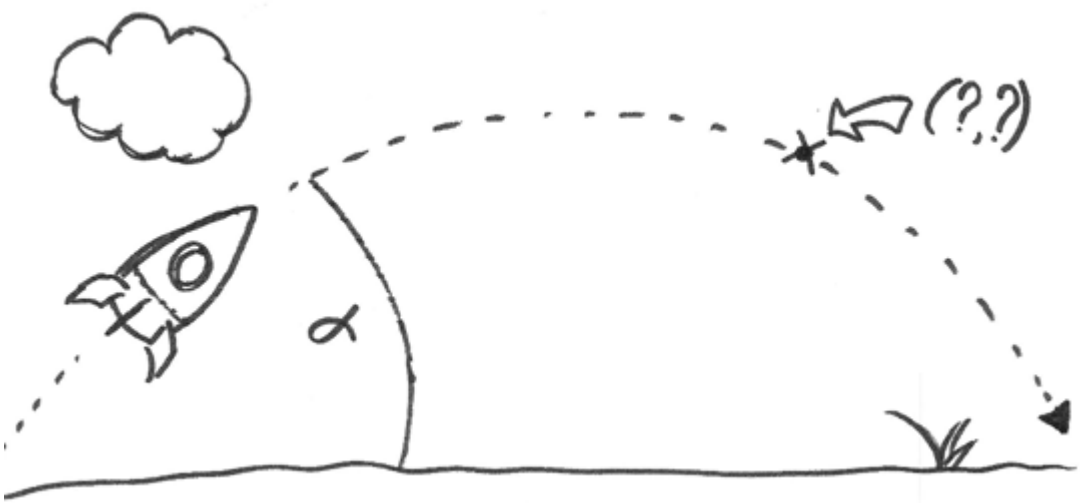
Basic Game Math Part 1 – Coordinate Spaces

Table of Contents

Introduction

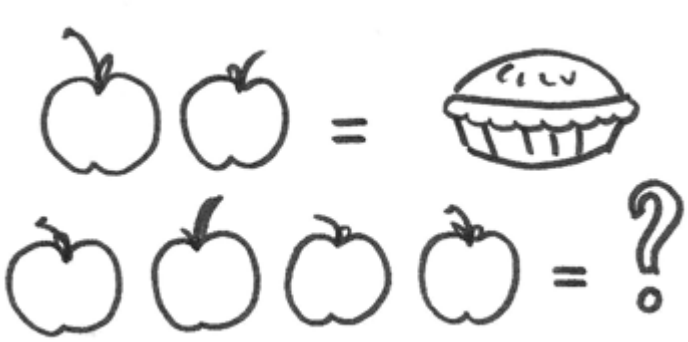


In Game Programming you often have to deal with objects that move across screen from position to position in a world that either looks flat as a cartoon or has depth like the real world. As a game programmer that means you'll have to come to terms with things like positions, dimensions, movement, speed,

velocity, vectors, angles in degrees and radians, trigonometry, dot products, etc, etc. All in all quite a number of things you'll have to master to reach your goal of Best. Game. Dev. Evah.

In this series, I'll try to take to you through a bunch of these terms and explain what they mean with examples abound. The first part will be about distances and dimensions also known as the Cartesian Coordinate System, which allows you to understand how things are positioned on screen, and more importantly will allow us to be on the same page when we move on to Vectors. As far as possible the game world will be a flat cartoony world and we'll postpone the intricacies of dealing with depth until later.

What you should already know



For some reason, most people without fail are able to solve riddles like this:

"Your friend requires 2 apples to make 1 apple pie, but you loooove apple pie so you give him 4 apples. How many pie's can he make ?"

