The Distance Formula

(Also available in WeScheme)

Students apply their knowledge of the Pythagorean Theorem and Circles of Evaluation to develop a function for the distance formula.

Lesson Goals	Students will be able to: • Explain how the distance formula is related to the Pythagorean theorem. • Write a function for the distance formula.
Student-Facing Lesson Goals	• Let's investigate how the Pythagorean theorem can help us find the distance between two game characters. • Let's write a function that takes in 2 points and returns the distance between them.
Prerequisites	• Simple Data Types • Contracts • Functions: Contracts, Examples & Definitions • Solving Word Problems with the Design Recipe
Materials	• PDF of all Handouts and Page • Lesson Slides • Printable Lesson Plan (a PDF of this web page)
Supplemental Materials	• Additional Printable Pages for Scaffolding and Practice • *Sample Game Starter File* • *Sample Game with Distance Lines Made Visible*
Relevant Resources	• This short video introduces viewers to the nearly 4000 year old Babylonian tablet known as Plimpton 322, which contains a table of Pythagorean Triples that long predates Pythagoras, as well as to Babylonians use of the base 60 system.

Key Points for the Facilitator	• *Note: This lesson assumes that students already have a basic understanding of the Pythagorean Theorem and how to use it. This lesson is designed to build on what they know and deepen their understanding!* • The distance formula is an excellent review of Circles of Evaluation. Have students work out the expression in small groups to foster discussion.

Glossary

conditional :: a code expression made of questions and answers
coordinate :: a number describing an object's location
hypotenuse :: the side opposite the 90-degree angle in a right triangle
Pythagorean Theorem :: the relationship between the squares of the sides of a right triangle; can be used to find diagonal distances on the coordinate plane

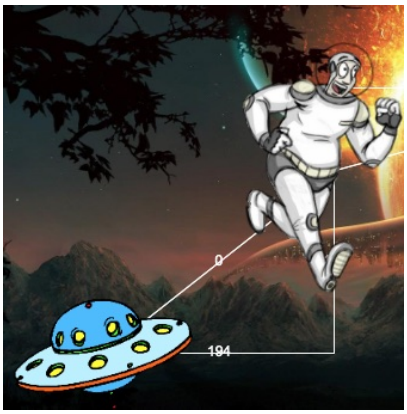
Distance in 1 Dimension *15 minutes*

Overview

Students discover the need for distance calculation (first in one dimension, then in two) in video games.

Launch

Sign in to code.pyret.org (CPO) and open your saved Game Starter Files.



At this point:

- The Target and Danger should be moving on their own.
- The Player should respond to keypresses.
- The Target and Danger should re-appear after they leave the screen.

It's almost fully-playable!

Here's a link to a Sample Game Starter File you can use if you're demoing on the board.

- What seems to be missing from this game?
  - *The characters aren't doing anything when they collide.*
- What does it mean for characters to 'hit' one another? To collide?
  - *They have to be close enough to touch.*
- How will the computer know when the characters have collided?
  - *When the coordinates of the characters are really close to each other.*

In the following activity, students will role play a collision between two characters.

Draw a "number line" on the floor or across a wall of your classroom as the backdrop for your movement, and select a volunteer to represent a character in the game (either `TARGET` or `DANGER`), while represent the `PLAYER`. Emphasize that this represents *one dimension* (perhaps the x-axis). Both of you should stand on the number line, 8-10 steps away from one another.

Each image in the game is located based on its center. Make sure that you and your volunteer stand with feet as close together as possible, representing the infinitely small point that identifies your center. Have you and the volunteer raise your arms to form a "T shape", representing the outer edges of the characters.

Ask the class how far apart you and your volunteer are. How they would calculate this if you were standing on a number line and they could see the actual coordinates under your feet? The goal is to illicit the response that students would subtract the smaller coordinate from the larger one (or subtract in any order and take the absolute value).

Side-step towards each other one step at a time, each time asking the class, "We are colliding: True or False?" *Be sure to only accept "true" and "false" as responses - not "yes" and "no"!*

After a few iterations, try switching places and repeating. *Point out that students always subtract the smaller number from the larger one, regardless of the character order!* **The results are always positive.**

Do this until students can clearly see that collision happens when the two characters are *touching* or *overlapping* in some way - NOT when they are *at the same point*.

Investigate

Our game computes 1-dimensional distance (vertical or horizontal) using a function called `line-length`. Let's explore how it works!

Optional: If you want to provide students with the questions below, use Line Length Explore.

- Find the `line-length` function in your game files and take a minute to look at the code.
- What do you notice?
  - *Both of the examples do the same thing, even though the numbers are given in a different order.*
  - *It's a piecewise function!*
  - *It uses inequalities.*
- What do you wonder?
- Click "Run", and practice using `line-length` in the Interactions Area with different values for `a` and `b`.
- What does the `line-length` function *do?*
  - *It always subtracts the smaller number from the larger number and evaluates to a positive distance!*

- Why does it use ***conditionals***?
  - *To determine whether or not to subtract the numbers in the given order or to swap the order to get a positive result.*

Synthesize

- Why is the distance between two points always positive?
  - *Because distance has nothing to do with direction - it takes just as long to drive from Seattle to Wichita as it does to drive from Wichita to Seattle.*

Why line-length?

Students learn early on that distance in 1-dimension is computed via $| x_2 - x_1 |$, and that distance is always a positive value. The Pythagorean Theorem teaches students that the length of the hypotenuse is computed based on the distance in the x- and y-dimension.

