolean equality `==` and `===`.

The double equals operator (`==`) is pretty generous in determining truthiness, for example each of the following is considered `true` in JavaScript when using the `==` operator, but would be `false` using the `===` operator:

```
1 == true;
"1" == true;
5 == "5";
null == undefined;
"" == false;
```

We use the `==` operator in this course because it's more forgiving, but it's important to be aware that it can sometimes report back truth when you really didn't intend it to (in which case you might want to use the more strict `===` operator)

Code Studio levels

Lesson Overview

Student Overview

Make a Prediction

2

(click tabs to see student view)

Video: Booleans

Teacher Overview

Student Overview

[View on Code Studio](#)



Discussion Goals

Students should be able to explain that a Boolean expression is something that is either true or false, similar to a yes or no question. The more formal way to say this is that Boolean expressions **evaluate** to either true or false. That means that when the computer processes a Boolean expression, it checks to see whether the expression describes a situation that is true or false, and then uses the value of either true or false wherever the expression is found.

Some examples of Boolean expressions that evaluate to true are $3 > 1$ and $4 \leq 7$, but press students to think of expressions that might be better represented by variables, such as `studentAge < 70` or `sizeOfClass > 2`.

Some examples of Boolean expressions that evaluate to false are $4 == 7$, `schoolName == "Hogwarts"`, and `currentYear < 1000`.

Boolean Comparison

4

5

6

7

(click tabs to see student view)

Booleans and Comparison Operators

Student Overview

Video: Conditional Statements

Teacher Overview

Student Overview

[View on Code Studio](#)



Discussion Goals

The broad point of this question is that programmers use if statements when they want the program to run differently in response to different situations. Encourage students to think of particular situations in which this would be the case. For example, they might want their characters to move faster when a "bonus" is in effect, or maybe they want more enemies to appear when the player reaches a certain level. Maybe they want the program to react in some way if a user presses a key or clicks the mouse, or they want a character to change animations if it touches a particular item.

Basic Conditionals

10

11

(click tabs to see student view)

If Statements

Student Overview

Levels

Extra

Extra

(click tabs to see student view)

Wrap Up (5 min)

Adding Conditionals

Journal: Think back to all of the programs you've written so far; how might you use conditionals to improve one of your programs from past lessons? What condition would you check, and how would you respond to it?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

► **AP** - Algorithms & Programming



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 12: Conditionals and User Input

Game Lab

Overview

Following the introduction to booleans and **if** statements in the previous lesson, students are introduced to a new block called `keyDown()` which returns a boolean and can be used in conditionals statements to move sprites around the screen. By the end of this lesson students will have written programs that take keyboard input from the user to control sprites on the screen.

Purpose

One common way conditionals are used is to check for different types of user input especially key presses. Having a way for a user to interact with a program makes it more interesting and dynamic. Without interaction from the user it is very difficult to create a game. Therefore the introduction of conditionals and user inputs for decision making is the first big step toward creating games.

Assessment Opportunities

1. Use conditionals to react to keyboard input

See Level 7 in Code Studio.

2. Move sprites in response to keyboard input

See Level 7 in Code Studio.

Agenda

Warm Up (5 min)

Taking Input

Activity (40 min)

Keyboard Input

Wrap Up (5 min)

Considering Conditions

[View on Code Studio](#)

Objectives

Students will be able to:

- Use conditionals to react to keyboard input
- Move sprites in response to keyboard input

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Boolean Expressions** - Video ([download](#))

Introduced Code

- `keyDown(code)`

Teaching Guide

Warm Up (5 min)

Taking Input

- 🗣️ **Discuss:** So far all of the programs you've written run without any input from the user. How might adding user interaction make your programs more useful, effective, or entertaining? How might a user provide input into your program?

Discussion

The goal here isn't to get into the technical specifics of how programs can take input (students will get to that in the online portion of the lesson), but rather to get students thinking about how allow user input could change the programs they've made. Encourage students to think back to Unit 1 and the various computer inputs and outputs they explored then. Which inputs would be most useful for the types of programs they've been making?

Activity (40 min)

Keyboard Input

Transition: Send students to Code Studio

🖥️ Code Studio levels

Lesson Overview 🖥️

Student Overview

Keyboard Input

🖥️ 2

🖥️ 3

🖥️ 4

🖥️ 5

🖥️ 6

🗑️ 🖥️ 7

(click tabs to see student view)

Editing Images 📖

Student Overview

Keyboard Input

🖥️ 9

(click tabs to see student view)

Levels

🖥️ Extra

🖥️ Extra

(click tabs to see student view)

Wrap Up (5 min)

Considering Conditions

Prompt: To get students to continue thinking about how conditionals can be used in programming, prompt them to come up with scenarios in games or programs they use regularly that might be triggered by conditionals.

Discuss: Have students share responses. Student responses might include:

- If my username and password are correct, log me into Facebook
- If Pacman has collected all the balls, start the next level
- If my keyboard or mouse hasn't moved in 10 minutes, turn on the screensaver

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

► AP - Algorithms & Programming



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 13: Other Forms of Input

Game Lab

Overview

In this lesson students continue to explore ways to use conditional statements to take user input. In addition to the simple `keyDown()` command learned yesterday, students will learn about several other keyboard input commands as well as ways to take mouse input.

Purpose

Students have learned how to make simple decisions with conditionals. Sometimes however we want to make decision based on if the condition we asked about originally was false or we want to make a decision based on multiple conditions being true. That's where else statements and more complex conditionals come in. Else statements are a second statement which is attached to a if statement. Else statements execute when the if statement it is attached to is false. You can think of it as "if something is true do thing 1 else do thing 2."

This concept is introduced alongside several new key and mouse input commands, allowing students to gradually build up programs that input in different ways.

Assessment Opportunities

1. Use an if-else statement to control the flow of a program.

See Level 5 in Code Studio.

2. Respond to a variety of types of user input.

Use the wrap up discussion to check students' understanding of the different types of user input. You may also informally check their ability to use them as they progress through the Code Studio lesson.

Agenda

Warm Up (5 minutes)

Check for Understanding

Activity (40 minutes)

If/Else and More Input

Wrap Up (5 minutes)

Wrap Up

[View on Code Studio](#)

Objectives

Students will be able to:

- Use an if-else statement to control the flow of a program.
- Respond to a variety of types of user input.

Vocabulary

- **Conditionals** - Statements that only run under certain conditions.

Introduced Code

- `keyWentDown(code)`
- `keyWentUp(code)`
- `mouseDidMove()`
- `mouseDown(button)`
- `mouseWentDown(button)`
- `mouseWentUp(button)`
- `sprite.visible`
- If/else statement

Teaching Guide

Warm Up (5 minutes)

Check for Understanding

Remarks

Today we'll be picking up today where we left off yesterday - using conditionals to write programs that respond to user input. Let's refresh what we learned yesterday.

Prompt:

- What is a Boolean? (eg. a true/false value)
- What is the relationship between a Boolean and a Conditional? (eg. a conditional asks a Boolean question and runs code if the answer is true)
- What are some examples of comparison operators that result in a Boolean? (eg. >, <, ==)
- What is the difference between = and == ? (eg. = is used to assign a value, == is used to check if two values are equal)

Remarks

That's great! Today we're going to look at a way to make our conditionals even more powerful, and see some new ways to get user input.

Activity (40 minutes)

If/Else and More Input

Transition: Move the class to Code Studio, and have students complete the prediction level as a class or in small groups, then talk about what they found.

Video: Watch the video as a class and review the discussion questions together.

Code Studio levels

Lesson Overview

Student Overview

Predict

Student Overview

Video: If/Else Statements

Teacher Overview

Student Overview

Discussion Goals

View on Code Studio



Make sure students are thinking of situations in which they want two different things to happen, depending on the situation. For example, they may say that they want one animation if the sprite is moving to the left and a different animation if the sprite is moving to the right. Challenge the students to think about when they would just use an `if` block, and when an `ifelse` block is necessary.

Input with If-Else

4

5

(click tabs to see student view)

If-Else Statements

Student Overview

More Input

7

8

9

10

11

12

13

(click tabs to see student view)

Levels

Extra

Extra

(click tabs to see student view)

Wrap Up (5 minutes)

Wrap Up

- ✓ **Prompt:** You now have many different ways to detect user input. With a partner, choose three different user input commands and think of an example of when you might use them. Be ready to share with the class!

✓ Assessment Opportunity

This discussion serves as a brief review and assessment of the new user input commands. As students share out, press them to explain why their choice is better than other, similar choices (mouseDown / mouseWentDown / mouseWentUp). If any commands are missing after all the groups have shared out, elicit the missing ones from the group before moving on.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 14: Project - Interactive Card

Game Lab | Project

Overview

In this cumulative project for Chapter 1, students plan for and develop an interactive greeting card using all of the programming techniques they've learned to this point.

Purpose

This end of chapter assessment is a good place for students to bring together all the pieces they have learned (drawing, variables, sprites, images, conditionals, user input) in one place. Students should still be working with code that is easily readable and doesn't involve very many high level abstractions. Giving students the opportunity to really be creative after learning all these new concepts will help to engage them further as we head into Chapter 2.

Assessment Opportunities

Use the project rubric attached to this lesson to assess student mastery of learning goals of this unit.

Agenda

Warm Up (10 min)

Demo Project Exemplars (Level 2)

Activity (2 days)

Unplugged: Interactive Card Planning

Levels: Implementing Interactive Card (Level 3 - 7)

Peer Review

Iterate - Update Code

Reflect

Wrap Up (10 minutes)

Sharing Cards

View on Code Studio

Objectives

Students will be able to:

- Use conditionals to react to keyboard input or changes in variables / properties
- Sequence commands to draw in the proper order
- Apply an iterator pattern to variables or properties in a loop

Preparation

☐ Read the Forum

☐ Print out a copy of the project guide for each student

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teacher

- **Interactive Card** - Exemplars

For the Students

- **Interactive Card** - Project Guide

[Make a Copy](#)

- **Interactive Card** - Rubric [Make a Copy](#)

- **Interactive Card** - Peer Review

[Make a Copy](#)

- **Computer Science Practices** - Reflection

[Make a Copy](#)

Teaching Guide


Warm Up (10 min)

Demo Project Exemplars (Level 2)

Goal: Students see an example of a final project and discuss the different elements that went into making it.

Display: On the projector, run the exemplar for students. Since this is on level 2 of the progression, if its easier for students to do it on their own computers you could do that as well.

Code Studio levels

- Interactive Card 
- **Student Overview**

Example Project

View on Code Studio



Run the program a few times and answer the following questions:

- 1) Which elements appear to use drawing commands?
- 2) Which elements appear to be Sprites?
- 3) For each Sprite, which properties are being updated?
- 4) Where do you see conditionals being used?
- 5) Are there elements that you don't understand?

Discuss: Have students share their observations and analyses of the exemplar.

Encourage the class to consider that there are multiple approaches to programming anything, but that there may be clues as to how something was created. In particular, when they are sharing their thoughts ask them to specify the following :

- Clues that suggest a Sprite was used
- Clues that suggest a conditional was used
- Clues that suggest an iterator pattern was used

Display: Show students the rubric. Review the different components of the rubric with them to make sure they understand the components of the project.

Activity (2 days)

Unplugged: Interactive Card Planning

Goal: Students should plan out what they want to create before they head to the computer so that once they get to the computer they are just executing the plan.


Distribute: Hand out the project guide to students. This is the tool students will use to scope out their projects before getting onto the computers. Give students some time to brainstorm the type of card they want to create and who the recipient will be.

Steps

- 1) The first layer of the interactive card is a background drawn with just the commands in the Drawing drawer. The front of the Activity Guide provides a grid for students to lay out their background, a reference table of drawing commands, and an area for students to take notes and write pseudocode.
- 2) Next students think through the Sprites they'll need, filling out a table of each Sprite's label, images, and properties

3) Finally students consider the conditionals they'll need in order to make their card interactive.

Levels: Implementing Interactive Card (Level 3 - 7)

 **Transition:** Once students have completed their planning sheet, it's time to head to the Code.org website for Lesson 10. The short level sequence asks students to complete each element of their project.

Code Studio levels

- Making an Interactive Card 
- **Student Overview**

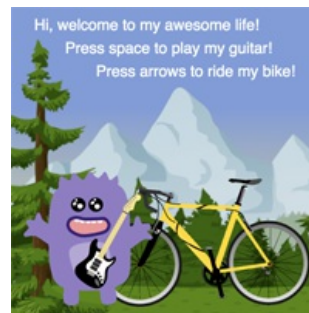
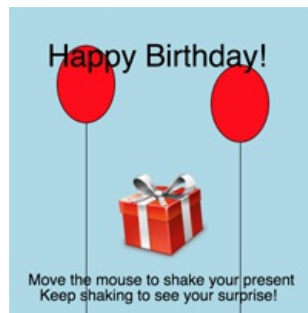
Your Interactive Card

[View on Code Studio](#)



In the next few levels, you'll be completing your own interactive card. Here are some examples to give you some ideas. Don't forget to look at the code to see how they work.

