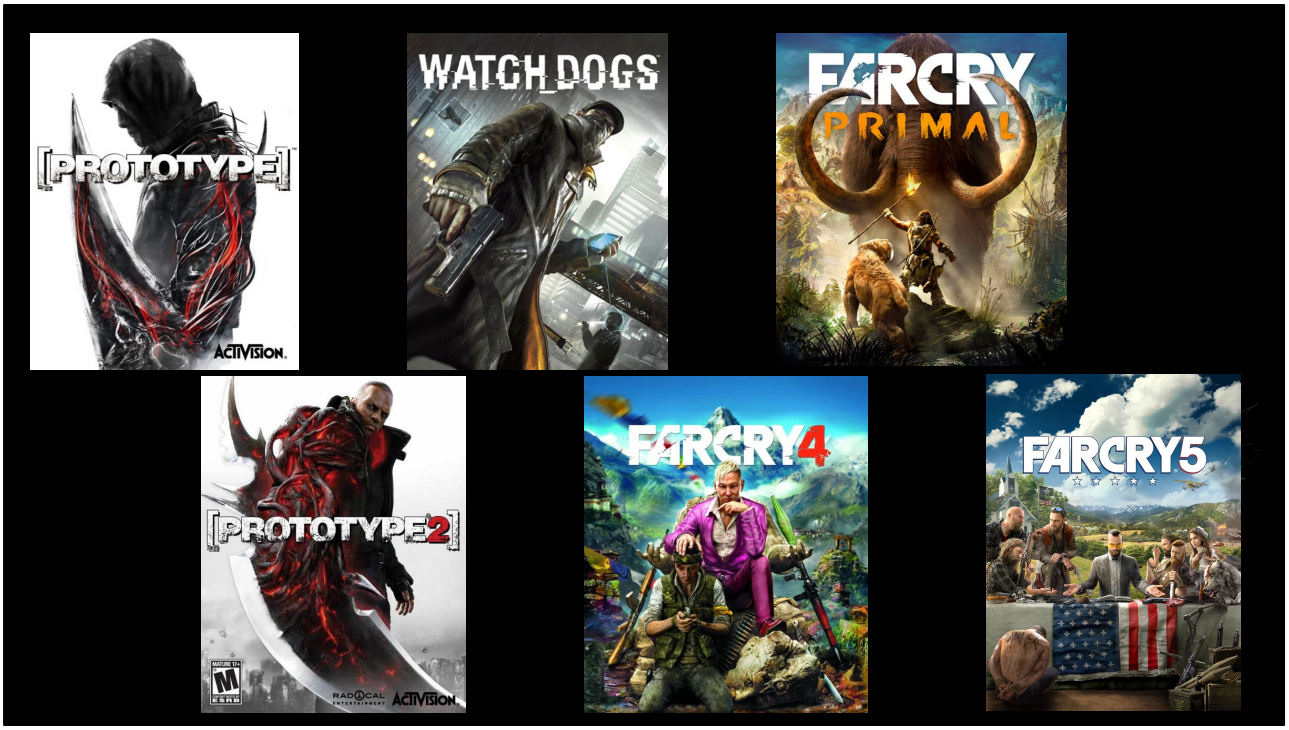
A Junior Coder Survival Guide

Keith O'Conor
CTO, Romero Games

Want to talk a bit about **THINGS I WISH I'D KNOWN** when getting into the industry, and some things to **KEEP IN MIND** for those who just have, or are planning to...



