Unit 3 - Interactive Animations and Games ('24-'25)

▼ Teacher resources | ▼ Printing Options

In the Interactive Animations and Games unit, students 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.

- ▶ Chapter 1 Overview
- ▶ Chapter 2 Overview
- ▶ Implementation Guidance for Interactive Animations and Games

Finished Teaching This Unit?

Answer this **short survey** to let the Code.org curriculum team know how the unit went.

Week 1	Lesson 1: Programming for a Purpose	Lesson 2: Plotting Shapes	Lesson 3: Drawing in Game Lab	Lesson 4: Shapes and Parameters	Lesson 5: Variables
Week 2	Lesson 6: Random Numbers	Lesson 7: Mini-Project - Robot Faces	Lesson 8: Sprites	Lesson 9: Sprite Properties	Lesson 10: Text
Week 3	Lesson 11: Mini-Project - Captioned Scenes	Lesson 12: The Draw Loop	Lesson 13: Sprite Movement	Lesson 14: Mini-Project - Animation	Lesson 15: Conditionals
Week 4	Lesson 16: Keyboard Input	Lesson 17: Mouse Input	Lesson 18: Project - Interactive Card		Lesson 19: Velocity
Week 5	Lesson 20: Collision Detection	Lesson 21: Mini-Project - Side Scroller	Lesson 22: Complex Sprite Movement	Lesson 23: Collisions	Lesson 24: Mini-Project - Flyer Game
Week 6	Lesson 25: Functions	Lesson 26: The Game Design Process	Lesson 27: Using the Game Design Process		
Week 7	Lesson 28: Project - Design a Game				

Key	☐ Instructional Lesson ✔ Assessment ✂ Unplugged Lesson

View calendar

Active section:

Select a section

Show All Lessons | Hide All Lessons

Chapter 1: Images and Animations

Lesson 1: Programming for a Purpose

To kick off a unit devoted to problem-solving and developing animations and games, students begin by investigating the design of different animations and games. Students look at a variety of animations and games and attempt to match each design with a potential user. Then students choose a user and attempt to prototype an animation or game design for them on paper or in a digital template. To conclude the activity, students consider what it means to be an animation and game designer and create resources for other users.

Question of the Day: How can we design animations and games based on the needs of a user?

1 **Animation and Game Templates**

2 **Check-In Survey**

Lesson 2: Plotting Shapes

The primary purpose of this lesson is to introduce students to the coordinate system they will use in Game Lab. Students begin by exploring the challenges of communicating how to draw with shapes and then transition to using 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 images to communicate before a final debrief discussion.

Question of the Day: How can we clearly communicate how to draw something on a screen?

1 **Drawing Shapes**

Lesson 3: Drawing in Game Lab

This lesson is designed 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.

Question of the Day: How can we communicate to a computer how to draw shapes on the screen?

- 1 — **Predict - Introduction to Game Lab**
- 2-6 — **Skill Building**
  - 2 3 4 5 6
- 7 — **Practice**
- 8 — **Assessment**
- 9 — **Challenges**

Lesson 4: Shapes and Parameters

In this lesson, students continue to build skills and 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. Throughout the lesson, students will need to reason about the x-y coordinate plane, consider the order of their code, and slightly increase their programs' complexity.

Question of the Day: How can we use parameters to give the computer more specific instructions?

- 1 — **Predict**
- 2-6 — **Skill Building**
  - 2 3 4 5 6
- 7 — **Practice**
- 8 — **Assessment**

9 Challenges

Lesson 5: Variables

In this lesson, students learn how to use variables to label a value. Students begin the lesson with a very basic description of the purpose of a variable within the context of the storage component of the input-output-storage-processing model. Students then complete a level progression that reinforces the model of a variable as a way to label or name a number. Students should 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 random numbers, in which they will see a more powerful use for variables, and in later lessons, students will continue to expand their understanding of variables and experience more advanced ways they can be used.

Question of the Day: How can we use variables to store information in our programs?

1 Prediction

2 Predict

3-5 Skill Building

3 4 5

6 Practice

7 Assessment

8 Challenges

Lesson 6: Random Numbers

This lesson introduces randomness, which is important both as a way to make programs more interesting and also to motivate the use of variables. Students are introduced to the `randomNumber()` block and how it can be used to create new behaviors in their programs. They then learn how to update variables during a program. Combining all of these skills, students draw randomized images.

Question of the Day: How can we make our programs behave differently each time they are run?

1 Predict

2-5 Skill Building

2 3 4 5

6 Practice

7 Assessment

8 Challenges

Lesson 7: Mini-Project - Robot Faces

After a quick review of the code they have learned so far, students are introduced to their first creative project of the unit. Using the problem-solving process as a model, students define the robot face that they want to create, prepare by thinking of the different code they will need, try their plan in Game Lab, then reflect on what they have created. They also have a chance to share their creations with their peers. The open-ended nature of this lesson also provides flexibility for the teacher to decide how long students should spend on their work, depending on the scheduling demands of the particular course implementation.

Question of the Day: How can we use shapes, variables, and randomness to express our creativity?

1 Review Shape Blocks

2 Create Your Robot Face

Lesson 8: Sprites

In order to create more interesting and detailed images, students are introduced to the sprite object which allows for one variable name to control both the shape and all its aspects. The lesson starts with a discussion of the various information that programs must keep track of, then presents sprites as a way to keep track of that information. Students then learn how to assign each sprite an image, which will greatly increase the complexity of what they can draw on the screen.

Question of the Day: How can we use sprites to help us keep track of lots of information in our programs?

1 Predict

2-7 Skill Building

2 3 4 5 6 7

8 Check for Understanding

9 Practice

10 Assessment

11 Challenges

Lesson 9: Sprite Properties

In the last lesson, when students were introduced to sprites, they focused mainly on creating a sprite and assigning it an animation. This lesson starts to dig into what makes sprites such a powerful programming construct - that they have properties that can be modified as a program is *running*. This lays the foundation for much of what students will be doing in the rest of the unit in terms of accessing and manipulating sprite properties to create interesting behaviors in their programs. The lesson starts with a review of what a sprite is, then students move on to Game Lab to practice more with sprites, using their properties to change their appearance. They then reflect on the connections between properties and variables.

Question of the Day: How can we use sprite properties to change their appearance on the screen?

1 Prediction

2-3 Skill Building

2 3

4 Practice

5 Assessment

6 Challenges

Lesson 10: Text

This lesson introduces Game Lab's text commands, giving students more practice using the coordinate plane and parameters. This is the last type of element that students will be placing on the screen - after this, students will focus on how they can control the movement and interactions of these elements. The lesson begins with asking students to caption a cartoon created in Game Lab. They then move on to Code Studio where they practice placing text on the screen and controlling other text properties, such as size. Students who complete the assessment early can go on to learn more challenging blocks related to text properties.

Question of the Day: How can we use text to improve our scenes and animations?

1 Prediction

2-3 Skill Building

2 3

4 Practice

5 Assessment

6 Challenges

Lesson 11: Mini-Project - Captioned Scenes

After a quick review of the code they have learned so far, students start working on their next creative project of the unit. Using the problem-solving process as a model again, students define the scene that they want to create, prepare by thinking of the different code they will need, try their plan in Game Lab, then reflect on what they have created. They also have a chance to share their creations with their peers. The open-ended nature of this lesson also provides flexibility for the teacher to decide how long students should spend on their work, depending on the scheduling demands of the particular course implementation.

Question of the Day: How can we use Game Lab to express our creativity?

1 Captioned Scene Example

2 Create a Background

3 Add Sprites

4 Add Text

5 Review Your Scene

Lesson 12: The Draw Loop

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. Students should leave the lesson understanding that the commands in the draw loop are called after all other code but

are then called repeatedly to create animation. 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.

Question of the Day: How can we animate our images in Game Lab?

- 1 **Predict**
- 2 **Skill Building**
- 3 **Predict**
- 4-5 **Skill Building**
  - 4
  - 5
- 6 **Practice**
- 7 **Assessment**
- 8 **Challenges**

Lesson 13: Sprite Movement

This lesson builds on the draw loop that students learned previously to create programs with *purposeful* motion. Students learn how to control sprite movement using a construct called the counter pattern, which incrementally changes a sprite's properties. Students first brainstorm different ways that they could animate sprites by controlling their properties, then explore the counter pattern in Code Studio. After examining working code, students try using the counter pattern to create various types of sprite movements. The skills that students build in this lesson lay the foundation for all of the animations and games that they will make throughout the rest of the unit.

Question of the Day: How can we control sprite movement in Game Lab?

- 1 **Prediction**
- 2-5 **Skill Building**
  - 2
  - 3
  - 4
  - 5
- 6 **Practice**
- 7 **Assessment**

8 Challenges

Lesson 14: Mini-Project - Animation

This lesson is a chance for students to get more creative with what they have learned as students are asked to combine different methods that they have learned to create an animated scene. Students first review the types of movement and animation that they have learned and brainstorm what types of scenes might need that movement. They then begin to plan out their own animated scenes, which they create in Game Lab.

Question of the Day: How can we combine different programming patterns to make a complete animation?

1 Example Animated Scene

2 Draw a Background

3 Add Sprites

4 Add Text

5 Add Movement

6 Review Your Animated Scene

Lesson 15: Conditionals

This lesson introduces booleans and conditionals, which allow a program to run differently depending on whether a condition is true. Students start by playing a short game in which they respond according to whether particular conditions are met. They then move to Code Studio, where they learn how the computer evaluates Boolean expressions, and how they can be used to structure a program.

Question of the Day: How can programs react to changes as they are running?

1 Prediction

2 Quick Check

3-5 Skill Building

3 4 5

6 Practice

7 Assessment

8 Challenges

Lesson 16: Keyboard Input

One common way conditionals are used is to check for different types of user input, especially key presses. 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.

Question of the Day: How can our programs react to user input?

1 Prediction

2-4 Skill Building

2 3 4

5 Practice

6 Assessment

7 Challenges

Lesson 17: Mouse Input

In this lesson, students continue to explore ways to use conditional statements to take user input - this time with the mouse. They will also expand their understanding of conditionals to include *else*, which allows for the computer to run a certain section of code when a condition is true, and a different section of code when it is not. This concept is introduced alongside several new mouse input commands, allowing students to gradually build up programs that use input in different ways.

Question of the Day: What are more ways that the computer can react to user input?

1 Prediction

2-5 Skill Building

2 3 4 5

6 Practice

7 Assessment

8 Challenges

Lesson 18: Project - Interactive Card

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. In this project, students plan for and develop an interactive greeting card using all of the programming techniques they've learned to this point. Giving students the opportunity to really be creative after learning all these new concepts will help to engage them further as they head into Chapter 2.

Question of the Day: What skills and practices are important when creating an interactive program?

1 Demo App: Interactive Card

2 Interactive Card Examples

3-7 Project Work

3 4 5 6 7

8 Reflection

Chapter 2: Building Games

Lesson 19: Velocity

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. After a brief review of how they used the counter pattern to move sprites in previous lessons, students are introduced to the idea of hiding those patterns in a single `velocity` block. Students then head to Code Studio to try out new blocks that set a sprite's velocity directly, and look at various ways that they are able to code more complex behaviors in their sprites. Over the next several lessons, students will see how this method of managing complexity allows them to produce more interesting sprite behaviors.

Question of the Day: How can programming languages hide complicated patterns so that it is easier to program?

1 Predict

2-7 Skill Building

2 3 4 5 6 7

8 Practice

9 Assessment

10 Challenges

Lesson 20: Collision Detection

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. Working in pairs, students further explore this idea in the context of the intentionally complex mathematical challenge of determining whether two sprites are touching. Students then use a single block, the `isTouching()` block, to represent this complexity and to create different effects when sprites collide. By the end of the lesson, students should understand that by using a single block to represent this complexity, it becomes much easier to write and reason about code and appreciate the value of using abstractions.

Question of the Day: How can programming help make complicated problems more simple?

1 Sample Game

2-5 Skill Building

2 3 4 5

6 Practice

7 Assessment

8 Challenges

Lesson 21: Mini-Project - Side Scroller

This lesson is another chance for students to get more creative with what they have learned. Students use what they have learned about collision detection and setting velocity to create a simple side-scroller game. After looking at a sample side-scroller game, students brainstorm what sort of side-scroller they would like to make, then use a structured process to program the game in Code Studio. This lesson can be shortened or lengthened depending on time constraints.

Question of the Day: How can the new types of sprite movement and collision detection be used to create a game?

- 1 Side Scroller Example
- 2 Draw Your Background
- 3 Create Your Sprites
- 4 Player Controls
- 5 Looping
- 6 Sprite Interactions
- 7 Scoring & Scoreboard
- 8 Review Your Game

Lesson 22: Complex Sprite Movement

This lesson does not introduce any new blocks and in fact, only uses patterns students have seen in Chapter 1 and demonstrates how combining these patterns, in particular the abstractions students learned in the previous two lessons, allows them to build new behaviors for their sprites. Specifically, this lesson has students learn how to combine the velocity properties of sprites with the counter pattern to create more complex sprite movement. After reviewing the two concepts, they explore various scenarios in which velocity is used in the counter pattern and observe the different types of movement that result, such as simulating gravity. They then reflect on how they were able to get new behaviors by combining blocks and patterns that they already knew.

Question of the Day: How can previous blocks be combined in new patterns to make interesting movements?

- 1 Prediction
- 2-4 Skill Building
  - 2
  - 3
  - 4

5 Practice

6 Assessment

7 Challenges

Lesson 23: Collisions

This lesson introduces collisions, another useful abstraction that will allow students to manipulate their sprites in entirely 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`). This is the last time they will learn a new sprite behavior, and following this lesson students will transition to focusing on how they organize their increasingly complex code.

Question of the Day: How can programmers build on abstractions to create further abstractions?

1 Code Prediction

2-5 Skill Building

2 3 4 5

6 Practice

7 Assessment

8 Challenges

Lesson 24: Mini-Project - Flyer Game

This lesson is another chance for students to get more creative with what they have learned. Students use what they have learned about simulating gravity and the different types of collisions to create simple flyer games. After looking at a sample flyer game, students brainstorm what sort of flyer games they would like, then use a structured process to program the game in Code Studio.

Question of the Day: How can the new types of collisions and modeling movement be used to create a game?

1 Flyer Game Example

2 Make Your Sprites

3-5 Player Controls

3 4 5

6 Sprite Movement

7 Sprite Interactions

8 Review Your Game

Lesson 25: Functions

In previous lessons, students have learned to use a number of abstractions in their programs which 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 that will serve to organize their code, make it more readable, and remove repeated blocks of code. Students first think about what sorts of new blocks they would like in Game Lab, and what code those blocks would contain inside. Afterward, 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.

Question of the Day: How can programmers use functions to create their own abstractions?

1 Predict

2-3 Skill Building

2 3

4 Predict

5 Practice

6 Quick Check

7-9 Collector Game

7 8 9

10 Challenges

Lesson 26: The Game Design Process

This lesson introduces students to the process they will use to design games for the remainder of the unit which is centered around a project guide that asks students to define their sprites, variables, and functions *before* they begin programming their game. Students begin by playing a game on Game Lab where the code is hidden, discussing what they think the sprites, variables, and functions would need to be to make the game. For the purposes of heavily scaffolding the software development process, students are then given a completed project guide that provides starter code and shows one way to implement the game. Students are then walked through this implementation process through a series of levels and have an opportunity to make improvements to the game to make it their own in the final level. In the subsequent lessons, students will need to complete a greater portion of the guide independently, and for the final project, they will follow this process largely independently.

Question of the Day: How does having a plan help to make a large project easier?

- 1 — Same Game
- 2 — Plan Your Project
- 3-4 — Set Up Sprites
  - 3
  - 4
- 5-7 — Control Your Player
  - 5
  - 6
  - 7
- 8-12 — Sprite Interactions
  - 8
  - 9
  - 10
  - 11
  - 12
- 13 — Challenges

Lesson 27: Using the Game Design Process

In this multi-day lesson, students use the problem-solving process from Unit 1 to create a platform jumper game. This lesson also 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. Students begin the lesson 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.

Question of the Day: How can the problem-solving process help programmers to manage large projects?

1 Sample Platform Jumper Game

2 Build a Platform Jumper

3-6 Platform Jumper - Background and Variables

3 4 5 6

7-9 Platform Jumper - Platforms

7 8 9

10-12 Platform Jumper - Items

10 11 12

13-16 Platform Jumper - Player

13 14 15 16

17 Platform Jumper Review

18 Challenges

Lesson 28: Project - Design a Game

Students will plan and build their own game using the project guide from the previous two lessons. 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.

Question of the Day: How can the five CS practices (problem-solving, persistence, communication, collaboration, and creativity) help programmers to complete large projects?

1 Sample Games

2-5 Project - Background and Variables

2 3 4 5

6-10 Project - Sprites and Interactions

6 7 8 9 10

11 Review Your Game

12 End-Of-Unit Survey

Post-Project Test

This lesson is locked - you need to become a verified teacher to unlock it. Learn more.