Examples



- Levels
-  4
-  5
-  6
-  7

Student Instructions

[View on Code Studio](#)



Laying Out Your Background

Before beginning this project, you should have already completed the Interactive Card Planning activity, and you'll want to have that paper with you as you develop your program. Preparation is one of the **most important** elements of successfully creating a program!

Do This

Refer to your planning activity sheet to help you lay out the shapes that will become the background to your card.

- First, figure out what the lowest layer in your image is (this should use the `background()` block) and add it to the very top of the draw loop.
- Next, layer each additional drawing block in the order you want them to appear in the stack.
- Finally, add a comment to the top of this section of code to describe what it does, and if you have any particularly complicated chunks of code within (such as code to draw a tree or a house), add a descriptive comment to that as well.

Challenge: Can you use variables or `randomNumber()` to add some subtle animation to your background layer?

Student Instructions

[View on Code Studio](#)

Adding Sprites

Now that you have the more static elements of your card layed out, it's time to add the Sprites. Your Sprites should provide the primary animations and interactions for your card - so feel free to get creative here and have fun.

Do This

Check out the Sprites table on the back of your planning sheet. For each Sprite in your table:

- Initialize the Sprite at the top of your program with `createSprite()` .
- Find or create the image(s) for the Sprite and set it with `setAnimation()` .
- Inside the `draw()` loop update any Sprite properties that we will be constantly animating (we'll deal with conditionals in a minute).

Student Instructions

[View on Code Studio](#)

User Input

You've got a background, you've got Sprites, now it's time to give your user something to do!

Do This

On the interactions table from your planning sheet, find all of the interactions that rely on user input (key presses and mouse movements). For each of those interactions:

- Add an `if` block (or `if-else` block if you need a fallback action) inside the `draw()` loop.
- Add the appropriate input block for your condition (such as `keyDown()` or `mouseDown()`).
- Add the necessary actions inside the `if` block.

Challenge: Can you create more sophisticated conditionals by nesting them or using compound booleans?

Student Instructions

[View on Code Studio](#)

Other Conditionals

The **surprise** in your card comes from conditionals that don't directly respond to user input, but to some other element of your card. This could be triggered by a variable that gets updated as the user interacts with your card, or a Sprite moving into a certain part of the screen.

Do This

For each of the remaining items on your interactions table:

- Add an `if` block (or `if-else` block if you need a fallback action) inside the draw loop.
- Add the appropriate Boolean comparison block to the condition (eg. `<` , `>` , or `==`).
- Add the necessary actions inside the `if` block.

Challenge: Can you create more sophisticated conditionals by nesting them or using compound booleans?

Peer Review

Distribute: Give each student a copy of the peer review guide.


Students should spend 15 minutes reviewing the other student's card and filling out the peer review guide.

Iterate - Update Code

Circulate: Students should complete the peer review guide's back side and decide how to respond to the feedback they were given. They should then use that feedback to improve their cards.

Reflect

Using the rubric, students should assess their own project before submitting it.

 Send students to Code Studio to complete their reflection on their attitudes toward computer science. Although their answers are anonymous, the aggregated data will be available to you once at least five students have completed the survey.

Code Studio levels

- Levels
-  8

Student Instructions

[View on Code Studio](#)



Finishing Touches

Now's your chance to put some finishing touches on your card. We've included some new blocks that you haven't seen before, so take some time to look around and try out some new blocks.

Do This

Consider adding any of the following to finish up your card:

- Text
- Additional images for your sprites
- Subtle animation in the background
- Sound effects (Can you figure out now to do this?)
- More ways for a user to interact with your card

Wrap Up (10 minutes)

Sharing Cards

Goal: Students share their creations with the class.

Share: Find a way for students to share their cards with each other, and with the intended recipient. It will likely be helpful to use the share link for the project so that students can share the project with other students.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 15: Velocity

Game Lab

Overview

After a brief review of how they used the counter pattern to move sprites in previous lessons, students are introduced to the properties that set velocity and rotation speed directly. As they use these new properties in different ways, they build up the skills they need to create a basic side scroller game.

Purpose

This lesson launches a major theme of the chapter: that complex behavior can be represented in simpler ways to make it easier to write and reason about code.

In this lesson they are taught to use the velocity blocks to simplify the code for moving a sprite across the screen. This marks a shift in how new blocks are introduced. Whereas previously blocks were presented as enabling completely new behaviors, they are now presented as simplifying code students could have written with the blocks previously available. Over the next several lessons, students will see how this method of managing complexity allows them to produce more interesting sprite behaviors.

Assessment Opportunities

1. Use the velocity and rotationSpeed blocks to create and change sprite movements

See Level 9 in Code Studio. (Level 6 can also be used to check student use of the rotationSpeed block.)

2. Describe the advantages of simplifying code by using higher level blocks

In the wrap up, check students' descriptions of the blocks that they would create and why they would want to create them.

Agenda

Warm Up (15 min)

Activity (75 minutes)

Learning to Use the Velocity Blocks

Wrap Up (5 min)

Journal

[View on Code Studio](#)

Objectives

Students will be able to:

- Use the velocity and rotationSpeed blocks to create and change sprite movements
- Describe the advantages of simplifying code by using higher level blocks

Introduced Code

- `sprite.rotationSpeed`
- `sprite.velocityX`
- `sprite.velocityY`


Teaching Guide

Warm Up (15 min)

Demonstrate: Ask for a volunteer to come to the front of the class and act as your "sprite". Say that you will be giving directions to the sprite as though you're a Game Lab program.

When your student is ready, face them so that they have some space in front of them and ask them to "Move forward by 1". They should take one step forward. Then repeat the command several times, each time waiting for the student to move forward by 1 step. You should aim for the repetitiveness of these instructions to be clear. After your student has completed this activity, have them come back to where they started. This time repeat the demonstration but asking the student to "Move forward by 2" and have the student take 2 steps each time. Once the student has done this multiple times ask the class to give them a round of applause and invite them back to their seat.

Prompt: I was just giving instructions to my "sprite", but they seemed to get pretty repetitive. How could I have simplified or streamlined my instructions?

 **Discuss:** Give students a minute to write down thoughts before inviting them to share with a neighbor. Then have the class share their thoughts. You may wish to write their ideas on the board.

Remarks


One way to simplify these instructions is just tell our sprite to keep moving by 1 or 2, or however many steps we want. This would make instructions as humans easier to understand, and as we're about to see there's a similar way to make our code simpler as well.

Discussion

Goal: The earlier demonstration should have reinforced the fact that repeatedly giving the same instruction is something you would never do in real life. You would instead come up with a way to capture that the instruction should be repeated, like "keep moving forward by 1."

Show: Sprite Velocity Video

Code Studio levels

- Video: Velocity 
- **Teacher Overview**
- **Student Overview**

[View on Code Studio](#)




Discussion Goals

It may not be obvious to students why the velocity block is so powerful. The immediate answer is that the velocity block allows a programmer to set the velocity at the beginning of the program and not have to worry about the counter pattern inside the draw loop (as Game Lab will take care of that). If students are having trouble of thinking of situations in which the velocity block provides a big advantage, assure them that they will tackle some problems in the coming lesson that they will need this block for.

As students give you examples, try to elicit answers that use both positive and negative numbers, and that use the x and y positions as well as sprite rotation.

Display: On the board, write the following pieces of code.

- `sprite.velocityX = 4;`
- `sprite.velocityY = -1;`
- `sprite.rotationSpeed = 2;`

 **Prompt:** We just saw how these new blocks help us simplify the counter pattern we used to move sprites. On a sheet of paper write down the counter pattern each of these lines of code is replacing.

Discuss: Have students write their ideas on a sheet of paper before discussing their responses as a class. Eventually write the correct answers on the board next to each, as shown below.

- `sprite.x = sprite.x + 4;`
- `sprite.y = sprite.y - 1;`
- `sprite.rotation = sprite.rotation + 2;`

Remarks

These new "higher level" blocks are helping us to write code that we actually already knew how to write.

They're simplifying our code for us by hiding some of the unnecessary details. As we're about to see, however, these new blocks will change the kind of programs and games we're able to write.

Teaching Tip

Check for Understanding: This is just a quick check for understanding after the video. Students are quickly prompted to see if they understood the main point of the video and if not you have an opportunity to reinforce it before moving into the Code Studio levels. Leave the translation from the velocity blocks to their associated counter pattern on the board to reference throughout the lesson.

Activity (75 minutes)

Learning to Use the Velocity Blocks



Transition: Move students to Code Studio

Circulate: These levels introduce the `velocityX`, `velocityY`, and `rotationSpeed` properties that you just discussed with students. Check in with students to see how they are doing and keep track of when everyone has made it to the end of level 10.

Teaching Tip

Inside or Outside the Draw Loop?: For the first few puzzles, check to make sure that students are setting the velocities and rotation speeds **outside** the draw loop, immediately after they create their sprites. The code will work for the first few puzzles, even if they set the velocities inside the draw loop, but it will cause problems later.

Code Studio levels

Lesson Overview

Student Overview

Velocity

 3 4 5 6 7 8 9

(click tabs to see student view)

Sprite Velocity

Student Overview

Side Scroller

 11 12 13

(click tabs to see student view)

Levels

 Extra

(click tabs to see student view)

If students are finished early, encourage them to experiment with different aspects of the scroller game they created. They can change the animations, create a background, etc.

Wrap Up (5 min)

Journal

✔ **Prompt:** You learned a few new blocks today. As first glance, these blocks did the same sorts of things we'd already done with the counter pattern, but made it simpler for us to do them. As you went through the puzzles, though, you started doing some interesting movements that we hadn't been able to do before.

- Describe one of those movements, and how you made it.
- Describe another block that you'd like to have.
 - What would you name it?
 - What would it do?
 - What code would it hide inside?
 - How would it help you?

Remarks

All of the movements that we did today are possible without the new blocks, but it would be very complicated to code them. One of the benefits of blocks like velocity is that when we don't have to worry about the details of simple movements and actions, we can use that extra brain power to solve more complicated problems. As you build up your side scroller game, we'll keep looking at new blocks that make things simpler, so we can build more and more complicated games.

✔ Assessment Opportunity

As students describe the blocks that they would make, ensure that they are relating the specific code that a block would use to the higher-level concept of what it would do. For example, a "velocity" block would move a sprite across the screen, and it would include blocks that use the counter pattern on a position property.

Students should describe advantages of having these blocks, such as not needed to re-write the code all the time, or making it easier to read what the program is doing.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 16: Collision Detection

Game Lab

Overview

Students learn about collision detection on the computer. Working in pairs, they explore how a computer could use sprite location and size properties and math to detect whether two sprites are touching. They then use the `isTouching()` block to create different effects when sprites collide, including playing sounds. Last, they use their new skills to improve the sidescroller game that they started in the last lesson.

Purpose

This lesson formally introduces the use of abstractions, simple ways of representing underlying complexity.

In the last lesson, students were exposed to the idea of using one block to represent complex code. Students further explore this idea in the context of the intentionally complex mathematical challenge of determining whether two sprites are touching. By using a single block to represent this complexity, in this case the `isTouching` block, it becomes much easier to write and reason about code, and students can appreciate the value of using abstractions. In later lessons, students will continue to build on the `isTouching()` abstraction to create more complex sprite interactions.

Assessment Opportunities

1. **Detect when sprites are touching or overlapping, and change the program in response.**
See Level 7 in Code Studio.
2. **Describe how abstractions help to manage the complexity of code**

In the Wrap Up discussion, make sure students can identify how more abstract blocks can help with creating larger or more complex programs.

Agenda

Warm Up (10 min)

Activity

Collisions Unplugged
`isTouching()`

Wrap Up (5 min)


View on Code Studio

Objectives

Students will be able to:

- Detect when sprites are touching or overlapping, and change the program in response.
- Describe how abstractions help to manage the complexity of code

Preparation

 Print copies of the activity guide such that each pair of students has a part A and a part B

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teacher

- **Collision Detection** - Exemplar

For the Students

- **Collision Detection (Version B)** - Activity Guide [Make a Copy](#)
- **Collision Detection (Version A)** - Activity Guide [Make a Copy](#)

Vocabulary

- **Abstraction** - a simplified representation of something more complex. Abstractions allow you to hide details to help you manage complexity, focus on relevant concepts, and reason about problems at a higher level.
- **Debugging** - Finding and fixing problems in an algorithm or program.
- **If-Statement** - The common programming structure that implements "conditional statements".

Introduced Code

- `sprite.isTouching(target)`
- `sprite.debug`

Teaching Guide

Warm Up (10 min)

Review: Briefly remind students of the side scroller game they made in the last lesson, and ask them to share out any ideas they had for improving their game in the future.

💡 **Display:** On a projector or just a computer screen demonstrate the game that appears in the first level in Code Studio for this lesson. The teacher or a student can control the frog to jump over a mushroom. Make sure at least some jumps are successful and some are not to see that the collision detection is working.

Prompt: An interesting improvement in this game is that the mushroom moves when the frog touches it. Can you think of any way that the computer could use the sprites' properties to figure out whether they are touching each other?

💬 **Discuss:** Allow the students to brainstorm ideas for how the computer could determine whether the two sprites are touching. List their ideas on the board and tell them that they'll have a chance to try out their theories in a moment.