A little bit **ABOUT MYSELF**

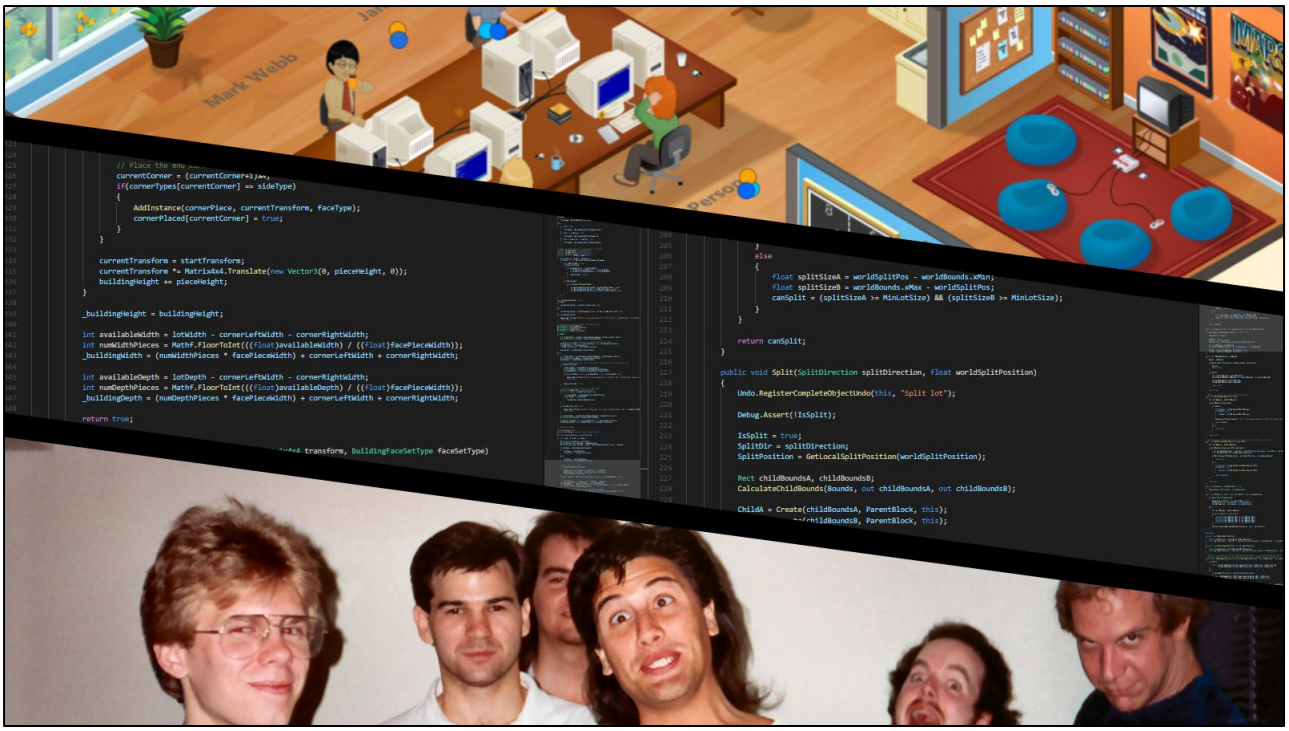
Ph.D. at **Trinity College Dublin**

Canada for ten years

Radical - **GRAPHICS PROGRAMMER** - Prototype series

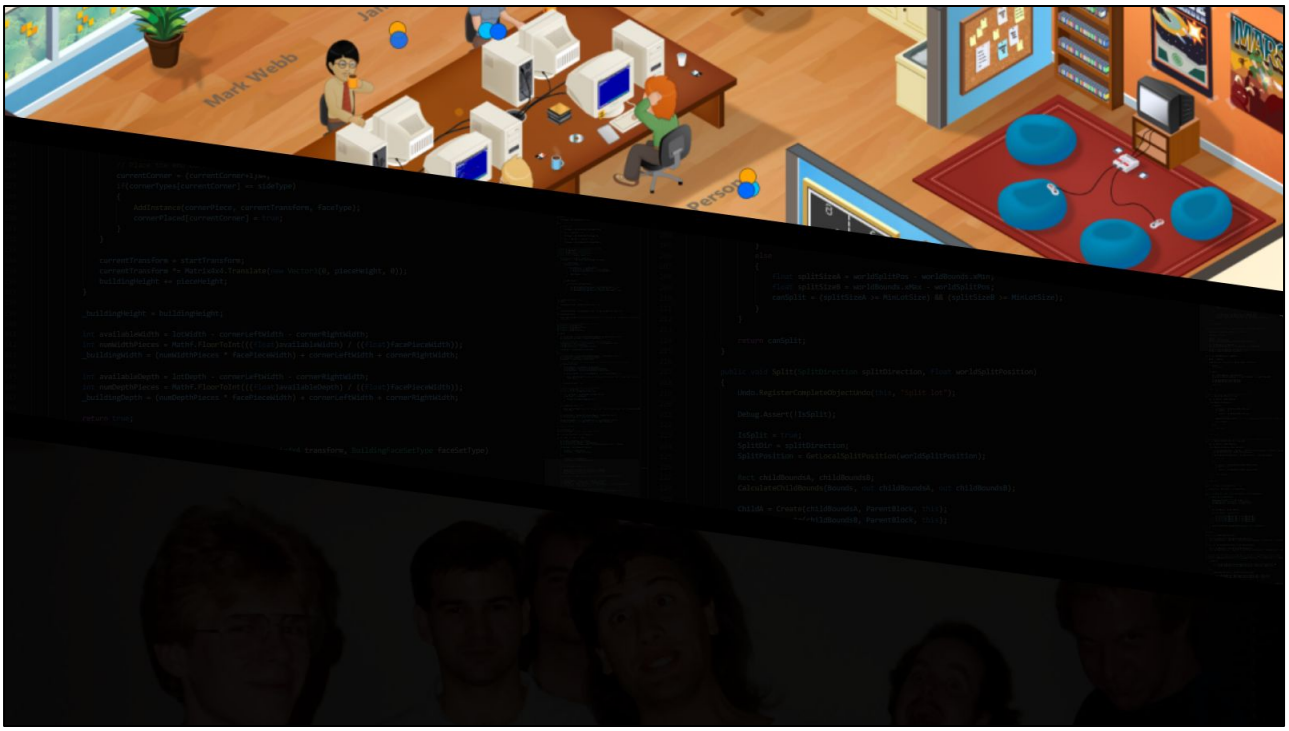
Ubisoft Montreal - **TECHNICAL LEAD** - Watch_Dogs, Far Cry

Moved home **LAST YEAR** to join John & Brenda at Romero Games



Three parts to being a junior game coder:

- Getting a **JOB**
- Day-to-day **CODING**
- Being part of a **TEAM**



To get a job, you **NEED TO DO** two things:

- **KNOW** your stuff
- **MARKET** yourself

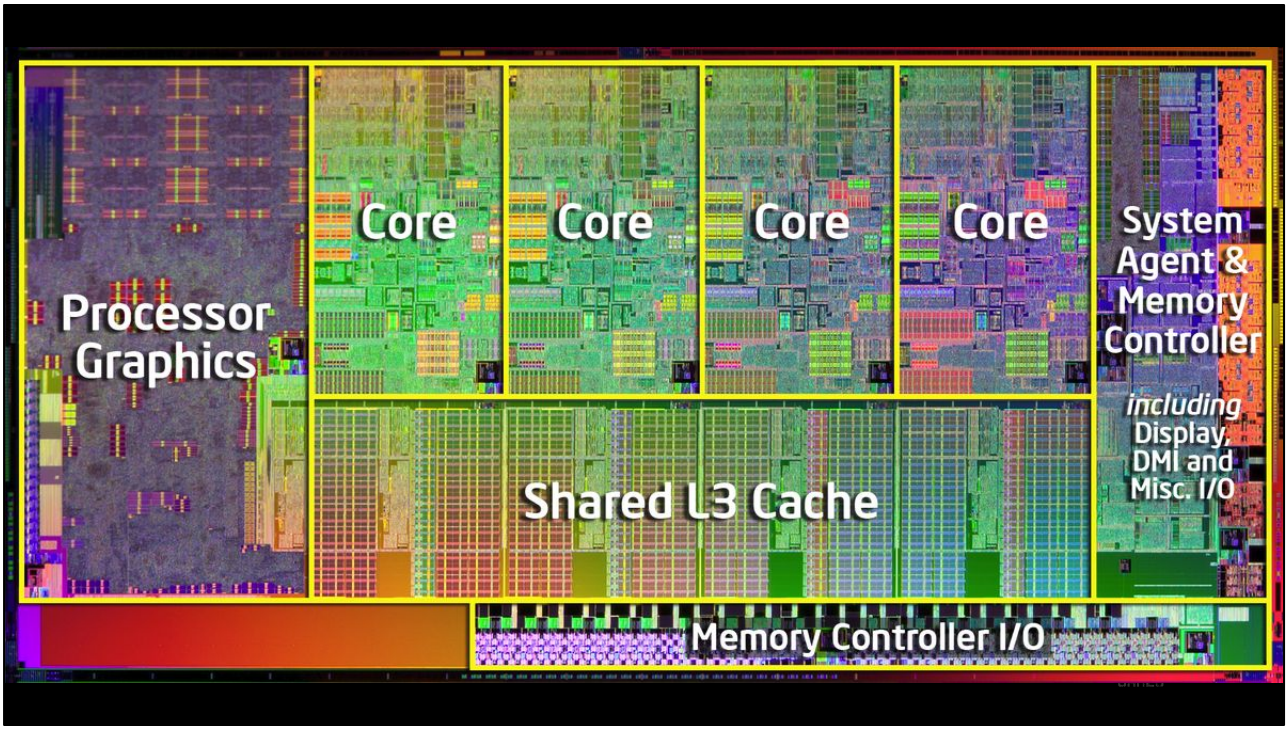
OBVIOUS requirement #1: you need to **KNOW A PROGRAMMING LANGUAGE**

C++ is the **BEST LANGUAGE** to learn for gamedev

Seems to be **CONTROVERSIAL** for some reason, and disappointing to see some **UNIVERSITIES** shying away from it

If you're not being taught it, **LEARN IT YOURSELF**. I was taught Java in college!

It's true, **C#** is popular because of Unity - although Unity's engine itself is written in C++

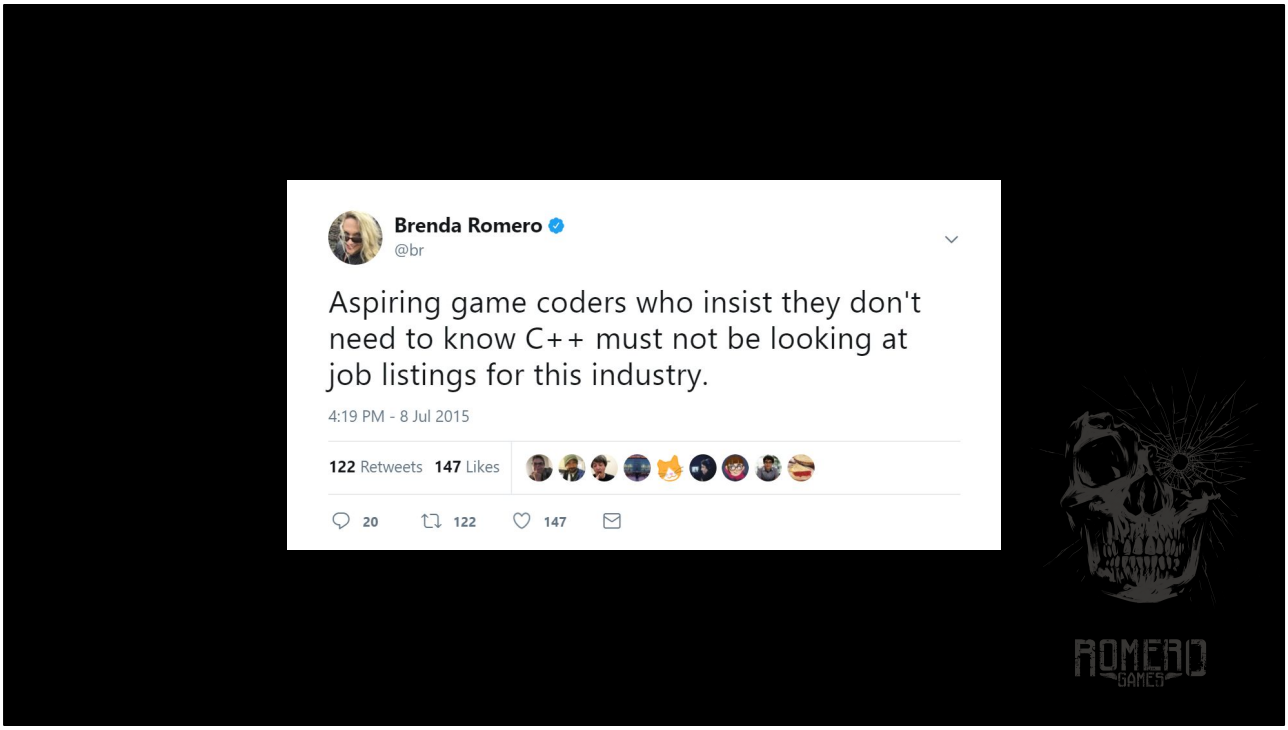


EMPLOYERS LIKE C++ EXPERIENCE, even for Unity jobs:

Understanding **CPU**s: registers, instructions, function calls, virtualization...

Understanding **MEMORY**: cache usage, allocation patterns

...understanding the **FUNDAMENTALS**, which leads to a better coder



Brenda put it best - just look at the **JOBS**

For **CONSOLES** and **AAA**, C++ knowledge is non-negotiable

Being **MULTILINGUAL** (C++, C#, Lua, Python) is a good sign of a well-rounded coder.
Picking up other higher-level languages are much easier **ONCE YOU UNDERSTAND C++**

OBVIOUS requirement #2: **MAKE GAMES**!