Lesson 1: Programming for a Purpose

45 minutes

Overview

To kick off a unit devoted to problem-solving and developing animations and games, students begin by investigating the design of different animations and games. Students look at a variety of animations and games and attempt to match each design with a potential user. Then students choose a user and attempt to prototype an animation or game design for them on paper or in a digital template. To conclude the activity, students consider what it means to be an animation and game designer and create resources for other users.

Question of the Day: How can we design animations and games based on the needs of a user?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Before the Lesson
- **Preparing for the Unit**

Warm Up - Option 1
- **Problem Solving Word Search**
- **The Problem Solving Process**

Warm Up - Option 2 (5 minutes)
- **Revisit The Problem Solving Process**

Activity (35 minutes)
- **Recommending Animations and Games**
- **Designing an Animation or Game**

Wrap Up (5 minutes)
- **Reflection**

Objectives

Students will be able to:

- Create a prototype of an animation or game design to meet the needs of a user using the problem-solving process
- Identify features of an animation or game design that match the needs of users
- Understand the steps of the problem-solving process

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Programming for a Purpose** - Slides ▾ Make a Copy

For the students

- **(Warm Up) Word Search** - Activity Guide ▾ Make a Copy
- **Animation and Game Design Template** - Resource ▾ Make a Copy
- **Animation and Game Design for Users** - Activity Guide

Make a Copy

- **Problem Solving and Design** - Resource Make a Copy
- **The Problem Solving Process** - Video (**Download**)

Teaching Guide

Before the Lesson

Preparing for the Unit

Getting Started with Code.org: Consider watching our **Getting Started with Code.org** video series for an overview of how to navigate lesson plans, setup a classroom section, and other important features of the Code.org platform. Each video also has a support article if you'd prefer to read or print instructions - **click here to learn more**.

Setup a Classroom Section: You can use a class section in Code.org to manage your students, view their progress, and assign specific curriculum - **click here to learn more**.

If you are using a learning management system, there may be additional steps to sync your classes with Code.org:

- **Click here** for steps to setup your classes with Google Classroom
- **Click Here** for steps to setup your classes with Clever

Become a Verified Teacher: Lesson plans and levels have additional resources and answer keys for Verified Teachers, which is quick process that verifies your position at an educational institution. **Click here to complete a form** and you should have access to verified teacher resources in ~1 business day. Verified teachers also have access to the **"Teacher's Lounge"** section of the forums.

Get Inspired: Consider watching our **Teacher Tips video playlist**, featuring current CS Discoveries teachers.

Technical Requirements: For the very best experience with all Code.org content, we recommend consulting with your school or district's IT department to *ensure specific sites **are allowed** and **are not blocked***. **Click here to see a list of sites to unblock**.

You can also find mobile and tablet support details, hardware recommendation information such as minimum Internet connection speed, smallest screen size supported, and other hardware recommendations, as well as a list of supported browsers and platforms at the same **technical requirements website**.

Warm Up - Option 1

Teaching Tip

Two Warm-Up Options: This lesson has two warm-up options, depending on whether or not students have been previously introduced to the Problem-Solving Process.

Option 1 is for classrooms that have not been introduced to the Problem-Solving Process. The warm-up is longer than normal because it introduces the Problem-Solving Process through a brief activity and a video.

Option 2 is for classrooms that have been introduced to the Problem-Solving Process in a previous unit. The warm-up is shorter and acts as a quick review of the Problem Solving Process.

Problem Solving Word Search (10 minutes)

Remarks

> In this unit, we'll be learning how to build interactive animations and games. While creating these animations and games, you might encounter some problems. The word "problem" can refer to lots of different situations - I could say I have a problem for homework, a problem with my brother, and a problem with my car, and all three mean very different things.

Prompt: What's an example of a problem you've had to solve recently?

Circulate & Distribute: Have students share their thoughts with a neighbor. Listen for examples that may be worth using as examples throughout the lesson.

As students share, distribute a copy of the **(Warm Up) Word Search** to each student, face down. Make sure students don't start the word search ahead of time.

Remarks

> Since we will most likely be faced with some problems to solve in this unit, today, we are going to learn about and practice an effective strategy for when we need to solve problems. We'll start with a simple problem: solving a word search.

Prompt: What is the problem you are trying to solve when you do a word search?

Discuss: Allow students to discuss their ideas with a classmate before sharing them with the entire class.

Discussion Goal: Guide students to the idea that the problem of a word search is that there are hidden words throughout the grid of letters and thus the goal of a word search is to find all the words in the list.

Prompt: What are the different strategies that someone could use for finding words? What are the pros and cons of each strategy?

Discuss: Allow students to silently record their ideas in writing for a minute. Afterward, invite them to share what they wrote with a neighbor and then finally bring the whole class together to develop a classwide list of strategies.

Discussion Goal: This is a quick, low-stakes brainstorm that asks students to bring in their own experiences with word searches and identify strategies they may have used or could use. Student responses may vary but may include the following strategies:

- Scan each row from left to right followed by scanning each column from top to bottom.
- Check surrounding letters. Once you found the first letter or a key letter in a word you're searching for, check the letters around it to see if you should keep going. For example, if you have a word with Q in it, if there's no U next to it, then you need to keep searching.
- Look for words that have less-common letters in them such as words with a Q, X, J, or Z
- Look for words with letter pairs, such as double letters or QU
- Look for more than one word at a time.
- Turn the puzzle upside down

Do This: Direct students to pick ONE strategy from the list that they want to try and put a "#1" next to it in their journals to indicate it was their first strategy used. Once they pick a strategy, direct students to use that strategy in their word search.

Teaching Tip

It is important to let the students know that if at any time the strategy they picked is not working, they should feel free to go back to the list of strategies and pick a different one, making sure to put a number next to the new strategy.

Circulate: Give students a few minutes to find as many words as possible. It is okay if they don't find every word. You may bring everyone back together once everyone has found at least several of the words on the list.

Prompt: Display and have students reflect on each of the following prompts:

- Were you successful in solving this word search problem? How do you know?
- What strategy did you pick and how did it work for you?
- Did anyone change strategies at any point ... why or why not?

Teaching Tip

As you lead this discussion, you may choose to do a quick raise of hand poll to find out who was successful, or on track to be successful if they didn't get a chance to finish. Additionally, you can also instruct students to pick two of the questions to answer.

Discussion Goal: This discussion serves as a way to get students to reflect on how they did solving this problem and to give students a moment to recognize if they changed strategies and why or if they would solve a problem like this differently next time.

The Problem Solving Process (5 minutes)

Display: Show students the **The Problem Solving Process** video in the slides.

Questions to Consider with Video:

- What are the 4 steps to the Problem-Solving Process?

Teaching Tip

Videos are used throughout the curriculum to spark discussions, supplement key concepts with additional explanations and examples, and expose students to the various roles and backgrounds of individuals in computer science.

While interacting with the video, turn on closed captioning so students can also read along as they watch.

To encourage active engagement and reflection, use one or more of the strategies discussed in the **Guide to Curriculum Videos**.

Discuss and Display: Briefly discuss with students what parts of the warm-up activity they felt fell into each step of the problem-solving process before displaying the example answers in the slides:

- Define: when we put into our own words what the problem of a word search is and what the goal is
- Prepare: the brainstorming of different strategies that could be used to solve a word search
- Try: when we picked a strategy and tried to find the words
- Reflect: when we reflected on how they did at the end

Teaching Tip

This is a good point to remind students that the problem-solving process is a cycle and there should be times when they need to repeat steps. For example, if they went back to the list of strategies and picked a different one at any point when solving this problem, they were not only doing the “Reflect” step without realizing it but going back to the “Prepare” and “Try” steps again.

Remarks

> Now that we're familiar with the problem-solving process, let's explore how we can use it when designing animations and games.

Warm Up - Option 2 (5 minutes)

Teaching Tip

Two Warm-Up Options: This lesson has two warm-up options, depending on whether or not students have been previously introduced to the Problem-Solving Process.

Option 2 is for classrooms that have been introduced to the Problem-Solving Process in a previous unit. The warm-up is shorter and acts as a quick review of the Problem-Solving Process.

Revisit The Problem Solving Process (5 minutes)

Remarks

> In this unit, we'll be learning how to build animations and games. Today, we are going to remind ourselves about the Problem Solving Process and eventually we will apply it to the animations and games we create.

Prompt: What do you remember about the Problem-Solving Process?

Discussion Goal: Based on student responses, the teacher should lead students through a discussion about the four steps of the Problem-Solving Process and what they remember to be involved in each step.

Prompt: Based on what you know about the Problem-Solving Process, what do you think each step could look like when designing an animation or game?

Discussion Goal: Even though students haven't designed animations or games yet, guide students to imagine the steps involved from having the first inspiration for an animation or game to having it created. Some ideas may include:

- Define: Deciding what you want to make an animation or game about or who you want to make it for
- Prepare: Looking at similar animations and games and deciding what kinds of colors, images, or characters you want to use, or sketching out the animation/game interface, or even planning out how to program the animation/game
- Try: Creating a basic version of the animation/game (even if it doesn’t have all of the features)
- Reflect: Showing it to friends to get feedback and improving the animation/game design

As students finish discussing, transition to the main activity.

Activity (35 minutes)

Recommending Animations and Games

Remarks

> In this unit, we are going to learn how to create animations and games, which starts with thinking about who we're creating an animation or game for. Sometimes we make animations and games about our own personal interests, but we should also think about designing them for other people. Eventually, we will think about applying the problem-solving process to help us actually program our animations and games, but today, we are going to focus on how to apply the problem-solving process when *designing* animations and games to meet the needs of our users and ourselves.

Group: Put students in pairs.

Distribute: Pass out a copy of the **Animation and Game Design for Users** activity guide to each student.

Teaching Tip

Answer Keys & Exemplars: An answer key or exemplar is provided for verified teachers as part of the resources in this lesson plan. If you do not see an answer key or exemplar listed as a resource, **follow these steps** to become a verified teacher.

Overview: Have students read through the directions on the first page of the activity guide. The first part of this activity asks students to match different users to an animation or game template that suits their needs. Students will need to visit Code Studio to view the templates.

Code Studio: Direct students to the "Sample Animation and Games" levels on Code Studio.

1 Animation and Game Templates

Teaching Tip

While most of these examples are pulled directly from lessons that students will complete later in the unit, a couple 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.

Circulate: Have students work through this activity in pairs, encouraging discussion about why exactly they are choosing each template. This is a great place to discuss different animation and game designs and the needs of each user.

This activity guide asks students to consider why people make animations and games and what sorts of interests can be expressed with them.

Teaching Tip

Value Reasoning over Correctness: Each user is designed with a particular template in mind, but students are encouraged to use their own reasoning and logic when recommending a template to users. It's okay for students to not make an "obvious" choice as long as they're thinking carefully about their reasoning and are able to justify their choices.

The animations and games are intentionally simple, to help set expectations about the animations and games that students will be creating over the course of the unit.

Share Out: Have students discuss the following questions in small groups before bringing them to a full class discussion:

- Which users were the easiest to find matches for?
- Which users were the hardest to find matches for?

Based on your observations while circulating, you should also identify a few specific users to ask the class to discuss, especially users for whom multiple students chose different templates for that user. Encourage students to hear different justifications for their choices, emphasizing that the rationale is more important than the choice itself.

Assessment Opportunity

As students discuss their reasoning for their choices, check to ensure that they are identifying the particular user's needs and characteristics, rather than general reasons to prefer a certain app or game. You may want to challenge students to distinguish their own needs and preferences from those of the described users.

Designing an Animation or Game

Remarks

> Now we've been asked to design an animation or game ourselves for a client. We can create our own design, but it should meet the needs of our users.

Teaching Tip

Note on Timing: Depending on how long the warm-up took, especially if the class watched the problem-solving video, you may run out of time to fully complete this activity. That's okay! It can be something students come back to as they progress throughout the unit, or you may decide to extend this activity into a new class period and stretch out the design and feedback process. This task can be implemented in a variety of ways to match the constraints of the classroom.

Overview: Have students flip to the back of their activity guide and read the overview of the next part of the activity. Students will choose a user to create a basic animation/game design for, following the Problem-Solving Process. Students can sketch their design on pen and paper, or use the provided Google Slides template to create their design. These designs are meant to be sketches - students will not have enough time to "perfect" their designs.

Distribute: Pass out copies of the **Problem Solving and Design** resource. Students can use this when preparing their design, and it can be a reference throughout the unit.

Circulate: Monitor students as they complete this activity, ensuring that students are creating designs that match the needs of the user they chose. Monitor student conversations as well to make sure both voices are being heard during discussions and decisions. Students will be working in pairs throughout the unit, so it can be good to intervene early to reinforce good collaboration habits.

(Optional) Share Out: Depending on how much time is left in class, allow students to trade with other groups and share their designs. Encourage students to share with another group who picked the same user so they can see similarities and differences in their designs.

Assessment Opportunity

Following the Problem-Solving Process: You can use student responses on the activity guide to determine how well they followed each step of the problem-solving process. This process will be reinforced throughout the unit, especially on projects.

Wrap Up (5 minutes)

Question of the Day: How can we design animations and games based on the needs of a user?

Prompt: Reflect - What was one thing that you think your design does well to meet the needs of the user? What's an area that you think can still be improved to meet the needs of your user?

Circulate: Students can answer these prompts on their activity guide in the Reflect section of the Problem Solving Process.

Share Out: Have students share their responses, highlighting similarities between responses even when the users were different.

Reflection

Code Studio: Have students answer 5 quick survey questions at the beginning of this unit. Once at least 5 students have completed the survey you will be able to view the anonymized results in the Teacher Dashboard. Some of these questions will be asked again at the end of the first project, which can be helpful in seeing student growth and shifts in attitudes throughout the unit.

Check-In Survey

Lesson 2: Plotting Shapes

45 minutes

Overview

The primary purpose of this lesson is to introduce students to the coordinate system they will use in Game Lab. Students begin by exploring the challenges of communicating how to draw with shapes and then transition to using 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 images to communicate before a final debrief discussion.

Question of the Day: How can we clearly communicate how to draw something on a screen?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
Communicating Drawing Information

Activity (35 minutes)
Drawing with a Computer

Wrap Up (5 minutes)
Journaling

Objectives

Students will be able to:

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

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Plotting Shapes** - Slides ▾ Make a Copy

For the students

- **Drawing Shapes (Version A)** - Activity Guide ▾ Make a Copy
- **Drawing Shapes (Version B)** - Activity Guide ▾ Make a Copy

Teaching Guide

Warm Up (5 minutes)

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.

[pull-right]

[/pull-right]

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 Share 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 then with the entire class.

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 the 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.

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.

Question of the Day: How can we clearly communicate how to draw something on a screen?

Activity (35 minutes)

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.

1 **Drawing Shapes**

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.

Discussion Goal: Rather than doing a live demo of the tool, use this discovery-based strategy to let students explore the tool themselves. Afterward, use the debrief to ensure all students are aware of the 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.

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

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.

Drawing Shapes

Distribute activity guides to each pair, ensuring that one student (Student A) receives Version A and the other student (Student B) receives 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 of 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 drawings 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.

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.

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.

Wrap Up (5 minutes)

Question of the Day: How can we clearly communicate how to draw something on a screen?

Journaling

Prompt: Have students reflect on each of the following prompts:

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, 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.

- 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.

Lesson 3: Drawing in Game Lab

45 minutes

Overview

This lesson is designed 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.

Question of the Day: How can we communicate to a computer how to draw shapes on the screen?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
- **Programming Images**

Activity (35 minutes)
- **Simple Drawing in Game Lab**
- **Share Drawings**

Wrap Up (5 minutes)

Objectives

Students will be able to:

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

Preparation

- Prepare projector or other means of showing videos if you wish to watch as a class
- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Drawing Shapes** - Resource
- **Drawing in Game Lab** - Slides
  ▼ Make a Copy

For the students

- **Drawing in Game Lab - Part 1** - Video (**Download**)
- **Drawing in Game Lab - Part 2** - Video (**Download**)

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

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

Teaching Guide

Warm Up (5 minutes)

Programming Images

Prompt: 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?

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

Discussion Goal: Guide students to realize that the way we communicate with each other may be more informal than how we communicate with a computer. For example, communicating with a digital assistant requires precise language and commands, usually starting with "Hey" and then the name of the device. Similarly, as we start coding in this unit, students will need to use a precise set of commands to communicate with a computer.

Remarks

> In order to give instructions to a computer, we need to use a language that a computer understands. 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.

Question of the Day: How can we communicate to a computer how to draw shapes on the screen?

Activity (35 minutes)

Simple Drawing in Game Lab

Group: Place students in pairs to program together.

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 Pair Programming in the **CSD Guide to Teaching and Learning Strategies**

Transition: Send students to Code Studio.

1 Predict - Introduction to Game Lab

Teaching Tip

Facilitating Predict Levels: Predict levels are a great opportunity for students to think critically about code and engage in class discussions. Consider having students think of their own prediction and discuss with a partner before typing in their response. Once they're run the code, bring the class together for a full-group discussion to discuss how their predictions were similar or different from the resulting program. Use this as an opportunity to address any misconceptions that students may have had about the code initially.