💡 Teaching Tip

Showing the Game: Avoid signing students into Code Studio at this point to play the game themselves. They have a fairly significant unplugged activity immediately afterwards and it will likely lead to difficulties in transitioning.

💬 Discussion

Goal: This purpose of this discussion is just to get some ideas on the board for students to use in the next activity. There's no need to actually evaluate or try them, because students will be working together to do so immediately after the discussion.

Activity

Collisions Unplugged

Group: Group students into pairs.

🗣️ Remarks

Now you're going to have a chance to try out the strategies that you came up with as a group. Each activity guide has four sheets of paper. One partner should take the papers with the "A" on the top, and the other should take the papers with the "B" on the top. You're each going to draw two secret sprites on the chart, and your partner will try to figure out whether or not they are touching based on the same information that the computer will have about each sprite's properties, so don't let your partner see what you are drawing.

Distribute the activity guides to each set of partners. Ensure that one partner has taken Version A and the other has taken Version B.

Each student will have a line on which to draw two squares. The student chooses the location and the size of each of the squares, and then records the information about the squares in a table. They then switch tables (not drawings) and try to determine whether or not the two sprites are touching based on the width of each sprite and the distance between them.

The math to determine whether the sprites are touching is as follows:

1. Subtract the x (or y) positions of the sprites to find the distance between their centers.
2. Divide the width (or height) of each square by 2 to get the distance from the center to the edge.
3. If the distance between the centers of the sprites is greater than the sum of the distances from their centers to their edges, the sprites are not touching.
4. If the distance between the centers of the sprites is equal to the sum of the distances from their centers to their edges, the sprites are barely touching.
5. If the distance between the centers of the sprites is less than the sum of the distances from their centers to their edges, the sprites are overlapping.

Circulate: Support students as they complete the worksheet. If students are not sure how to determine whether the sprites are touching, encourage them to use one of the ideas on the board. Remind them that they are not being graded on whether they are right or wrong, but on their ability to use the problem solving process. If any students are finished early, challenge them to find a method that will work for sprites anywhere on the grid, not just on the same line.

Share: After students have all had a chance to test their solutions, ask them to share what they discovered.

Remarks

People can use a lot of different strategies to solve a problem like this. Because computers can't "see" the drawings in the same way that people can, they need to use math to figure out whether two things are touching. We looked at how this can work along a line, but we can combine these methods to work anywhere on the game screen.

isTouching()


Transition: Have the class log onto Unit 3 - Code Studio Levels for Lesson 16.

Display: Show bubble 2, and allow students to look in and discuss in pairs how they might code the behavior that they see in the program.

Discussion

Goal: The goal of this discussion is for students to see that the math in the code is the same math described in the unplugged activity. The code combines the tests for x and y in nested "if" statements.

Code Studio levels

- Sample Game 
- **Student Overview**


Frog Jump

View on Code Studio



Look at the frog jumping game to the left. It looks like the game from the last lesson, but the frog moves the mushroom if it hits it. What code do you think would help the computer to know whether two sprites are touching?


The purpose of this level is for students to see that they can test whether two sprites are touching with the blocks that are already available to them. The code is complicated, but it only uses blocks students have already learned to detect when two sprites are touching.


 **Discuss:** Ask the students how the code below relates to what they did in the unplugged activity.

Remarks

Even though this code works, it can be hard to read, and it would be easy to make a careless error when you're writing it. It would also take some time to write out every single time, so a programmer saved the code into one block that we can use to figure out whether two sprites are touching, and we don't need to write all of this code. Because this is something a programmer wants to do over and over again, someone has created a block called `isTouching()` that already has all of the math inside of it. Hiding the details of how something is done to make it easier to program is called **abstraction**.

Code Studio levels

- Collision Detection 
- **Teacher Overview**
- **Student Overview**

The code in this level is overwhelming. The point is not that students understand every line, but that they see that it's possible to check whether sprites are touching just by using their positions. **View on Code Studio** 

They should understand that the `isTouching` block used in the next level automatically runs the complicated code that they see here, but that it's hidden inside the block to make programming easier.

Balloon

[View on Code Studio](#)

The code below uses the sprites' x and y positions to check whether they are touching. It will change the balloon sprite's animation when the tack touches it. Use the arrow keys to move the tack until it touches the balloon.

Do This

- You do not need to change any code on this level.
- Read the if statements inside the draw loop and find the different sprite properties and how they are compared.
- Discuss the code with your partner. Would you want to write this code every time you checked whether sprites were touching?

- Levels
- 4
- 5
- 6
- 7

Student Instructions

[View on Code Studio](#)

isTouching()

Writing out the math each time you want to check whether two sprites are touching can take a while, so a programmer created the `isTouching` block, which can check whether one sprite is touching another sprite (the **target**). The computer is still doing the same math as in the previous program, but you don't have to worry about it because another programmer already did that work.

Do This

Inside the draw loop, drag the `isTouching` block into the if block. (**Show me where**)

Hint: Don't forget to change the "sprite" to "balloon" and the "target" to "tack".

Student Instructions

[View on Code Studio](#)

Applesauce

When the apple hits the blender, the blender should turn on.

Do This

Use the `isTouching` block to make the blender shake back and forth when the apple sprite touches the blender sprite. The shaking motion is already coded using the random block, so you just have to check when the two sprites are touching.

Challenge: Can you make the apple disappear when it touches the blender?

Student Instructions

[View on Code Studio](#)

Making Sounds

You can also use code to play a blender sound.

Do This

Use the `playSound` block from the "World" drawer to play the "https://docs.code.org/sounds/blender.mp3" sound when the apple touches the blender. You will need to paste the address of the sound into the block, so it looks like this:

```
playSound(▼ "https://docs.code.org/sounds/blender.mp3");
```

Student Instructions

[View on Code Studio](#)



Rainbow Horse

When the rainbow touches the horse, it should turn into a unicorn.

Do This

Use the `if`, `isTouching`, and `setAnimation` blocks to change the horse sprite's image when the rainbow touches it. The unicorn image is already loaded in the animations tab for you.

Assessment

Key Concepts:

Create and modify objects (sprites) to manage the complexity of on-screen elements.

Assessment Criteria:

- ▶ Extensive Evidence
- ▶ Convincing Evidence
- ▶ Limited Evidence
- ▶ No Evidence

Code Studio levels

- Levels
- 8
- 9
- 10

Student Instructions

[View on Code Studio](#)



Debugging Collisions

The balloon is popping before the tack touches it. When sprites aren't doing what you expect, you can use the `debug` block to get more information about why the sprites are behaving that way. Can you find out what's wrong in the code below?

Do This

- Run the code and use the arrow keys to move the tack to pop the balloon.
- In the code below, change `balloon.debug = false` to `balloon.debug = true`.
- Add a new `debug` block to the code and set the tack sprite's debug property to true.
- Run the code again, then discuss with your partner why the balloon is popping early.

Challenge: Can you use the animations tab to resize the balloon picture so it pops at the correct time?

Student Instructions

[View on Code Studio](#)



Scoreboard

You can also use `isTouching` to decide whether you should increase the score. In this game, the score is stored inside the 'score' variable. It is displayed on the screen using the `text` block. (**Show me where**)

Do This

- Use the `if` and `isTouching` blocks to determine whether the bunny has caught the carrot. (**Show me where**)
- If it has, do the following three things:
- Use the counter pattern on the score variable to increase the score.
- Reset the carrot's x position off the right-hand side of the screen.
- Set the carrot's y position to a random number between 10 and 390.

Student Instructions

[View on Code Studio](#)



Improve Your Game

Now that you know how to use `isTouching` and `playSound` , you can make some fun things happen when your sprites run into each other.

Note: The `playSound` block now has an extra parameter called "**loop**" that can be set to true or false. If this parameter is true, the sound will continue to play in a loop.

Do This

- Add at least two effects when your sprites collide.

Challenge: Add a scoreboard to the top of your screen.

- Collision Detection Review
- **Student Overview**

[View on Code Studio](#)



Wrap Up (5 min)

- ✔ **Prompt:** At the beginning of the lesson, you saw that it's possible to do everything that the `isTouching` block does without using the block at all. What makes this block useful?

Remarks

One of the great things about programming, is that once you've figured out a solution to a problem, you can often program it into the computer to be used over and over again. When you don't have to worry about the details of that particular problem, you can take on bigger challenges, as you did today.

Prompt: What were some different things that your sprites did when they interacted? What is one type of interaction you'd like, but you haven't seen yet? Out of those types of interactions, are there any that you think are useful enough that you would make a block for them?

Share: Once students have finished journaling, allow them to share out the different type of interactions they'd like to see. Let them know that you will be learning more ways for sprites to interact later in the unit.

✔ Assessment Opportunity

Students should note that even though they could program the code to detect whether sprites are touching each time, it is error prone and would take up more time and energy than having a block that performs the same code automatically. The new block helps them to take on more difficult challenges by reducing the risk of errors and freeing them to think about the bigger picture.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 17: Complex Sprite Movement

Game Lab

Overview

Students learn to combine the velocity properties of sprites with the counter pattern to create more complex sprite movement. In particular students will learn how to simulate gravity, make a sprite jump, and allow a sprite to float left or right. In the final levels of the Code Studio progression students combine these movements to animate and control a single sprite and build a simple game in which a character flies around and collects a coin. Students are encouraged to make their own additions to the game in the final level.

Purpose

This lesson does not introduce any new blocks and in fact only uses patterns students have seen in Chapter 1. Instead it demonstrates how combining these tools, in particular the abstractions students learned in the previous two lessons, allows them to build new behaviors for their sprites. This highlights the broader point that abstractions not only simplify code, but also can themselves be used as building blocks of even more complex behavior.

The lesson features some of the most challenging programming in the unit. Students will be combining multiple programming constructs, including the velocity properties, the counter pattern, user interaction, and collision detection. The patterns students use in this lesson are generally useful for building games and students can and will reuse them later in the unit. In particular students will return to their flyer game from this lesson in the following one, and many of the mechanics students learn in this game reappear in lesson 17.

Assessment Opportunities

1. **Use sprite velocity with the counter pattern to create different types of sprite movement**

See Level 7 in Code Studio.

2. **Explain how individual programming constructs can be combined to create more complex behavior**

In the wrap up discussions, check that students can explain how they are creating new types of movement without new blocks.

Agenda

Warm Up (10 mins)

Activity (60 mins)

Wrap Up (10 mins)

[View on Code Studio](#)

Objectives

Students will be able to:

- Use sprite velocity with the counter pattern to create different types of sprite movement
- Explain how individual programming constructs can be combined to create more complex behavior

Preparation

Teaching Guide

Warm Up (10 mins)

Review: On the board write up the four major concepts that students will be combining in today's lesson, `isTouching()`, `keyDown()`, `sprite.velocityX / sprite.velocityY`, and the "counter pattern". Ask students to discuss with a neighbor and remind one another what each of these four constructs is used for and how they work. As a class define each based on these discussions.

Remarks

In the last several lessons we've learned a lot of powerful programming constructs that have let us make much more interesting games. Today we're going to explore how combining these together will give us even more control over the types of games we can make.

Activity (60 mins)

Transition: Move students to Code Studio. With the exception of the discussion after the first level students will be working in Game Lab until the end of the lesson.

Code Studio levels

- Velocity and the Counter Pattern 
- **Teacher Overview**
- **Student Overview**

[View on Code Studio](#)



Velocity and the Counter Pattern

Teaching Tip

This level introduces the primary new programming pattern of this lesson, combining the counter pattern with sprites' velocity properties. Encourage students to take seriously their predictions before actually running the code.

[View on Code Studio](#)



Using the counter pattern with a sprite's x and y property makes a sprite move smoothly across the screen. In this program **the counter pattern is being used with the** `sprite.velocityX` **property** instead.

Predict

What do you think will happen when the code is run? Why? Once you're ready you can run the code to find out.

Discuss: After students have made their predictions about how the code in the first level has run quickly discuss why they saw the car start to move more quickly. Note that whereas before they used the counter pattern to increase a car's position, now it is being used to increase the car's velocity.

Levels 3 - 5: These levels give students practice using velocity inside of the counter pattern.

Code Studio levels

- Levels
-  3
-  4

-  5

Student Instructions

[View on Code Studio](#)



Velocity and the Counter Pattern

As you just saw, using a `sprite.velocityX` property with the counter pattern will change a sprite's velocity during the program. This makes the sprite speed up. Do a little practice using this pattern yourself.

Do This

This program already makes a car move across the screen, but it's going very slowly.

- Use the counter pattern with the sprite's `velocityX` property to make the car speed up. (**Show me where**)

Student Instructions

[View on Code Studio](#)



Falling Rock

The rock should speed up as it falls down the screen. Can you use the same counter pattern with `velocityY` inside the draw loop to make the rock go faster and faster as it falls?

Do This

- Use the counter pattern with the sprite's `y` velocity to make the rock speed up as it falls. (**Show me where**)

Challenge: Can you make the rock spin as it falls?

Student Instructions

[View on Code Studio](#)



Rising Bubble




This program makes a bubble rise up the water. Can you make it get faster as it rises?

