Unit 8
Collision Detection

Unit Overview

Agenda

20min Introduction
10min 1D Distance
30min The Distance Formula
20min Collide?
10min Closing

Students discuss and then prove the Pythagorean theorem, and use this theorem - in conjunction with Booleans - in their games to detect when a collision has occurred.

Learning Objectives
Students will:

- Understand the challenge of the distance formula in two dimensions.
- Be familiar with the Pythagorean theorem.
- Draw connections between geometry and real-world problems.

Product Outcomes

- Students write `distance` and `collide?`

State Standards
See Bootstrap Standards Matrix provided as part of the Bootstrap curriculum.

Length: 90min

Materials and Equipment

- ☐ Cutouts of Pythagorean Theorem packets [1, 2] - 1 per cluster
- ☐ Computers w/DrRacket or WeScheme
- ☐ Student workbooks
- ☐ Class posters (rules, basic skills, calendar, warnings, design recipe, language table)
- ☐ Pens or pencils for students

Preparation

- ☐ Write agenda on board
- ☐ All student computers should have their game templates pre-loaded, with their image files linked in
- ☐ Class posters
- ☐ Seating arrangements: ideally clusters of desks/tables
- ☐ Load pre-class assessment on student machines

Pre-Class Assessment: Review of Unit 7

- *This data helps us track student performance. All data collected is anonymous, and participation is optional.*
- If you would like us to see the assessment results for your class, contact us.

Language Table

Types	Functions
Number	+ - * / sq sqrt expt
String	string-append string-length
Image	rectangle circle triangle ellipse radial-star scale rotate put-image
Boolean	= > < string=? and or

Introduction — Time: 20 minutes

- *Have students begin with the online assessment.*
- Right now, in your games, what happens when the player collides with another game character? Nothing! We need to write a function change that.
- This is going to require a little math.
- Scroll to the `line-length` and `collide?` functions in your game files.
- Suppose I'm the player, moving around up here at the front of the class. Can I have a volunteer come up here and be the Target?
- *Have a volunteer stand up in front of the class, swinging their arms around, helicopter-style.*
- I need to know how far apart I am from the danger, so I'll know when we've collided. Can anyone tell me how far apart we are? Have we collided?
- *Take a step forward.* How far are we now? Have we collided? *Repeat until you collide.*
- As you can see, it's important to be able to calculate how far apart to characters are, in order to know when they've hit one another.
- In one dimension, this is pretty easy. If the characters are on the same line, you just subtract one coordinate from another, and you have your distance.
- *Demonstrate this on the board, using a number line. Ask the following questions, filling in the blanks for contract as students give answers:*
- How many inputs does `line-length` take? What is the name of the input? What is the Domain? What is the Range?
- This function, `line-length`, will take two positions along a number line input and give back how far apart they are.
- I'd like to have one volunteer stand up and be our function. Raise your hand if you'd like to volunteer!
- Your name is now "line-length". Whenever I call your name, I will also give you two numbers, and your job is to tell me the difference between them - just like the code on the board! Let's try one example "line-length twenty ten!" *your volunteer should reply with "ten"*
- Raise your hand if you'd like to try! *Take some volunteers, and make sure line-length is doing the right thing (including bugs if the first number is smaller than the second!).*
- What is the name of the second function? How many inputs does it take? What is the name of the input? What's the Domain? What's the Range?
- Who can tell me what this function does? Raise your hand if you have an idea.
- `collide?` takes in two numbers, and then gives them to line-length! When line-length responds, it checks to see if the difference is greater than 5.
- I'd like to have one volunteer stand up and be `collide?`. Raise your hand if you'd like to volunteer!
- Your name is now "collide?". Whenever I call your name, I will also give you two numbers, and your job is to say "true" if the difference between them is greater than five. What will you say if the difference is smaller than five?
- Let's try an example: "collide? ten twenty"! *Make sure collide? calls on line-length!.* Let's have some more examples...
- So what's the problem, if the second number is bigger? *we keep getting negative numbers!*
- Thank you! You can both have a seat now. A round of applause for our brave volunteers!

