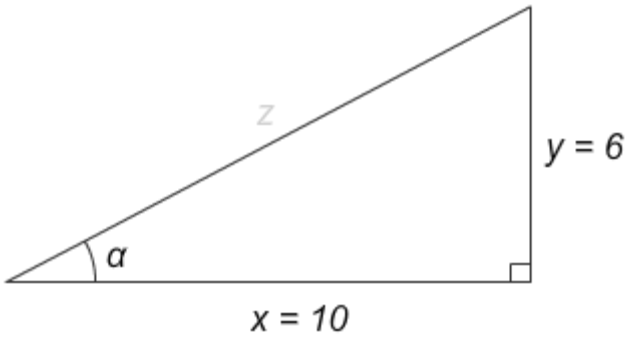
Mathematics in game development: Trigonometry

In game development, there are a lot of situations where you need to use the trigonometric functions. The functions are used to calculate an angle of a triangle with one corner of 90 degrees. By converting the equation, it is also possible to calculate a side of a triangle with one angle given.

These calculations come in handy for circular forms and placements as well. In this article, we'll handle some usage of these trigonometric functions. Also, we'll go through examples for how to apply these techniques into game development.

Tangent

With the Tan function, you can calculate the angle of a triangle with one corner of 90 degrees, when the smallest sides of the triangle are given:



In this example, we want to calculate the angle α. We do this by using the equation:

$$Tan(\alpha) = y / x$$

Because we want to know what *α* is, and actually don't care about what *Tan(α)* is, we perform a Tan^{-1} on both sides of the equal sign. This results into the equation:

$$\alpha = Tan^{-1}(y / x)$$

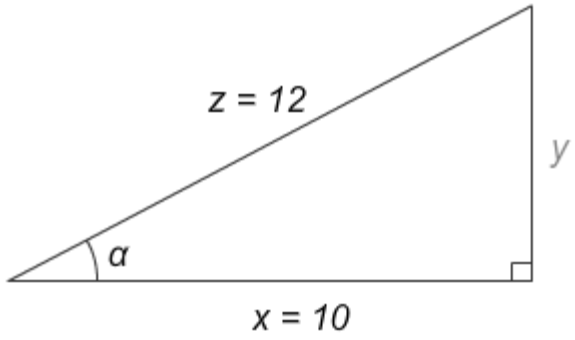
Now we can calculate the angle α. We do this by simply fill in the given values, and let the calculator (or computer) do the work.

$$\alpha = Tan^{-1}(y / x)$$
$$\alpha = Tan^{-1}(6 / 10)$$
$$\alpha = Tan^{-1}(.6)$$
$$\alpha = 31^{o}$$

Cosine and Sine

In the previous example we've calculated angle α, with two sides of the triangle as a given. Now, we were only able to do this with the Tangent, because these very sides were given. If only the x and the z were given (z is the longest side), we wouldn't be able to use Tangent.

To calculate α when only the x and z were given, we'd have to use the Cosine function, which is very much like Tangent:



$$Cos(\alpha) = x / z$$

Again, here we don't want to know $Cos(\alpha)$. We'd like to know α, so we use the same technique as we did before (applying Cos^{-1} on both sides of the equal sign).

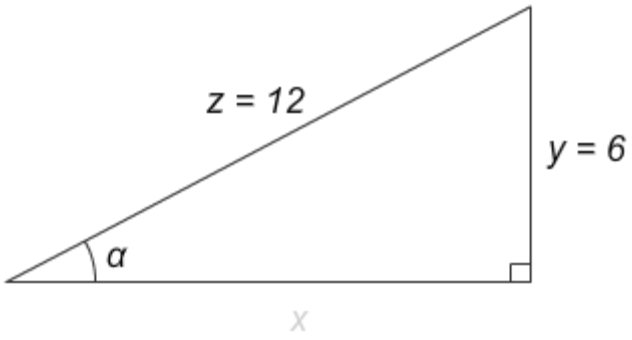
$$\alpha = Cos^{-1}(x / z)$$

From here on we only have to fill in the given values, do some calculations, and were done.

$$\alpha = Cos^{-1}(10 / 12)$$
$$\alpha = Cos^{-1}(.8)$$
$$\alpha = 34^{o}$$

Now, there still is a combination of sides that we didn't discuss yet. That is, when we only know the value of z and y. In this situation we aren't able to use the Tangent or the Cosine. We'd have to use the Sine functions.