Do This

- Use the counter pattern and the sprite's `y` velocity to make the bubble move up more quickly.

Levels 6 - 7: These levels introduce using velocity and the counter pattern to slow down a sprite, eventually moving it in the opposite direction. This leads to introducing how this pattern could be used to simulate gravity.

Code Studio levels

- Levels
-  6
-   7

Student Instructions

[View on Code Studio](#)



Slowing Things Down

Now that you've had some practice speeding things up, can you use the counter pattern to slow sprites down?

Do This

The car is going to run into the water! You'll need to use the counter pattern to slow it down.

- Use the `sprite.velocityX` block with a counter pattern to slow the car down by 0.25 as it moves across the screen.
- Discuss with your Partner: What do you think will happen when the car finally stops?

Challenge: Add code that makes the car slow down only if his velocityX is greater than 0.

Student Instructions

[View on Code Studio](#)



Simulating Gravity

In the last level you slowed down the car with the `sprite.velocityX` block and the counter pattern. It almost looked like the car was getting pulled to the left.

If you use this same pattern with the `sprite.velocityY` block it will look like your sprite is always being pulled down, which is exactly what gravity does!




Do This

The rock is thrown in the air but it never falls back down.

- Use the `sprite.velocityY` block with the counter pattern to make the rock slow down and then fall in the other direction.
- Experiment with different values in your counter pattern. Do you want the rock to slow down quickly or gradually? What looks most realistic to you?
- Discuss with your partner: Why are you setting the rock's initial velocity outside the draw loop? Why are you changing the sprite's velocity inside the draw loop?

Levels 8 - 10: Students begin working on a flyer game. In these levels they use the programming constructs they've learned to make their main character move. The character responds to simulated gravity, jumps, and floats left and right.

Code Studio levels

- Levels
-  8
-  9
-  10

Student Instructions

[View on Code Studio](#)



Jumping

Increasing a sprite's y velocity inside the counter pattern can simulate gravity. By adding user interactions you can make your sprite appear to jump as well. For starters you'll make a simple jump, and then make it more realistic looking in the next level.

Do This

A sprite has already been created for you that falls because its y velocity is increased inside the draw loop. You'll need to make this sprite appear to jump.

✓ Assessment

Key Concepts:

Model complex movement on a coordinate plane

Assessment Criteria:

- ▶ Extensive Evidence
- ▶ Convincing Evidence
- ▶ Limited Evidence
- ▶ No Evidence

- Inside the `if` block that checks whether the up arrow has been pressed, set the sprite's y velocity to -5. (**Show me where**)
- Discuss with a neighbor: Why does this code run the way it does? How would using a number besides -5 affect the way the code works? How could you jump higher or lower?

Student Instructions

[View on Code Studio](#)



Floating Right

You're now using the counter pattern with the sprite's Y velocity to simulate gravity and jumping. If you use the sprite's X velocity in the counter pattern then you can make your sprite float from side to side as well.

Do This

In this level you'll make your sprite start floating to the right when the right arrow is pressed.

- Add an `if` statement inside your draw loop below the one you created for the "up" arrow.
- Use the `keyDown` block to make the `if` statement respond to when the "right" arrow is pressed.
- Inside the `if` block use the counter pattern with the `sprite.velocityX` block to add 0.1 to the sprite's X velocity.

Run your code to see how it works. The sprite should start floating to the right when you press the right arrow and jump when you press "up". You'll make the left arrow work in the next level.

Student Instructions

[View on Code Studio](#)



Floating Left

In the last level you got detailed instructions on how to make your sprite start floating to the right. This time you'll need to make your sprite float to the left on your own. You should be pretty comfortable with using velocity and the counter pattern together at this point. If you're having trouble, talk to a neighbor or review some of the past levels.




Do This

- Add code to your draw loop that will make the sprite start moving to the left when the "left" arrow is down.
- Make sure you're using velocity and the counter pattern together.

Once your code is working share what you wrote with a partner. Is your sprite easy to control? Does changing the amount you add or subtract in the counter patterns you wrote affect the way the game feels? What kind of game might be fun to make with a player that moves like this?

Levels 11 - 13: Students add a coin to the game for their character to collect. In the last level they are encouraged to update the game themselves. Use this opportunity in particular to encourage students to use other programming patterns they've learned in this unit, e.g. creating a scoreboard.

Code Studio levels

- Levels
-  11
-  12
-  13

Student Instructions

[View on Code Studio](#)



Add a Coin

In the next few levels you'll add to your program to make a simple game. In this game the player will collect points to increase the score. This is a good chance to see how different kinds of movement can affect the way a game feels, and it will also just help you practice programming skills.

Do This

In this level you'll just be adding a new coin sprite to the game. You should be working at the top of your program, outside the draw loop.

- Use the `createSprite()` block to create a new sprite. Make sure to give it a descriptive name such as `coin`.
- Use the `sprite.x` and `sprite.y` properties of the sprite to give it a random X and Y position between 0 and 400.
- In the Animation Tab there is already a coin animation. Use the `sprite.setAnimation()` block to give your sprite this animation.

Test your code before moving on. When you run the game, you should see a coin sprite appear somewhere randomly on the screen.

Student Instructions

[View on Code Studio](#)



Reset Coin

When your character touches the coin you should reset it somewhere on the screen.

Do This

- Place an `if` block inside of your draw loop.
- Use the `sprite.isTouching()` block as the condition to detect when the character touches the coin.
- Inside the `if` block write code that sets the coin's X and Y position to random numbers between 0 and 400.
 - **Hint: You've already written this code elsewhere in your program.**

Test your code before moving on. When your player touches the coin, it should move somewhere else on the screen.

Student Instructions

[View on Code Studio](#)



Make It Your Own

You now have the basic mechanics of your game in place, so it's time to make it your own. What do you want to happen? Should the character get points every time it collects a coin? Can you add a scoreboard like you learned in the last lesson? Do you want to make another coin? What about a "bad coin" that takes away points?

Do This

Make at least one improvement to the game that makes it your own. Be prepared to share your changes and improvements with your classmate.

- Velocity and the Counter Pattern Review 
- **Student Overview**

[View on Code Studio](#)



Wrap Up (10 mins)

Share: Have students share with their classmates what additions they made to their final flyer game. Have students focus not just on how the game works, but what the code to create that kind of functionality looks like.

🗣️ **Prompt:** On your paper make two lists. First make a list of new things you can program to do after today's lesson. On the second list write down all the new blocks you learned today.

Discuss: Have students share their lists with classmates. Afterwards share lists as a class. They should hopefully have listed many new sprite movements but students haven't actually learned any new blocks in this lesson.

✔️ **Prompt:** Today we built lots of new sprite movements like gravity and jumping, but none of this required us to learn new blocks. How were you able to do new things without learning any new blocks?

Discuss: Lead a quick follow-up to your initial discussion about this point.

🗣️ **Remarks**

We're going to keep learning a few more tools in Game Lab, but as we do remember what we saw today. To create new kinds of programs you don't always need to learn new blocks. Most of the time the creativity of programming comes from learning to combine things you already know in new and creative ways.

Discussion

Goal: This conversation should highlight that students did not learn any new blocks in today's lesson, they just learned new ways to combine blocks and patterns they had learned previously. The broader point here is that programming is not always about learning new blocks, but being creative about combining the tools you already know how to use in the language.

✔️ Assessment Opportunity

Check that students can explain the new programming structures and algorithms that they were able to use to get the new behaviors in the program.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

► **AP** - Algorithms & Programming



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 18: Collisions

Game Lab

Overview

Students program their sprites to interact in new ways. After a brief review of how they used the `isTouching` block, students brainstorm other ways that two sprites could interact. They then use `isTouching` to make one sprite push another across the screen before practicing with the four collision blocks (`collide` , `displace` , `bounce` , and `bounceOff`).

Purpose

This lesson introduces collisions, another useful abstraction that will allow students to manipulate their sprites in entirely new ways. While students could theoretically have written their own `displace`, `collide`, or `bounce` commands, the ability to ignore the details of this code allows them to focus their attention on the high level structure of the games they want to build.

This lesson is also intended to give students more practice using the new commands they have learned in the second chapter. It is actually the last time they will learn a new sprite behavior, and following this lesson students will transition to focusing more on how they organize their increasingly complex code.

Assessment Opportunities

1. Model different types of interactions between sprites.

See Level 16 in Code Studio.

2. Describe how abstractions can be built upon to develop even further abstractions

In the discussion between Levels 4 and 5 in Code studio, make sure students talk about the importance of higher level blocks, such as `isTouching` , and that while these blocks don't provide new functionality, hiding the complexity of the code inside of a single block allows them to tackle more complex problems.

Agenda

Warm Up

Activity

Code Studio Levels

Wrap Up (10 mins)

Share Out and Journal 3-2-1

[View on Code Studio](#)

Objectives

Students will be able to:

- Model different types of interactions between sprites.
- Describe how abstractions can be built upon to develop even further abstractions

Vocabulary

- **Abstraction** - a simplified representation of something more complex. Abstractions allow you to hide details to help you manage complexity, focus on relevant concepts, and reason about problems at a higher level.

Introduced Code

- `sprite.bounce(target)`
- `sprite.bounceOff(target)`
- `sprite.collide(target)`
- `sprite.displace(target)`
- `setCollider(type, xOffset, yOffset, width/radius, height, rotationOffset)`
- `sprite.bounciness`

Teaching Guide

Warm Up


Display: If you have the ability, project the animation on the first level in Code Studio for this lesson. Otherwise ask pairs of students to look at it together. It shows two sprites, one moving across the screen towards the other and eventually pushing the one when they collide.

Discussion

Goal: The goal of this discussion is for students to brainstorm ways to solve the problem of having one sprite push another across the screen. There's no need for students to come to a consensus, because they will each have a chance to try out a solution in the next level in Code Studio. Students should understand that it is possible to use blocks to produce the desired movement just with the blocks that they have already learned.

Code Studio levels

- Lesson Overview 
- **Student Overview**

Prompt: Using the blocks we already know how to use, how could we create the sprite interaction **View on Code Studio** we can see in this program? 

Remarks

We have a lot of great ideas for how we might make one sprite push another across the screen. Now that you've prepared, you can try out your ideas in Code Studio. One big part of the problem is figuring out when the two sprites are touching, but because we already figured out how to do that and can now use the `isTouching` block, we don't need to think about it anymore. We can focus on the new part of the problem.

Activity

Code Studio Levels

Levels 2-4: In these levels students are shown a sprite interaction. They then implement their ideas for creating the sprite interaction they observed. The first time they can implement the ideas from the group. The second time challenge them to implement that behavior independently.

Code Studio levels

- Sprite Interactions 
- **Student Overview**

Sprite Interactions

View on Code Studio



So far you've been able to create simple sprite interactions by using the `sprite.isTouching()` block. For example, you've reset a coin to a different location on the screen when a character touches it. Now it's time to start making sprites have more complex interactions.

Do This

- Run the program and observe the interaction between the two sprites.

- Discuss with a neighbor: Using only the commands you already know how could you create this kind of interaction?
There are many ways to do it, but here are some blocks to consider:
 - `sprite.isTouching()`
 - `sprite.velocityX`
 - `sprite.velocityY`
 - `sprite.x`
 - `sprite.y`

Be ready to share your ideas with your classmates.

- Levels
-  3
-  4

Student Instructions

[View on Code Studio](#)



Program a Sprite Interaction

You should have discussed with your classmates how you could create the sprite interaction you saw in the last level. Now it's your turn to program it yourself. How can you make the giraffe move the monkey off the screen?

Do This

The giraffe is already moving across the screen toward the monkey but the sprite interaction itself hasn't been programmed.

- Use the plan you developed with your classmates on the last level to program the sprite interaction yourself.

Student Instructions

[View on Code Studio](#)



Write Your Own Sprite Interaction

In the last level you should have written code for a sprite interaction that you developed with your classmates. This time try to write the program on your own, but you can use the patterns you saw in the last level.

Do This

The elephant should **push the hippo off the screen**. Notice that the elephant moves at a random Y velocity each time the program runs.

- Using the patterns from the last level, write code that makes the elephant push the hippo off the screen.

Prompt: This was a challenging problem, but we were able to solve it. What helped us to solve this problem?

Remarks

All of these things are very important, and they come up in Computer Science a lot. One thing that was particularly helpful was the `isTouching` block, which hid the complicated code that tells us whether the two sprites are touching. There's also a `displace` block that hides the code we just wrote, and some other blocks that hide the code for some other types of sprite interactions. You'll have a chance to try out these blocks in the next few levels, and use them to improve your flying game.

Levels 5-11: These levels introduce how to use the 4 new collision blocks students will learn in this lesson.










✓ Assessment Opportunity

Make sure students talk about the importance of higher level blocks, such as `isTouching`, and that while these blocks don't provide new functionality, hiding the complexity of the code inside of a single block allows them to tackle more complex problems.

This is also a good time to call out how far the students have progressed in their skills since the beginning of the unit. This problem would have seemed almost impossible at the beginning of the year. Some things that made the problem easier to solve were:

- Preparation: The students brainstormed and thought about solutions before trying out their code.
- Cooperation: Students worked as a group to come up with a solution
- Abstraction: Students were able to use the `isTouching` and `velocityY` blocks to hide part of the solution's complexity.