Digging Deeper: For more tips about programming levels, see the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

Discussion Goal: The parameters in this level will set the *x* and *y* values for the square. Students should recall this from the previous lesson, but what they may not realize is that the top left corner of the square is what will be placed at the *100 x* and *100 y* locations. Using this new information, they will be able to revisit this code at the next level and play with the *x* and *y* parameters to gain a better understanding.

Video: Show students the **Drawing in Game Lab - Part 1** video in the slides.

Teaching Tip

Videos are used throughout the curriculum to spark discussions, supplement key concepts with additional explanations and examples, and expose students to the various roles and backgrounds of individuals in computer science.

While interacting with the video, turn on closed captioning so students can also read along as they watch.

To encourage active engagement and reflection, use one or more of the strategies discussed in the **Guide to Curriculum Videos**.

Questions to Consider with Video:

- Where can you find more information about how to use the blocks?
- What's an advantage of using block mode?

Discussion Goal:

- *Where can you find more information about how to use the blocks?*
  - Make sure students understand how to access the block documentation by clicking on the blocks inside the toolbox and clicking "see examples".
- *What's an advantage of using block mode?*
  - 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.*

Teaching Tip

Using Resources: Below you can find recommendations for using the many resources students are introduced to in the lesson. You could consider creating a "Resource Chart" to keep track of these options and support students to be self-sufficient as they progress through levels.

- **Videos:** Watched as a class, but students can always return to them.
- **Help and Tips Tab:** This tab contains all of the relevant videos and reference guides for a particular level.
- **Reference Guides:** Contain text and diagrams explaining content. These are intended as helpful student resources, not class readings. They are a good place to go for review after learning content or when students get stuck in levels. You may decide to print these and have them available for students as they work through levels.
- **Documentation and Examples:** Hovering over a block will show a short description of what the block does. Clicking the "See Examples" link will open the documentation for that block.
- **Level Instructions:** Instructions may introduce small pieces of new content. Each level features a "Do This" section explaining what students are supposed to do in that level. Set the expectation early that reading these instructions, not just the "Do This" section, is important.

2-3 **Skill Building**

2 3

Teaching Tip

Facilitating Skill Building Levels: Skill Building levels are designed to continue teaching new skills and blocks through exploration, trial-and-error, and using worked examples from pre-supplied code. Students are still getting familiar with the concepts in the lesson and will need strong support throughout these levels to build confidence, debug their code, and cement their understanding.

Consider having students complete Skill Building levels in pairs using **Pair Programming**, which has students use one computer and trade between being a Driver or a Navigator. This process is highlighted in **this video**, which you can show to the class. You can have students switch roles based on a timer, or switch every time they complete a level.

Digging Deeper: For more tips about programming levels, see the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

Video: Show students the **Drawing in Game Lab - Part 2** video in the slides. **Questions to Consider with Video:**

- What's the difference between stroke and fill?

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

4-6 **Skill Building**

4 5 6

Teaching Tip

Text-to-Speech Options: The instructions panel includes two options that can support comprehension for students.

- **Text to Speech** which reads aloud the instructions for students
- **Microsoft Immersive Reader** which opens a new panel for the instructions and gives controls to change the text size, contrast, or translate to another language.

Click here to learn more about these options

7 Practice

Teaching Tip

Facilitating Practice Levels: Practice levels are designed for students to apply their knowledge from the previous levels and develop fluency in using the new blocks of code to solve problems. Students can choose which practice levels they would like to complete, and it's not necessary for a student to complete each practice level before continuing.

Students tend to be more engaged and respond better when they have an authentic choice about how to continue their learning. Allow students to choose practice levels according to their interests and level of comfort, and consider providing opportunities for students to demonstrate and explain their solutions to the practice levels they chose to the entire class.

Digging Deeper: For more tips about programming levels, see the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

8 Assessment

Teaching Tip

Facilitating Assessment Levels: Assessment levels contain a single task that requires applying the skills and concepts from the level in order to solve. Students should complete these levels individually and you can use your judgment of how much external help students should have. Assessment levels also contain a rubric that can be used for formative assessment and a box to provide feedback to students - **click here to learn more about using rubrics and giving feedback to students**.

Digging Deeper: For more tips about assessing student work, see the **CSD Guide to Assessment**.

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

9 **Challenges**

Teaching Tip

Facilitating Challenge Levels: Challenge levels are designed as extensions to the concepts and skills students learn throughout a lesson. Challenge levels tend to focus on more open-ended tasks for students to complete, or opportunities to combine several skills from previous lessons together into one program.

Challenge levels do **not** need to be completed for students to meet the core objectives of a lesson. Instead, every task in a challenge level is meant to supplement and enrich the learning objectives of a lesson, but are not required for future lessons. Students can still demonstrate mastery of the objectives of a lesson without completing any of the challenge levels.

Digging Deeper: For more tips about programming levels, see the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

Share Drawings

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

Wrap Up (5 minutes)

Question of the Day: How can we communicate to a computer how to draw shapes on the screen?

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?

Share: Allow students to share out their responses with the class.

Lesson 4: Shapes and Parameters

45 minutes

Overview

In this lesson, students continue to build skills and 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. Throughout the lesson, students will need to reason about the x-y coordinate plane, consider the order of their code, and slightly increase their programs' complexity.

Question of the Day: How can we use parameters to give the computer more specific instructions?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
- **Shapes of Different Sizes**

Activity (35 minutes)
- **Programming with Parameters**

Wrap Up (5 minutes)
- **Journal**

Objectives

Students will be able to:

- Use and reason about drawing commands with multiple parameters

Preparation

- Review the level sequence in Code Studio
- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Shapes and Parameters** - Resource
- **Shapes and Parameters** - Slides
  - ▼ Make a Copy

Vocabulary

- **Parameter** - Additional information provided as input to a block to customize its functionality

Introduced Code

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

Teaching Guide

Warm Up (5 minutes)

Shapes of Different Sizes

Prompt: The `rect` block has two inputs that control where it's drawn - the x and y position. If you wanted these commands to draw different sizes of rectangles, what additional inputs would you need to give these blocks?

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

Discussion Goal: This discussion introduces the vocabulary word "parameter" and also helps students understand the need for parameters. Students will be seeing versions of the `ellipse()` and `rect` block in this lesson that have additional 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 rectangles to be a different size, the block will need an input that lets the programmer decide how large to make it.

Remarks

> 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.

Key Vocabulary: Parameter - Additional information provided as input to a block to customize it's functionality

Question of the Day: How can we use parameters to give the computer more specific instructions?

Activity (35 minutes)

Programming with Parameters

Group: Put students into pairs for the programming activity.

Transition: Move students onto Code Studio.

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

1 **Predict**

Discussion Goal: The new parameters in this level will change the width and height of the rectangle block. Some students may guess this, but even if students are unable to make the initial prediction, that's okay! Once they run their code, they should be able to look at the result and notice that the rectangles are longer and wider than before. Using this new information, they should revisit their initial prediction and adjust it with this new information.

2-6 **Skill Building**

2 3 4 5 6

Teaching Tip

Normalizing Mistakes and Supporting Debugging: As programming levels become more complex, students may find themselves with bugs in their code that they need to untangle. If this happens frequently, this can be a demoralizing experience for students and can affect their self-perception of how capable they are in class.

To counter this, we recommend normalizing bugs and mistakes as something that happens to everyone - it's just part of the process. You can show students our **Debugging Video**, which includes several students normalizing mistakes and discussing debugging strategies that students can use. Additionally, consider displaying the **Student Guide to Debugging** for students to reference throughout the unit and having **Bug Report Quarter-Sheets** available for students to use.

7 **Practice**

8 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

9 **Challenges**

Wrap Up (5 minutes)

Journal

Question of the Day: How can we use parameters to give the computer more specific instructions?

Prompt: You use parameters to control your shape's location and size. Can you think of any other situations in which parameters might be useful?

Share: Allow students to share out their ideas.

Discussion Goal: Student answers will vary, but they should all follow the pattern of giving more specific information about how to carry out a task. If students have trouble thinking of something, you may want to give some examples of things a computer could do, such as ring an alarm (with a parameter for a certain time) or play a song (with a parameter for the song title).

Lesson 5: Variables

45 minutes

Overview

In this lesson, students learn how to use variables to label a value. Students begin the lesson with a very basic description of the purpose of a variable within the context of the storage component of the input-output-storage-processing model. Students then complete a level progression that reinforces the model of a variable as a way to label or name a number. Students should 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 random numbers, in which they will see a more powerful use for variables, and in later lessons, students will continue to expand their understanding of variables and experience more advanced ways they can be used.

Question of the Day: How can we use variables to store information in our programs?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
Input-Output-Storage-Processing

Activity (35 minutes)
Programming with Variables

Wrap Up (5 minutes)
Reflection

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
- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Naming Variables** - Resource
- **Variables** - Slides ▾ Make a Copy

For the students

- **Introduction to Variables** - Video (**Download**)
- **Variables** - Resource
- **Variables** - Video

Vocabulary

- **Variable** - A label for a piece of information used in a program.

Introduced Code

- `var x = ___;`
- `var x;`

Teaching Guide

Warm Up (5 minutes)

Input-Output-Storage-Processing

Prompt: All computers do four things: input, output, processing, and storage. Where do you see input, output, storage, and processing in Game Lab?

Teaching Tip

Input, Output, Processing, and Storage: These terms are introduced in the Problem Solving and Computing unit in CS Discoveries, so students may already be familiar with them. However, if they need a refresher, you may decide to show the **What Do Computers Do?** video which highlights these four functions.

Share: Allow students to share out their answers.

Discussion Goal: Allow students to share out their different ideas, but eventually bring the conversation back to storage to tie into today's lesson. Students' answers may include:

- input: values passed as parameters, typing into the Game Lab workspace
- output: shapes shown on the Game Lab screen
- storage: remembering the code
- processing: the If/then and matching that turn the code into the pictures on the screen

Remarks

> Today we're going to focus on storage. We're going to look at variables, which are a very common way for computers to store information in a program.

Key Vocabulary: variable - a label for a piece of information used in a program

Question of the Day: How can we use variables to store information in our programs?

Activity (35 minutes)

Programming with Variables

Transition: Send students to Code Studio.

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

1 Prediction

Discussion Goal: Students may notice that the term `xPosition` is associated with the value 50, and so when we see the term `xPosition` farther down in the code, they may think it will use the value of 50 in that spot and draw the circle on the left side of the screen. When describing this relationship, they may not have the correct words to describe what's happening and may wonder what the purple `var` keyword means. Encourage students to use other familiar words when describing this relationship, like "container" or "placeholder", before introducing vocabulary

Video: Show students the **Introduction to Variables** video in the slides.

Teaching Tip

To encourage active engagement and reflection, use one or more of the strategies discussed in the **Guide to Curriculum Videos**.

Questions to Consider with Video:

- What are variables used to do?
- How do you create a variable and assign it a value?
- What can go into a variable?

Discussion Goal: 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.

2 Predict

Discussion Goal: During this discussion, encourage students to use the vocabulary they saw in the previous video, referring to `xPosition` and `whatsTheY` as variables and describing the equal sign as "gets the value of". The concept is the same as the first predict level, with the added emphasis on using correct vocabulary. For example: "I see that the variable `xPosition` gets the value of 300, so I know it will be drawn on the right side of the screen"

3-5 Skill Building

3 4 5

Teaching Tip

Level 5: If students use variable names that start with numbers, include spaces, or break other rules, the code may be forced into text mode the next time that they go to that level or refresh the page. To get back into block mode, students will first need to fix the problem with the variable names. Use the red error squares to see where the bugs most likely are, and once they are gone, click the "block mode" button at the top right of the workspace.

6 **Practice**

7 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

8 **Challenges**

Wrap Up (5 minutes)

Reflection

Question of the Day: How can we use variables to store information in our programs?

Prompt: Give students the following prompts:

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 thing they should understand before moving on to the next lesson.

- 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.

Lesson 6: Random Numbers

45 minutes

Overview

This lesson introduces randomness, which is important both as a way to make programs more interesting and also to motivate the use of variables. Students are introduced to the `randomNumber()` block and how it can be used to create new behaviors in their programs. They then learn how to update variables during a program. Combining all of these skills, students draw randomized images.

Question of the Day: How can we make our programs behave differently each time they are run?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (35 minutes)

Programming Images

Wrap Up (5 minutes)

Objectives

Students will be able to:

- Generate and use random numbers in a program
- Update a value stored in a variable

Preparation

- Review the level progression in Code Studio
- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Random Numbers** - Slides
  ▼ Make a Copy

For the students

- **Random Numbers** - Resource

Teaching Guide

Warm Up (5 minutes)

Prompt: So far, our programs have done the same thing every time that we run them. Are there any times that you'd want a program to do something differently each time it was run?

Discuss: Allow students time to write down some ideas, then discuss them as a group.

Discussion goal: The goal of this discussion is to set the context for the introduction of random numbers. Students may come up with various ideas related to user interaction or gathering input from other sources. Allow them to discuss the different ideas that they have, but eventually, turn the conversation to the idea of randomness.

Remarks

> So far, we've wanted our programs to do exactly as we've coded, and most of our surprises have been bugs. Today we're going to look at how we can code random behaviors into our programs so that we can get some good surprises.

Question of the Day: How can we make our programs behave differently each time they are run?

Activity (35 minutes)

Programming Images

Transition: Move students onto Code Studio

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

1 **Predict**

Discussion Goal: Students should predict that *something* will happen with the x-coordiante of the ellipse, even if they are unsure what will happen. Students may say that the value will be either 200 or 400, or they may say the value will be between 200 and 400. This creates the effect of drawing an ellipse somewhere on the right side of the screen.

When facilitating the class discussion, ensure that students look at their neighbors screen and compare - this helps them see that different behaviors are occuring for different people. Also encourage students to run the code multiple times to see the result.

2-5 **Skill Building**

2 3 4 5

6 **Practice**

7 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

8 **Challenges**

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 minutes)

Question of the Day: How can we make our programs behave differently each time they are run?

Prompt: So far, we've only looked at random numbers. Are there any other things that you might like to be random in your program?

Share: Allow students to share out what sorts of random things they might like in their programs.

Discussion Goal: The discussion is intended to have students think about the wider implications of randomness in games and other programs. Although there is no block to generate random data other than numbers, in later lessons students will learn techniques that will allow them to use random numbers to randomly choose from a variety of behaviors.

Lesson 7: Mini-Project - Robot Faces

45 minutes

Overview

After a quick review of the code they have learned so far, students are introduced to their first creative project of the unit. Using the problem-solving process as a model, students define the robot face that they want to create, prepare by thinking of the different code they will need, try their plan in Game Lab, then reflect on what they have created. They also have a chance to share their creations with their peers. The open-ended nature of this lesson also provides flexibility for the teacher to decide how long students should spend on their work, depending on the scheduling demands of the particular course implementation.

Question of the Day: How can we use shapes, variables, and randomness to express our creativity?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
- **Review**

Activity (35 minutes)
- **Step 1: Define - Describe your Image**
- **Step 2: Prepare - Design your Image**
- **Step 3: Try - Develop your Image**
- **Gallery Walk**

Wrap Up (5 minutes)
- **Journal**

Objectives

Students will be able to:

- Apply variables, shapes, and randomNumber concepts to create a program.
- Use a structured process to plan and develop a program.

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Mini-Project - Robot Faces** - Slides
  ▾ Make a Copy

For the students

- **Activity Guide - Robot Face Planning** - Activity Guide
  ▾ Make a Copy
- **Problem Solving with Programming** - Resource
  ▾ Make a Copy
- **Robot Faces** - Rubric
  ▾ Make a Copy

Teaching Guide

Warm Up (5 minutes)

Review

Prompt: What are some ways we can use variables with random numbers to make our programs different each time we run them?

Share: Allow students to share what they remember as a group review.

Discussion Goal: The goal of this discussion is to review how to use variables with random numbers when drawing shapes. Students may share ways that they have used variables with random numbers in previous lessons as well as new ideas that they haven't seen yet.

Remarks

> You've learned how to do some really cool things in Game Lab. Today you'll have a chance to put them together to make a unique robot face to share with the world. That means instead of trying to recreate someone else's idea, you get to come up with an idea of your own. You can make your robot look however you want; you can choose the shape of its head, how many eyes or mouths it has, if it has antennas, and more. There is no limit, so let's come up with something original and unexpected!

Question of the Day: How can we use shapes, variables, and randomness to express our creativity?

Teaching Tip

Facilitating Mini-Projects: Mini-Projects act as checkpoints in the curricula and cover the subset of skills students have seen so far in the unit. They are designed for 1-2 days of implementation as a way to check in with how well students understand the course content so far.

You may decide to extend these projects as a way to support or challenge students, which could allow you to revisit difficult concepts or support students who may have missed lessons and are trying to catch up. However, we recommend deciding this ahead of time and being firm with students about how much time they have for each project - otherwise, it's easy for projects to drag out to multiple days and for students' work to spiral beyond the scope of this project.