So **EASY** these days - so many **RESOURCES** for learning & doing

Good **MOTIVATION**: game jams! On your own, or with others.

24/48hrs are fun, but **I LIKE 1GAM** for students because it means you're more likely to tackle and **THINK THROUGH** problems that you might otherwise hack around in a short jam

First step in getting a job - **APPLY**!

Your main role in finding a job: make it **EASY** for people to **WANT TO INTERVIEW** you

APPLICATION & COVER LETTER: Be professional - nobody will hire a sloppy coder
CV: max 2 pages, don't pad, spell check
WEBSITE: Clean & easy, Contact details, CV, Portfolio (doesn't have to be **FLASHY,** get to the point)
SOURCE: Use source control, make it good code (clear, concise, easy to read, well-commented, well-structured)
EXECUTABLES: No missing dependencies(eg. missing DLLs), **TEST** on another machine
VIDEOS: Embed on the site
SOCIAL: You will be googled, behave!

Most important: **CODE** - you will be **JUDGED**, "Would I want this code in my codebase?"

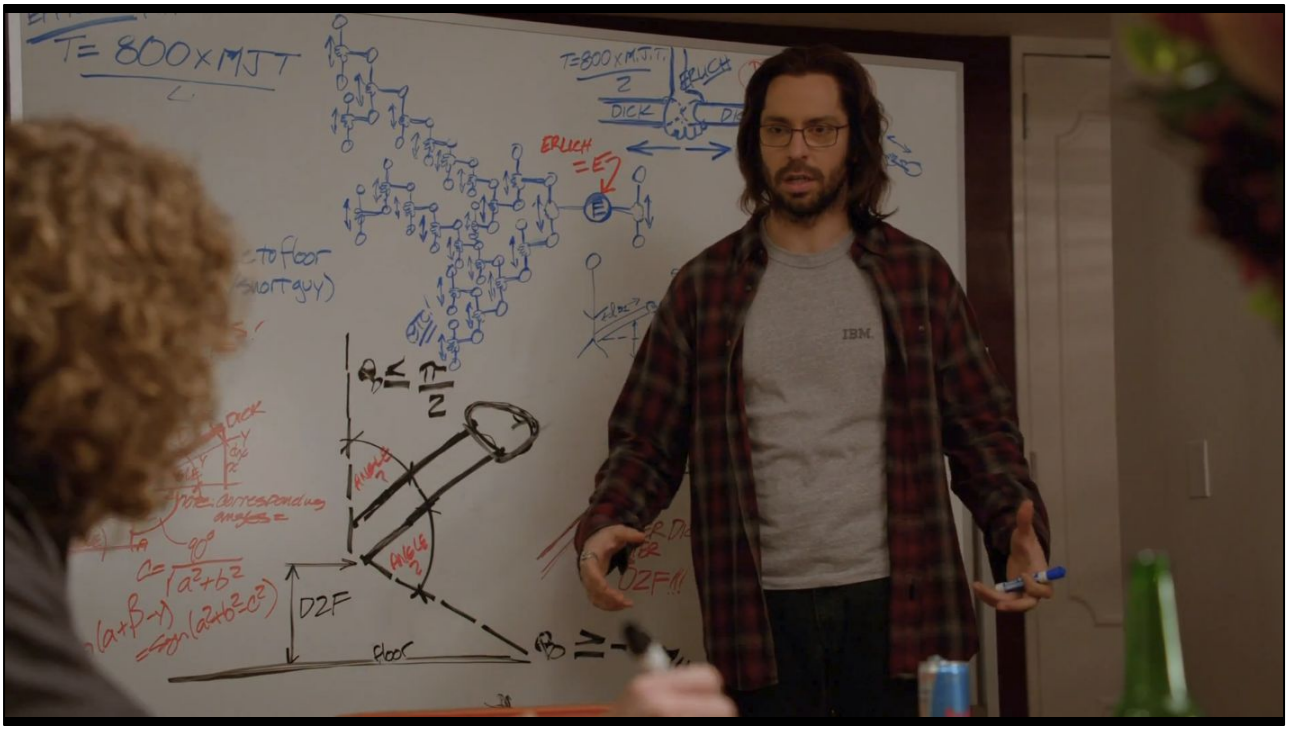
You'll finally get a call for an **INTERVIEW**

Do your homework. Play **THEIR GAMES** - you'll likely be asked about them (**BE DIPLOMATIC!** And don't fanboy or overly criticize)

Steer conversation towards **YOUR BEST WORK** (rehearse talking about it beforehand)

Two-way process; have **QUESTIONS FOR THEM** (anything; project, tools, life in studio, city, etc.)