Code Studio levels

- Levels
-  5
-  6
-  7
-  8
-  9
-  10
-  11

Student Instructions

[View on Code Studio](#)



Displace

The interaction you've been programming is so common that there's a block designed to do the interaction for you. `sprite.displace()` that will make one sprite push the other when they touch. The code underlying this block might look a lot like what you just wrote, but now you no longer need to worry about writing those details yourself.

Do This

Someone tried to use the `sprite.displace()` block to make the **elephant push the hippo**, but there is a bug. Can you change the code so that the elephant pushes the hippo off the screen?

- Find the line of code where the `sprite.displace()` block is used and fix the error.

Student Instructions

[View on Code Studio](#)



More Collision Blocks

Three new types of sprite interactions have been added to the toolbox, `sprite.collide()`, `sprite.bounce()`, and `sprite.bounceOff()`. How do you think they'll affect the sprites?

Do This

- Switch out the `displace` block for the `sprite.collide()` , `sprite.bounce()` , and `sprite.bounceOff()` blocks. (**Show me where**)
 - **Hint: If you're having trouble doing this with blocks then switch over to text mode.**
- Discuss with a neighbor: What is the difference between the four different sprite interactions? What do you think the purpose of each block is?

Student Instructions

[View on Code Studio](#)



Collision Types

There are four types of collisions that we use in Game Lab: `displace` , `collide` , `bounce` , and `bounceOff` . These blocks will cause a certain type of interaction between the **sprite** and its **target**.

Do This

Choose the best block to model the basketball bouncing off the floor. (**Show me where**) *Hint: You can try the different blocks out, or read more about them in the "Help and Tips" tab.

Student Instructions

[View on Code Studio](#)



Debugging Sprite Interactions

Sprite interactions just run some code when they're called. The interactions are not "remembered" by the game. If you want one sprite to bounce or collide with another then it needs to be a part of the draw loop. If you forget then this can lead to unexpected behavior.

Do This

The turtle can be moved with the arrow keys. It's not supposed to be able to walk through the tree, but something is wrong in the code. Can you find and correct the bug in the code?

- Run the code and try to make the turtle collide with the tree.
- Look through the code and discuss with your partner what the problem is.
- Correct the code, then run it again to make sure it works.

Student Instructions

[View on Code Studio](#)



Debug

Sometimes sprites will behave in ways that are unexpected. There is a `sprite.debug` property you can use to better understand why the sprites interact the way that they do.

Do This

These two coins are round, so you would expect them to bounce in a certain way. Something weird is happening though!

- Run the code and watch the way that the coins interact.
- Use the `sprite.debug` block to make `debug 'true'` for both the sprites and run the code again.
- Change the gold coin's starting x position to 51 and run the code again.
- Discuss with a partner: Why do you think the coins are bouncing strangely?



setCollider

Sprites interact based on the size and shape of their collider, not the images that are assigned to them. You can only see the collider when debug mode is turned on. You can change the shape of the collider using the `sprite.setCollider()` block, which lets you pick between a "rectangle" or a "circle". By default all colliders are "rectangle".

Do This

- Find the `sprite.setCollider()` block for the gold coin, and change it from "rectangle" to "circle".
- Add a new `sprite.setCollider()` block for the silver coin, and choose "circle" for the shape of the collider.
- Run the code again to see how the sprites bounce.



Bounciness







So far, `bounceOff` has made sprites bounce away from other objects as fast as they bounced into them. In the real world, almost everything slows down just a little bit when it bounces off something else. You can use the `bounciness` block to tell your sprite how much to slow down or speed up when it bounces off something else.

Do This

- Read the code below and press "Run" to see the behavior of the basketball and pool ball.
- Use a `bounciness` block to set the bounciness of your soccer ball.
- Run the code again to see how the sprites bounce off the floor.

Levels 12-15: These levels guide students through adding collisions to the flyer game they started in the previous lesson, eventually inviting students to add their own modifications to the game.

Code Studio levels

- Levels
-  12
-  13
-  14
-  15
-   16



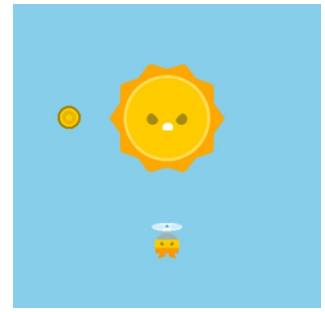
Flyer Game - Add Obstacle

This is the flyer game you built in the last lesson. For the next several levels, you'll be adding an obstacle sprite to the game, using some of the sprite interactions you just learned. At the end you'll have a chance to keep adding on ideas of your own.

Do This

Add an obstacle sprite to the game. You can use whatever image you like from the animation tab but the example shown here uses a sun. Right now you just need to add the sprite to your game and give it an animation.

- Add a new sprite to your game called "obstacle".
- In the animation tab create a new animation for your obstacle. In the example a sun image was chosen.
- Use the `sprite.setAnimation()` block to give your sprite the image you chose.
- Run the code and make sure the sprite appears where you want it on the screen. You may need to set its X, Y, and scale properties to get it to look the way you want.



Student Instructions

[View on Code Studio](#)



Flyer Game - Interacting with the Obstacle

You don't want your player to be able to move through the obstacle, so you'll need to use one of the sprite interactions.

Do This

- Add code to your game that prevents your player from moving through the obstacle.
- If you use one of the bounce interactions, decide whether you want to reset the bounciness of your character.
- Discuss with a neighbor: Which sprite interaction did you decide to use? Is there more than one sprite interaction that works the way you'd expect?

Student Instructions

[View on Code Studio](#)



Flyer Game - Coin Behind the Obstacle

Right now your coin is moving to random locations. That means sometimes it even will appear behind your obstacle, so your character can't get to it. Using sprite interactions you can fix this problem.

Do This

- Add code to your game that prevents the coin from moving behind the obstacle. Don't be afraid to try out ideas just to see how they work.
- Discuss with a neighbor: Which sprite interaction did you decide to use? Is there more than one sprite interaction that works the way you'd expect?

Student Instructions

[View on Code Studio](#)



Flyer Game - Change Colliders

Right now your colliders are all rectangular. Switch them over to circles to get more interesting and realistic bounces and collisions.

Do This

- Use the `sprite.setCollider()` block to change the colliders of your sprites to circles.
- Set your sprites' debug properties to true to make sure your game is working the way you want.
- Play your game to make sure it's working the way you want.

Student Instructions

[View on Code Studio](#)



Flyer Game - Make It Your Own

Time to make this game your own by using what you've learned about sprite interactions.

Do This

Add at least one more aspect to your game that uses sprite interactions. There's some ideas below or you can choose to add features of your own. Make sure you're ready to share your ideas with your classmates.

- Create invisible sprites at the edge of your game to keep your character from bouncing out.
- Add platforms to the game for your character to navigate around.
- Add another obstacle to your game.
- Create another idea of your own.

- Sprite Interactions Review 📖
- **Student Overview**

- Levels
- 🖥️ Extra

Key Concepts:

Model complex movement on a coordinate plane.

Assessment Criteria:

- ▶ Extensive Evidence
- ▶ Convincing Evidence
- ▶ Limited Evidence
- ▶ No Evidence

[View on Code Studio](#)



Student Instructions

[View on Code Studio](#)



Free Play

Use what you've learned to create whatever you like. When you're finished, you can click [Share](#) to send your creation to a friend, or [Remix](#) to send it to your Projects Gallery.

Wrap Up (10 mins)

Share Out and Journal 3-2-1

Share: Allow students time to play each other's flying games. Ask them to focus not just on the new behavior that they added but also the code they used to create it.

Journal: Have students write and reflect about the following prompts.

- What are three things you saw in someone else's game that you really liked?
- What are two improvements you'd make to your game if you had more time?
- What's one block you'd like to have in Game Lab, and how would it work?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

► **AP** - Algorithms & Programming



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 19: Functions

Game Lab

Overview

Students learn how to create functions to organize their code, make it more readable, and remove repeated blocks of code. An unplugged warmup explores how directions at different levels of detail can be useful depending on context. Students learn that higher level or more abstract steps make it easier to understand and reason about steps. Afterwards students learn to create functions in Game Lab. They will use functions to remove long blocks of code from their draw loop and to replace repeated pieces of code with a single function. At the end of the lesson students use these skills to organize and add functionality to the final version of their side scroller game.

Purpose

In the first four lessons of this chapter students have learned to use a number of abstractions in their programs, including the velocity properties, `isTouching`, and collisions. These abstractions have allowed them to build much more complex programs while ignoring the details of how that behavior is created. In this lesson students learn to build abstractions of their own by creating functions.

Students will primarily use functions to break code into logical chunks that are easier to reason about. This foreshadows the chapter's transition from building technical skill to the organizational processes used to develop software.

Assessment Opportunities

1. **Create and use functions for blocks of code that perform a single high-level task within a program**

See Level 12 in Code Studio.

2. **Explain the advantages of using functions in a program.**

In the wrap up discussion, make sure students understand that functions can increase both the readability and organization of code.

3. **Explain how abstractions allow programmers to reason about a program at a higher level**

In the wrap up, students should make the connection between functions and abstraction, that functions allow a programmer to name a bit of code so that they can think about it at that higher level, rather than worry about all the details.

[View on Code Studio](#)

Objectives

Students will be able to:

- Create and use functions for blocks of code that perform a single high-level task within a program
- Explain the advantages of using functions in a program.
- Explain how abstractions allow programmers to reason about a program at a higher level

Vocabulary

- **Function** - A named group of programming instructions. Functions are reusable abstractions that reduce the complexity of writing and maintaining programs.

Introduced Code

- Define a function
- Call a function

Agenda

Warm Up (10 mins)

High Level vs. Low Level Instructions

Activity (60 mins)


Wrap Up (10 mins)

Teaching Guide

Warm Up (10 mins)

High Level vs. Low Level Instructions

Prompt: Imagine you needed to write a 5-step set of instructions for going through your morning. What would they be? Write down your steps and be ready to share.

 **Discuss:** Student should write 5-step instructions for their morning and share with a neighbor. Ask a couple of students to share with the class.

Prompt: This set of instructions is pretty easy to follow and understand. They're at the level you might think about when describing your day to a friend. Now let's go a level deeper. Pick one of your 5 steps and split it into 5 smaller steps you need to complete that larger task. Be ready to share your ideas again.

Discuss: Students should share their smaller steps. Again ask some volunteers to subsequently share their original step and how they split it up.

Discuss: This is getting interesting. It seems like the first time we gave our steps we were "hiding" some of the details necessary to complete the task. Let's try this once more. Take one of your 5 smaller steps and split it up again into 5 even smaller steps.

Discuss: Ask students to share their steps one more time. Again ask some volunteers to share how they broke up one of their steps into even smaller steps.

Prompt: Imagine we had split every one of your first 5 steps into 5, and then split all of those steps again. This would mean we would have a high level set of instructions, and at the bottom a really low-level or detailed set of instructions. Be ready to respond to the following questions.

- When would the high level set of steps you just wrote be most appropriate?
- When would the lowest level set of steps be most appropriate?
- Which one is easiest to reason about or understand?

Discuss: Ask students to share their thoughts and opinions. After a couple of minutes move the conversational to the transitional comment below.

Remarks

Sometimes details are important, but often high level steps are much easier to reason about and make it clear what's happening. In programs the same thing is true. We've learned that blocks like velocityX or isTouching actually just contain code that we could've written ourselves. Using these commands, or abstractions, is really helpful since we can think about code at a high level. Today we're going to learn how to group lots of commands to create a single new block our own. In programming when we create a new block like this we call it a function.

Activity (60 mins)

Transition: Move students into Code Studio where they will learn to create an call functions.

Code Studio levels

Discussion

Goal: This conversations demonstrates to students that they often use a high level name or description for much more complex behavior. It is motivating the value of grouping or combining lots of smaller steps under a larger name and provides some of the justifications for creating functions within programs. Namely, high level steps make it easier to understand and reason about a process.

Content Corner

Breaking Down Processes: The order in which they are breaking down a larger task here also mirrors how they'll be asked to write code with functions. Often programmers first write the name of the function they intend to write based on what it should do before actually going in and writing the details.

Discussion Goals

[View on Code Studio](#)

Make sure students understand role of the two steps in using functions, as well as seeing functions as a form of "chunking" or abstraction. The function definition should include all of the code that they want to run, and the name of the function should be a short description of the purpose of the code. The function should be called at each place in the program that the student wants that block of code to run.

Functions

 3 4 5 6 7 8 9

(click tabs to see student view)

Collector Game

 10 11  12

(click tabs to see student view)

Functions Review

Student Overview

Levels

 Extra

(click tabs to see student view)

Wrap Up (10 mins)

- ✓ **Prompt:** Why would we say that functions allow us to "create our own blocks?" Why is this something we'd want to do?

Discuss: Have students discuss at their table before talking as a class.

- ✓ **Prompt:** Write down your own definition of an abstraction? Why would a function count as an abstraction?

Discuss: Have students discuss at their table before talking as a class.