This function is also a lot like the ones before. The only differences are the input values:



$$Sin(\alpha) = y / z$$

Again, we apply both sides of the equal sign with the function Sin^{-1}, which gives us:

$$\alpha = Sin^{-1}(y / z)$$

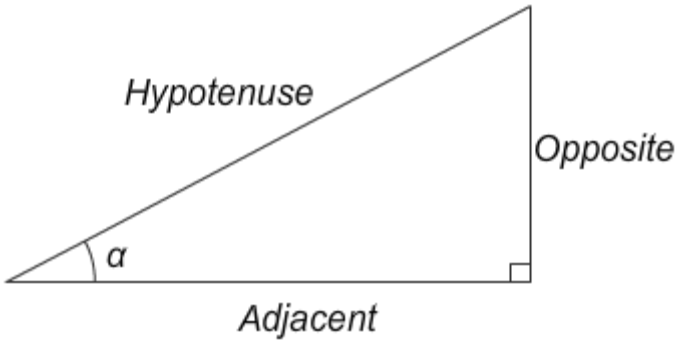
And finally, we fill in the values that were given to calculate α.

$$\alpha = Sin^{-1}(6 / 12)$$
$$\alpha = Sin^{-1}(.5)$$
$$\alpha = 30^{o}$$

When to use Cos, Sin or Tan

The Tangent, Sine and Cosine are commonly shortened as Tan, Sin and Cos. In the previous examples we've worked with all three the functions.

We've seen that we choose our function (Sin, Cos or Tan) based on the values that are given. So, based on the sides of the triangle that we know, we choose our function.



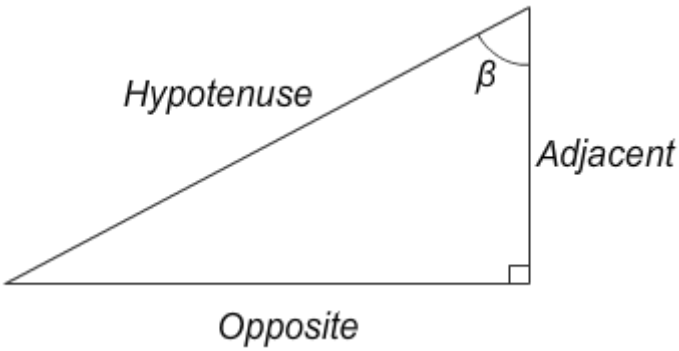
Sin(α) = Opposite / Hypotenuse

Cos(α) = Adjacent / Hypotenuse

Tan(α) = Opposite / Adjacent

We see that we can give these sides of the triangle a name, that represent their position relative to the corner that we want to calculate. The side that goes from our corner to the corner of 90 degrees, is called the adjacent. The side that doesn't point to our corner is called the opposite. And finally, the side that isn't pointing at the corner of 90 degrees is called the hypotenuse.

If we'd want to calculate the other corner, we'd have to rename the sides. This is because the name of a side represents it's position relative to the corner that we want to calculate. Changing the corner, means changing the sides.



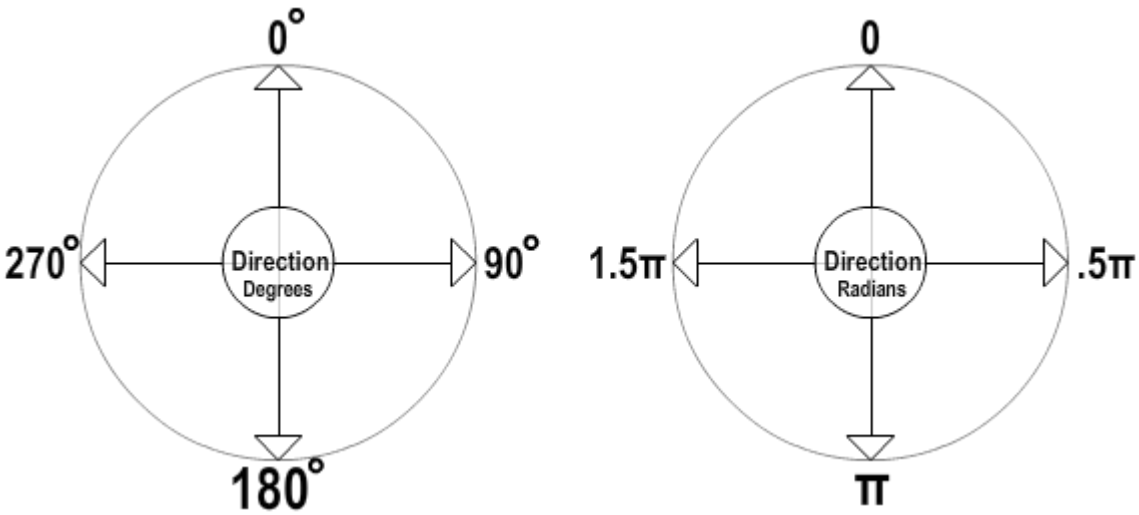
We see that when we want to calculate β instead of α, the opposite and adjacent sides swap. Only the hypotenuse doesn't change, because this is always the side that doesn't point at the corner of 90 degrees. The equations will still work the same, as long as you enter the right values:

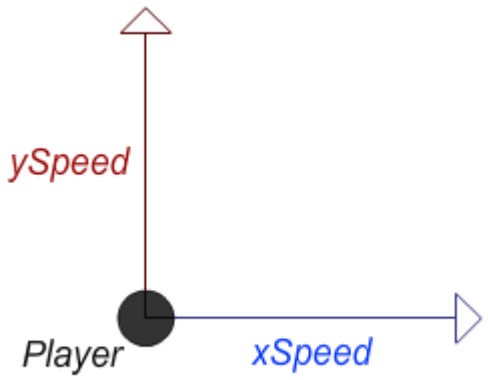
Sin(β) = Opposite / Hypotenuse
Cos(β) = Adjacent / Hypotenuse
Tan(β) = Opposite / Adjacent

Calculating player direction

So, how can we put this into practice when creating games? Well, let's make an example. We're going to calculate the direction of a player that has a xSpeed and ySpeed.

First off, in the previous examples we've used degrees to represent the angle of a corner in a triangle. In programming, we don't work with degrees. We work with radians. This is just another way to represent an angle. 360 degrees equals 2π radian.





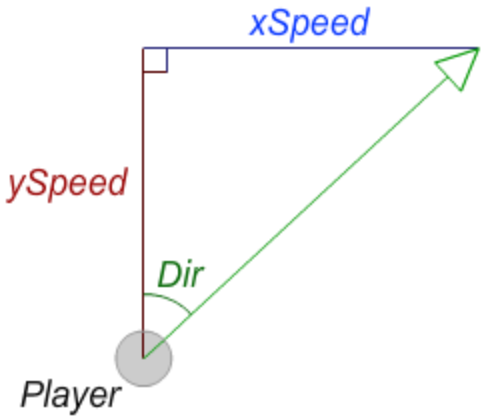
In the image on the left we see a player (black circle), that has a xSpeed and ySpeed. These speed values are added to the player's position.

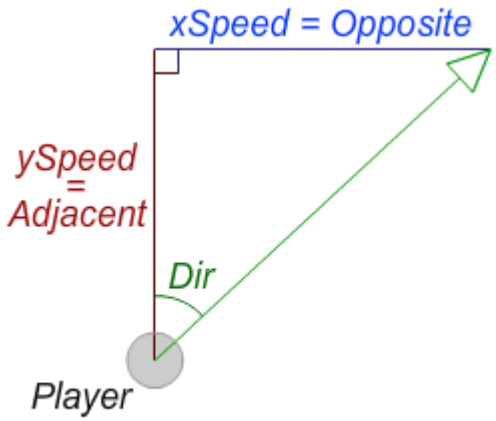
A lot of times in games, you want to know what direction the player is heading. With the knowledge of the Sin, Cos and Tan functions we've just learned, we can calculate the direction of the player.

But we first need to create theses lines into a triangle with one corner of 90 degrees.

In the image on the right, we have our values drawn in a triangle that we can use for calculations. It is a very good practice to draw your geometrics when using math in programming. It helps you focus on what you want to calculate, and it keeps things organized.

When looking at this triangle, we should be able to choose one of our trigonometric functions. First, we have to name every side that is given, based on its position relative to the corner we want to calculate.