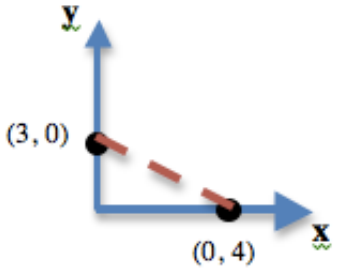
1D Distance — Time: 10 minutes

- If all we did was subtract the second number from the first, the function would only work half the time! Suppose your target is standing at 20, and the player is at 10. What is 20 minus 10? What if their positions were reversed?
- We have to make sure we are always subtracting the bigger number from the smaller one!
- So actually there are two conditions: one is if the first number is bigger, and the other is if the second is bigger. What do we do, when we have multiple **cond**itions?
- Turn to **page 26**, and see if you can write this function so that it always subtracts the smaller number from the bigger one. We've already done the examples for you, but you'll need to first fill out the contract, and then move on to the code.
- What should we do if a is less than or equal to b? Figure it out, and write the code. We've put in two EXAMPLES for you, but you'll need to uncomment them (delete the semicolons at the front). Make sure both EXAMPLEs work!

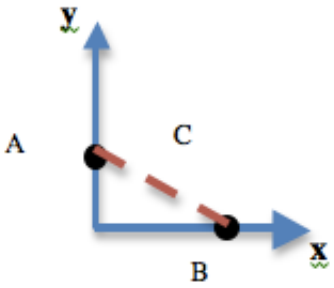
- Now look at the definition of `*distances-color*`. Currently it's just the empty string: "". Try changing it to a real color, like "black" or "white" or "magenta".
- When you click Run, you will now see lines representing the x-length, y-length, and distance between the target, danger and player. These numbers come from your line-length function! Make sure there are no negative numbers!

The Distance Formula — Time: 30 minutes

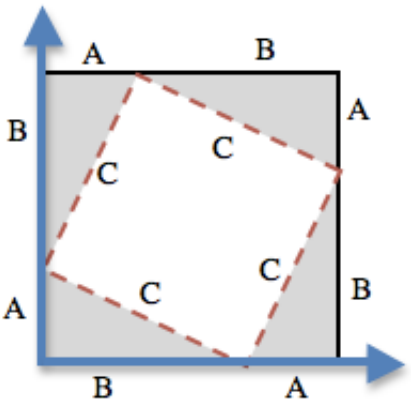
- Unfortunately, the 2d distance looks like 0. Is that right? NO! That's because you haven't written the code to calculate the distance in two dimensions! All you have is something that tells you the length in the x- and y-dimension.
- Ancient civilizations had the same problem: they also struggled with distance in two dimensions. An ancient Greek named Pythagoras finally figured it out.
- *Draw normal Cartesian coordinate plane, with a point on each axis. Label the coordinates (3,0) and (0, 4).*



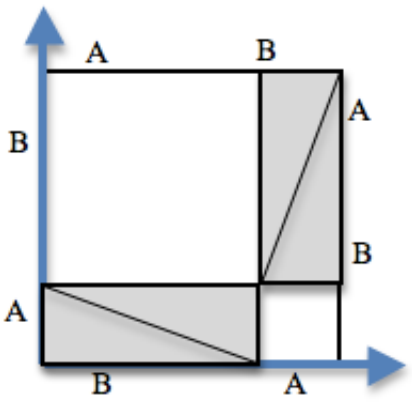
- How can we find the distance between these two points? How can we find the length of the dotted line, also called the Hypoteneuse? Well, let's start with what we do know: the dotted line sort of makes a triangle, and we know the `line-length` of the other two sides. Let's label them "A," "B" and "C." What is the line-length of A? *Have students answer. This will typically involve subtraction, but point out that subtraction can sometimes give back negative numbers!*
- To make our lives easier, we can use the function `line-length`. Just like we did with the butterfly example, we can write functions that call each other for help!
- In our example, `(line-length A)` is 4, but we still don't know C. *On the board, write:*
  What is the length of C?
- *Pass out Pythagorean Proof materials to each group, and have them review all of their materials:*
  - *A large, white square with a smaller one drawn inside*
  - *Four gray triangles, all the same size*



- Everyone has a packet with the same materials, but each group's triangles are a little different. That's okay, though: what we're going to do works with all sizes, so you'll each get to test it out on your own triangles.
- First, I'd like you all to take ONE of the triangles, and place it on the center of a desk, so that it matches the triangle on the board. Do you see the sides labeled "A," B" and "C"? Do they match the sides on the board? Good! On YOUR desks, all of the A's are the same size, all of the B's are the same size, and all of the C's are the same size.
- Now take your triangles, and place them on the big white square so that all of the As, Bs and Cs line up. You can follow along with what I have on the board, too. (See diagram with inscribed square.)