Remarks

Functions are a useful tool for helping us write and organize more complex pieces of code. As we start looking to end of the unit and your final project, being able to keep your code organized will be an important skills.

✓ Assessment Opportunity

Goal: Use this first prompt to review what students learned today. When they create a function they are creating their own block that they can call or use whenever they like. They saw at least two primary motivations for creating functions today including.

- Simplifying code by breaking it into logically named chunks
- Avoiding repeated code by making one block you can use multiple times.

✓ Assessment Opportunity

Goal: Students should review the definition of abstraction as "a simple way of representing something complex". Note that a function is an abstraction because it allows you to create one simple name for a more complex block of code.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

► AP - Algorithms & Programming



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 20: The Game Design Process

Game Lab

Overview

This lesson introduces students to the process they will use to design games for the remainder of the unit. This process is centered around a project guide which asks students to define their sprites, variables, and functions before they begin programming their game. In this lesson students begin by playing a game on Game Lab where the code is hidden. They discuss what they think the sprites, variables, and functions would need to be to make the game. They are then given a completed project guide which shows one way to implement the game. Students are then walked through this process through a series of levels. As part of this lesson students also briefly learn to use multi-frame animations in Game Lab. At the end of the lesson students have an opportunity to make improvements to the game to make it their own.

Purpose

This lesson introduces multi-frame animations, and is the first in a sequence centered around the process of building software.

While previous lessons focused on using abstractions to manage the complexity of the code, this lesson focuses on managing the complexity of the software development process. The lesson heavily scaffolds the software development process by providing students a completed project guide, providing starter code, and walking students through its implementation. In the subsequent lessons students will need to complete a greater portion of this guide independently, and for the final project they will follow this process largely independently.

Assessment Opportunities

1. Implement different features of a program by following a structured project guide

In the Wrap Up discussion, make sure students are clear on how the project guide helps them to break their programming work into manageable chunks, and that they understand how listing the sprites and functions ahead of time can help them focus on specific bits of code rather than the entire program at once.

Agenda

Warm Up (15 mins)

Play Cake Defender

Stop: Review Project Guide

View on Code Studio

Objectives

Students will be able to:

- Implement different features of a program by following a structured project guide

Preparation

Print copies of the the project guide if you will be giving students physical copies. Please note that this project guide is intentionally filled out. (See notes in Lesson Plan.)

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- Defender Game - Project Guide

Make a Copy

Activity (60 mins)

Multiframe Animations

Implement Project Guide


Wrap Up (20 mins)

Using the Project Guide

Teaching Guide

Warm Up (15 mins)

Play Cake Defender


 **Transition:** This lesson begins immediately in Code Studio. On the first level they will be find a game but will not be able to see the code. They should play the game and follow the instructions which ask them to list the variables, sprites, and functions they think are necessary to create this game.



Teaching Tip

Keeping Focus: Students can easily get distracted by the fun of playing the game. Let them play for a while but eventually encourage them to follow the on-screen instructions and make a list of the variables, sprites, and functions that would be necessary to play the game.

Code Studio levels

- Sample Game 
- **Student Overview**

Defend Your Cake!

View on Code Studio




This is an example of a defender game that you'll build by the end of this lesson. To defend your cake, move the alien with arrow keys to block the lady bugs and push them into the water.


Do This

Turn to a classmate and make a list of the following information.

- How many sprites are there in this game. Which are they?
- What variables are needed to make this game? What do they store?
- If you were to split the code of this game into functions what do you think they would be? What are the major pieces of behavior you'd need to create in your code?

Stop: Review Project Guide

 **Discuss:** Students should have individually created a list of variables, sprites, and functions they would create to make the defender game they played. Ask students to share their lists with a neighbor before discussing as a class.

 **Distribute:** Give each student or pair of students a copy of the project guide.

Prompt: Compare the components of the game you thought would be included to the ones on this project guide. Do you notice any differences?

Discuss: As a class compare the list you had on the board to the list of variables, sprites, and functions on the project guide. Note the similarities. Where there are differences try to understand why. Don't approach one set as "right" vs. "wrong" but just confirm both would be able to make the game students played.

 **Remarks**

Discussion

Running the Conversation: You can write "Variables", "Sprites", and "Functions" on the board and record their ideas below each. Ask students to justify their decisions but don't feel the need to settle on one right answer.

There's usually lots of ways you can structure a program to get it to work the way you want. The important thing when writing complex or large programs is that you start with a plan. Today we're going to look at how we could implement this plan to build our own defender game. By the end of the lesson you'll not only have built your game, but you'll know how to change it and make it your own. Let's get going!

Teaching Tip

Project Guide: The project guide is intentionally filled out for students so that they can experience using it as a reference when programming. This should give them more context when filling out their own project guide in the next two lessons.

You can give each student their own copy for reference, but you might also choose to print one copy per pair, share digital copies, or just display the guide on the projector. So long as it is available for reference, any approach will work fine.


Activity (60 mins)

Multiframe Animations

Transition: Students should move back to Code Studio.

Before actually implementing the plan students will need to quickly review one new skill, how to use multiframe animations. Students will quickly learn how to create, modify, and rename animations as well as pick them from the library.

Code Studio levels

- Plan Your Project 
- **Student Overview**

Stop

[View on Code Studio](#)



Before you move on you'll need to look at the Project Guide for this project. Wait for instructions from your teacher as well.

- Levels
-  4
-  5

Student Instructions

[View on Code Studio](#)



Using Multiframe Animations

In the sample defender game the sprites themselves were animated. Before getting started on programming this game, take a minute to get familiar with this new way of animating sprites.

Do This

This program already includes several sprites but they don't yet have any animations.

- Go to the Animation Tab and check out the multi-frame animations already added to your project. Choose one for each of your characters.
- Remember you can use `setAnimation` to give your sprites animations you've created in the Animation Tab.

Student Instructions

[View on Code Studio](#)



Slow Down

Nice work! Time to start learning how to control these multiframe animations.

Do This

Your sprites should be animated but they're moving really quickly.

- Head back to the Animation Tab. Underneath each animation you should see a slider.
- Use these sliders to slow down your animations so they look more realistic.















Implement Project Guide

Students are given a large amount of starter code in this project. The sprites, variables, and functions have all already been given to them. The work of this project is writing the code for the individual functions. These levels guide students through how to implement those functions. As students move through the levels point out how the project guide is being used.

The most challenging skill students use in these levels is recognizing the need to create new functions to replace repeated code. Students need to build this skill on their own but these levels demonstrate an instance where this might happen.

Code Studio levels

- Levels
-  6
-  7
-  8
-  9
-  10
-  11
-  12
-  13
-  14
-  15
-  16
-  17

Student Instructions



[View on Code Studio](#)


Editing Multiframe Animations

Your sprites will look a lot more realistic if they turn around when they're moving. You can switch back and forth between multiframe animations when the user presses different keys.

Do This

Read this code and run the program. Make sure you know how the sprite responds to the arrow keys.

- In the Animation Tab, create a copy of the alien animation by clicking the following button: 
- Use the tool to flip your animation. Make sure you flip both frames using this button: 
- Rename your new animation.
- Use your new animation and old animation so that the alien faces the correct direction when moving. Where do you think you'll need to set the sprite's animation in your code?

Student Instructions

[View on Code Studio](#)


Getting Started: Set Animations

You should have already reviewed the planning guide for this project. A lot of the work to turn this project guide into a working game has already been started. Based on the project guide you're going to do the rest of this work.

Do This

Before we get started you'll want some better animations for each of your sprites.

- In the Animation Tab are animations for each of your sprites. Go look at what they are.
- In your code give each sprite its appropriate animation.**Use the ones provided for now** but later you'll be able to go change them.
- Head to **Level 4** if you need help remembering how to do this.

Student Instructions

[View on Code Studio](#)



Moving The Enemies

It's time to start writing the code that will move your sprites. To begin you'll need to get your enemy sprites to a random position and moving across the screen.

Do This

At the top of your program, after you create each enemy sprite, write code that will move it to the correct position and give it the correct velocity.

- Use `sprite.x` to set the x position to 0.
- Use `sprite.y` to set the y position should be a random number between 150 and 250.
- Use `sprite.velocityX` to set the x velocity to 2.

Test your program. Your enemy sprites should now be moving across the bridge.

Student Instructions

[View on Code Studio](#)



Touching the Cake

If the enemies get all the way across to the cake you should place them back at the left side of the screen and decrease the score. To start you'll **write code for only one of your enemies** .

Do This

Inside the `enemiesTouchCake` function you'll need to write code that checks when a ladybug is touching the cake, resets its position, and changes the score.

- Use an `if` and `isTouching` to detect whether `enemy1` has touched the cake.
- Inside your `if` block place code that:
 - sets `enemy1's` x position back to 0.
 - sets `enemy1's` y position to a random number between 150 and 250.
 - uses the counter pattern to decrease the score by 2.

(Hint: You can reuse some code you already wrote)

Test your code. One of your ladybugs should now reset when it gets across to the cake, and the score should go down by 2.

Student Instructions

[View on Code Studio](#)



Touching the Cake: Second

Ladybug

Your first enemy sprite should now be resetting when it gets to the cake. Now you'll want the other ladybug to reset as well.

Do This

Inside the `enemiesTouchCake` function you should have written code that resets `enemy1` .

- Copy the entire if-statement you wrote in the last level (Ctrl-C).
- Paste the code inside of the `enemiesTouchCake` function, just below the last one (Ctrl-V).
- Change the name of the sprite in that code from `enemy1` to `enemy2` .

Test your code. Now both bugs should reset when they touch the cake.

Student Instructions

[View on Code Studio](#)



Creating Functions

Your program now includes code in two places to set the enemies on the left side of the screen at a random y location. You can create functions to reset each of your two enemies to remove repetitions from your program. This will make your program easier to read, allow you to change it more easily, and allow you to quickly reset your sprites at other points in your program if you need to.

```
enemy1.x = 0;  
enemy1.y = randomNumber(150, 250);  
enemy1.velocityX = 2;
```

Do This

- At the bottom of your program create two new functions, `setEnemy1` and `setEnemy2` .
- Inside each of these functions place the code that sets the enemies on the left side of the screen and gives them a random y position.
- Wherever the code for `setEnemy1` and `setEnemy2` appears in your program replace them with a call to the functions you just created.

```
function setEnemy1() {  
  enemy1.x = 0;  
  enemy1.y = randomNumber(150, 250);  
  enemy1.velocityX = 2;  
}
```

Student Instructions

[View on Code Studio](#)



Moving Left and Right

Now that your enemy sprites are moving correctly, it's time to write the code to move your player. For now you'll just need to get your character moving left and right and changing its animations.

Do This

For this level you'll be writing code inside the `movePlayer` function.

- Use an `if` block along with `keyDown` to detect when the "right" arrow is pressed.
- Use `sprite.x` and the counter pattern increase the player's x position by 3.
- Use another `if` block to move the player to the left when the "left" arrow is pressed. This time you'll need to decrease the player's x position.

Test your game. Your character sprite should now move left and right when you press the left and right arrows.

Student Instructions

[View on Code Studio](#)



Moving Up and Down

You'll want your player sprite to move up and down as well.

Do This

For this level you'll still be writing code inside the `movePlayer` function.

- Use an `if` block along with `keyDown` to detect when the "up" arrow is pressed.
- Use `sprite.y` to make the player go up by 3 using the counter pattern.
- Use another `if` block to move the player down when the "down" arrow is pressed.

Test your code. Your character should now move in all 4 directions.

Student Instructions

[View on Code Studio](#)



Change Player Animations

Right now your player is always facing the same direction. You can make things look a lot more realistic by switching between animations. Your player should switch between a left-facing and right-facing animation depending on which key was last pressed. Remember, you can quickly copy and edit animations inside the Animation Tab.

Do This

- Inside the Animation Tab copy the animation of your player sprite.
- Flip each frame of the new animation so that the sprite is facing in the opposite direction.
- Rename your new animation.
- Use the `setAnimation` command inside the `movePlayer` function so that the player changes the direction it is facing when the "left" and "right" arrows are pressed.

Student Instructions

[View on Code Studio](#)



Displace Enemies

It's time to write code for some more sprite interactions. Your player sprite should displace the enemy sprites.

Do This

For this level you'll be writing code inside the `displaceEnemies` function.

- Write code that makes player displace both enemy sprites.
- Test your program to make sure your player is displacing enemies but they keep moving right after the player moves away.

Hint: You can use `sprite.debug` to see your sprites' colliders if you need to debug your program.

Student Instructions

[View on Code Studio](#)



Touching the Water

The last part of the game that you'll need to write is the code to reset the sprites when they touch the water. Luckily you should have already written functions that reset each sprite, so you'll just need a good way to know when either sprite

leaves the bridge. Start by writing the code for a single enemy and then copy-paste and make small changes to create code for your second enemy.

Do This

For this level you'll be writing code inside the `enemiesTouchWater` function.