However, if we rewrite this in a slightly less attractive format:

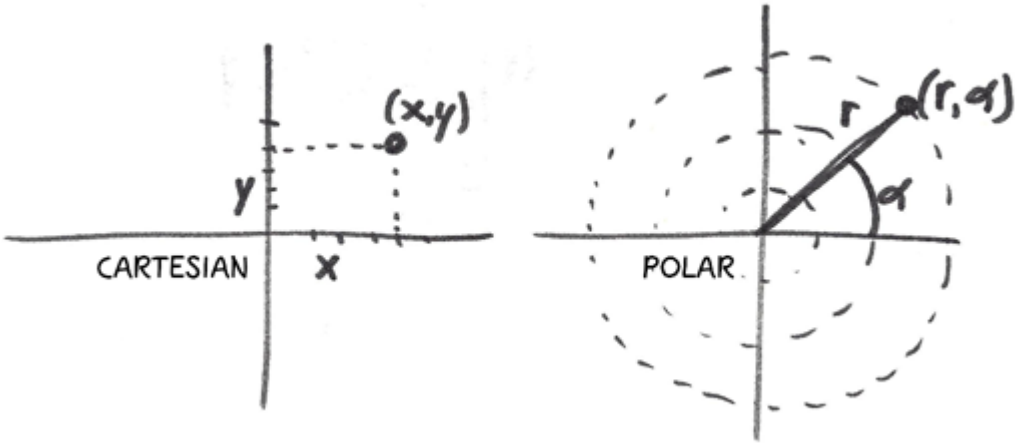
y = 2 * x, (where x is the amount of pies, and y is the number

of apples,) they freeze up like a deer in the headlights.

I get it, pie's are familiar and much more tasty, however this series does assume you have some basic algebra skills such as adding, subtracting, multiplying etc and solving simple formula's such as the one above with one unknown. If not, do not despair, build up your basic math skills (check out betterexplained.com or khanacademy.org for example) and get back later.

Let's get started.

"Get to the point" aka "the Cartesian Coordinate System"



As said before, Game programming often deals with positions and movement of objects in some way. To clearly and without ambiguity indicate where objects should be placed in a certain space, people much smarter than me came up with what we call a ***coordinate system***, also called ***frame of reference***. Coordinate being another word for *position* and system denoting that we are dealing with *a set of rules* for how to resolve those coordinates. *Resolving* here means turning some sort of abstract notation like (200,300) into an actual position on screen.

Two of the most well known systems to do so are called the ***Cartesian*** coordinate system and the ***Polar*** Coordinate System. Both are valid ways to tell us where a specific point in space is but in this post we'll look into the *Cartesian* Coordinate System first.

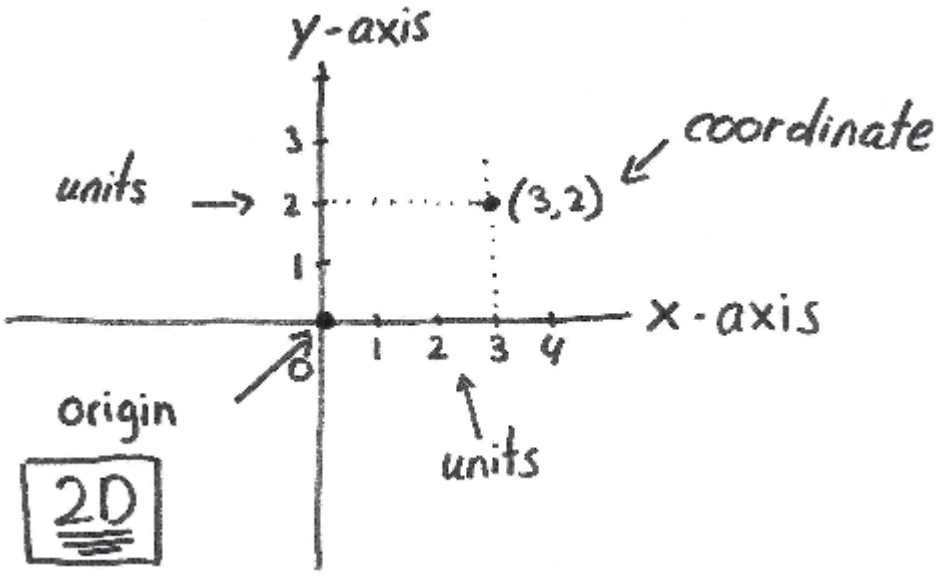
So let's pick this apart, the Cartesian Coordinate System:

- "Cartesian" because the system is named after it's inventor Rene Descartes, a (pretty brilliant) french philosopher and mathematician
- "Coordinate" because the system deals with positions
- "System" because it consists of a number of elements that together lets you unambiguously resolve the positions it describes.