In the image on the left we have named the sides of the triangle. Now, we have to choose our function.

Sin(α) = Opposite / Hypotenuse

Cos(α) = Adjacent / Hypotenuse

Tan(α) = Opposite / Adjacent

In this situation, we have the opposite and adjacent values given; the xSpeed and ySpeed. So the functions we want to choose is Tan (because this function deals with opposite and adjacent).

To calculate the direction of the player (Dir), we use the Tangent:

$$Tan(Dir) = Opposite / Adjacent$$

This leads us to the equation:

$$Tan(Dir) = xSpeed / ySpeed$$

Again, by applying the inverted function of Tan to both sides of the equal sign, we get an equation that will return the player's direction.

$$Dir = Tan^{-1}(xSpeed / ySpeed)$$

There it is; an equation to calculate the players direction based on its xSpeed and ySpeed. We can now test this equation by entering a few values. Let's say the player would be going up, and a little bit to the right. In that situation the ySpeed would be bigger than the xSpeed. The resulted direction of the player should than be smaller than 45 degrees. But remember; we don't use degrees in programming. Instead, we use radians. So we want a result that is smaller than 0.25π.

$$Dir = Tan^{-1}(xSpeed / ySpeed) = Tan^{-1}(2 / 10) = Tan^{-1}(.2) = 11^{o} = .06\pi$$

Yes, we were right. The direction of the player is smaller than 0.25π. Now let's try to get the direction larger than 0.25π by making the xSpeed bigger than the ySpeed:

$$Dir = Tan^{-1}(xSpeed / ySpeed) = Tan^{-1}(11 / 5) = Tan^{-1}(2.2) = 66^{o} = 0.37\pi$$

There we have our direction greater than 0.25π. To get exactly 0.25π, we'd only have to make the xSpeed and the ySpeed exactly the same. This would make the player go diagonal.

And that's how we calculate the direction of the player. Although, this equation alone isn't enough to use in a game. It still has some problems. Let's say the player goes directly to the right, with no vertical speed. This would mean the ySpeed would be zero. Let's see what happens.

$$Dir = Tan^{-1}(xSpeed / ySpeed) = Tan^{-1}(10 / 0) = Tan^{-1}(NaN) = NaN$$

Here we try to divide 10 by zero. This is impossible because it would result in a infinite number, and the computer seems to have some problems with that. It would return a NaN value (Not a Number), and that would probably turn into an error.

So to solve this problem we need to make a somewhat more intelligent algorithm. When the ySpeed is zero, we check if the xSpeed is positive, or negative. If it's positive, we set the direction to 0.5π (90 degrees). If it's negative, we set it to 1.5π (270 degrees).


```
//Check if player is going horizontal
if (ySpeed == 0)
{
	//Check if going to the right
	if (xSpeed > 0)
	{
		direction = .5 * Math.PI;
	}
	//Check if going to the left
	else
	{
		direction = 1.5 * Math.PI;
	}
}
//If the player isn't going horizontal
else
{
	direction = Math.Atan( xSpeed / ySpeed );
}
```

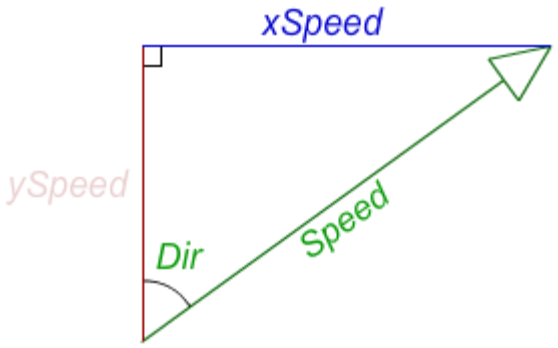
In this code we first ask if the players ySpeed is zero. If this would be true, we shouldn't apply the Tan function, for that would lead us to an error. Instead of applying the Tan function, we ask if the xSpeed is greater than zero. If that would be true, we know the player is heading right, with a direction of 0.5π. If the xSpeed would be smaller than zero, we know the player is moving left, with a direction of 1.5π.

If the ySpeed is greater or smaller than zero, the inverted Tan function is applied, for it would not turn into an error. With this code set, we've finally made an algorithm that calculates the players direction based on its xSpeed and ySpeed.