- Use an `if` statement to check whether `enemy1` is off the top of the bridge by checking whether its `y` value is below 140. Within your `if` statement:
 - use your `setEnemy1` function to reset the sprite.
 - add 1 to the score.
- Use an `if` statement to check whether `enemy1` is off the bottom of the bridge by checking whether its `y` value is above 260. Within your `if` statement:
 - use your `setEnemy1` function to reset the sprite.
 - add 1 to the score.
- Test your program for the first enemy sprite. Make sure the sprite is resetting and the score goes up.
- Once it is working copy and paste the code you wrote to create the same behavior for `enemy2`. You'll need to change the name of the sprite and the name of the functions you use.

Student Instructions

[View on Code Studio](#)



Make It Your Own

You just walked through someone else's plan for creating a game, so now it's time to make it your own. What additional features or challenges do you want to create?

Do This

Select one of the challenges below to add to the game or come up with a challenge of your own.

- Change the visuals of the game so that your player, enemies, or cake look different.
- End the game when the enemies get to the cake and print the score. For an extra challenge end the game only after 3 enemies get through.
- Randomize the speed of the enemies.
- Create a new background that shows up when players reach a higher score.

Wrap Up (20 mins)

Using the Project Guide

✔ **Prompt:** Today, you used a filled-out project guide as you completed your program.

- How did the project guide help you as you coded?
- What do you think will be important to remember when you fill out your own project guide?

Teaching

This last level encourages students to make the game their own. If students have made their way to this point they have all the skills they need to progress through the curriculum, so there is no pressure to complete any of the modifications suggested in this level. If you have time, however, getting practice planning and implementing new features will be a useful skill. Even just modifying the animations of the game is an easy way students can make the game their own.

✓ Assessment Opportunity

As students answer the questions, make sure that they understand that the project guide is there to help them organize their work, so that they can look at smaller parts of the problem rather than thinking about it all at one time. They can use it to focus on one part of their program at a time as they code.

In the next lesson, as they fill out their own project guides, they will need to think carefully about the different parts of their program before they start coding. You may want to remind them of the "Prepare" part of the Problem Solving Process, and to think about how they might need to go back and tweak the project guide even after they have started coding.

Code Studio levels

- Levels

Share: Once students have completed the project they can share their work with their classmates. Encourage students to showcase the additional code they wrote and explain how it has changed the way the game works.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ AP - Algorithms & Programming



This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 21: Using the Game Design Process

Game Lab

Overview

In this multi-day lesson, students use the problem solving process from Unit 1 to create a platform jumper game. They start by looking at an example of a platform jumper, then define what their games will look like. Next, they use a structured process to plan the backgrounds, variables, sprites, and functions they will need to implement their game. After writing the code for the game, students will reflect on how the game could be improved, and implement those changes.

Purpose

Students have already learned all of the programming constructs that they need to make a game. This lesson reviews many of those concepts while introducing them to a structured process that will help them to manage the work. It builds on the use of the Project Guide in the previous lesson by having students complete more of this project guide independently before using it to build a game. This activity prepares students to write their own game from scratch for the final project.

Assessment Opportunities

1. **Identify core programming constructs necessary to build different components of a game**
- In the project guide, check that the student has identified key functions, sprites and variables needed in the program, and that the general description of the program is accurate.
2. **Implement different features of a program by following a structured project guide**
- See Level 20 in Code Studio.

Agenda


- Warm Up
- Activity
- Play Alien Jumper
- Discuss Project Guide
- Share out
- Wrap Up
- Journal

View on Code Studio

Objectives

- Students will be able to:
- Identify core programming constructs necessary to build different components of a game
 - Implement different features of a program by following a structured project guide

Preparation

 Print one copy of the project guide for each student or pair of students

Links


Heads Up! Please make a copy of any documents you plan to share with students.

- For the Teacher
- Planning Your Platform Game - Exemplar
- For the Students
- Planning Your Platform Game - Project Guide
- Make a Copy

Teaching Guide

Warm Up

Prompt: The Problem Solving Process helps us work through all kinds of problems. Think about the problem of building a larger piece of software, like the game we built in the last lesson. What did each of the 4 steps look like? Why were they important?

 **Discuss:** Students should brainstorm quietly and write down what each step might be. Afterwards, lead a share out discussion. You can record ideas on the board. Possible parts of each step include:

- Define: Figuring out what you want the game to look like, how it should work, who will play it.
- Prepare: Plan ahead what your code will look like. Decide on a structure for your game.
- Try: Write the code following your plan.
- Reflect: Test your code, play the game to make sure it works, get feedback from other people to make the game better.

Discussion

Goal: Students should share their thoughts but if it doesn't come up naturally then suggest the examples provided. This discussion will motivate the use of the project guide for building a game later in the lesson.

Remarks

When you build software, the problem solving process can be a helpful guide. Obviously we need to write the code, but being careful to define what you want to build, making a good plan to build it, and reflecting afterwards on how to improve it are all part of making good software. Today we're going to use this process to make a new game.


Activity

Play Alien Jumper

Distribute: Give each student a copy of the project guide.

Remarks

We're going to be building a jumper game today. You'll have a chance to play a sample game, then plan out how you would create the game on your Project Guide.

 Move students to Code Studio. On the first level they will be find a game but will not be able to see the code. They should play the game and follow the instructions which ask them to list the variables, sprites, and functions they think are necessary to create this game.

Code Studio levels

- Sample Platform Jumper Game 
- **Student Overview**

Platform Jumper

[View on Code Studio](#)



The game on the left is an example of a platform jumper. Press "Run" to play it. You can make the alien jump with the up arrow, and move it to the left and right with the arrow keys. You score by collecting stars, and if you score high enough, the background will change.

You already know how to use all the blocks you need to make a game just like this one, and you'll be making your own platform jumper in this lesson.

Discuss Project Guide

Circulate: Students should complete the project guide in the style of the one they saw in the previous lesson. They will likely want to keep the game up as they try to determine the behavior each of the sprites will have.

Share: Students share out their plans for making the game. Reassure them that there are many correct ways to make the same piece of software, and that they will have a chance to try out their ideas in code studio.

💡 Teaching Tip

Making the game will take at least two class periods. If there's not enough time for all students to finish the lesson, groups of students can work to code different aspects, then share their code with each other. For example, one group could work on the platforms, one on the stars, and another on the player. Student who have finished early can choose more challenges from the later levels.

🖥️ Code Studio levels

- CSD U3 platform intro_2018_2019 📄
- **Student Overview**

Build a Platform Jumper

View on Code Studio



In the next several levels, you'll be building a platform jumper game. Before you move on you'll need to look at the Project Guide for this project. Wait for instructions from your teacher before clicking to the next level.

- Levels
- 🖥️ 4
- 🖥️ 5
- 🖥️ 6
- 🖥️ 7
- 🖥️ 8
- 🖥️ 9
- 🖥️ 10
- 🖥️ 11
- 🖥️ 12
- 🖥️ 13
- 🖥️ 14
- 🖥️ 15
- 🖥️ 16
- 🖥️ 17

Student Instructions

View on Code Studio



Background

The first thing that you will create for your game is the background. The sample game had two different backgrounds that were chosen according to the user's score. The first background has already been created for you. Look at the background1 function in the code below to see how it works. **Show me where**)

In order for the background function to do something, you have to call it inside the draw loop. **Show me where**)

There is also an empty function named background2 . **(Show me where)** You will need to fill that function with new code to make a different background, then test the code by calling the function inside the draw loop.

Do this

- Read the code for `background1` .
- Fill the `background2` function with new code for a second background.
- Test your `background2` function by calling it inside the draw loop.

Hint: It's much easier to copy, paste, and make small changes to your code in text mode.

Student Instructions

[View on Code Studio](#)



Score Variable

Now that you've created your backgrounds, you'll need to choose when each background is drawn. For that, you'll need a score variable to hold information about your player's score.

You should always give your variables a starting value at the very beginning of the program. That way, they are available for any code that comes after.

Do This

- Create a score variable at the beginning of your game program. (**Show me the block**)
- Set the score equal to 0.

Student Instructions

[View on Code Studio](#)



Choosing your Background

Now that you have your score variable, you can use it to choose the right background for your game. You can see an example of changing your background according to your score in **Lesson 19 Level 11**

Do This

- Inside the draw loop, use an `if` statement and your two background functions to draw your background according to your score level.
- Test your code by changing the start score to 100, then running to code to see whether the background changes.

Challenge: If you have a third background, you can click the plus sign at the bottom of the `if` block. Another space will appear for your third background function, as well as a place to check the score again.

Student Instructions

[View on Code Studio](#)



Make the Scoreboard

You'll also need a scoreboard so the player can keep track of the score. There's already a `showScore` function written, but it only shows the text "Score" and not the actual score. (**Show me where**) You can see an example of a working scoreboard in **Lesson 16 Level 9**.



Do This

- Read the code in the `showScore` function.
- Call the function inside the draw loop, right after you draw the backgrounds.

- Use the `text` block to display the score at the top of the screen.

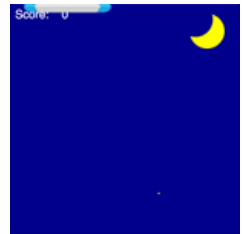
Student Instructions

[View on Code Studio](#)



Create a Platform Sprite

Now that you have your background and your variables, it's time to create your sprites. Usually, it will be easiest to start with the sprites that are part of the environment, such as your platforms. The sample game had two platform sprites, but you'll make just one first, then test it before copying and pasting the code to make the second.



Do This

- Look at your worksheet and choose a platform sprite to create.
- In the "Create sprites" area at the top of your code, create your new sprite with the `createSprite` block, giving it the correct position and label (name). (**Show me the block**)
- Use the `setAnimation` and `velocityY` blocks to give your sprite the correct image and downward velocity.
- Test the sprite to make sure that it's moving in the correct way. You might need to adjust its velocity.

Hint: The sprite will go off the screen and not come back. You'll make it loop back around in the next level.

Student Instructions

[View on Code Studio](#)



Loop the Platform Sprite

Right now, your platform sprite moves down, but it doesn't loop back up to the top of the screen. You can look at **Lesson 15 Level 13** to see an example of a sprite looping around a screen.



Do This

- Use the `function` block to create a `loopPlatforms` function at the bottom of your code.
- Use the `if` block inside the function to check whether the platform has gone off the bottom of screen and, if it has, move it back to the top of the screen.
- Call the function inside the draw loop, in the "update sprites" area.
- Run the code to test your sprite.

Hint: What will `platform.y` be when the sprite moves off the bottom of the screen? What should `platform.y` be when you put it back at the top of the screen?

Student Instructions

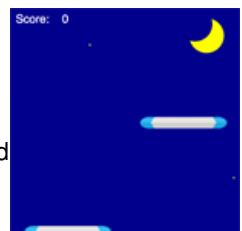
[View on Code Studio](#)



Make your Second Platform

Making a second platform will be easier than making the first, because you can copy and paste a lot of the code, then make a few small changes. This is a lot easier in **text mode**, so be sure to try it out if you haven't already.

You'll need to copy two parts of your code: the part at the beginning where you made the platform, and the part in your `loopPlatforms` function, where you looped the platform back to the top of the screen.



Do This

- Copy the code you used to create the first platform sprite (`createSprite` , `setAnimation` , and `velocityY`), and paste it directly beneath the original code.
- Change the names of the sprite in the new lines. For example, if you named your original sprite "platform", you could name this one "platform2".
- Change the starting position of your new platform sprite.
- Inside your `loopPlatforms` function, copy the if statement, then paste it directly underneath the original code, inside the function.
- Change the sprite name in the new lines of code.
- Run your code to test it.

Challenge: You can make your platforms appear at random x positions when they loop back to the top of the screen.

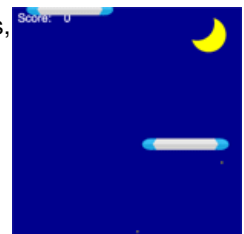
Student Instructions

[View on Code Studio](#)



Create an Item

Next, you need to add the items that fall from the top of the screen. These move just like the platforms, but faster. In order to make the game more interesting, the items start at a random location above the screen. For the sample game, the item's x position is a random number between 50 and 350, and the y position is a random number between -30 and -60.



Do This

- Use the `createSprite` block to make an item sprite in the "create sprites" section of your code.
- Use the `randomNumber` block inside your `createSprite` block to start the item at a random x and y position.
- Use `setAnimation` and `velocityY` to give your sprite the correct image and make it fall from the top of the screen.
- Run the code to test your sprite.

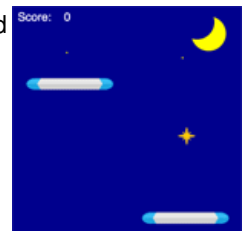
Student Instructions

[View on Code Studio](#)



Loop your Item

Now that your item is falling, you can add code to loop it back to the top. This is similar to what you did for the platform sprite, but the item sprite will loop back to a random x and y location when it goes to the top of the screen.



Do This

- Create a `loopItems` function that uses an `if` block to check whether the item sprite is off the bottom of the screen, then sends the item back to a random x and y position, just as it did when you first created the sprite.
- Call the function inside the draw loop.
- Run the code to test your sprite.