Aim for them: find out **WHAT YOU KNOW**, and **HOW YOU THINK**

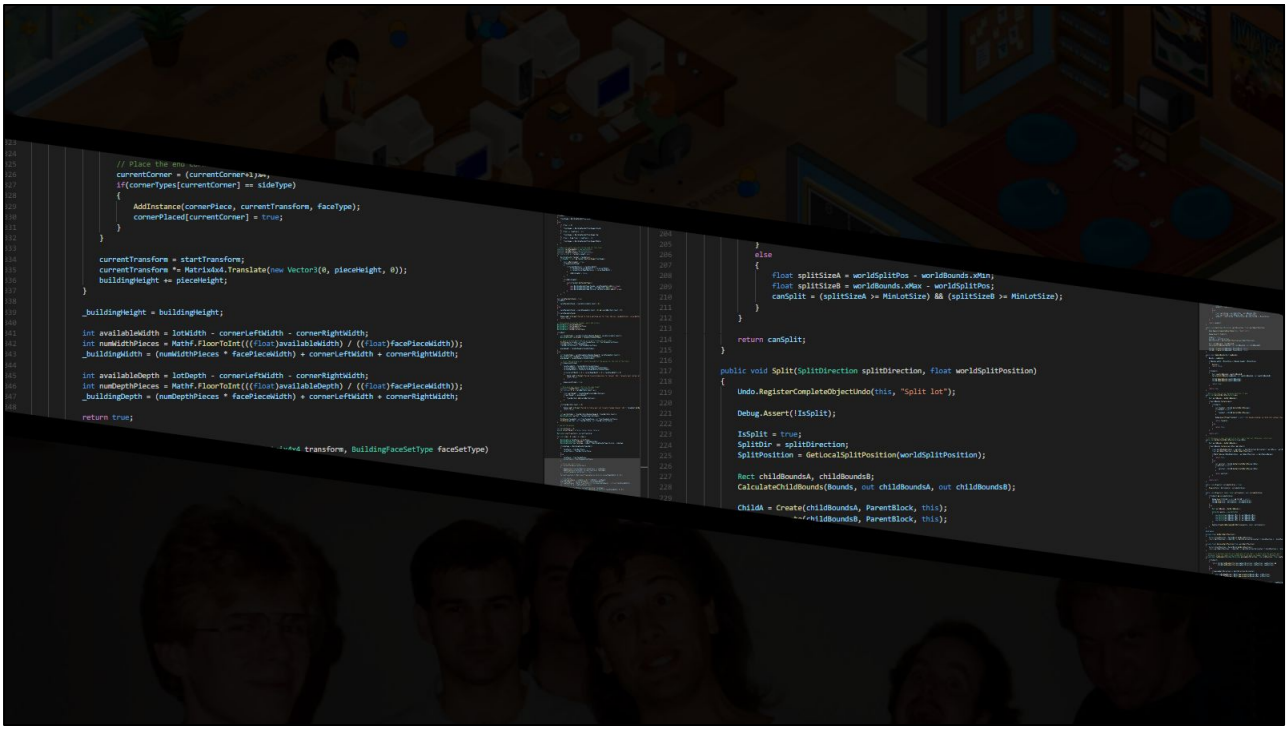


TECH INTERVIEWs can be anything - practice the **COMMON** code questions

I've been asked to write string reversal on a **WHITEBOARD,** been sat in front of **WORDPAD,** had tech interviews that lasted **FULL DAYS**, been interviewed by a room of **9 PEOPLE**...

Stressful!
If you get **STUCK,** stay **CALM**
Identify **ASSUMPTIONS**,
Talk through your **THOUGHT PROCESS**
Ask for **CLARIFICATION**

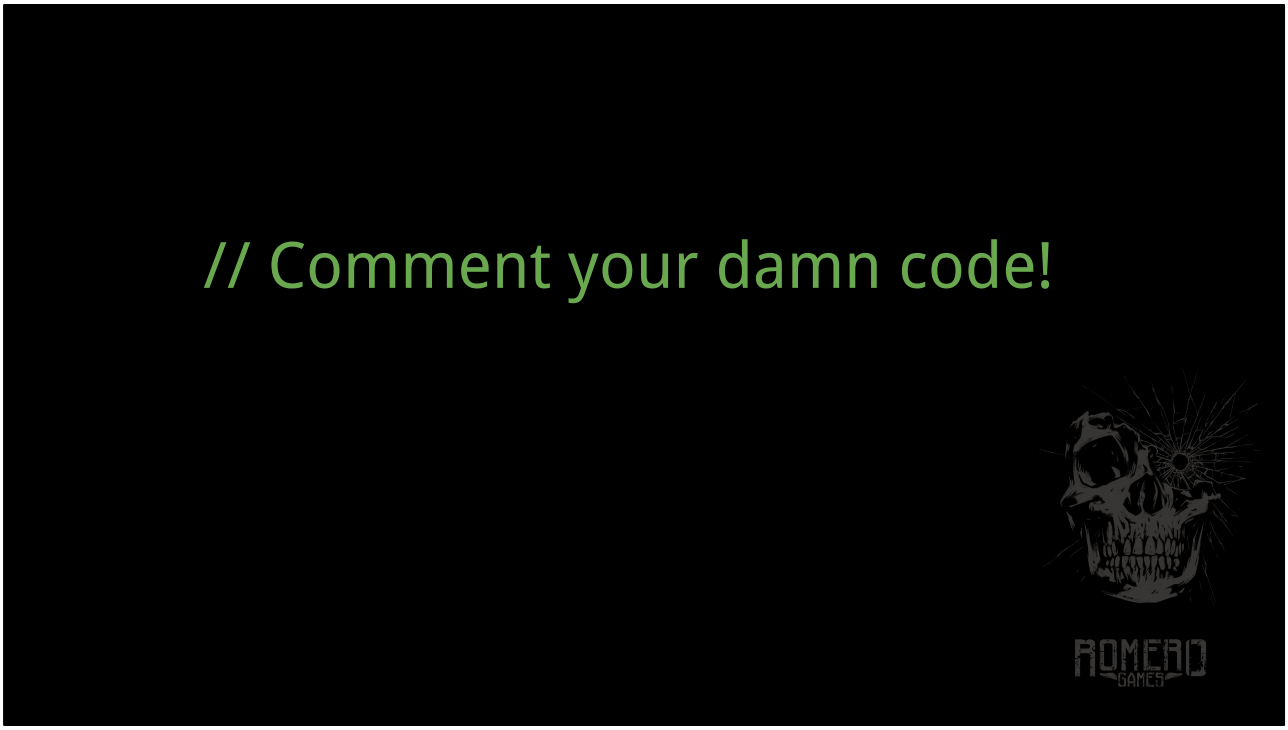
Last resort; say you **DON'T KNOW**, describe how you'd **FIND OUT**



Once you do get a job, comes the fun part - **CODING**!

SO MANY things I could about here, so instead I wanted to touch on a couple of the **MOST COMMON PITFALLS** hit by juniors

But first, one **SPECIFIC** to get out of the way...