We see that we use the Math class to get certain functions and numbers. We got our PI from the Math class, as well as the Atan function (inverted Tan function). A lot of mathematical stuff is found in this class in a lot of programming languages.

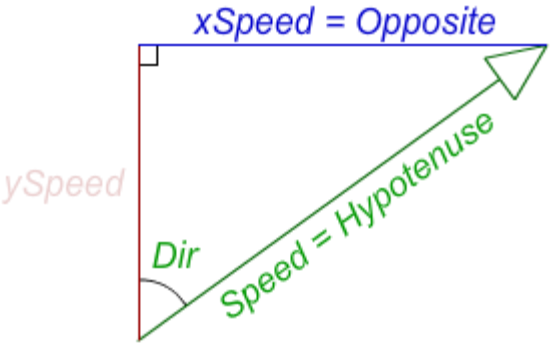
Calculating player xSpeed and ySpeed

In the previous example we've learned how to create a algorithm that calculates the direction of player with the xSpeed and ySpeed as given values. But what to do when we'd want to calculate the xSpeed and ySpeed with the direction and speed of the player as the given values?



In this example we know the direction and speed of the player. Speed here is the value of the longest side of the triangle.

To choose our function (Sin, Cos or Tan), we first define the sides of the triangle based on the position of the side relative to the corner we know.



So here the xSpeed represents the Opposite side. The speed represents the Hypotenuse. Now we have to find the function that handles the Opposite and Hypotenuse:

Sin(α) = Opposite / Hypotenuse

Cos(α) = Adjacent / Hypotenuse

Tan(α) = Opposite / Adjacent

The Sin function handles both the Opposite as the Hypotenuse, so that's going to be the function that we're going to work with. This leads us to the equation:

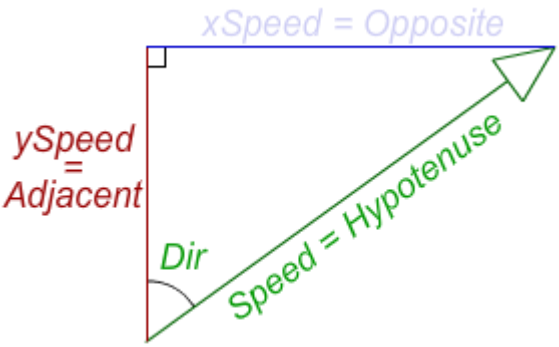
Sin(α) = Opposite / Hypotenuse
Sin(Dir) = xSpeed / Speed

Now, we don't want to know what Sin(Dir) is. We want to know what xSpeed is. By multiplying both sides of the equal sign with Speed, we get the following equation:

*Sin(Dir) * Speed = xSpeed / Speed * Speed*
*Sin(Dir) * Speed = xSpeed*
*xSpeed = Sin(Dir) * Speed*

And there we have our equation to calculate xSpeed by Dir and Speed as the given values. Of course, when we are able to calculate the xSpeed, it is also possible to calculate the ySpeed.

This is actually very similar as how we've calculated the xSpeed.



Again, we have to find the function we need to use based on the sides that are given.

Sin(α) = Opposite / Hypotenuse

Cos(α) = Adjacent / Hypotenuse

Tan(α) = Opposite / Adjacent

The Cos function handles both the Adjacent as the Hypotenuse, so that's going to be the function that we're going to work with. This lead us to the equation:

Cos(α) = Adjacent / Hypotenuse
Cos(Dir) = ySpeed / Speed
*ySpeed = Cos(Dir) * Speed*

And that gives us the equation to calculate the ySpeed from the players direction and speed. In code, these equations would look like this:

```
//Calculate xSpeed and ySpeed based on Dir and Speed
xSpeed = Math.Sin(Dir) * Speed;
ySpeed = Math.Cos(Dir) * Speed;
```

Conclusion

So that's it for this article. We learned that Sin, Cos and Tan is used a lot in game development. We learned how to use Sin, Cos and Tan to calculate corners of triangles with one corner of 90 degrees. We learned how to choose either Sin, Cos or Tan for our equations according to the values that we know. We learned how to put this into practice in game developing, and how to create an algorithm to calculate the direction of the player with its xSpeed and ySpeed. We've learned that in game developing, almost all programming languages use radians instead of degrees. We also learned how to set up the equations to calculate the player's xSpeed and ySpeed with its direction and speed.

