

Line Intersection (in C#)

Author: Mathias Verboven
Date: 18/11/2009

Introduction:

When you start programming games or other graphical applications that are a little more complex than usual there comes a moment where you have to find out if something is colliding with each other. Whether it be 2 graphical objects in the same layer in a 2D painting system or a player colliding with a bullet. The principle is the same, something is overlapping with something else and we have to find out where and how it does that.

There are plenty of ways one can do collision detection and I'll be covering a simple one in this tutorial. Line to line collision detection.

This tutorial uses C# as the programming language but it can be applied to others without much conversion.

Let's begin.

An important part of programming is setting everything up so you can reuse parts of your program, it's a big part of what we call object oriented programming (OOP).

We first create a class to hold our 'line' data. This class is called 'Line'.

```
public class Line {  
    public float x1, x2;  
    public float y1, y2;  
  
    public Line(float x1, float y1, float x2, float y2) {  
        this.x1 = x1; this.y1 = y1;  
        this.x2 = x2; this.y2 = y2;  
    }  
}
```

As you can see my Line class is really simple the only thing it holds as data is the 2 points which define the start and end of the line. It's a small class but it's much better to have this than to keep an array somewhere with 4 float values for each line you create.

The constructor needs 4 values to initialize the line. It's always a good idea to initialize your objects with values. Either force the programmer to give in values or use temporary ones.

Later when we do our math calculation we need to check if a value is inside a range. For this I have created another small class called 'Help' which in this tutorial only has 1 function 'isBetween'.

```

static class Help {
    static public Boolean isBetween(float value, float min, float max) {
        if (value >= min && value <= max) return true;
        return false;
    }
}

```

We use the 'static' word so we do not have to instantiate this object to use it. As a helper class I just want to access any functions or values I store in it without the extra code to instantiate it.

The idea of creating yet another class for something as simple as a single function is to organize your code. Once the program gets bigger you will create more and more functions that you can use somewhere else. It's a good idea to group these functions in well named classes. 'Help' might not be a good name, but it will do for now.

The actual code that will perform the intersection check is this:

```

private void ReCal() {
    float x1, x2, x3, x4, y1, y2, y3, y4;
    float ua, ub, ud;
    float x, y;

    x1 = l1.x1; x2 = l1.x2; x3 = l2.x1; x4 = l2.x2;
    y1 = l1.y1; y2 = l1.y2; y3 = l2.y1; y4 = l2.y2;

    ud = ((y4 - y3) * (x2 - x1) - (x4 - x3) * (y2 - y1));

    if (ud != 0) {
        ua = ((x4 - x3) * (y1 - y3) - (y4 - y3) * (x1 - x3)) / ud;
        ub = ((x2 - x1) * (y1 - y3) - (y2 - y1) * (x1 - x3)) / ud;

        if (Help.isBetween(ua, 0, 1) && Help.isBetween(ub, 0, 1)) {
            x = x1 + ua * (x2 - x1);
            y = y1 + ua * (y2 - y1);
        }
    }
}

```

To get an explanation of the actual math behind it visit this website:

<http://local.wasp.uwa.edu.au/~pbourke/geometry/lineline2d/>

Basically when the value of both ua and ub is between 0 and 1 the lines intersect somewhere on the line itself. To get that point we just multiply ua, which will act as a percentage with 0 being the start of the line and 1 being the end of the line, with the line itself.

The X and Y coordinates we get back from this multiplication defines the intersection point.

This code is just a basic example and might get adjusted in a real project. Group a few of these lines and you have a collision body of a character in an actual game.