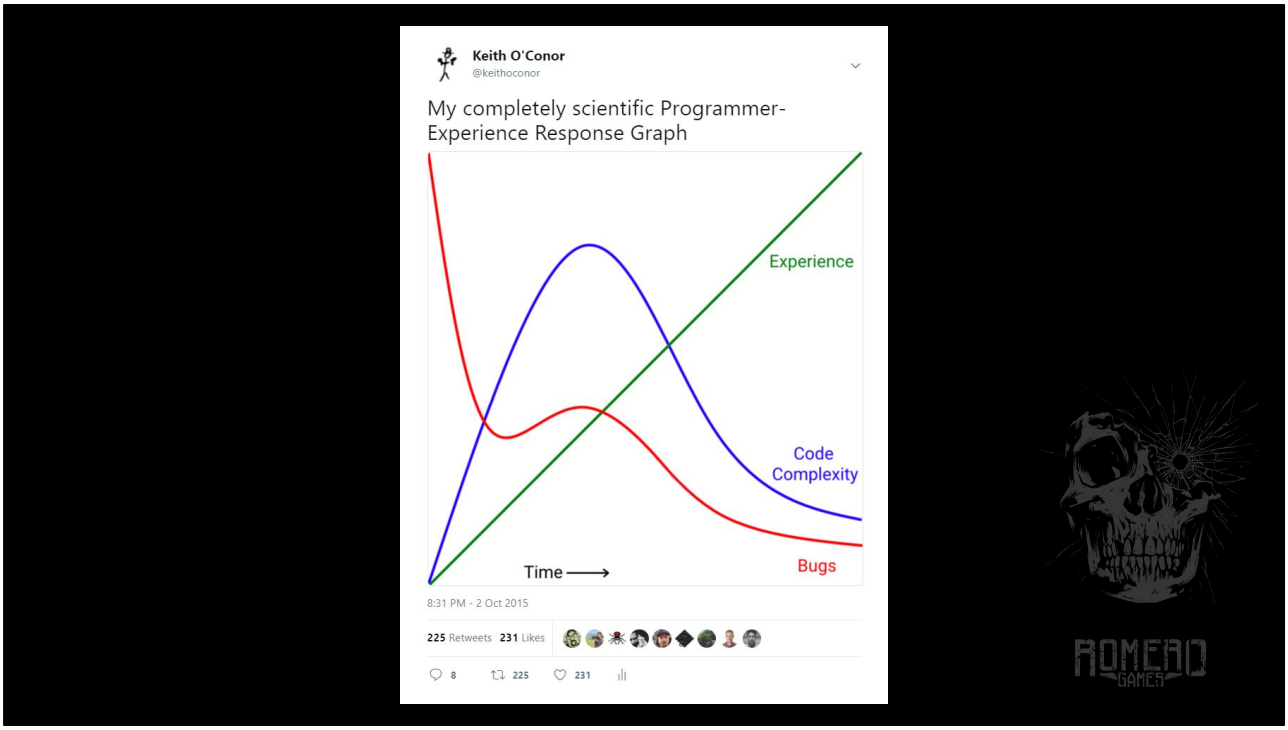


EASY TO FORGET but so important for a **HEALTHY CODEBASE…**

What seems **OBVIOUS** to you while knee-deep in the problem isn't necessarily so

Write comments **FOR OTHERS**, but also for you **SIX MONTHS FROM NOW** - you'll write a lot of code between now and then

But I'm more interested in talking about **HIGHER LEVEL** concerns...



One of the **MOST IMPORTANT LESSONS** a junior coder can learn - or any coder really - is about **CODE COMPLEXITY**

Juniors in particular have a strong tendency to **OVER-GENERALIZE** their solutions They want to solve this problem once and forever, and try to deal with **EVERY EVENTUALITY** and possible use.

Generalization means more **ABSTRACTION,** which means more **COMPLEXITY**, which means **HARD TO UNDERSTAND** code, which means more **BUGS**

Or worse, juniors want to **SHOW OFF** by solving problems in clever way with coding acrobatics. Same result: **COMPLEX**, **BUG-PRONE** code.

With **SENIORITY** comes a strong desire to keep things **SIMPLE** yet neatly functional. Good code "**JUST WORKS**" and is **READABLE**, **DEBUGGABLE** and **MAINTAINABLE**.

Linux

```
                1 MINGW64 /d/git/linux (master)
$ loc --include "(?:.+\.c|.+\.h)$" .
-------------------------------------------------------------------------------
Language             Files        Lines        Blank      Comment         Code
-------------------------------------------------------------------------------
C                    25456     17371714      2493594      1573298     13304822
C/C++ Header         19960      5125613       497892       700287      3927434
-------------------------------------------------------------------------------
Total                45416     22497327      2991486      2273585     17232256
-------------------------------------------------------------------------------
```

Frostbite

```
[dev-ew-sthlm] D:\ew\TnT>loc --include ".+\.(?:cpp|c|h|cs|py)$" .
-------------------------------------------------------------------------------
 Language             Files        Lines        Blank      Comment         Code
-------------------------------------------------------------------------------
 C++                  30627      9135470      1730556       552712      6852202
 C#                   29737      6847496      1200459       583281      5063756
 C/C++ Header         33115      5363305      1053585       732290      3577430
 Python                8427      2507009       358697       224974      1923338
 C                     2581      1808117       223365       265961      1318791
 Makefile                14         5676          866            5         4805
-------------------------------------------------------------------------------
 Total                104501     25667073      4567528      2359223     18740322
-------------------------------------------------------------------------------
```

Easier said than done - game development is **INHERENTLY COMPLEX** - Frostbite

Where complexity is **UNAVOIDABLE**, make it **OBVIOUS** and **UNDERSTANDABLE**

BEST GUIDELINE: err on the side of only fixing the immediate problem, but leave room to build on.
Some generality can be prudent; keep **SHORT-TERM PLANS** in mind when solving problems
LONG-TERM PLANS CHANGE, so complexity can easily be premature.