Student Instructions

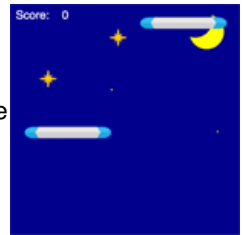
[View on Code Studio](#)



Make your Second Item

Next, you'll copy and paste the code for your first item to create a second item. Remember that this is a lot easier in **text mode**, so be sure to try it out if you haven't already.

You'll need to copy two parts of your code: the part at the beginning where you made the item, and the part in your `loopItem` function, where you looped the item back to the top of the screen.



Do This

- Copy the code you used to create the first item sprite (`createSprite` , `setAnimation` , and `velocityY`), and paste it directly beneath the original code.
- Change the names of the sprite in the new lines. For example, if you named your original sprite "star", you could name this one "star2".
- Inside your `loopItems` function, copy the if statement, then paste it directly underneath the original code, inside the function.
- Change the sprite name in the new lines of code.
- Run your code to test it.

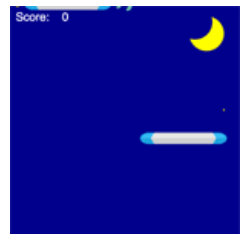
Student Instructions

[View on Code Studio](#)



Create your Player

Now you can create your player sprite. Just like the item sprites, the player sprite will fall from the top of the screen. Unlike the items, your player sprite will get faster as it falls, just like real falling objects. This is what allows it to jump up, and fall back down.



Do This

- Use the `createSprite` block to make a player sprite with the label and starting position that you put on your worksheet.
- Use `setAnimation` to give it the correct image.
- Create a `playerFall` function that makes the sprite fall from the top of the screen. The code inside the function should use `velocityY` in a counter pattern, just as when you made the falling rock in **Lesson 17 Level 4**.
- Call the `playerFall` function inside the draw loop.
- Run the code to test your sprite.

Student Instructions

[View on Code Studio](#)



User Controls

Next, you should add user controls so that you can move your player around. Your player needs to move to the left when the left arrow key is pressed, to the right when the right arrow key is pressed, and jump when the space bar is pressed.

Do This

- Create a new `controlPlayer` function in the "functions" area of your code.
- Inside the `controlPlayer` function, use the `if` , `keyDown` , and `sprite.x` blocks to make your player move to the left and right according to the arrow keys. Look at **Lesson 12 Puzzle 7** for examples.
- Inside the `controlPlayer` function, use the `if` , `keyDown` , and `velocityY` blocks to make your player jump when the up arrow is pressed. Look at **Lesson 15 Puzzle 11** for an example.
- Call the `controlPlayer` function inside the draw loop.
- Run the game and test your code.



Player Interactions

The last part of making your game is programming the player interactions with the other sprites. First, your player needs to land on the platforms.

Do This

- Create a `playerLands` function and add it to the "functions" area of your code.
- Inside the function, use the `collide` block so that your player can land on both the platforms.
- Call the function inside the draw loop.
- Run the code to test your function.



Collect Items


Last, you'll want your player to collect the items falling from the top of the screen.

Do This

- Create a `collectItems` function and add it to the "functions" area of your code.
- Use the `if` and `isTouching` blocks to change the x and y position of the items when the player touches them. You can look at the `loopItem` function for clues in how to reset the item position.
- Inside your `if` statement, add a counter pattern that will increase the score every time the player touches an item. Look at **Lesson 16 Puzzle 9** for an example.
- Call the function inside the draw loop, in the "update sprites" area of your code.
- Run the code to test your function.

Students work in pairs to create the alien jumper game.

Code Studio levels

- Plane Jumper 
- **Student Overview**

Plane Jumper



Here's another example of a platform jumper, but it has a few more features. You can use it to get ideas to improve your own game. For example, there is a coin sprite that gives the player an extra life.

Choose one or more of the following changes and add them to your game. **Choose new animations for your player, platform, and items.** Make it impossible for your player to go off the left or right of the screen. **Add a different type of item for the player to collect or avoid.** Add a variable that keeps track of how many lives the player has, and end the game if the player runs out.

- Bunny Jumper 
- **Student Overview**

Bunny Jumper



Here's another example of a jumper. In this one, the items get faster when they fall, and bounce off the platforms.

Choose one or more of the following changes and add them to your game. **Make your player's animation change direction when the player changes direction.** Add another background and make it appear when the score gets even higher. * Make your items interact with the platforms in some way.

- Levels
-   20

Student Instructions

[View on Code Studio](#)



Improve Your Game

Improve your game by adding in two or more of the features you saw in the last two examples. You can use the list below to help you.

- Choose new animations for your player, platform, and items.
- Prevent your player from going off the side of the screen.
- Add a different type of item for the player to collect or avoid.
- Add a variable that keeps track of how many lives the player has, and end the game if the player runs out.
- Make your player's animation change direction when the player changes direction.
- Change the player to only jump when it is on a platform.
- Add another background and make it appear when the score gets even higher.
- Make your items interact with the platforms in some way.

Students have an opportunity to improve their game after being exposed to two other versions of a platform jumper.

Share out

Share: Students share their games with their classmates.

Wrap Up

Journal

Prompt: Have students reflect on their development of the **five practices of CS Discoveries** (Problem Solving, Persistence, Creativity, Collaboration, Communication). Choose one of the following prompts as you deem appropriate.

- Choose one of the five practices in which you believe you demonstrated growth in this lesson. Write something you did that exemplified this practice.
- Choose one practice you think you can continue to grow in. What's one thing you'd like to do better?
- Choose one practice you thought was especially important for the activity we completed today. What made it so important?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

Assessment

Key Concepts:

Using a structured process to plan and develop a program

Assessment Criteria:

- ▶ Extensive Evidence
- ▶ Convincing Evidence
- ▶ Limited Evidence
- ▶ No Evidence



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 22: Project - Design a Game

Game Lab | Project

Overview

Students will plan and build their own game using the project guide from the previous two lessons to guide their project. Working individually or in pairs, students will first decide on the type of game they'd like to build, taking as inspiration a set of sample games. They will then complete a blank project guide where they will describe the game's behavior and scope out the variables, sprites, and functions they'll need to build. In Code Studio, a series of levels prompts them on a general sequence they can use to implement this plan. Partway through the process, students will share their projects for peer review and will incorporate feedback as they finish their game. At the end of the lesson, students will share their completed games with their classmates. This project will span multiple classes and can easily take anywhere from 3-5 class periods.

Purpose

This lesson is the culmination of Unit 3 and provides students an opportunity to build a Game Lab project of their own from the ground up. The scaffolding provided by the project guide and the practice they have using it are intended to assist students in scoping their projects and seeing their ideas through to completion. This project is an opportunity to showcase technical skills, but they will also need to collaborate with their partner, provide constructive peer feedback, and repeatedly use the problem solving process as they encounter obstacles along the way. This project should be student-directed whenever possible, and provide an empowering and memorable conclusion to this unit of CS Discoveries.

Assessment Opportunities

Use the project rubric attached to this lesson to assess student mastery of learning goals of this unit. You may also choose to assign the post-project test through code studio.

Agenda

Warm Up (10 mins)

Review Project Guide

Activity (80-200 mins)

Define - Scope Game

Prepare - Complete Project Guide

Try - Write Code

Reflect - Peer Review

Iterate - Update Code

[View on Code Studio](#)

Objectives

Students will be able to:

- Independently scope the features of a piece of software
- Create a plan for building a piece of software by describing its major components
- Implement a plan for creating a piece of software

Preparation

☐ Print copies of the project guide, one for each student / pair of students

☐ Print copies of the rubric, one for each student / pair of students

☐ Print copies of the peer review guide, one for each student / pair of students

☐ Review sample games in Code Studio

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teacher

- **Make Your Own Game** - Exemplars

For the Students

- **Make Your Own Game** - Project Guide

[Make a Copy](#)

- **Make Your Own Game** - Rubric

[Make a Copy](#)

- **Make Your Own Game** - Peer Review

[Make a Copy](#)

- **Computer Science Practices** - Reflection

[Make a Copy](#)

Wrap Up (10 mins)

Share

Reflect

Post-Project Test



Teaching Guide

Warm Up (10 mins)

Review Project Guide

Group: This project can be completed individually or in pairs. At your discretion you may choose to have students form larger groups as well.

Distribute: Each student or group of students should be given a copy of the project guide. As a class review the different steps of the project and where they appear in the project guide. Direct students towards the rubric so that they know from the beginning what components of the project you will be looking for.

Activity (80-200 mins)


Define - Scope Game

Circulate: Students should spend the first 15-20 minutes playing the sample games, reviewing past work, and discussing as a group the type of game they'd like to build. If they want they can sketch ideas on scratch paper or in


Prepare - Complete Project Guide

Circulate: Once students have discussed their ideas for the project they should complete the project guide. While this should be a fairly familiar process, encourage students to make each component as clear and detailed as they can at this point. Planning ahead can help them identify issues in their plan before they'll need to make more significant changes to their code.


Try - Write Code

 **Transition:** Students are now ready to program their games on Code Studio. These levels provide some guidance on how students may go about implementing their project guide. None of the steps are significantly different from what students have seen from the previous two lessons. If they wish, students can work in a different order than the one suggested in these levels.

Code Studio levels

- Lesson Overview 
- **Teacher Overview**
- **Student Overview**

View on Code Studio to access answer key(s)

- Design Your Game 
- **Student Overview**

View on Code Studio

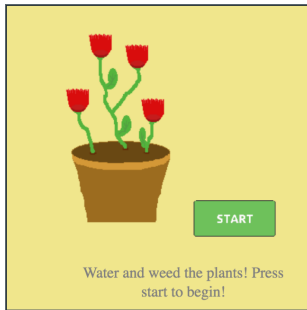
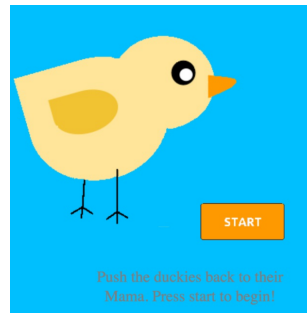

View on Code Studio


Create your own game

View on Code Studio


Now that you have all the skills you need, it's time to make your own game!

With a partner, brainstorm some different ideas for your game. You can think about the games you've already seen, or look at some more sample games to give you ideas.



Once you have settled on a type of game with your partner, fill out the Project Guide with the backgrounds, variables, sprites, and functions that you will need to make the game. You'll spend the next few levels creating your game.

- Levels
- 3
- 4
- 5
- 6
- 7

Student Instructions

[View on Code Studio](#)

Create your Variables

First, you'll need to create all of your variables and put them in the variables area of your code **Show me the block Show me the area in the code**

Don't forget, each variable needs a label (name) and a starting value. You can change the value of the variable later in your code.

Student Instructions

[View on Code Studio](#)

Create your Backgrounds

Next, you'll create all of the background functions that you need for your game. Some games only have one background, and others have more than one that's chosen according to user score or another aspect of gameplay. You'll need to create a function for each separate background in your game. You'll write the code to choose the correct background in the next level.

- **Show me the block to create a new function**
- **Show me the area in the code to put my function**

After you create your functions, test them by calling them inside the draw loop, one background per test.

- **Show me the block to call my function**

Student Instructions

[View on Code Studio](#)

Display Boards

Now that your backgrounds are working, you can add your display boards. Most games have a score board, but you might also want to display information about player level or lives remaining. Look at **Lesson 16 Puzzle 9** for an example of how to make a scoreboard.

For each display board: **Create a function to display the information** Call the function in the draw loop

Be sure to test your boards by changing the starting value of your variables and making sure the board also changes when you run the code.

Student Instructions

[View on Code Studio](#)

Choose your Backgrounds

Now that you have the backgrounds that you need, you'll write the code to choose the correct background. You've seen this done in **Lesson 19 Level 11**.

After you've written the code, test it by changing the starting value of your variables and making sure the correct background shows up.

Student Instructions

[View on Code Studio](#)

Create your Animations

Next you will create your animations in the animation tab. Don't forget to make multiple animations if you want your sprite to change appearance according to how it's moving.

Reflect - Peer Review

Distribute: Give each student a copy of peer review guide.

Students should spend 15 minutes reviewing the other group's game and filling out the peer review guide.

Iterate - Update Code

Circulate: Students should complete the peer review guide's back side where they decide how to respond to the feedback they were given. They should then use that feedback to improve their game.


Wrap Up (10 mins)

Share

Share: Give students a chance to share their games. If you choose to let students do a more formal presentation of their projects the project guide provides students a set of components to include in their presentations including:

- The original game they set out to build
- A description of the programming process including at least one challenge they faced and one new feature they decided to add
- A description of the most interesting or complex piece of code they wrote
- A live demonstration of the actual game

Reflect

 Send students to Code Studio to complete their reflection on their attitudes toward computer science. Although their answers are anonymous, the aggregated data will be available to you once at least five students have completed the survey.

Code Studio levels

- Levels
-   13

Student Instructions

View on Code Studio



Post-Project Test

The post-project test is found at the bottom of the Interactive Animations and Games unit overview page on Code Studio (studio.code.org/s/csd3-2019).

This test is locked and hidden from student view by default. In order for students to see and take this test, you'll need to unlock it by clicking the "Lock Settings" button and following the instructions that appear.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming



This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.