Activity (35 minutes)

Distribute: Pass out copies of the **Activity Guide - Robot Face Planning**. Students should use this worksheet to guide them through the Problem-Solving Process and plan out the robot face they create at the end of this lesson.

Step 1: Define - Describe your Image

Circulate: As students fill out a description of the robot face they want to create, circulate the room to ask students about their ideas and help guide them to make sure they are thinking of how they could include variables and random features. Remind students about the `shape`, `regularPolygon`, `line`, `arc`, and `point` blocks so that they know they are not limited to just `ellipse` and `rect` blocks.

Step 2: Prepare - Design your Image

As students answer questions about the design of their robot face, continue to ensure that they are thinking about variables and random features. After they have completed their designs, ensure that they can identify which blocks will be needed for the different features on the face.

Teaching Tip

Scoping Student Projects: Students may ideate projects that are beyond the skills they currently have or that would take longer than the allotted time to implement. Rather than asking students to choose a different project, consider asking students to imagine a more scaled-down version of their initial idea. As an analogy, if a student's initial idea is the "Run" step, imagine a less intense version that represents what the "Walk" step would look like. If necessary, you can keep going back further to a "Crawl" step as well.

Digging Deeper: This is sometimes referred to as the Minimal Viable Product - you can learn more about this process and adapt it into your project strategies by reading this article: **Making Sense of MVP** by Henrik Kniberg

Step 3: Try - Develop your Image

Remarks

> Many of you are ready to start creating your programs. One thing that could make this challenging is the *blank screen* effect: unlike previous levels, you won't have any starter code or direction on what to create. This means you might not be sure what exactly you're supposed to do, or you might run into bugs you need to fix which can be frustrating. Luckily, we can also use the problem-solving process to help with these types of projects as well!

Distribute: Hand out a copy of the **Problem Solving with Programming** to pairs of students. Encourage students to look over the guide.

Display: Show the slide with the problem solving process graphic.

Remarks

> If you feel stuck or you're not sure what to do next, remember you can always follow the steps of the problem-solving process to **define** your next step, **prepare** for what you want to code, **try** it out, then **reflect** on whether or not it solved your problem.

Circulate: Encourage students to use the steps in the Problem-Solving Process for Programming when they get stuck or are unsure of what to do next.

Teaching Tip

Not all bullets in the Problem Solving Process will be applicable to every problem a student has. Instead, encourage them to pick one or two from each category to try each time they are stuck

After you have checked the designs, allow students to log into Code Studio and code their programs. They will have a chance to review all of the blocks they have learned before they start on their robot faces, as some may help them create the features of their robot faces. They can complete these activities as a review or use them as resources while they work on their projects.

Transition: Send students to Code Studio.

1 **Review Shape Blocks**

2 **Create Your Robot Face**

Teaching Tip

Debugging Strategies: As students design and implement their own project ideas, they may find themselves with new bugs that they need to untangle and you may find yourself looking at completely unfamiliar code as students look for help troubleshooting their errors. To help smooth out the debugging experience, consider the following strategies:

- Review the **Teacher Guide to Debugging** for some common questions and strategies to help support students in debugging their code
- Have students follow the steps in the **Student Guide to Debugging** and use the **Bug Report Quarter-Sheets** as an initial step in the debugging process. This helps students prepare and communicate their issues before asking for help.
- If students haven't seen it yet, consider showing the **Debugging Video** to the class to reinforce debugging best practices.

Digging Deeper: Consider supplying students with an object to talk to as part of the debugging process. This is sometimes known as Rubber Duck Debugging - you can learn more on the website **https://rubberduckdebugging.com/**

Gallery Walk

Allow students to walk around the room and see the robot faces that each of their classmates has created. Celebrate all of the different ideas that students were able to implement with the same basic blocks.

Teaching Tip

You may choose to formalize this process by having each student write down one positive quality of each project, or having students "draw names" to comment on one particular classmate's work.

Assessment Opportunity

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

Wrap Up (5 minutes)

Journal

Question of the Day: How can we use shapes, variables and randomness to express our creativity?

Prompt: What was one especially creative way you saw someone else use the blocks today?

Share: Have students share what they appreciated about their classmates' projects. You may want to do this "popcorn" style, with each student who responds choosing the next person to share.

Discussion Goal: This discussion should serve as a celebration of what the students have accomplished. As students share what they have seen, encourage them to learn from each other and ask questions if they are not sure how to do something. Highlight how students were able to do very different things with the same tools.

Lesson 8: Sprites

45 minutes

Overview

In order to create more interesting and detailed images, students are introduced to the sprite object which allows for one variable name to control both the shape and all its aspects. The lesson starts with a discussion of the various information that programs must keep track of, then presents sprites as a way to keep track of that information. Students then learn how to assign each sprite an image, which will greatly increase the complexity of what they can draw on the screen.

Question of the Day: How can we use sprites to help us keep track of lots of information in our programs?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
- **How Much Information?**

Activity (35 minutes)
- **Introduction to Sprites**
- **Skill Building**
- **Check for Understanding**
- **Practice**
- **Assessment**
- **Challenges**

Wrap Up (5 minutes)
- **Journal**

Objectives

Students will be able to:

- Create and use a sprite

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Animation Tab** - Resource
- **Sprites** - Resource
- **Sprites** - Slides ▾ Make a Copy

For the students

- **Introduction to Sprites** - Video (**Download**)
- **The Animation Tab** - Video (**Download**)

Vocabulary

- **Sprite** - A character on the screen with properties that describe its location, movement, and look.

Introduced Code

- `drawSprites()`

- `sprite.setAnimation(label)`
- `var sprite = createSprite(x, y, w, h)`

Teaching Guide

Warm Up (5 minutes)

How Much Information?

Review: So far we've written programs that put simple shapes on the screen. List of all of the different pieces of information that you have used to control *how* these shapes are drawn.

Prompt: If you 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?

Share: Allow students to share out their lists.

Discussion Goal: 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?

Remarks

> Today we'll learn how to create characters in our animations called **sprites**. These sprites will help us keep track of all of the information that we need in our programs.

Question of the Day: How can we use sprites to help us keep track of lots of information in our programs?

Activity (35 minutes)

Introduction to Sprites

Transition: Send students to Code Studio

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

1 **Predict**

Video: Show students the **Introduction to Sprites** video in the slides.

Questions to Consider with Video:

- What is a sprite?
- What problem do sprites solve?

Discussion Goal: From the video, students should think of sprites as a *container* that helps us manage different properties of an image and helps us solve the problem of organizing a lot of information about how something should be drawn to a screen. It's okay if students also bring in some of their own experiences with animations and sprites, but emphasize that in Game Lab: a sprite isn't the animation or image itself, but the container that we can attach an image to. This helps manage expectations in the next few levels, where a sprite is just a gray rectangle until we learn to add animations to it.

Teaching Tip

To encourage active engagement and reflection, use one or more of the strategies discussed in the **Guide to Curriculum Videos**.

Remarks

> Keeping track of many shapes and the different variables that control characteristics of those shapes can get very complex. Computer scientists created something called an object which allows for one variable name to control both the shape and all its additional characteristics. In Game Lab, students will use a certain type of object called a sprite. A sprite is just a rectangle with properties for controlling its look.

Content Corner

Sprites and Objects: Behind the scenes, a sprite is a type of data called an object. Objects are programming structures that encapsulate data and behaviors into a single item. We don't need to introduce the formal idea of an object in this unit, but sprites are a good example of an object and students will learn to manipulate their properties throughout the unit.

Key Vocabulary:

- **Sprite** - A character on the screen with properties that describe its location, movement, and look.

2-3 Skill Building

2 3

Video: Show students the **The Animation Tab** video in the slides.

Questions to Consider with Video:

- What are the steps to adding an image to a sprite?

Discussion Goal: 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.

Teaching Tip

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, and that two or more sprites might share the same animation.

Skill Building

4-7 **Skill Building**

4 5 6 7

Check for Understanding

8 **Check for Understanding**

Practice

9 **Practice**

Assessment

10 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

Challenges

11 **Challenges**

Wrap Up (5 minutes)

Key Vocabulary:

- **Sprite** - A character on the screen with properties that describe its location, movement, and look.

Journal

Question of the Day: How can we use sprites to help us keep track of lots of information in our programs?

Prompt: So far we've been able to change a sprite's location and image. What else might you want to change about your sprites?

Share: Allow students to share out their ideas.

Discussion Goal: This discussion prompts students to think about the different properties that a sprite might have, and prepares them for the next lesson, which explicitly covers sprite properties.

Lesson 9: Sprite Properties

45 minutes

Overview

In the last lesson, when students were introduced to sprites, they focused mainly on creating a sprite and assigning it an animation. This lesson starts to dig into what makes sprites such a powerful programming construct - that they have properties that can be modified as a program is *running*. This lays the foundation for much of what students will be doing in the rest of the unit in terms of accessing and manipulating sprite properties to create interesting behaviors in their programs. The lesson starts with a review of what a sprite is, then students move on to Game Lab to practice more with sprites, using their properties to change their appearance. They then reflect on the connections between properties and variables.

Question of the Day: How can we use sprite properties to change their appearance on the screen?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (35 minutes)

Wrap Up (5 minutes)

Objectives

Students will be able to:

- Use dot notation to update a sprite's properties

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Sprite Properties** - Resource
- **Sprite Properties** - Slides
  ▼ Make a Copy

For the students

- **Sprite Properties** - Video (**Download**)

Vocabulary

- **Dot notation** - the way that sprites' properties are used in Game Lab, by connecting the sprite and property with a dot.
- **Property** - A label for a characteristic of a sprite, such as its location and appearance

Introduced Code

- `sprite.rotation`
- `sprite.scale`
- `sprite.x`
- `sprite.y`

Teaching Guide

Warm Up (5 minutes)

Prompt: What is your definition of a sprite? What sprite properties do you know how to use? What other sprite properties might be useful?

Allow students time to reflect on their own and then with a partner before sharing out to the entire group. It's okay if students do not have a canonical definition of a sprite, but they should recognize that a sprite is a part of the program that has several different properties that control its location and appearance.

Remarks

> So far, we've only been able to control our sprite's location and animation, but today, we're going to learn how to update other sprite properties so we can make even better programs.

Question of the Day: How can we use sprite properties to change their appearance on the screen?

Activity (35 minutes)

Transition: Send students to Code Studio.

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

1 **Prediction**

Video: Show students the **Sprite Properties** video in the slides.

Key Vocabulary:

- **Property:** A label for a characteristic of a sprite, such as its location and appearance
- **Dot notation:** the way that sprites' properties are used in Game Lab, by connecting the sprite and property with a dot.

Teaching Tip

To encourage active engagement and reflection, use one or more of the strategies discussed in the

Guide to Curriculum Videos.

2-3 Skill Building

2 3

4 Practice

5 Assessment

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

6 Challenges

Wrap Up (5 minutes)

Question of the Day: How can we use sprite properties to change their appearance on the screen?

Journal Prompt: What is one way sprite properties are the same as variables? What's one way that sprite properties are different from variables?

Discuss: Allow students to discuss in pairs or small groups before sharing out to the entire group.

Discussion Goal: Students may note that sprite properties and variables are similar in that they both store information. They are different in that variables can be anything, but sprites have particular properties that are used in certain ways on the screen.

Lesson 10: Text

45 minutes

Overview

This lesson introduces Game Lab's text commands, giving students more practice using the coordinate plane and parameters. This is the last type of element that students will be placing on the screen - after this, students will focus on how they can control the movement and interactions of these elements. The lesson begins with asking students to caption a cartoon created in Game Lab. They then move on to Code Studio where they practice placing text on the screen and controlling other text properties, such as size. Students who complete the assessment early can go on to learn more challenging blocks related to text properties.

Question of the Day: How can we use text to improve our scenes and animations?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
Journal

Activity (35 minutes)

Wrap up (5 minutes)
Journal

Objectives

Students will be able to:

- Place text on the screen using a coordinate plane.
- Use arguments to control how text is displayed on a screen.

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

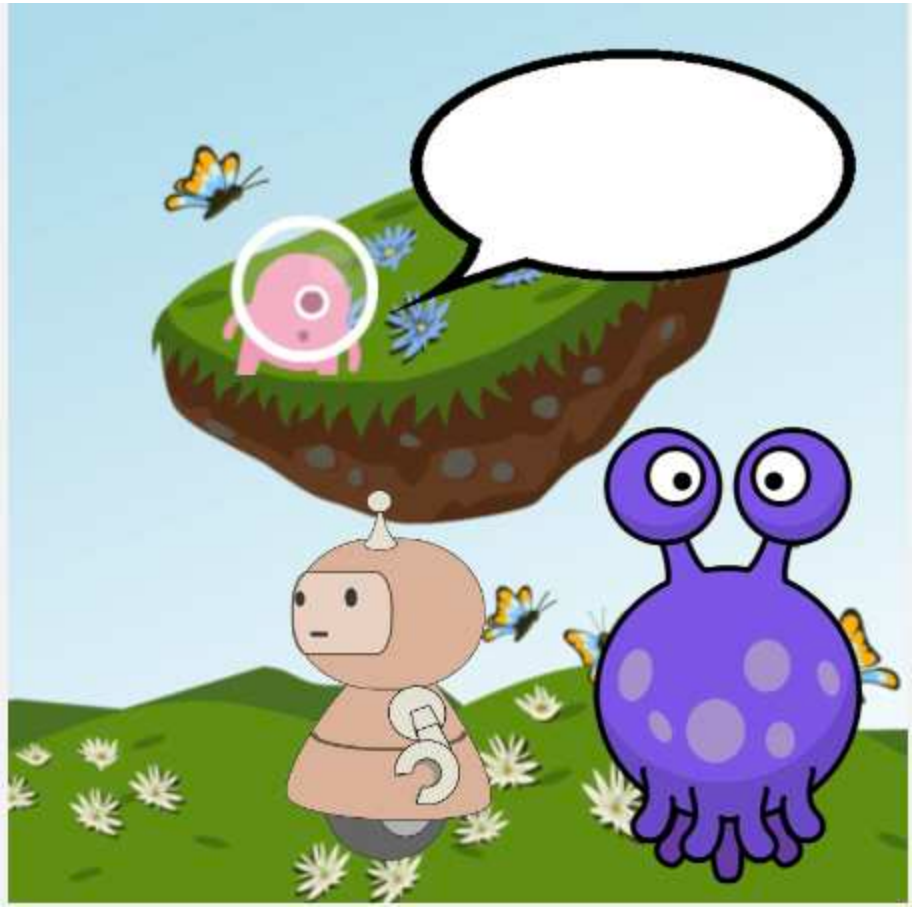
For the teachers

- **Text** - Slides ▾ Make a Copy

Teaching Guide

Warm Up (5 minutes)

Journal



Display: Display the cartoon for students to see.

Prompt: Look at the cartoon that was made in Game Lab. What do you think the alien should be saying?

Share: Allow volunteers to share out their ideas.

Remarks

> We've had a lot of fun drawing things and using our sprites, but there's been one thing missing from our Game Lab pictures: text! Today we're going to learn how to add text to Game Lab projects.

Question of the Day: How can we use text to improve our scenes and animations?

Activity (35 minutes)

Transition Send students to Code Studio.

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating

programming levels with students.

1 **Prediction**

2-3 **Skill Building**

2 3

4 **Practice**

5 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

6 **Challenges**

Wrap up (5 minutes)

Journal

Question of the Day: How can we use text to improve our scenes and animations?

Prompt: You've drawn with both text and shapes on the screen.

- What are two ways drawing with text is similar to drawing shapes?
- What is one way that drawing with text is different from drawing with shapes?

Share: If there is time, allow students to share out their answers.

Discussion Goal: Student answers to the question may vary, but some similarities are that they both use the coordinate plane and that they are drawn automatically, unlike sprites, which must use the `drawSprites()` command. One possible difference is that it is more difficult to control the exact size of text since the amount of text and font size are not as specific as height and width parameters.

Lesson 11: Mini-Project - Captioned Scenes

45 minutes

Overview

After a quick review of the code they have learned so far, students start working on their next creative project of the unit. Using the problem-solving process as a model again, students define the scene that they want to create, prepare by thinking of the different code they will need, try their plan in Game Lab, then reflect on what they have created. They also have a chance to share their creations with their peers. The open-ended nature of this lesson also provides flexibility for the teacher to decide how long students should spend on their work, depending on the scheduling demands of the particular course implementation.

Question of the Day: How can we use Game Lab to express our creativity?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
- **Review**

Activity (35 minutes)
- **Step 1: Define - Describe Your Scene**
- **Step 2: Prepare - Design Your Image**
- **Step 3: Try - Develop Your Scene**
- **Gallery Walk**

Wrap up (5 minutes)
- **Journal**

Objectives

Students will be able to:

- Use a structured process to plan and develop a program.

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Mini-Project - Captioned Scenes** - Slides ▾ Make a Copy

For the students

- **Captioned Scenes** - Rubric ▾ Make a Copy
- **Problem Solving with Programming** - Resource ▾ Make a Copy
- **Sprite Scene Planning** - Activity Guide ▾ Make a Copy

Teaching Guide

Warm Up (5 minutes)

Review

Prompt: Write down as many blocks as you can remember from Game Lab. Make sure you know what each one does, especially the inputs, or parameters, for each of the blocks.