JOHN: "You're going to be writing new code later because you'll be smarter"

[screenshot courtesy of @repi]

At the **OTHER END** of the spectrum… resist the urge to **HACK** it together to just get it working now!
Deadlines always loom, there's always more work to do.

Sometimes it's hard for **JUNIORS TO DISTINGUISH** between a hack and regular code.
Ask, "Is this going to make my life harder in the future?" Experience helps to know the difference.

Sometimes a hack is **JUSTIFIED** as a short-term stopgap.
But beware, **TECHNICAL DEBT** is a silent and deadly **KILLER** of projects.

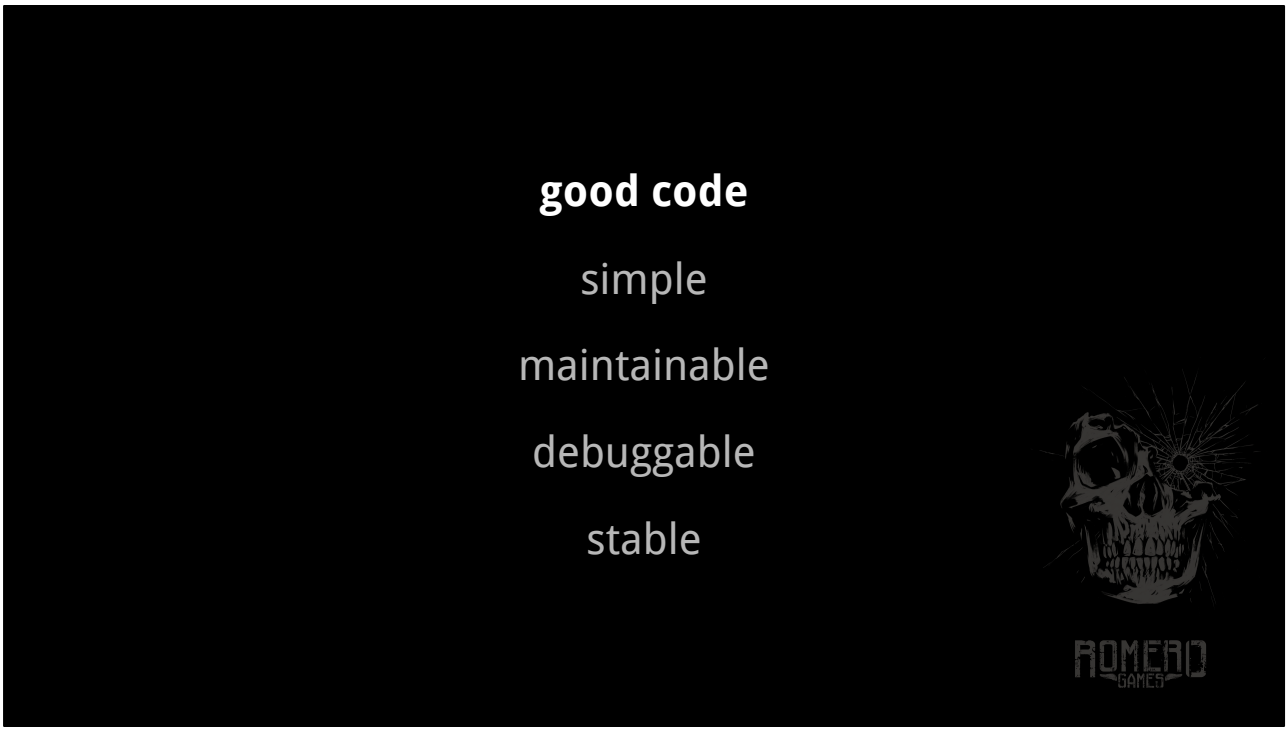
In a similar vein, a **98% SOLVED** problem is still a **PROBLEM** - only solving the general case isn't always enough

Recent bug; after **HOURS** of debugging, finally found **THE CAUSE** in an obscure part of the build pipeline….

```
else
{
    // todo
}
```

Realistically, most ‘todo’s remain that way for the **REST OF THE PROJECT**, and probably into the next and beyond.

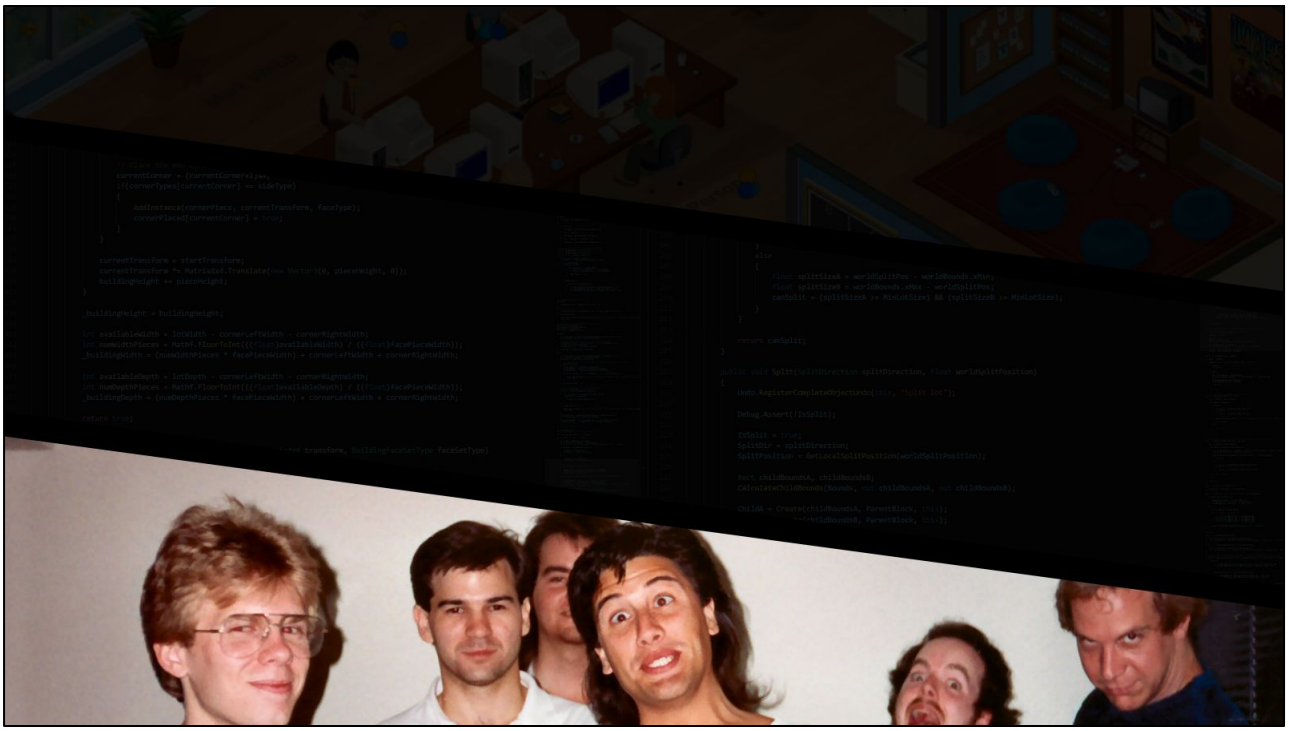
UNREAL ENGINE codebase**: 6,315** occurrences of 'todo' at time of writing