- Now we have four triangles, each with a side A, B and C. We also have two squares: the inner square, whose sides are a C, and the outer square, whose sides are (A+B).
- Raise your hand if you know how to find the area of a square. Take a volunteer.
- What's the area of the white, inner square? C^2. *On the board, write:*
  white space = C^2
- Move your triangles so they match the drawing on the board (see diagram, or this animation). Now we have two small, white squares. Is there more white on the board now than there was when we had just a single big one? Why or why not?



- Since we didn't change the size of the outer square, and all we did was move stuff around inside it, we know there is still the same amount of white space as there was before - it's just broken into two parts now. *Refer back to the previous board writing:*

  whitespace = C^2
- What is the area of the smaller white square? We know that both of its sides are of length A, so its area must be A^2.
- What about the bigger white square? We know that both of its sides are of length B, so its area must be B2. So now we have two ways of writing the area of the white space:

  whitespace = $C^2 = A^2 + B^2$
- Well, if we know that A and B are 3 and 4, so let's fill that in.

  whitespace = $C^2 = 3^2 + 4^2$
- What is 3 squared? 4 squared?

  whitespace = $C^2 = 9 + 16$
- What's 9+16?

  whitespace = $C^2 = 25$
- Okay, so we know that C^2 is 25...but remember, we want C by itself. What is the square root of 25? It's five!
- Pythagoras proved that you can get the square of the hypotenuse by adding the squares of the other two sides. In your games, you're going to use the horizontal and vertical distance as the two sides of your triangle, and use the Pythagorean theorem to find the length of that third side.
- Turn to **Page 27** of your workbook - you'll see the formula written out. Let's do this one **together**, since it's been a long time since we've done Circles of Evaluation as a class.
- What's the simplest expression inside this giant thing? `(line-length 4 0)`! Let's draw the circle for that. *Walk students through the entire thing...*
- So now we've got code that tells us the distance between those two points. But we want to have it work for any two points! It would be great if we had a function that would just take the x's and y's as input, and do the math for us.
- Let's think again about the problem statement, and the function header. Turn to **page 28**, use the Design Recipe to write your distance function. Feel free to use the work from the previous page as your first example, and then come up with a new one of your own. Raise your hand when you are done with the contract, and when you've circled and labeled your two examples.
- Take a few minutes to type in your examples and your code and try it.
- Do the distances look right? The shortest distance between any two points is a straight line. So can either of the legs a or b be longer than c? Can c be longer than a+b?

Collide? Time: 20 minutes

- So now we have a function called `distance`: two coordinates go in and one value comes out, representing the distance between those coordinates.
- If we write `(update-danger 10)`, we get back the new location of the danger after it's at position 10. If we write `(distance 100 200 95 65)`, we get back the distance between the coordinates (100, 200) and (95, 65).
- But what do we want to **do** with this distance?
- *Using visual examples, ask students to guess the distance between an danger and a player at different positions. How far apart do they need to be before one has hit the other?*
- Make sure students understand what is going on by asking questions: If the collision distance is small, does that mean the game is hard or easy? What would make it easier?
- At the bottom of **page 29** you'll find the Problem Statement for collide?. Fill in a Design Recipe Worksheet, and then write the code. Remember: you WILL need to make use of the distance function you just wrote!

Closing Time: Time: 10 minutes

- *Have everyone walk around and play each other's games.*

- CONGRATULATIONS! You have finished the code for your videogames!!!
- Who can tell us one thing we learned today?
- Who saw someone else in the class do something great?
- Cleanup, dismissal.