Share: Allow students to share out what they remember as a group review.

Discussion Goal: The goal of this discussion is to review the different blocks and skills that students have learned. As students talk about the different blocks, try to steer the conversation into how they can be used creatively, to link to the main topic of the day.

Remarks

> You've learned how to do some really great things since our last mini-project in Game Lab. Today you'll have a chance to put them all together to make an interesting scene to share with the world. That means instead of trying to recreate someone else's idea, you're going to get to come up with an idea of your own, so it's time to get creative!

Question of the Day: How can we use Game Lab to express our creativity?

Teaching Tip

Facilitating Mini-Projects: Mini-Projects act as checkpoints in the curricula and cover the subset of skills students have seen so far in the unit. They are designed for 1-2 days of implementation as a way to check-in with how well students understand the course content so far. You may decide to extend these projects as a way to support or challenge students, which could allow you to revisit difficult concepts or support students who may have missed lessons and are trying to catch up. However, we recommend deciding this ahead of time and being firm with students about how much time they have for each project - otherwise, it's easy for projects to drag-out to multiple days and for student's work to spiral beyond the scope of this project.

Activity (35 minutes)

Distribute: Pass out copies of the activity guide. Students should use this worksheet to guide them through the Problem-Solving Process and plan out the captioned scene they create at the end of this lesson.

Optional Transition You can choose to either send students to Code Studio to see an example of a captioned scene in level 1 or display the example in the slides.

1 **Captioned Scene Example**

Teaching Tip

Facilitating Group Projects: If students are working in pairs or small teams to complete projects, consider showing these two videos to the class:

- **How Teamwork Works**

- **Dealing with Disagreements**

Depending on your goals with this project, consider having teams complete a **Student Guide to Team Planning**, which reinforces the message in the video

Step 1: Define - Describe Your Scene

Circulate: As students fill out a description of the captioned scene they want to create, circulate the room to ask students about their ideas and help guide them to make sure they are thinking of how they are going to use sprites and text to meet the project requirements.

Step 2: Prepare - Design Your Image

As students answer questions about the design of their captioned scene, continue to ensure that they are thinking about implementing sprites and text. After they have completed their designs, ensure that they can identify which blocks will be needed for the different features of their scene.

Teaching Tip

Scoping Student Projects: Students may ideate projects that are beyond the skills they currently have or that would take longer than the allotted time to implement. Rather than asking students to choose a different project, consider asking students to imagine a more scaled-down version of their initial idea. As an analogy, if a student's initial idea is the "Run" step, imagine a less intense version that represents what the "Walk" step would look like. If necessary, you can keep going back further to a "Crawl" step as well.

Digging Deeper: This is sometimes referred to as the Minimal Viable Product - you can learn more about this process and adapt it into your project strategies by reading this article: **Making Sense of MVP** by Henrik Kniberg

Step 3: Try - Develop Your Scene

Remarks

> Many of you are ready to start creating your programs. One thing that could make this challenging is the *blank screen* effect: unlike previous levels, you won't have any starter code or direction on what to create. This means you might not be sure what exactly you're supposed to do, or you might run into bugs you need to fix which can be frustrating. Luckily, we can also use the problem-solving process to help with these types of projects as well!

Distribute: Hand out a copy of the **Problem Solving with Programming** to pairs of students. Encourage students to look over the guide.

Display: Show the slide with the problem solving process graphic.

Remarks

> If you feel stuck or you're not sure what to do next, remember you can always follow the steps of the problem-solving process to **define** your next step, **prepare** for what you want to code, **try** it out, then **reflect** on whether or not it solved your problem.

Circulate: Encourage students to use the steps in the Problem-Solving Process for Programming when they get stuck or are unsure of what to do next.

Teaching Tip

Not all bullets in the Problem Solving Process will be applicable to every problem a student has. Instead, encourage them to pick one or two from each category to try each time they are stuck

After you have checked the designs, allow students to log into Code Studio and code their programs.

Transition: Send students to Code Studio.

2 Create a Background

Teaching Tip

Scaffolded Project: The tasks for this mini-project are broken down for students with each level focusing on a particular part of the project (background, sprites, text, etc). Encourage students to follow the instructions in each level, focusing on one task at a time.

3 Add Sprites

Teaching Tip

Debugging Strategies: As students design and implement their own project ideas, they may find themselves with new bugs that they need to untangle and you may find yourself looking at completely unfamiliar code as students look for help troubleshooting their errors. To help smooth out the debugging experience, consider the following strategies:

- Review the **Teacher Guide to Debugging** for some common questions and strategies to help support students in debugging their code
- Have students follow the steps in the **Student Guide to Debugging** and use the **Bug Report Quarter-Sheets** as an initial step in the debugging process. This helps students prepare and communicate their issue before asking for help.
- If students haven't seen it yet, consider showing the **Debugging Video** to the class to reinforce debugging best practices.

Digging Deeper: Consider supplying students with an object to talk to as part of the debugging process. This is sometimes known as Rubber Duck Debugging - you can learn more on the website **https://rubberduckdebugging.com/**

4 Add Text

5 Review Your Scene

Gallery Walk

Allow students to walk around the room and see the pictures that each of their classmates has coded. Celebrate all of the different ideas that students were able to implement with the same basic code.

Teaching Tip

You may choose to formalize this process by having each student write down one positive quality of each project, or having students "draw names" to comment on one particular classmate's work.

Assessment Opportunity

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

Wrap up (5 minutes)

Journal

Question of the Day: How can we use Game Lab to express our creativity?

Prompt: What was one especially creative way you saw someone else use the blocks today?

Share: Have students share out what they appreciated about their classmates' projects. You may want to do this "popcorn" style, with each student who responds choosing the next person to share.

Discussion Goal: This discussion should serve as a celebration of what the students have accomplished. As students share out what they have seen, encourage them to learn from each other and ask questions if they were not sure how to do something. Highlight how students were able to do very different things with the same tool.

Lesson 12: The Draw Loop

45 minutes

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. Students should leave the lesson understanding that the commands in the draw loop are called after all other code but are then called repeatedly to create animation. 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.

Question of the Day: How can we animate our images in Game Lab?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (35 minutes)

Wrap Up (5 minutes)

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
- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Flipbook Example - by Marnic Bos** - Video
- **Random Dot Flipbook**
- **Random Sprite Flipbook**
- **The Draw Loop** - Resource
- **The Draw Loop** - Slides
  ▾ Make a Copy

For the students

- **Introduction to the Draw Loop** - Video (**Download**)

Introduced Code

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

Teaching Guide

Warm Up (5 minutes)

Display: Show **Flipbook Example - by Marnic Bos - 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 share as a whole group.

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 being one of those many pictures--a single image within an animation. Then transition to the fact that soon students will be creating animations of their own.

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.

Question of the Day: How can we animate our images in Game Lab?

Activity (35 minutes)

Video: Show students the **Introduction to the Draw Loop** video in the slides.

Teaching Tip

To encourage active engagement and reflection, use one or more of the strategies discussed in the **Guide to Curriculum Videos**.

Question to Consider with Video:

- What does the draw function do?

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 the 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.

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

Do This: Send students to Code Studio.

The first prediction level program is the first time they will see the "draw loop" in action.

Discuss: Students should consider and discuss:

- What will be drawn on the screen do and why?

Discussion Goal: Predictions will vary but after viewing the Draw Loop video, students should be able to predict that the program will draw an ellipse over and over and over to the screen. Students may also predict and should notice once they run the program that the ellipse is drawn in random locations on the screen. One last thing to point out is the fact that the code to give the ellipse a color is *outside* the draw loop and runs *before* the program enters the draw loop.

1 Predict

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

2 Skill Building

The second prediction level program has one small difference that will make it run a little differently.

Discuss: Students should consider and discuss:

- What will this program do?

Discussion Goal: Notice that the blue background is never visible because it is immediately drawn over by the red background. Also, there's only ever one yellow dot visible since as the draw loop runs over and over it is placing down new backgrounds.

3 Predict

Teaching Tip

Inside versus outside the draw loop

This is a good time to make sure students understand that code **outside** the draw loop is used to **set up the program**. It is for how you want your program to start. Code **inside** the draw loop is for things that are **changing as the program is running**.

4-5 Skill Building

4 5

6 Practice

7 Assessment

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

8 Challenges

Wrap Up (5 minutes)

Question of the Day: How can we animate our images in Game Lab?

Prompt: How does the draw loop help us make animations?

Assessment Opportunity

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.

Review: Return to the resources students saw at the beginning of the lesson (Video, physical flip books) or under Help and Tips during the lesson. Address misconceptions that have arisen in the lesson.

Lesson 13: Sprite Movement

45 minutes

Overview

This lesson builds on the draw loop that students learned previously to create programs with *purposeful* motion. Students learn how to control sprite movement using a construct called the counter pattern, which incrementally changes a sprite's properties. Students first brainstorm different ways that they could animate sprites by controlling their properties, then explore the counter pattern in Code Studio. After examining working code, students try using the counter pattern to create various types of sprite movements. The skills that students build in this lesson lay the foundation for all of the animations and games that they will make throughout the rest of the unit.

Question of the Day: How can we control sprite movement in Game Lab?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
Reviewing Sprite Properties

Activity (35 minutes)

Wrap Up (5 minutes)
Journal

Objectives

Students will be able to:

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

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Sprite Movement** - Slides
  ▾ Make a Copy

For the students

- **Sprite Movement** - Video (**Download**)

Teaching Guide

Warm Up (5 minutes)

Reviewing Sprite Properties

Review: 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.

Prompt: What kinds of animations could you make if you could control these properties? Consider adding and subtracting from properties, or even updating multiple properties at the same time.

Discuss: Allow students to share their ideas with a partner before sharing with the entire class.

Discussion Goal: 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?

Question of the Day: How can we control sprite movement in Game Lab?

Activity (35 minutes)

Transition: Send students to Code Studio.

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

1 **Prediction**

Video: Show students the **Sprite Movement** video in the slides.

Teaching Tip

To encourage active engagement and reflection, use one or more of the strategies discussed in the **Guide to Curriculum Videos**.

Questions to Consider with Video:

- What is the counter pattern?
- How does the counter pattern move sprites across the screen?

Discussion Goal:

- *What is the counter pattern?*

- 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.

- *How does the counter pattern 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 for 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.

2-5 **Skill Building**

2 3 4 5

6 **Practice**

7 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

8 **Challenges**

Wrap Up (5 minutes)

Journal

Question of the Day: How can we control sprite movement in Game Lab?

Prompt: You've seen two ways to create animations with the draw loop: random numbers and the counter pattern. What is one type of movement that you'd want to use random numbers for? What is one type of movement that you would want to use the counter pattern for? Are there any movements that might combine the counter pattern and random numbers?

Discuss: Allow students to discuss with a partner before sharing with the entire class.

Discussion Goal: Student answers may vary, but movements such as shaking use random numbers, and movements that are more purposeful use the counter pattern. It's less important that students have a specific answer than that their answers reflect a growing understanding of how the different programming

constructs work.

Students may have seen random numbers used on one property (rotation) and the counter pattern used on another (x position) in one of the challenges, but if there is time, encourage them to think about what would happen if you used the counter pattern to add or subtract a random number to a sprite property, or to combine the two constructs in other ways.

Lesson 14: Mini-Project - Animation

45 minutes

Overview

This lesson is a chance for students to get more creative with what they have learned as students are asked to combine different methods that they have learned to create an animated scene. Students first review the types of movement and animation that they have learned and brainstorm what types of scenes might need that movement. They then begin to plan out their own animated scenes, which they create in Game Lab.

Question of the Day: How can we combine different programming patterns to make a complete animation?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
- **Review**

Activity (35 minutes)
- **Step 1: Define - Describe Your Scene**
- **Step 2: Prepare - Design Your Scene**
- **Step 3: Try - Develop Your Scene**
- **Gallery Walk**

Wrap up (5 minutes)
- **Journal**

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Mini-Project - Animation** - Slides
  - Make a Copy

For the students

- **Animated Scene** - Rubric
  - Make a Copy
- **Animated Scene** - Activity Guide
  - Make a Copy
- **Problem Solving with Programming** - Resource
  - Make a Copy

Teaching Guide

Warm Up (5 minutes)

Review

Prompt: Write down as many types of movement and animations as you can remember from the previous lesson. Make sure you know what blocks and patterns you need to make those movements, and when those movements would be useful.

Share: Allow students to share out what they remember as a group review.

Discussion Goal: The goal of this discussion is to review the different types of movement and animations that students have learned. As students talk about the different methods, make sure that they are mentioning why that type of movement would be useful. Press them to be specific about what they might animate using the various methods.

Remarks

> Now that we can control the way that our sprites move a little better, we're going to have a chance to put everything together to make an animation from scratch.

Question of the Day: How can we combine different programming patterns to make a complete animation?

Teaching Tip

Facilitating Mini-Projects: Mini-Projects act as checkpoints in the curricula and cover the subset of skills students have seen so far in the unit. They are designed for 1-2 days of implementation as a way to check-in with how well students understand the course content so far. You may decide to extend these projects as a way to support or challenge students, which could allow you to revisit difficult concepts or support students who may have missed lessons and are trying to catch up. However, we recommend deciding this ahead of time and being firm with students about how much time they have for each project - otherwise, it's easy for projects to drag-out to multiple days and for student's work to spiral beyond the scope of this project.

Activity (35 minutes)

Distribute: Pass out copies of the **Animated Scene**. Students should use this worksheet to guide them through the Problem-Solving Process and plan out the animated scene they create at the end of this lesson.

Teaching Tip

Facilitating Group Projects: If students are working in pairs or small teams to complete projects, consider showing these two videos to the class:

- **How Teamwork Works**
- **Dealing with Disagreements**

Depending on your goals with this project, consider having teams complete a **Student Guide to Team Planning**, which reinforces the message in the video

Optional Transition You can choose to either send students to Code Studio to see an example of an animated scene in level 1 or you can show the example in the slides.

1 **Example Animated Scene**

Step 1: Define - Describe Your Scene

Circulate: As students fill out a description of the animated scene they want to create, circulate the room to ask students about their ideas and help guide them to make sure they are thinking of how they could incorporate movement in a scene with the draw loop.

Step 2: Prepare - Design Your Scene

As students answer questions about the design of their scenes, continue to ensure that they are thinking about movement and using the draw loop. After they have completed their designs, ensure that they can identify which sprites will be needed and what sprite properties they need to animate for desired movement in their scene.

Teaching Tip

Scoping Student Projects: Students may ideate projects that are beyond the skills they currently have or that would take longer than the allotted time to implement. Rather than asking students to choose a different project, consider asking students to imagine a more scaled-down version of their initial idea. As an analogy, if students initial idea is the "Run" step, imagine a less intense version that represents what the "Walk" step would look like. If necessary, you can keep going back further to a "Crawl" step as well.

Digging Deeper: This is sometimes referred to as the Minimal Viable Product - you can learn more about this process and adapt it into your project strategies by reading this article: **Making Sense of MVP** by Henrik Kniberg

Step 3: Try - Develop Your Scene

Distribute: Hand out a copy of the **Problem Solving with Programming** to pairs of students or have them take this resource out if they still have it after the last project. Encourage students to look over the guide.

Display: Show the slide with the problem solving process graphic.

Remarks

> Many of you are ready to start creating your programs. Don't forget about using the problem solving process to help with the *blank screen* effect. If you feel stuck or you're not sure what to do next, remember you can always follow the steps of the problem-solving process to **define** your next step, **prepare** for what you want to code, **try** it out, then **reflect** on whether or not it solved your problem.

Circulate: Encourage students to use the steps in the Problem-Solving Process for Programming when they get stuck or are unsure of what to do next.

Teaching Tip

Not all bullets in the Problem Solving Process will be applicable to every problem a student has. Instead, encourage them to pick one or two from each category to try each time they are stuck

After you have checked the designs, allow students to log into Code Studio and code their programs.

Transition: Send students to Code Studio.

2 **Draw a Background**

Teaching Tip

Scaffolded Tasks: The tasks for this mini-project are broken down for students with each level focusing on a particular part of the project (background, sprites, text, etc). Encourage students to follow the instructions in each level, focusing on one task at a time.

3 Add Sprites

Teaching Tip

Debugging Strategies: As students design and implement their own project ideas, they may find themselves with new bugs that they need to untangle and you may find yourself looking at completely unfamiliar code as students look for help troubleshooting their errors. To help smooth out the debugging experience, consider the following strategies:

- Review the **Teacher Guide to Debugging** for some common questions and strategies to help support students in debugging their code
- Have students follow the steps in the **Student Guide to Debugging** and use the **Bug Report Quarter-Sheets** as an initial step in the debugging process. This helps students prepare and communicate their issue before asking for help.
- If students haven't seen it yet, consider showing the **Debugging Video** to the class to reinforce debugging best practices.

