Unit 8
Collision Detection

Unit Overview

Students return to the Pythagorean Theorem and distance formula they used in Bootstrap 1, this time with data structures and the full update-world function.

Learning Objectives:

- Reinforce their understanding of the distance formula
- Identify collision as yet another sub-domain which requires different behavior of the update-world function

Agenda

5 min	Introduction
10 min	1D Distance
20 min	The Distance Formula
10 min	Collide?
40 min	update-world
5 min	Closing

Product Outcomes:

- Students will write the distance function
- Students will write the collide? function
- Students will use different Cond branches to identify collisions in their games

State Standards See our Common Core Standards Table provided as part of the Bootstrap curriculum.
Length: 90 minutes

Materials and Equipment:

- *Computers w/DrRacket or WeScheme*
- *Student workbooks*
- *Design Recipe Sign*
- *Language Table*
- *Cutouts of Cat and Dog images*
- *Cutouts of Pythagorean Theorem packets [1, 2] - 1 per cluster*
- *The Ninja World 6 file [NW6.rkt from source-files.zip | WeScheme] preloaded on students' machines*

Preparation:

- *Language Table Posted*
- *Seating arrangements: ideally clusters of desks/tables*

Introduction (Time 5 minutes)

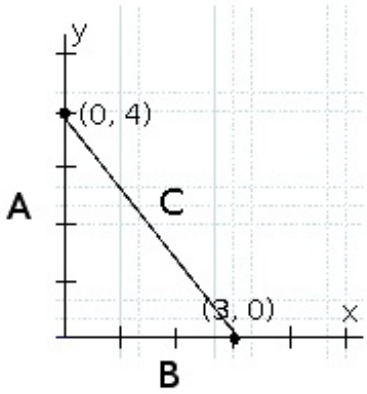
- Right now, in your games, what happens when the player collides with another game character? Nothing! We need to write a function change that.
- This is going to require a little math, but luckily it's exactly the same as it was in Bootstrap 1!
- *Draw a number line, and place the cat and dog upon it. How far apart are the cat and dog? Move them. How about now? Move them. Now? Move them such that the dog and cat have switched positions. How about now?*
- In one dimension, finding the distance is pretty easy. If the characters are on the same line, you just subtract one coordinate from another, and you have your distance. However, if all we did was subtract the second number from the first, the function would only work half the time! When I switched the cat and dog, did you still say "dog minus cat"? No! It turned into "cat minus dog"! How did you know?

1D Distance (Time 10 minutes)

- We have to make sure we are always subtracting the bigger number from the smaller one!
- So there are really 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 conditions?
- Turn to Page 36, and see if you can write examples for this function so that it subtracts the smaller number from the bigger one. Start with an example for 23 and 5...then do an example with 5 and 23 in the other order.
- Now we have an idea of the results for our `cond` statement. But what's the other thing that we need in `cond`? Tests! We want to see whether the first number is greater than the second number...how would we write that in Racket code?
- And what would the result for that test be? If a is greater than b, which would we subtract from which?
- Take one minute: Write down the definition for `line-length`.

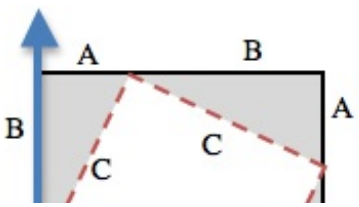
The Distance Formula (Time 20 minutes)

- Unfortunately you still 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.
- *Draw normal Cartesian coordinate plane, with two points on it, of the coordinates (0, 4) and (3, 0).*
- How can we find the distance between these two points? How can we find the length of the dotted line, also called the Hypoteneuse? 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!