Let's have a look at which elements we are talking about.

Cartesian Coordinate System elements

Check out the image of a Cartesian Coordinate System below:



What elements does this system have?

Number of dimensions, axis & origin

In the image above we see we have two degrees of freedom.
We can move things from left to right, and we can move things up-down.

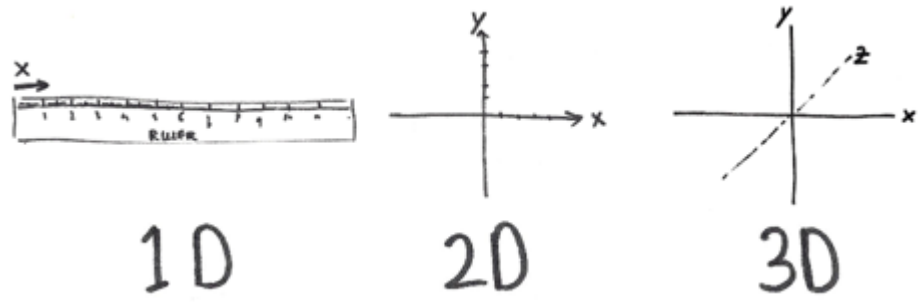
We call this *two dimensional*.

We can also have *one* dimensional, which is analogous to moving along a straight line, eg a ruler.
We can also have *three* dimensional, which is analogous to moving freely in the world, left-right, up-down, back & forth.

The number of dimensions is visible from the *amount of axis* we have.
Each axis denotes a degree of freedom we can move along.

In two dimensions, these are *usually* labelled x & y where positive x points to the right, and positive y points up.
Also to keep it simple for now, we also assume axis are perpendicular to each other (they form a straight cross).

In one dimension we would only have one axis x.
In two dimensions we would have two axis, x & y.
In three dimensions we would have three axis, x, y & z.
Below these three options are shown side by side:



As can be seen in the images above, all axis runs from negative values to positive values, with zero in the middle.

By convention the axis intersect at zero. We call this the *origin*.

Coordinates

Coordinates are denoted by numbers, which correspond to a specific point in our coordinate system.
Each and every coordinate matches with one exact spot in that system.

For 1 dimensional systems, a coordinate is a single number like 5.
For 2 dimensional systems, a coordinate is a tuple, which means two numbers, like (3,4).
Similarly for 3 dimensional systems, a coordinate is a triplet, aka three numbers, like (1,5,2)

The numbers indicate how many steps you have to take along a specific axis.
Sort of like finding a treasure using a treasure map.
Each number in a coordinate corresponds to a specific axis in the following order: x,y,z

In other words:

- (5) means: take 5 steps along the x axis.
- (3,4) means: take 3 steps along the x axis and 4 along the y.
- (1,5,2) means: take 1 step along the x axis, 5 along the y, and 2 along the z axis.

Of course, we need to know how big those steps should be, otherwise we'd still be lost.
This brings us to *units*.

Units

Turning a coordinate like (3,4) into a specific location in our coordinate system is called "*resolving a coordinate*".

In the case of (3,4) this requires taking 3 steps along the x axis and 4 along the y.
How big those steps are is determined by the unit size of the system, indicated by the little tick marks along the axis.

The size of these units are constant along a specific axis, and when it comes to gaming, usually also the same for every axis.
Although you can choose the size of these units arbitrarily, usually they aren't. Here are some non arbitrary examples:

- High school math: the unit size will often be 1 centimeter or an inch
- 2D Gaming: unit size will often be in pixels, since your screen is made up of them

Choices, choices…

It's good to realize what freedoms we have within the Cartesian Coordinate System and which things are fixed.

Fixed system elements:

- we have dimensions
- we have axis
- we have an origin
- we have units

Freedom of choice:

- the **amount** of dimensions we want to use
- the **direction** of our axis. Why not have positive y point downwards like some game engines do?
- the **size** of the units

Conclusions

This concludes it for this post, which laid some ground work for my next post, which will be about Vector basics.
Hope it was helpful to you and see you next time.