Digging Deeper: Consider supplying students with an object to talk to as part of the debugging process. This is sometimes known as Rubber Duck Debugging - you can learn more on the website **https://rubberduckdebugging.com/**

4 Add Text

5 Add Movement

6 Review Your Animated Scene

Gallery Walk

Allow students to walk around the room and see the pictures that each of their classmates has coded. Celebrate all of the different ideas that students were able to implement with the same basic code.

Teaching Tip

You may choose to formalize this process by having each student write down one positive quality of each project, or having students "draw names" to comment on one particular classmate's work.

Assessment Opportunity

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

Wrap up (5 minutes)

Journal

Question of the Day: How can we combine different programming patterns to make a complete animation?

Prompt: What was one interesting way that you saw sprite movement used today?

Share: Have students share out what they appreciated about their classmates' projects. You may want to do this "popcorn" style, with each student who responds choosing the next person to share.

Discussion Goal: This discussion should serve as a celebration of what the students have accomplished. As students share out what they have seen, encourage them to learn from each other and ask questions if they were not sure how to do something. Highlight how students were able to do very different things with the same tool.

Lesson 15: Conditionals

45 minutes

Overview

This lesson introduces booleans and conditionals, which allow a program to run differently depending on whether a condition is true. Students start by playing a short game in which they respond according to whether particular conditions are met. They then move to Code Studio, where they learn how the computer evaluates Boolean expressions, and how they can be used to structure a program.

Question of the Day: How can programs react to changes as they are running?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
Introducing Conditionals

Activity (35 minutes)
Conditionals

Wrap Up (5 minutes)
Considering Conditions

Objectives

Students will be able to:

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

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Booleans and Comparison Operators** - Resource
- **Conditionals** - Slides
  ▼ Make a Copy

For the students

- **Boolean Expressions** - Video (**Download**)
- **Conditional Statements** - Video (**Download**)
- **If Statements** - Resource

Vocabulary

- **Boolean Expression** - in programming, an expression that evaluates to True or False.

- **Condition** - Something a program checks to see whether it is true before deciding to take an action.
- **Conditionals** - Statements that only run when certain conditions are true.

Introduced Code

- `___ != ___`
- `___ < ___`
- `___ <= ___`
- `___ == ___`
- `___ > ___`
- `___ >= ___`
- `if (condition) { statement }`

Teaching Guide

Warm Up (5 minutes)

Introducing Conditionals

Display: Display the following questions on the board and ask students to answer them.

Teaching Tip

The last two prompts ask students to react to information that may change as they are answering the questions. In order to drive this point home, consider moving from the front of the room and back, as well as alternately tapping and not tapping a pencil, so that students realize that their answers may change according to when they are answering each question.

1. If your last name has more than five letters, draw a square on your paper.
2. If your last name less than seven letters, draw a circle.
3. If you are wearing anything green, add 3 + 2.
4. If the teacher is tapping their pencil, draw an 'X'.
5. If the teacher is in the front of the room, fold your paper in half.

Prompt: When we program, we give the computer instructions on what to do. How are the instructions you just followed different from the instructions that we have been giving in Game Lab?

Think-Pair-Share: After students have had a chance to write down their answers, allow them to talk to a partner before sharing out as a class.

Discussion Goal: This discussion should prepare students to investigate programs that react to changing conditions. Students should realize that there is no "one way" that these questions will be answered, and that the answers will change according to changing conditions.

Remarks

> These instructions were a little different because we first had to decide whether something was true or not before we knew what we should do. In programming, instructions that depend on whether or not something is true are called **conditionals**, and the thing that is checked is called the **condition**. Conditionals are especially useful when we want the program to react to situations that change while the program is running.

Key Vocabulary:

- **Condition** - Something a program checks to see whether it is true before deciding whether to take an action.
- **Conditionals** - Statements that only run under certain conditions.

Question of the Day: How can programs react to changes as they are running?

Activity (35 minutes)

Conditionals

Transition: Send students to Code Studio.

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

1 **Prediction**

Video: Show students the **Boolean Expressions** video in the slides.

Teaching Tip

To encourage active engagement and reflection, use one or more of the strategies discussed in the **Guide to Curriculum Videos**.

Questions to Consider with Video:

- What is a Boolean expression?
- What's an expression that would evaluate to true?
- What's an expression that would evaluate to false?

Discussion Goal:

- *What is a Boolean expression?*
  - **Boolean Expression** - in programming, an expression that evaluates to True or False.
  - 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.

- *What's an expression that would evaluate to true?*
  - Some examples of Boolean expressions that evaluate to true are `3 > 1` and `4 <= 7`, but press students to think of expressions that might be better represented by variables, such as `studentAge < 70` or `sizeOfClass > 2`.
- *What's an expression that would evaluate to false?*
  - Some examples of Boolean expressions that evaluate to false are `4 == 7`, `schoolName == "Hogwarts"`, and `currentYear < 1000`.

2 **Quick Check**

3-4 **Skill Building**

3 4

Video: Show students the **Conditional Statements** video in the slides.

Question to Consider with Video:

- When would you want to use an if statement?

Discussion Goal: 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.

5 **Skill Building**

6 **Practice**

7 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

8 **Challenges**

Wrap Up (5 minutes)

Considering Conditions

Question of the Day: How can programs react to changes as they are running?

Key Vocabulary:

- **Condition** - Something a program checks to see whether it is true before deciding to take an action.
- **Conditionals** - Statements that only run under certain conditions.
- **Boolean Expression** - In programming, an expression that evaluates to True or False.

Prompt: Now that you know how conditionals work, where you do think that they are used in games or other programs and apps that you already use?

Discuss: Have students share responses.

Discussion Goal: This discussion should get students thinking about how conditionals are used in games they have already seen, and help them connect those ideas to how they might want to use conditionals in their own programs. 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

Lesson 16: Keyboard Input

45 minutes

Overview

One common way conditionals are used is to check for different types of user input, especially key presses. 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.

Question of the Day: How can our programs react to user input?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
Taking Input

Activity (35 minutes)
Keyboard Input

Wrap Up (5 minutes)
Adding Conditionals

Objectives

Students will be able to:

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

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Keyboard Input** - Slides
  ▼ Make a Copy

Introduced Code

- `keyDown(code)`

Teaching Guide

Warm Up (5 minutes)

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 Goal: 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 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?

Question of the Day: How can our programs react to user input?

Activity (35 minutes)

Keyboard Input

Transition: Send students to Code Studio

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

1 **Prediction**

2-4 **Skill Building**

2 3 4

5 **Practice**

6 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

7 **Challenges**

Wrap Up (5 minutes)

Adding Conditionals

Question of the Day: How can our programs react to user input?

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

Lesson 17: Mouse Input

45 minutes

Overview

In this lesson, students continue to explore ways to use conditional statements to take user input - this time with the mouse. They will also expand their understanding of conditionals to include *else*, which allows for the computer to run a certain section of code when a condition is true, and a different section of code when it is not. This concept is introduced alongside several new mouse input commands, allowing students to gradually build up programs that use input in different ways.

Question of the Day: What are more ways that the computer can react to user input?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
- **3-2-1 Review**

Activity (35 minutes)
- **If/Else and More Input**

Wrap Up (5 minutes)
- **Wrap Up**

Objectives

Students will be able to:

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

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **If-Else Statements** - Resource
- **Mouse Input** - Slides ▾ Make a Copy

For the students

- **If/Else Statements** - Video (**Download**)

Vocabulary

- **Conditionals** - Statements that only run when certain conditions are true.

Introduced Code

- `if (condition) { statement1 } else { statement`
- `keyWentDown(code)`
- `keyWentUp(code)`
- `mouseDidMove()`

- `mouseDown(button)`
- `mouseWentDown(button)`
- `mouseWentUp(button)`
- `sprite.visible`

Teaching Guide

Warm Up (5 minutes)

3-2-1 Review

Prompt: What are three different things that you've been able to do with conditionals? What are two big things that everyone should remember when using conditionals? What's one thing you still want to learn how to program?

Share: Allow students to share out their responses.

Discussion Goal: Students may come up with many different specific scenarios, but make sure that they are differentiating between conditionals that respond to user input and those that respond to other changes in the program, such as a score increase, the location of a sprite, or player lives lost. Major things to remember may include putting the conditionals in the draw loop or paying attention to how the blocks are nested.

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.

Question of the Day: What are more ways that the computer can react to user input?

Activity (35 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.

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

1 **Prediction**

Video: Show students the **If/Else Statements** video in the slides.

Teaching Tip

To encourage active engagement and reflection, use one or more of the strategies discussed in the

Guide to Curriculum Videos.

Questions to Consider with Video:

- What's an example of when you would need an "if/else" statement?

Discussion Goal: 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 `if()/else()` block is necessary.

2-5 **Skill Building**

2 3 4 5

6 **Practice**

7 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

8 **Challenges**

Wrap Up (5 minutes)

Wrap Up

Question of the Day: What are more ways that the computer can react to user input?

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.

Lesson 18: Project - Interactive Card

90 minutes

Overview

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. In this project, students plan for and develop an interactive greeting card using all of the programming techniques they've learned to this point. Giving students the opportunity to really be creative after learning all these new concepts will help to engage them further as they head into Chapter 2.

Question of the Day: What skills and practices are important when creating an interactive program?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
- **Journal**

Activity (80 minutes)
- **Demo Project Exemplars (Level 1)**
- **Step 1: Define - Plan Your Card**
- **Step 2: Prepare - Design Your Card**
- **Step 3: Try - Develop Your Card**
- **Peer Review**
- **Iterate - Update Code**
- **Step 4: Reflect**

Wrap Up (5 minutes)
- **Reflection**

Objectives

Students will be able to:

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

Preparation

- Print out a copy of the project guide for each student
- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Extra Code in Challenge Levels** - Resource ▾ Make a Copy
- **Project - Interactive Card** - Slides ▾ Make a Copy

For the students

- **Computer Science Practices** - Activity Guide ▾ Make a Copy
- **Interactive Card** - Rubric ▾ Make a Copy

- **Interactive Card - Activity Guide** - Activity Guide ▾ Make a Copy
- **Interactive Card - Peer Review** - Activity Guide ▾ Make a Copy
- **Interactive Card - Student Checklist** - Resource ▾ Make a Copy
- **Problem Solving with Programming** - Resource ▾ Make a Copy

Teaching Guide

Warm Up (5 minutes)

Journal

Prompt: Think of one time you gave or received a card from someone. Who was that person? What was the purpose of the card? What about the card made it specific to that purpose?

Share: Have students share out their ideas.

Discussion Goal: This discussion introduces the chapter project: creating an interactive card. This is a good chance to tie in the 'card' concept to the problem-solving process, by defining the "problem" as cheering someone up, or letting them know that we are thinking about them.

Remarks

> We're going to start designing our own cards today. Because we have learned how to program, our cards are going to be interactive. At the end of the project, you'll be able to email or text your card to someone.

Question of the Day: What skills and practices are important when creating an interactive program?

Activity (80 minutes)

Demo Project Exemplars (Level 1)

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

Display: Show students the sample program.

1 **Demo App: Interactive Card**

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

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 and student checklist. Review the different components of the rubric with them to make sure they understand the components of the project.

Distribute: Pass out copies of the **Interactive Card - Activity Guide**. Student should use this activity guide to 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. Give students some time to brainstorm the type of card they want to create and who the recipient will be.

Also distribute a copy of the **Interactive Card - Student Checklist** to each student. Students can use this to self-assess and reflect as they develop their project.

Teaching Tip

Scoping Student Projects: Students may ideate projects that are beyond the skills they currently have or that would take longer than the allotted time to implement. Rather than asking students to choose a different project, consider asking students to imagine a more scaled-down version of their initial idea. As an analogy, if students initial idea is the "Run" step, imagine a less intense version that represents what the "Walk" step would look like. If necessary, you can keep going back further to a "Crawl" step as well.

Digging Deeper: This is sometimes referred to as the Minimal Viable Product - you can learn more about this process and adapt it into your project strategies by reading this article: **Making Sense of MVP** by Henrik Kniberg

Step 1: Define - Plan Your Card

Circulate: As students fill out a description of the interactive card they want to create, circulate the room to ask students about their ideas and help guide them to make sure they are thinking of how they could include all concepts they've learned about so far.

Step 2: Prepare - Design Your Card

As students fill in the activity guide about the design of their card, continue to support student ideas and brainstorms as they are asked to consider the different aspects of their card:

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 with each sprite's label, images, and properties.
3. Finally, students consider the conditionals they'll need in order to make their card interactive.

Optional: Students can visit Code Studio - Level 2 for more interactive card ideas if needed.

2 **Interactive Card Examples**

Teaching Tip

New Code and Previous Challenge Levels: Throughout the unit, students may have learned additional code in the challenge levels of different lessons. As students approach this project, they may want to revisit the code they learned in these levels or visit them for the first time to learn new codes to use in

this project. To help guide students back to previous levels, you can use the **Extra Code in Challenge Levels** as a resource to quickly find where new codes were introduced in earlier challenge levels.

Step 3: Try - Develop Your Card

Remarks

> Many of you are ready to start creating your programs. Don't forget about using the problem solving process to help with the *blank screen* effect. If you feel stuck or you're not sure what to do next, remember you can always follow the steps of the problem-solving process to **define** your next step, **prepare** for what you want to code, **try** it out, then **reflect** on whether or not it solved your problem.

Distribute: Hand out a copy of the **Problem Solving with Programming** to pairs of students or have students take this resource out if they kept it after the last project. Encourage students to look over the guide.

Display: Show the slide with the problem solving process graphic.

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

Teaching Tip

Scaffolded Tasks: The tasks for this mini-project are broken down for students with each level focusing on a particular part of the project (background, sprites, text, etc). Encourage students to follow the instructions in each level, focusing on one task at a time.

Variable Names Take an opportunity to go over the importance of naming variables with descriptive names. (Note that the Exemplar uses Surpise1 and Suprise2 as variable names so that it doesn't give away what the surprises are). You can have a discussion around variable names and how it can impact their program if they have vague names making it harder to identify their variables later as they code.

Circulate: Encourage students to use the steps in the Problem-Solving Process for Programming when they get stuck or are unsure of what to do next.

3-7 **Project Work**

3 4 5 6 7

Teaching Tip

Debugging Strategies: As students design and implement their own project ideas, they may find themselves with new bugs that they need to untangle and you may find yourself looking at completely unfamiliar code as students look for help troubleshooting their errors. To help smooth out the debugging experience, consider the following strategies:

- Review the **Teacher Guide to Debugging** for some common questions and strategies to help support students in debugging their code
- Have students follow the steps in the **Student Guide to Debugging** and use the **Bug Report Quarter-Sheets** as an initial step in the debugging process. This helps students prepare and communicate their issue before asking for help.
- If students haven't seen it yet, consider showing the **Debugging Video** to the class to reinforce debugging best practices.

Digging Deeper: Consider supplying students with an object to talk to as part of the debugging process. This is sometimes known as Rubber Duck Debugging - you can learn more on the website **https://rubberduckdebugging.com/**

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.

Students should use the rubric and student checklist to self-assess and make sure their project matches with the rubric.

Teaching Tip

Rubric and Checklist: Students have two resources they can use for self-reflection and making sure they are on the right track: the rubric and the student checklist. We recommend having students use the checklist for their own self-assessment and reflection, since it may be easier to digest and understand when reviewing their own project. However, we recommend teachers use the full rubric for evaluating projects to give more accurate feedback to students. You can see examples of this with the Sample Marked Rubrics resource at the top of the lesson plan (only visible to verified teachers)

Step 4: 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.

8 **Reflection**

Assessment Opportunity

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

Wrap Up (5 minutes)

Reflection

Question of the Day: What skills and practices are important when creating an interactive program?

Prompt: Have students reflect on their development of the five practices of CS Discoveries (Problem Solving, Persistence, Creativity, Collaboration, Communication).

- Choose one of the five practices which you demonstrated growth in during this lesson. Write something you did that showed 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 this project. What made it so important?

Lesson 19: Velocity

45 minutes

Overview

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. After a brief review of how they used the counter pattern to move sprites in previous lessons, students are introduced to the idea of hiding those patterns in a single `velocity` block. Students then head to Code Studio to try out new blocks that set a sprite's velocity directly, and look at various ways that they are able to code more complex behaviors in their sprites. Over the next several lessons, students will see how this method of managing complexity allows them to produce more interesting sprite behaviors.

Question of the Day: How can programming languages hide complicated patterns so that it is easier to program?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (35 minutes)

Wrap Up (5 minutes)

Journal

Objectives

Students will be able to:

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

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Velocity** - Resource
- **Velocity** - Slides ▾ Make a Copy

For the students

- **Velocity** - Video (**Download**)

Introduced Code

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

Teaching Guide

Warm Up (5 minutes)

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 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.

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."

Remarks

> Programming languages also have ways to simplify things for us. Today, we're going to look at some blocks in Game Lab that hide complicated coding patterns to make things easier for programmers.

Question of the Day: How can programming languages hide complicated patterns so that it is easier to program?

Activity (35 minutes)

Remarks

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

Transition: Move students to Code Studio

1 **Predict**

Discussion Goal: Student predictions will vary but some may predict that the new `velocity` block might make the sprite move faster than the counter pattern. Once they run the program, students will see that the two lines of code cause the same sprite movement - the difference being the `velocity` block hides the details of the counter pattern *and* happens *outside* the draw loop.