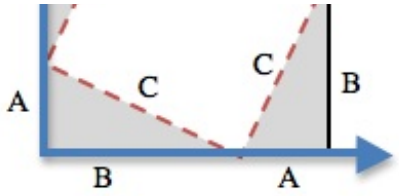


- To make our lives easier, we can use the function `line-length`.
- In our example, (line-length A) is 4 and (line-length B) is 3, but we still don't know C.
- *Pass out Pythagorean Proof materials to each group, and have them review all of their materials:*
  - *A large, white square with a smaller one drawn inside*
  - *Four gray triangles, all the same size*

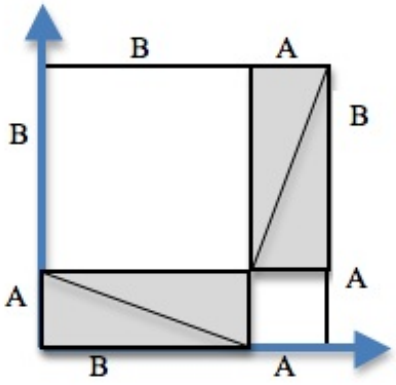
  Everyone has a packet with the same materials, but each group's triangles are a little different. That's okay, though: what we're going to do works with all sizes, so you'll each get to test it out on your own triangles.
- Watch this animation.
  Teachers, draw this on the board.
- First, take ONE of 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? C2.
- *On the board, write: white space = C^2*
- Move your triangles so they match the drawing on the board. 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 B^2.
- 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, we can 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 C2 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 game, you're going to use the horizontal and vertical distance between two characters as the two sides of your triangle, and use the Pythagorean theorem to find the length of that third side.



- Turn to Page 37 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 is the simplest expression inside this giant thing? `(line-length 4 0)`! Start with 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.
- Think again about the problem statement, and the function header. Turn to Page 38, and 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.
- When you're done, type your `line-length` and `distance` functions into your game, and see what happens.
- Does anything happen when things run into each other? No! We still haven't written a function to check whether they're colliding!

Collide? (Time 10 minutes)

- So what do we want to do with this distance?
- *Using visual examples, ask students to guess the distance between a 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 top of Page 39 you'll find the Problem Statement for `collide?`. Fill in the Design Recipe, and then write the code. Remember: you WILL need to make use of the `distance` function you just wrote!
- When you're done, take two minutes to type it into your game, under `distance`.

update-world (Time 40 minutes)

- Now that you have a function which will check whether something is colliding, we can go back to modifying your game code! Out of the four major functions in our game (`update-world`, `draw-world`, `keypress` and `big-bang`), which do you think we'll need to edit to handle collisions? `update-world`. We need to make some more branches for `cond`.
- In Ninja World, what do we want to happen when the cat collides with the dog? We want to put the dog offscreen, so that he can come back to attack again!
- Let's start with the test. How can we check whether the cat and dog are colliding? Do we have a function to check that? What do the inputs need to be? Do we have a `catX` in our world? How do we know what it is?
- Since the Cat's x-coordinate is always the same, let's check `draw-world` so that we can put it in. The cat's x-coordinate will always be 360! How do we pull the `catY` out of the world? How do we pull the `dogX` out of the world? Is there a `dogY` in the world? Where can we get that number?

-
```
[(collide? 360 (world-catY w) (world-dogX w) 400) ...result...]
```
- Remember that `update-world` gives back a world, so what should we do first in our result? `make-world`.
-
```
[(collide? 360 (world-catY w) (world-dogX w) 400) (make-world ...dogX...
                                                              ...rubyX...
                                                              ...catY...)]
```
- And what did we want to happen when our cat and dog collide? We wanted the dog to go off screen. Can you think of a number that puts the dog off the screen?
-
```
[(collide? 360 (world-catY w) (world-dogX w) 400) (make-world -100
                                                              ...rubyX...
                                                              ...catY...)]
```
- How about `rubyX`? Do we want it to change when the dog and cat collide? No! How about `catY`? No!
-
```
[(collide? 360 (world-catY w) (world-dogX w) 400) (make-world -100
                                                              (world-rubyX w)
                                                              (world-catY w))]
```
- It's time to figure out what happens in your game! Turn to Page 40 and write some more tests and results. When you've figured out what happens when each thing collides with your player and have written it down, type it into the computer.
- *Work in small groups to complete collision branches.*

Closing (Time 5 minutes)

- *Congratulations guys! You've finished every lesson, and now it's time to get to the fun stuff. I want you to go home and brainstorm...what else do you want your game to do? Next time we're going to add more things, so that your games are even cooler.*
- *Have students show each other their games!*