Most math textbook, however, show the distance formula without connecting back to that theorem.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

A student who asks whether it's a problem when $x_2 - x_1$ is negative is displaying a deep understanding of what's going on!

Using the `line-length` function explicitly connects the distance formula back to the 1-dimensional distance students know, allowing them to apply prior knowledge and better connect back to the ***Pythagorean Theorem*** itself.

This effectively rewrites the distance formula as:

$$\sqrt{| x_2 - x_1 |^2 + | y_2 - y_1 |^2}$$

Distance in 2 Dimensions *30 minutes*

Overview

Students extend their understanding of *distance* from one dimension to two.

Launch

We just practiced computing the distance in 1-dimension, which is useful if the Player and Danger have the same x- or y-coordinate. But how do we compute the distance between two points when both the x- *and* y-coordinates are different?

Here's a link to a Sample Game with Distance Lines Made Visible to use if you're demoing the instruction below on the board.

- Scroll down to `4. Collisions` in your game file and look for the `distances-color` definition. What is the value defined to be?
  - *Right now this value is defined to be the* empty string `""`.
- Change this to a color that will show up on your background, and click "Run". What happens?
  - *This setting draws lines from your Player to each of the other characters, and then uses those lines as the hypotenuse of right triangles! The legs of these triangles show the distance in 1 dimension each (on the x- and y-axis).*

In order to compute the *diagonal* distance between two characters in a video game, we'll need a special formula that considers *both the vertical and the horizontal* distances between them!

When we turned on `distances-color` in our game, we saw the diagonal distance between two characters represented as the ***hypotenuse*** of a right triangle.

- How do we find the hypotenuse of a right triangle if we know the measures of both of its legs?
  - *The Pythagorean Theorem!* $a^2 + b^2 = c^2$
- If we had one player at (0,0) and another player at (4,3), we'd see a right triangle and the lengths of the legs would be 3 and 4. How would we use the Pythagorean Theorem to find the hypotenuse of the triangle?
  - *We would add* 3^2 *and* 4^2*, or 9 and 16, to get 25. The square root of 25, or 5, is the length of the hypotenuse.*

Optional: If it's been a long time since your students have used the Pythagorean Theorem, now would be a good time to do some Pythagorean Theorem Practice.

Connecting Pythagorean Theorem to video games

We recommend carving out 4.5 minutes and wowing your students with Tova Brown's Video of a Geometric Proof of the Pythagorean Theorem and its application to finding distance between game characters. Then have them try explaining the proof to one another.

In our case, the lengths A and B are computed by the `line-length` function we already have!

Optional: On Writing Code to Calculate Missing Lengths we've provided screenshots from two games where the horizontal and vertical distances between the characters are shown. Students are asked to write the code to calculate the distance between these characters using the Pythagorean Theoream. You could also have them do the computations (using a calculator) and compare their results to what their code evaluates to.

Investigate

- Turn to Distance on the Coordinate Plane and look at how line-length is used in the code. See if you can figure out how to write the code for the second problem.
- Then turn to Circles of Evaluation: Distance between (0, 2) and (4, 5). Convert the expression to a Circle of Evaluation, and then to code.
- Then we'll make sure we really understand it all with Multiple Representations: Distance between two points by combining circles of evaluation, code, computation and a sketch on a graph.

For more practice writing code to generate the distance between two fixed points, complete Distance From Game Coordinates. *Optional:* more practice can be found at Distance From Game Coordinates 2.

Debrief these pages - or have students pair-and-share - before moving on to writing the full distance function. Explain to students that all of the practice they've done so far today focused on a screenshot of a moment in time. With the game stopped in that moment, we knew either the exact location of our characters or the exact distances between them. **As we play our games, however, the characters are constantly changing locations!**

In order to calculate the distance between two objects whose locations are constantly changing, we need to use variables!

- Turn to Distance (px, py) to (cx, cy) and use the Design Recipe to help you write a function that takes in two ***coordinate*** pairs (four numbers) of two characters (px, py) and (cx, cy) and returns the distance between those two points.
- HINT: The code you wrote in Circles of Evaluation: Distance between (0, 2) and (4, 5) can be used to give you your first example!
- When you're done, fix the broken `distance` function in your game file, click "Run" and check that the right triangles in your file now appear with reasonable distances for the hypotenuse.

Optional:

If we knew the lengths of the hypotenuse and one leg of the triangle, could we use the formula $A^2 + B^2 = C^2$ to compute the length of the other leg?

Take a look at the two examples on Comparing Code: Finding Missing Distances.

- There's a subtle difference between the two examples! What is it?
  - *In the first example, the length of the hypotenuse is missing. In the second example, the length of a leg is missing.*
- Can you explain why they need to be written differently?
  - *Finding the hypotenuse requires finding the square root of the $A^2 + B^2$, whereas finding a leg requires finding the square root of the difference between C^2 and B^2.*

Common Misconceptions

It is *extremely common* for students to put variables in the **wrong order**. In other words, their program looks like `…num-sqrt(num-sqr(line-length(x1,y1)) + num-sqr(line-length(x2, y2)))…` instead of `…num-sqrt(num-sqr(line-length(x2 x1)) + num-sqr(line-length(y2 y1)))…`

In this situation, remind students to look back at what they circled and labeled in the example steps. *This is why we label!*

Synthesize

- How does the length of the hypoteneuse rely on the length of each side?
- Where do you see one formula being used inside the other?

Additional Exercises

- Have students use the Design Recipe to solve Word Problem: line-length on their own.
- You might also want to have them modify `line-length` to make use of the absolute value function: `num-abs`.