Bugs suck. They’re frustrating, they can keep you sidetracked for hours, and fixing them doesn’t feel like making progress.

Debugging is a skill you can develop, rather than an annoyance you can prevent. This series of posts will help you improve that skill, starting with what you can do ***offline and on your own***.

**Narrow it down**

**Do you know *where* the problem is? Like, the exact line and the exact situation that causes the problem? It’s important to figure this out, unless you enjoy the dawning realisation that you just spent the last four hours looking in the wrong function.**

Your best friend in this case is trace(). Let’s say you’ve got some code like this:

[?](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[View Code](javascript:;) ACTIONSCRIPT3

|  |
| --- |
| enemy**.**position**.**x **+**= enemy**.**velocity**.**directionX **\*** enemy**.**velocity**.**speed**;**  enemy**.**position**.**y **+**= enemy**.**velocity**.**directionY **\*** enemy**.**velocity**.**speed**;**  **if** ( enemy**.**weapon**.**isCharged() )  {  enemy**.**fireAt( player**.**position**.**x**,** player**.**position**.**y )**;**  } |

…and you’re getting an error like this:

TypeError: Error #1009: Cannot access a property or method of a null object reference.

What a lot of new programmers do here is sit and stare at the above lines of code, trying to figure out just where the heck a problem could be occurring, and getting annoyed because it all *looks* fine.

Don’t do this.

You can easily narrow this down to a single line with a few trace statements:

[?](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[View Code](javascript:;) ACTIONSCRIPT3

|  |
| --- |
| trace( "a" )**;**  enemy**.**position**.**x **+**= enemy**.**velocity**.**directionX **\*** enemy**.**velocity**.**speed**;**  trace( "b" )**;**  enemy**.**position**.**y **+**= enemy**.**velocity**.**directionY **\*** enemy**.**velocity**.**speed**;**  trace( "c" )**;**  **if** ( enemy**.**weapon**.**isCharged() )  {  trace( "d" )**;**  enemy**.**fireAt( player**.**position**.**x**,** player**.**position**.**y )**;**  trace( "e" )**;**  }  trace( "f" )**;** |

You don’t have to write “sensible” traces that look professional, because the end-user is never going to see them (you’ll remove them as soon as you’ve solved the problem). Just stick a few letters in there. And don’t worry about whether it’s *really* necessary to have both the trace( "e" ) and trace( "f" ) statements there. Who cares?

Now your output will look something like this:

a

b

c

f

a

b

c

f

a

b

c

d

TypeError: Error #1009: Cannot access a property or method of a null object reference.

Follow this through in your code, and it’s easy to see that:

1. The problem only occurs after if ( enemy.weapon.isCharged() ) returns true
2. The error is thrown by the line enemy.fireAt( player.position.x, player.position.y );

The other thing that trace() is very useful for is getting a quick look at the values of variables at run-time. Now that we’ve figured out which line is causing the error, we can take a look at the inputs that cause it:

[?](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[View Code](javascript:;) ACTIONSCRIPT3

|  |
| --- |
| enemy**.**position**.**x **+**= enemy**.**velocity**.**directionX **\*** enemy**.**velocity**.**speed**;**  enemy**.**position**.**y **+**= enemy**.**velocity**.**directionY **\*** enemy**.**velocity**.**speed**;**  **if** ( enemy**.**weapon**.**isCharged() )  {  trace( "playerX:"**,** player**.**position**.**x )**;**  trace( "playerY:"**,** player**.**position**.**y )**;**  enemy**.**fireAt( player**.**position**.**x**,** player**.**position**.**y )**;**  } |

Note that I’ve removed all the trace( "a" ) lines as they’re no longer needed. When we run the game now, what output will we see?

TypeError: Error #1009: Cannot access a property or method of a null object reference.

Oh. Wait, what? Why aren’t those two traces being run? IT MUST BE A BUG IN FLASH. No, OK, obviously not, because Flash is perfect. Maybe the error is being thrown before the if this time? Or maybe… wait, why don’t we just check that:

[?](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[View Code](javascript:;) ACTIONSCRIPT3

|  |
| --- |
| enemy**.**position**.**x **+**= enemy**.**velocity**.**directionX **\*** enemy**.**velocity**.**speed**;**  enemy**.**position**.**y **+**= enemy**.**velocity**.**directionY **\*** enemy**.**velocity**.**speed**;**  **if** ( enemy**.**weapon**.**isCharged() )  {  trace( "weapon is charged" )**;**  trace( "playerX:"**,** player**.**position**.**x )**;**  trace( "playerY:"**,** player**.**position**.**y )**;**  enemy**.**fireAt( player**.**position**.**x**,** player**.**position**.**y )**;**  } |

Output:

weapon is charged

TypeError: Error #1009: Cannot access a property or method of a null object reference.

Ah. The traces themselves (or at least the first one) must be throwing the error. Time to narrow this down some more:

[?](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[View Code](javascript:;) ACTIONSCRIPT3

|  |
| --- |
| enemy**.**position**.**x **+**= enemy**.**velocity**.**directionX **\*** enemy**.**velocity**.**speed**;**  enemy**.**position**.**y **+**= enemy**.**velocity**.**directionY **\*** enemy**.**velocity**.**speed**;**  **if** ( enemy**.**weapon**.**isCharged() )  {  trace( "weapon is charged" )**;**  trace( "player:"**,** player )**;**  trace( "player pos:"**,** player**.**position )**;**  trace( "playerX:"**,** player**.**position**.**x )**;**  trace( "playerY:"**,** player**.**position**.**y )**;**  enemy**.**fireAt( player**.**position**.**x**,** player**.**position**.**y )**;**  } |

Output:

weapon is charged

player: [object PlayerEntity]

player pos: null

TypeError: Error #1009: Cannot access a property or method of a null object reference.

