

Lesson 19: Collision Detection

Overview

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

Students learn about collision detection on the computer. Working in pairs, they explore how a computer could use sprite location and size properties and math to detect whether two sprites are touching. They then use the `isTouching()` block to create different effects when sprites collide, and practice using the block to model various interactions.

Purpose

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

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

Assessment Opportunities

1. **Detect when sprites are touching or overlapping, and change the program in response.**

See Level 7 in Code Studio.

2. **Describe how abstractions help to manage the complexity of code**

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

Standards

Full Course Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

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

Links

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

For the teachers

- **CSD Unit 3 - Interactive Animations and Games** - Slides
- **Collision Detection** - Resource

For the students

- **Collision Detection (Version A)** - Activity Guide
- **Collision Detection (Version B)** - Activity Guide

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

- `debug`

Agenda

Lesson Modifications

Warm Up (5 minutes)

Activity (35 minutes)

Collisions Unplugged

isTouching()

Wrap Up (5 minutes)

- `isTouching(target)`

Teaching Guide

Lesson Modifications



Attention, teachers! If you are teaching virtually or in a socially-distanced classroom, please **click here** to access modifications that can be used during this lesson.

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

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.

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

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

 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.