Video: Show students the **Velocity** video in the slides.

Teaching Tip

To encourage active engagement and reflection, use one or more of the strategies discussed in the **Guide to Curriculum Videos**.

Questions to Consider with Video:

- Why might you want to use a velocity block instead of the counter pattern?
- Give an example of a counter pattern and how you could use a velocity block instead.

Discussion Goal:

- *Why might you want to use a velocity block instead of the counter pattern?*
  - 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 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.
- *Give an example of a counter pattern and how you could use a velocity block instead.*
  - 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.

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.

2-7 **Skill Building**

2 3 4 5 6 7

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

Teaching Tip

Inside versus outside the draw loop

This is a good time to remind students that code outside the draw loop is used to set up the program. It is for how you want your program to start. Code inside the draw loop is for things that are changing as the program is running, user interaction.

There may be some confusion that the new blocks are animation (changing position) and yet have gone outside the draw loop up until this point. That is because up until this point, the velocity has been set at the beginning of the program and not changed. When students want the velocity to change during the program, it will need to go inside the draw loop.

8 **Practice**

9 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

10 **Challenges**

Wrap Up (5 minutes)

Journal

Prompt: You learned a few new blocks today. At 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.

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

- 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 brainpower 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.

Lesson 20: Collision Detection

45 minutes

Overview

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. Working in pairs, students further explore this idea in the context of the intentionally complex mathematical challenge of determining whether two sprites are touching. Students then use a single block, the `isTouching()` block, to represent this complexity and to create different effects when sprites collide. By the end of the lesson, students should understand that by using a single block to represent this complexity, it becomes much easier to write and reason about code and appreciate the value of using abstractions.

Question of the Day: How can programming help make complicated problems more simple?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (35 minutes)

- **Collisions Unplugged**
- **isTouching()**

Wrap Up (5 minutes)

Objectives

Students will be able to:

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

Preparation

- Print copies of the activity guide such that each pair of students has a part A and a part B
- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Collision Detection** - Resource
- **Collision Detection** - Slides ▾ Make a Copy

For the students

- **Collision Detection (Version A)** - Activity Guide ▾ Make a Copy
- **Collision Detection (Version B)** - 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.

Introduced Code

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

Teaching Guide

Warm Up (5 minutes)

Display: On a projector or a computer screen, demonstrate the game that appears in the first level in Code Studio for this lesson.

1 **Sample Game**

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.

Prompt: An interesting aspect of this animation is that the sprites change when they touch each other. 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.

Discussion Goal: The 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.

Remarks

> This is a tough problem, and we're going to get to dig into it today. As we work on it, we're also going to look at ways that the computer can help us make these tough, complicated problems more simple.

Question of the Day: How can programming help make complicated problems more simple?

Activity (35 minutes)

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. 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: Send students to Code Studio.

2-5 **Skill Building**

2 3 4 5

Teaching Tip

Level 2: 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. 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.

This code does not include the y and height properties because the two sprites are interacting on the same horizontal line. If the bunny could move diagonally, then the code would be even more complicated.

Level 5: This level does not ask students to fix the bug in the program, which must be done in the animation tab. In order to make the collision work properly, students will need to crop the empty space around the visible part of the picture. The easiest way to do this is to click once on the "crop" icon in the animation tab, which will tightly crop to the smallest rectangle around the visible parts of the picture. Students may also use the rectangular select tool to specify what should be cropped away.

6 **Practice**

7 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

8 **Challenges**

Wrap Up (5 minutes)

Question of the Day: How can programming help make complicated problems more simple?

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?

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.

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. Using a simplified representation of something to hide the details so you can think about the big picture is called "abstraction", and it's a key tool programmers use to help them write complicated programs.

Key Vocabulary:

- **abstraction** - a simplified representation of something more complex.

Lesson 21: Mini-Project - Side Scroller

45 minutes

Overview

This lesson is another chance for students to get more creative with what they have learned. Students use what they have learned about collision detection and setting velocity to create a simple side-scroller game. After looking at a sample side-scroller game, students brainstorm what sort of side-scroller they would like to make, then use a structured process to program the game in Code Studio. This lesson can be shortened or lengthened depending on time constraints.

Question of the Day: How can the new types of sprite movement and collision detection be used to create a game?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (35 minutes)

- **Step 1: Define - Plan Your Game**
- **Step 2: Prepare - Design Your Game**
- **Step 3: Try - Program Your Game**
- **Step 4: Reflect**

Wrap up (5 minutes)

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Mini-Project - Side Scroller** - Slides
  - Make a Copy

For the students

- **Problem Solving with Programming** - Resource
  - Make a Copy
- **Side Scroller** - Rubric
  - Make a Copy
- **Side Scroller** - Activity Guide
  - Make a Copy

Teaching Guide

Warm Up (5 minutes)

Review

Ask students to think of all of the things that they have learned how to do in the unit so far, and display their answers to the class. This is a good time to check in on any concepts that have been challenging for students.

Remarks

> Now that you've learned how to detect sprite interactions, you can start making some more interesting games. Today, we're going to look at how you can use what you've learned to make a side scroller game.

Question of the Day: How can the new types of sprite movement and collision detection be used to create a game?

Teaching Tip

Facilitating Mini-Projects: Mini-Projects act as checkpoints in the curricula and cover the subset of skills students have seen so far in the unit. They are designed for 1-2 days of implementation as a way to check-in with how well students understand the course content so far. You may decide to extend these projects as a way to support or challenge students, which could allow you to revisit difficult concepts or support students who may have missed lessons and are trying to catch up. However, we recommend deciding this ahead of time and being firm with students about how much time they have for each project - otherwise, it's easy for projects to drag-out to multiple days and for student's work to spiral beyond the scope of this project.

Activity (35 minutes)

Distribute: Pass out copies of the **Side Scroller**. Student should use this activity guide to plan out and brainstorm how to program the side scroller game before they head to the computer so that once they get to the computer they are just executing the plan.

Optional Transition You can choose to either send students to Code Studio to play the Side Scroller example game or demo the game for the students.

1 **Side Scroller Example**

Step 1: Define - Plan Your Game

Circulate: As students fill out a description of the aspects of the side scroller they want to keep the same and what they want to change, circulate the room to ask students about their ideas and help guide them.

Step 2: Prepare - Design Your Game

As students fill in the activity guide about the side scroller game, continue to support student ideas and brainstorms as they are asked to consider the different aspects of programming the game:

1. Students are asked to identify parts of programming the game that they are not sure how to do yet. If students identify concepts that have previously been covered, suggest they go back and review.
2. Next, students are given the opportunity to identify new animations for the three sprites in the game.
3. The first layer of the game is a background. The activity guide provides a space for students to lay out their background and an area for students to take notes and write pseudocode.
4. Next, students think through how each sprite is programmed to move and interact with the other sprites and given an opportunity to decide if they want to change anything.

Step 3: Try - Program Your Game

Remarks

> Many of you are ready to start creating your game. Don't forget about using the problem solving process to help with the *blank screen* effect. If you feel stuck or you're not sure what to do next, remember you can always follow the steps of the problem-solving process to **define** your next step, **prepare** for what you want to code, **try** it out, then **reflect** on whether or not it solved your problem.

Distribute: Hand out a copy of the **Problem Solving with Programming** to pairs of students or have students take this resource out if they kept it after the last project. Encourage students to look over the guide.

Display: Show the slide with the problem solving process graphic.

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

2 Draw Your Background

Teaching Tip

Scaffolded Tasks: The tasks for this mini-project are broken down for students with each level focusing on a particular part of the project (background, sprites, player controls, etc). Encourage students to follow the instructions in each level, focusing on one task at a time.

Circulate: Encourage students to use the steps in the Problem-Solving Process for Programming when they get stuck or are unsure of what to do next.

3 Create Your Sprites

Teaching Tip

Debugging Strategies: As students design and implement their own project ideas, they may find themselves with new bugs that they need to untangle and you may find yourself looking at completely unfamiliar code as students look for help troubleshooting their errors. To help smooth out the debugging experience, consider the following strategies:

- Review the **Teacher Guide to Debugging** for some common questions and strategies to help support students in debugging their code
- Have students follow the steps in the **Student Guide to Debugging** and use the **Bug Report Quarter-Sheets** as an initial step in the debugging process. This helps students prepare and communicate their issue before asking for help.
- If students haven't seen it yet, consider showing the **Debugging Video** to the class to reinforce debugging best practices.

Digging Deeper: Consider supplying students with an object to talk to as part of the debugging process. This is sometimes known as Rubber Duck Debugging - you can learn more on the website **https://rubberduckdebugging.com/**

4 Player Controls

5 Looping

6 Sprite Interactions

7 Scoring & Scoreboard

8 Review Your Game

Assessment Opportunity

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

Step 4: Reflect

Direct students to complete the last step of their activity guide to reflect on how they did programming this game.

Wrap up (5 minutes)

Question of the Day: How can the new types of sprite movement and collision detection be used to create a game?

Prompt: What was one challenge in making this game? What is your advice for someone else who has the same challenge?

Lesson 22: Complex Sprite Movement

45 minutes

Overview

This lesson does not introduce any new blocks and in fact, only uses patterns students have seen in Chapter 1 and demonstrates how combining these patterns, in particular the abstractions students learned in the previous two lessons, allows them to build new behaviors for their sprites. Specifically, this lesson has students learn how to combine the velocity properties of sprites with the counter pattern to create more complex sprite movement. After reviewing the two concepts, they explore various scenarios in which velocity is used in the counter pattern and observe the different types of movement that result, such as simulating gravity. They then reflect on how they were able to get new behaviors by combining blocks and patterns that they already knew.

Question of the Day: How can previous blocks be combined in new patterns to make interesting movements?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (35 minutes)

Wrap Up (5 minutes)

Objectives

Students will be able to:

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

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Complex Sprite Movement** - Slides ▾ Make a Copy
- **Velocity and the Counter Pattern** - Resource

Teaching Guide

Warm Up (5 minutes)

Display: Show the two images of a frog jumping to the class.

Prompt: Here are two images of a frog jumping. The first is from the side scroller. Do you have any ideas for how to make the second type of jumping?

Think-Pair-Share Allow students to discuss their ideas with a classmate before sharing with the entire class.

Discussion Goal: It is unlikely that students will come up with the solution on their own, but encourage them to think of as many ideas as possible, and that they will keep working on the problem throughout the class.

Remarks

> We've learned a lot of new blocks that have helped us create some fun animations. Today, we're going to look at how we can use the blocks that we already know in new ways to make more interesting types of movements. By the end of the class, you'll be able to code the new type of jumping with blocks that you already know.

Question of the Day: How can previous blocks be combined in new patterns to make interesting movements?

Activity (35 minutes)

Transition: Move students to Code Studio.

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

1 **Prediction**

Teaching Tip

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

2-4 **Skill Building**

2 3 4

5 Practice

6 Assessment

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

7 Challenges

Wrap Up (5 minutes)

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 on 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 sprites to do after today's lesson. On the second list write down all the new blocks you learned today.

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.

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.

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.

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.

Lesson 23: Collisions

45 minutes

Overview

This lesson introduces collisions, another useful abstraction that will allow students to manipulate their sprites in entirely 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`). This is the last time they will learn a new sprite behavior, and following this lesson students will transition to focusing on how they organize their increasingly complex code.

Question of the Day: How can programmers build on abstractions to create further abstractions?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (35 minutes)

Wrap Up (5 minutes)

Objectives

Students will be able to:

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

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Collisions** - Slides ▾ 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.

Introduced Code

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

Teaching Guide

Warm Up (5 minutes)

Display: Display the animated image. It is also available as a level in code studio.

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

1 **Code Prediction**

Share: Allow students to share out their ideas.

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.

Remarks

> The first part of the problem is figuring out when the two sprites are touching, but we already figured out how to do that and can now use the **`isTouching`** block. That means we don't need to think about those details anymore. Using abstraction to hide the complicated details in that part of the problem means we can focus on the new part.

Teaching Tip

Students have seen this vocabulary before, but given its importance to the chapter, it is introduced again here.

Vocabulary Review:

- **abstraction** - a simplified representation of something more complex

Question of the Day: How can programmers build on abstractions to create further abstractions?

Activity (35 minutes)

Transition: Move students onto Code Studio - level 2 where they will code the idea they made during the warm-up to make the giraffe sprite push the monkey sprite across the screen. When students finish level 2, have them try level 3 where they will make an elephant sprite push a hippo sprite *down* the screen.

Prompt: These were challenging problems, but we were able to solve them. What helped us to solve these problems?

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.

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.

Transition: Have students continue working on the remaining levels.

2-5 **Skill Building**

2 3 4 5

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

6 **Practice**

7 **Assessment**

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

8 **Challenges**

Wrap Up (5 minutes)

Question of the Day: How can programmers build on abstractions to create further abstractions?

Vocabulary Review:

- **abstraction** - a simplified representation of something more complex

Prompt: How did having the `isTouching` block and the `velocityX` block make it easier to solve the problem of one sprite pushing another?

Assessment Opportunity

Students should understand that these two blocks represent partial solutions to the problem of one sprite pushing the other, and that by hiding the details of those partial solutions, they can more easily focus on how to fit those partial solutions together to solve larger and more complex problems.

Lesson 24: Mini-Project - Flyer Game

45 minutes

Overview

This lesson is another chance for students to get more creative with what they have learned. Students use what they have learned about simulating gravity and the different types of collisions to create simple flyer games. After looking at a sample flyer game, students brainstorm what sort of flyer games they would like, then use a structured process to program the game in Code Studio.

Question of the Day: How can the new types of collisions and modeling movement be used to create a game?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (35 minutes)

- **Step 1: Define - Plan the Game**
- **Step 2: Prepare - Design Your Game**
- **Step 3: Try - Program Your Game**
- **Step 4: Reflect**

Wrap up (5 minutes)

- **Share Out and Journal 3-2-1**

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Mini-Project - Flyer Game** - Slides
  - Make a Copy

For the students

- **Flyer Game** - Rubric
  - Make a Copy
- **Flyer Game** - Activity Guide
  - Make a Copy
- **Problem Solving with Programming** - Resource
  - Make a Copy

Teaching Guide

Warm Up (5 minutes)

Review

Ask students to think of all of the things that they have learned how to do in the unit so far, and display their answers to the class. This is a good time to check in on any concepts that have been challenging for students.

Remarks

> You've already learned all of the sprite interactions and types of movement that we will cover this unit. Today you'll have a chance to put them all together to make a flyer game.

Question of the Day: How can the new types of collisions and modeling movement be used to create a game?

Teaching Tip

Facilitating Mini-Projects: Mini-Projects act as checkpoints in the curricula and cover the subset of skills students have seen so far in the unit. They are designed for 1-2 days of implementation as a way to check-in with how well students understand the course content so far. You may decide to extend these projects as a way to support or challenge students, which could allow you to revisit difficult concepts or support students who may have missed lessons and are trying to catch up. However, we recommend deciding this ahead of time and being firm with students about how much time they have for each project - otherwise, it's easy for projects to drag-out to multiple days and for student's work to spiral beyond the scope of this project.

Activity (35 minutes)

Distribute: Pass out copies of the **Flyer Game**. Student should use this activity guide to plan out and brainstorm how to program the flyer game before they head to the computer so that once they get to the computer they are just executing the plan.

Optional Transition You can choose to either send students to Code Studio to play the flyer game example game or demo the game for the students.

1 **Flyer Game Example**

Step 1: Define - Plan the Game

Circulate: As students fill out a description of the aspects of the flyer game they want to keep the same and what they want to change, circulate the room to ask students about their ideas and help guide them.

Step 2: Prepare - Design Your Game

As students fill in the activity guide about the flyer game, continue to support student ideas and brainstorms as they are asked to consider the different aspects of programming the game:

1. Students are asked to identify parts of programming the game that they are not sure how to do yet. If students identify concepts that have previously been covered, suggest they go back and review.
2. Next, students are given the opportunity to identify new animations for the three sprites in the game.
3. The first layer of the game is a background. The activity guide provides a space for students to lay out their background and an area for students to take notes and write pseudocode.
4. Next, students think through how each sprite is programmed to move and interact with the other sprites and given an opportunity to decide if they want to change anything.

Step 3: Try - Program Your Game

Remarks

> Many of you are ready to start creating your game. Don't forget about using the problem solving process to help with the *blank screen* effect. If you feel stuck or you're not sure what to do next, remember you can always follow the steps of the problem-solving process to **define** your next step, **prepare** for what you want to code, **try** it out, then **reflect** on whether or not it solved your problem.

Distribute: Hand out a copy of the **Problem Solving with Programming** to pairs of students or have students take this resource out if they kept it after the last project. Encourage students to look over the guide.

Display: Show the slide with the problem solving process graphic.

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

2 **Make Your Sprites**

Teaching Tip

Scaffolded Tasks: The tasks for this mini-project are broken down for students with each level focusing on a particular part of the project (background, sprites, player controls, etc). Encourage students to follow the instructions in each level, focusing on one task at a time.