Aha! There’s the problem: player.position is null for some reason. Now that you know that, your next tasks are clear: figure out why it’s null, and fix it.

In this example, tracing the values of the variables involved helped us find the cause of the problem directly. Sometimes this won’t always be the case, of course, but it can still be very enlightening to know that the problem only occurs when the player is on the right side of the map, or the enemy is using a certain type of weapon, or two bullets have hit the player within the same tick, or the enemy is moving at high speed — any pattern that helps you figure out *when* a bug will happen will also help you figure out *why* that bug happens.

If you want to go beyond the trace statement, there’s an excellent free tool called [De MonsterDebugger](http://www.demonsterdebugger.com/) that I highly recommend.

**Clarify and Simplify**

You can’t debug code you don’t understand. And just because you wrote something yourself doesn’t mean you still remember exactly how all the pieces work together. As [Joel Spolsky says](http://www.joelonsoftware.com/articles/fog0000000043.html):

When you have a bug in your code that you see the first time you try to run it, you will be able to fix it in no time at all, because all the code is still fresh in your mind.

If you find a bug in some code that you wrote a few days ago, it will take you a while to hunt it down, but when you reread the code you wrote, you’ll remember everything and you’ll be able to fix the bug in a reasonable amount of time.

**But if you find a bug in code that you wrote a few months ago, you’ll probably have forgotten a lot of things about that code, and it’s much harder to fix. By that time you may be fixing somebody else’s code, and they may be in Aruba on vacation, in which case, fixing the bug is like science: you have to be slow, methodical, and meticulous, and you can’t be sure how long it will take to discover the cure**.

If you’ve got variables with silly names like fred and george, it’s going to stop you [being able to read your code](http://gamedev.michaeljameswilliams.com/2009/02/13/personal-coding-style/). Same goes for not bothering to add comments to long and complicated functions. Of course, this knowledge is only going to make you feel guilty, not actually help you debug. So what can you do *right now*?

You need to make the troublesome code as clear and simple in your mind as possible. Grab a piece of paper and a pen and start sketching out how it works. Maybe a diagram would clarify it, maybe working through the code as if you were the computer, or maybe just rewriting the method in plain English. Alternatively, you could try explaining the code out loud to someone else (they don’t have to be *real*).

The simple act of doing this will help you notice mistakes in your logic, and will solidify the way the code works in your mind. You’ll also have comments ready to add to the code so that you don’t have to think so hard next time.

***Simplification*** is all about reducing the number of things you have to concentrate on at any one time. Obviously, narrowing the problem down to one line and a specific situation helps with this, as does having single-purpose functions and readable code. But aside from that, there’s one huge mistake people make that goes directly against this: trying to fix two bugs at once.

Switching back and forth between one bug and another will just slow you down; your brain keeps breaking out of flow, and you have to “reload” the relevant information for each bug each time. Don’t bother. I’m not saying that you should avoid *having* multiple bugs at once — that’s ridiculous; if we could put a hard limit on the number of bugs, we’d set it at zero. Nor am I saying that you should fix one bug at a time, in between adding shiny new features. I’m saying you should focus on one bug at a time, and that means removing everything that could distract your attention to another bug:

* If you have little bugs that really irritate you but will only take a few seconds to fix, get rid of them right now. (These are often visual issues, like a misspelling in a menu screen.)
* **If you have two separate bugs that always occur at the same time, temporarily remove or greatly simplify the code for one of them (For instance, you might have a problem with the collision detection between your bullets and the enemies, and also with the fancy particle effects that stream out behind the bullet, so change collision detection to hitTests while you work on the particle effects.)**

Sometimes you’ll find that a bug is a lot more complicated than you imagined, in which case it’s fine to shift your attention to another one. Just don’t fall into the trap of switching again and again and again without actually fixing anything.

**Do Something Else**

Have you ever sat and scratched your head over a puzzle for hours, given up, and then suddenly realised, “Oh, *that’s* what it was!” hours later?

Sometimes it’s best to give your brain time to figure things out on its own, without your interference. So if it feels like you’ve been banging your head against a brick wall and getting nowhere solving your problem, the best thing to do might be to just relax and do something else for a while.

**Watch TV, make a sandwich, take a walk, have a chat, go to bed — just stop thinking about it. (You can check your emails or do some other work if you’re on the clock.) When you come back to the problem later, you might just have that “Oh!” moment and fix it in a few minutes. I can’t make any promises, though** ;)

In [the next part](http://gamedev.michaeljameswilliams.com/2009/05/17/fix-any-bug-online-help/) of this mini-series we’ll look at how to find answers on the web and in the documentation; in [the final part](http://gamedev.michaeljameswilliams.com/2009/05/17/fix-any-bug-online-help/) we’ll see how to get help from other people.

Do you have any tips on solo debugging that you’d like to share?

[Next in series](http://gamedev.michaeljameswilliams.com/2009/05/10/fix-any-bug-learn/)

Tagged as: [AS3](http://gamedev.michaeljameswilliams.com/tag/as3/), [bugs](http://gamedev.michaeljameswilliams.com/tag/bugs/), [debugging](http://gamedev.michaeljameswilliams.com/tag/debugging/), [errors](http://gamedev.michaeljameswilliams.com/tag/errors/), [trace](http://gamedev.michaeljameswilliams.com/tag/trace/)