Circulate: Encourage students to use the steps in the Problem-Solving Process for Programming when they get stuck or are unsure of what to do next.

3-5 **Player Controls**

3 4 5

Teaching Tip

Debugging Strategies: As students design and implement their own project ideas, they may find themselves with new bugs that they need to untangle and you may find yourself looking at completely unfamiliar code as students look for help troubleshooting their errors. To help smooth out the debugging experience, consider the following strategies:

- Review the **Teacher Guide to Debugging** for some common questions and strategies to help support students in debugging their code
- Have students follow the steps in the **Student Guide to Debugging** and use the **Bug Report Quarter-Sheets** as an initial step in the debugging process. This helps students prepare and communicate their issue before asking for help.
- If students haven't seen it yet, consider showing the **Debugging Video** to the class to reinforce debugging best practices.

Digging Deeper: Consider supplying students with an object to talk to as part of the debugging process. This is sometimes known as Rubber Duck Debugging - you can learn more on the website **https://rubberduckdebugging.com/**

6 **Sprite Movement**

7 **Sprite Interactions**

8 **Review Your Game**

Assessment Opportunity

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

Step 4: Reflect

Direct students to complete the last step of their activity guide to reflect on how they did programming this game.

Wrap up (5 minutes)

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 piece of advice you'd give to someone making this type of game?

Question of the Day: How can the new types of collisions and modeling movement be used to create a game?

Lesson 25: Functions

45 minutes

Overview

In previous lessons, students have learned to use a number of abstractions in their programs which 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 that will serve to organize their code, make it more readable, and remove repeated blocks of code. Students first think about what sorts of new blocks they would like in Game Lab, and what code those blocks would contain inside. Afterward, 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.

Question of the Day: How can programmers use functions to create their own abstractions?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

Agenda

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 how functions allow programmers to reason about a program at a higher level
- Explain the advantages of using functions in a program

Preparation

- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Functions** - Slides ▾ Make a Copy

For the students

- **Functions** - Video (**Download**)
- **Functions** - Resource

Vocabulary

- **Function** - A named bit of programming instructions.

Introduced Code

- `function myFunction() { // function body, including optional `
- `myFunction();`

Teaching Guide

Warm Up (5 minutes)

Your Personal Blocks

Prompt: What's one block you'd like to have in Game Lab? What would it do? What code would it use to work?

Think-Pair-Share: Allow students to explain their blocks to a partner before sharing out their ideas.

Discussion Goal: As students share their ideas, make sure they have a clear idea of the code that they would use to make the block, reminding them that all of the blocks they have learned this chapter (e.g. velocity, collisions) have been built from blocks that they already knew.

Remarks

> Today, we're going to learn how to create our own blocks so that we decide exactly how they will work. These special blocks are called functions, and they are one of the most powerful parts of programming.

Question of the Day: How can programmers use functions to create their own abstractions?

Activity (35 minutes)

Video: Show students the **Functions** video in the slides.

Teaching Tip

To encourage active engagement and reflection, use one or more of the strategies discussed in the **Guide to Curriculum Videos**.

Questions to Consider with Video:

- Think of a time when a function might have helped you write a program.
- What code would go in the **definition** of the function?
- When would you **call** the function?
- What would you name it?

Discussion Goal: Ensure students know the key vocabulary term **Function** and its definition as *a named bit of programming instructions.* Make sure students understand the 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 where the student wants that block of code to run.

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

The first prediction level program is the first time they will see functions in action.

Discuss: Students should consider and discuss:

- What will be drawn on the screen and why?

Discussion Goal: Predictions will vary but after viewing the Functions video, students should be able to predict that the program will draw what is inside the `drawBackground` and `drawPlanet` blocks of code. It is important to point out that the `drawPlanet` function was called twice which is why there are two planets drawn on the screen, each one a different color and location. Students should also notice the `drawStar` function being called *within* the `drawBackground` function multiple times rather than at the top of the program like the other function calls. You can take this opportunity to discuss how this helped organize and simplify the code since the stars are part of the background.

1 **Predict**

Teaching Tip

Guide to Programming Levels: Additional guidance for programming levels is provided in the **CSD Guide to Programming Levels**. This document includes strategies and best-practices for facilitating programming levels with students.

2-3 **Skill Building**

2 3

Teaching Tip

Introducing Functions

In these first few levels students are just being shown the syntax of functions and are not asked to write or create their own. It can be useful to explain creating a function as basically “creating a new block” just like another programmer created the “isTouching” or “velocity” blocks that they’ve seen actually contain other more complex code.

4 **Predict**

5 **Practice**

Teaching Tip

Why Use Functions

Level 4a, 4b, and 4c introduce three uses of functions, namely removing repetition in programs, allowing code to quickly be changed at multiple points, and providing organization in code. Students will need to write more of their own functions in these levels.

6 **Quick Check**

7-9 **Collector Game**

7 8 9

Teaching Tip

Level 6: Functions in Context

In these levels, students will use functions to organize code within a simple game. While it is not identical to the side scroller, many of the skills and uses of functions in these levels can and should be used when they complete their side scroller.

Assessment Opportunity

Formative Assessment: This level can be used as a formative assessment. A rubric is provided in the level, and written feedback can be given to students. **Click here to learn more about giving feedback to students**.

10 **Challenges**

Wrap Up (5 minutes)

Key Vocabulary:

- **Function** - a named bit of programming instructions

Prompt: Why would we say that functions allow us to "create our own blocks?" Why is this something we'd want to do? Why would a function count as an abstraction?

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

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
- Allowing a programmer to think at a higher level by hiding the details or a particular bit of programming instructions
- Avoiding repeated code by making one block you can use multiple times

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.

Remarks

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

Question of the Day: How can programmers use functions to create their own abstractions?

Lesson 26: The Game Design Process

45 minutes

Overview

This lesson introduces students to the process they will use to design games for the remainder of the unit which is centered around a project guide that asks students to define their sprites, variables, and functions *before* they begin programming their game. Students begin by playing a game on Game Lab where the code is hidden, discussing what they think the sprites, variables, and functions would need to be to make the game. For the purposes of heavily scaffolding the software development process, students are then given a completed project guide that provides starter code and shows one way to implement the game. Students are then walked through this implementation process through a series of levels and have an opportunity to make improvements to the game to make it their own in the final level. In the subsequent lessons, students will need to complete a greater portion of the guide independently, and for the final project, they will follow this process largely independently.

Question of the Day: How does having a plan help to make a large project easier?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)
- **Play Cake Defender**
- **Stop: Review Project Guide**

Activity (35 minutes)
- **Implement Project Guide**

Wrap Up (5 minutes)
- **Using the Project Guide**

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.)
- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **The Game Design Process** - Slides ▾ Make a Copy

For the students

- **Defender Game** ▾ Make a Copy

Teaching Guide

Warm Up (5 minutes)

Play Cake Defender

Transition: This lesson begins immediately in Code Studio. On the first level, students will 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.

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.

Discussion Goal: To help you run 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.

Remarks

> 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, including the sprites, variables, and functions you will want in your program. 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!

Question of the Day: How does having a plan help to make a large project easier?

Activity (35 minutes)

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

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.

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 made 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.

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.

1 **Same Game**

2 **Plan Your Project**

3-4 **Set Up Sprites**

3 4

5-7 **Control Your Player**

5 6 7

8-12 **Sprite Interactions**

8 9 10 11 12

13 **Challenges**

Teaching Tip

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.

Wrap Up (5 minutes)

Using the Project Guide

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

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 then 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.

- 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?

Lesson 27: Using the Game Design Process

90 minutes

Overview

In this multi-day lesson, students use the problem-solving process from Unit 1 to create a platform jumper game. This lesson also 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. Students begin the lesson 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.

Question of the Day: How can the problem-solving process help programmers to manage large projects?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (80 minutes)
- **Play Alien Jumper**
- **Discuss Project Guide**
- **Share out**

Wrap Up (5 minutes)
- **Journal**

Objectives

Students will be able to:

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

Preparation

- Print one copy of the project guide for each student or pair of students
- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Using the Game Design Process** - Slides ▾ Make a Copy

For the students

- **Planning Your Platform Game** - Activity Guide ▾ Make a Copy

Teaching Guide

Warm Up (5 minutes)

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. Afterward, 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, and 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, and 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 above. 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.

Question of the Day: How can the problem solving process help programmers to manage large projects?

Activity (80 minutes)

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.

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.

Assessment Opportunity

Identify core programming constructs necessary to build different components of a game

You can 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 in the Project Guide.

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. Students who have finished early can choose more challenges from the later levels.

1 **Sample Platform Jumper Game**

2 **Build a Platform Jumper**

3-6 **Platform Jumper - Background and Variables**

3 4 5 6

7-9 **Platform Jumper - Platforms**

7 8 9

10-12 **Platform Jumper - Items**

10 11 12

13-16 **Platform Jumper - Player**

13 14 15 16

17 **Platform Jumper Review**

Assessment Opportunity

You can use this level as a formative assessment for students. Click inside the level to view a rubric and

leave feedback to your students

18 **Challenges**

Share out

Share: Students share their games with their classmates.

Wrap Up (5 minutes)

Journal

Question of the Day: How can the problem solving process help programmers to manage large projects?

Prompt: Before you started coding your game, you first had to fill out a project guide with a plan. How did having a plan change the way that you coded your game? Will you do anything differently when you make your plan for your final project?

Lesson 28: Project - Design a Game

225 minutes

Overview

Students will plan and build their own game using the project guide from the previous two lessons. 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.

Question of the Day: How can the five CS practices (problem-solving, persistence, communication, collaboration, and creativity) help programmers to complete large projects?

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

Agenda

Warm Up (5 minutes)

Activity (215 minutes)
- **Review Project Guide**
- **Step 1: Define - Scope Game**
- **Step 2: Prepare - Complete Project Guide**
- **Step 3: Try - Write Code**
- **Step 4: Reflect - Peer Review**
- **Iterate - Update Code**
- **Share**

Wrap Up (5 minutes)
- **Journal**
- **Reflect**
- **Teacher End-Of-Unit Survey**

Objectives

Students will be able to:

- Create a plan for building a piece of software by describing its major components
- Implement a plan for creating a piece of software
- Independently scope the features of 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
- Check the **"Teacher's Lounge"** forum for verified teachers to find additional strategies or resources shared by fellow teachers
- If you are teaching virtually, consider checking our **Virtual Lesson Modifications**

Links

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

For the teachers

- **Extra Code in Challenge Levels** - Resource ▾ Make a Copy

After the Lesson

- **Post-Project Test**
- **End of Course Survey**

- **Project - Design a Game** - Slides
  Make a Copy

For the students

- **Computer Science Practices** - Activity Guide Make a Copy
- **Make Your Own Game** - Activity Guide Make a Copy
- **Make Your Own Game** - Rubric
  Make a Copy
- **Make Your Own Game - Peer Review** - Activity Guide
  Make a Copy
- **Make Your Own Game - Student Checklist** - Resource
  Make a Copy
- **Problem Solving with Programming** - Resource
  Make a Copy

Teaching Guide

Warm Up (5 minutes)

Prompt: Today, you'll start the final project of the unit, in which you will design and code your own game. Before you start, what are three skills or qualities that you think will be important as you complete this project?

Share: Allow students to share out their ideas.

Discussion Goal: This discussion serves to set cultural norms for the project. Students should expect that the project will be challenging, but that they will have plenty of opportunities to iterate on their work as they work together to learn as a community.

Remarks

> There are lots of practices that programmers use when working on large projects. As you design and build your game, try to reflect on how you are using the five practices of problem solving, persistence, communication, collaboration, and creativity.

Question of the Day: How can the five CS practices (problem solving, persistence, communication, collaboration, and creativity) help programmers to complete large projects?

Activity (215 minutes)

Group: This project can be completed individually or in pairs. At your discretion, you may choose to have students form larger groups as well.

Teaching Tip

Facilitating Group Projects: If students are working in pairs or small teams to complete projects, consider showing these two videos to the class:

- **How Teamwork Works**
- **Dealing with Disagreements**

Depending on your goals with this project, consider having teams complete a **Student Guide to Team Planning**, which reinforces the message in the video

Review Project Guide

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.

Distribute: Give each student a copy of the rubric or student checklist so that they know from the beginning what components of the project you will be looking for.

Step 1: 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 their journals.

1 **Sample Games**

Teaching Tip

Class Brainstorm: Before diving into individual projects, consider leading a class brainstorm of what some example projects could look like that fit this criteria. You may also decide to show one or two of the exemplar apps to help inspire the brainstorm. Using these ideas, students may find it easier to latch onto an initial idea for their project

Step 2: 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. Planning ahead can help them identify issues in their plan before they'll need to make more significant changes to their code.

Teaching Tip

Scoping Student Projects: Students may ideate projects that are beyond the skills they currently have or that would take longer than the allotted time to implement. Rather than asking students to choose a different project, consider asking students to imagine a more scaled-down version of their initial idea. As an analogy, if students initial idea is the "Run" step, imagine a less intense version that represents what the "Walk" step would look like. If necessary, you can keep going back further to a "Crawl" step as well.

Digging Deeper: This is sometimes referred to as the Minimal Viable Product - you can learn more about this process and adapt it into your project strategies by reading this article: **Making Sense of MVP** by Henrik Kniberg

Step 3: Try - Write Code

Distribute: Hand out a copy of the **Problem Solving with Programming** to pairs of students or have students take this resource out if they kept it after the last project. Encourage students to look over the guide.

Display: Show the slide with the problem solving process graphic.

Transition: Once students have completed their planning sheet, it's time to head to Code Studio to start programming. The 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.

Teaching Tip

New Code and Previous Challenge Levels: Throughout the unit, students may have learned additional code in the challenge levels of different lessons. As students approach this project, they may want to revisit the code they learned in these levels or visit them for the first time to learn new codes to use in this project. To help guide students back to previous levels, you can use the **Extra Code in Challenge Levels** as a resource to quickly find where new codes were introduced in earlier challenge levels.

Circulate: Encourage students to use the steps in the Problem-Solving Process for Programming when they get stuck or are unsure of what to do next.

2-5 **Project - Background and Variables**

2 3 4 5

Teaching Tip

Debugging Strategies: As students design and implement their own project ideas, they may find themselves with new bugs that they need to untangle and you may find yourself looking at completely unfamiliar code as students look for help troubleshooting their errors. To help smooth out the debugging experience, consider the following strategies:

- Review the **Teacher Guide to Debugging** for some common questions and strategies to help support students in debugging their code
- Have students follow the steps in the **Student Guide to Debugging** and use the **Bug Report Quarter-Sheets** as an initial step in the debugging process. This helps students prepare and communicate their issue before asking for help.
- If students haven't seen it yet, consider showing the **Debugging Video** to the class to reinforce debugging best practices.

Digging Deeper: Consider supplying students with an object to talk to as part of the debugging process. This is sometimes known as Rubber Duck Debugging - you can learn more on the website **https://rubberduckdebugging.com/**

6-10 **Project - Sprites and Interactions**

6 7 8 9 10

Step 4: Reflect - Peer Review

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

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

11 **Review Your Game**

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 game.

Students should refer to the rubric or student checklist to ensure they meet the requirements of the project.

Direct students to complete the last step of their activity guide to reflect on how they did programming this game.

Teaching Tip

Rubric and Checklist: Students have two resources they can use for self-reflection and making sure they are on the right track: the rubric and the student checklist. We recommend having students use the checklist for their own self-assessment and reflection, since it may be easier to digest and understand when reviewing their own project. However, we recommend teachers use the full rubric for evaluating projects to give more accurate feedback to students. You can see examples of this with the Sample Marked Rubrics resource at the top of the lesson plan (only visible to verified teachers)

Assessment Opportunity

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.

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

Wrap Up (5 minutes)

Journal

Question of the Day: How can the five CS practices (problem solving, persistence, communication, collaboration, and creativity) help programmers to complete large projects?

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

- Choose one of the five practices in which you believe you demonstrated growth in this unit. 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 project we completed today. What made it so important?

Reflect

Send students to Code Studio to complete a quick end-of-unit survey. Although their answers are anonymous, the aggregated data will be available to you once at least five students have completed the survey.

End-Of-Unit Survey

Teacher End-Of-Unit Survey

We also have a teacher end-of-unit survey to learn more about how the unit went for you and your students. While students take their survey, **please complete this end of unit survey for teachers** as well. Your feedback is valued and appreciated!

After the Lesson

Post-Project Test

Post-Project tests are included at the end of every unit. These include several multiple choice and matching questions as well as open ended reflections on the final project of the unit. These tests are aligned to the **learning framework** of each unit and are designed to assess parts of the framework that may not have been covered by the project rubrics. To holistically assess the learning objectives of the unit, the post-project test should be paired with the end-of-unit project which is the primary student assessment in each unit.

Unlocking The Tests: 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. **Click here for more information about unlocking and admistering assessments**

End of Course Survey

If this is the last unit of CS Discoveries that you are teaching, also have students take the end-of-course survey. See the **CSD Instructions resource** for more information about the End-of-Course survey and how to assign and see the results.