With all this in mind, there are three **THINGS TO CONSIDER** when working on any given code or feature:

How easy will the code be to **MAINTAIN** - is it readable, understandable, obvious and/or well-commented?
How easy is it to **DEBUG** if something goes wrong?
Is it likely to cause **STABILITY** problems if something goes wrong? How **RISKY** is it?

The last one is particularly important, because it not only affects you it **AFFECTS EVERYONE ELSE….**



That brings me to the last thing I wanted to talk about: **TEAMWORK**

One of the things I love about gamedev is the **CONTRAST** between coders, artists & designers

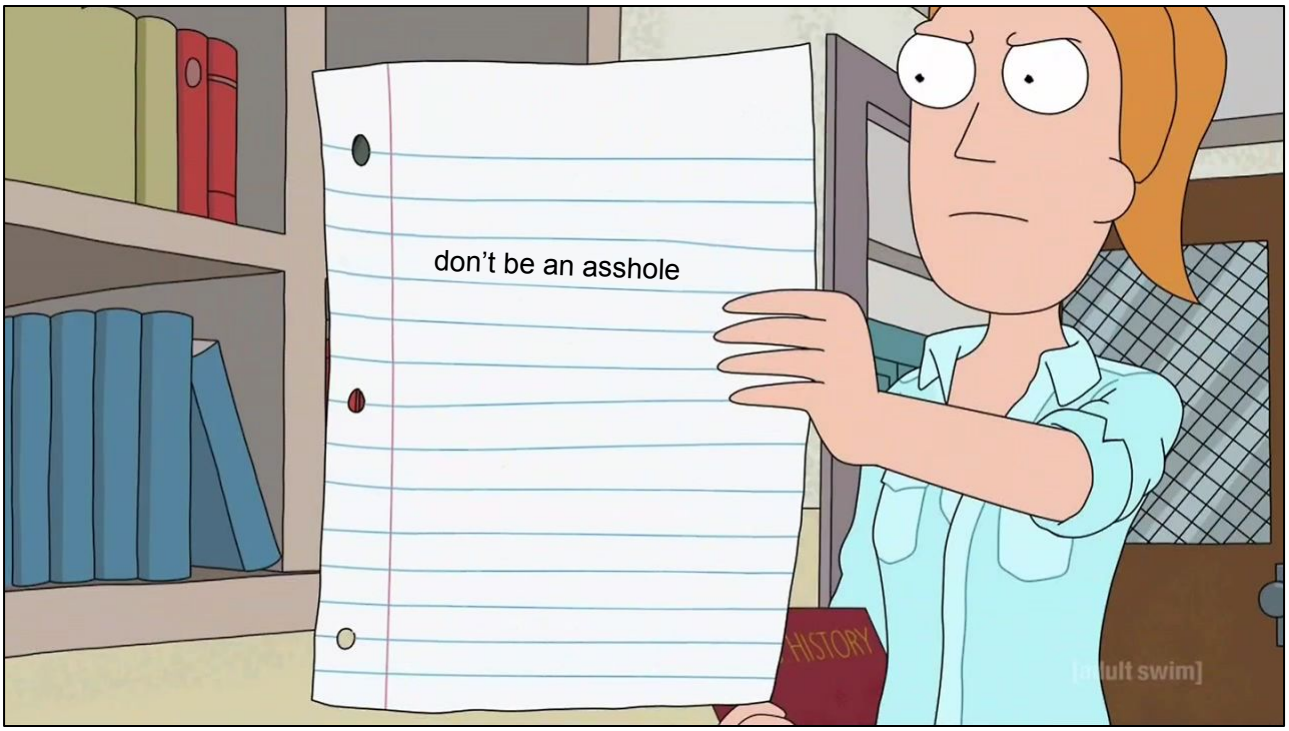
Everyone has an **AGENDA** of sorts; designers = **FUN**, artists = **BEAUTY**, coders = **FUNCTIONALITY**

An important part of being a game coder is learning to work with the other disciplines to **ACHIEVE A SHARED VISION**.

Without a doubt, the most important part of making it all happen is **COMMUNICATION**

I gave a talk at **STATE OF PLAY** earlier this year about communication, and my experiences working with other disciplines

There’s lots I could to say again, but I really wanted to **REITERATE ONE POINT**:



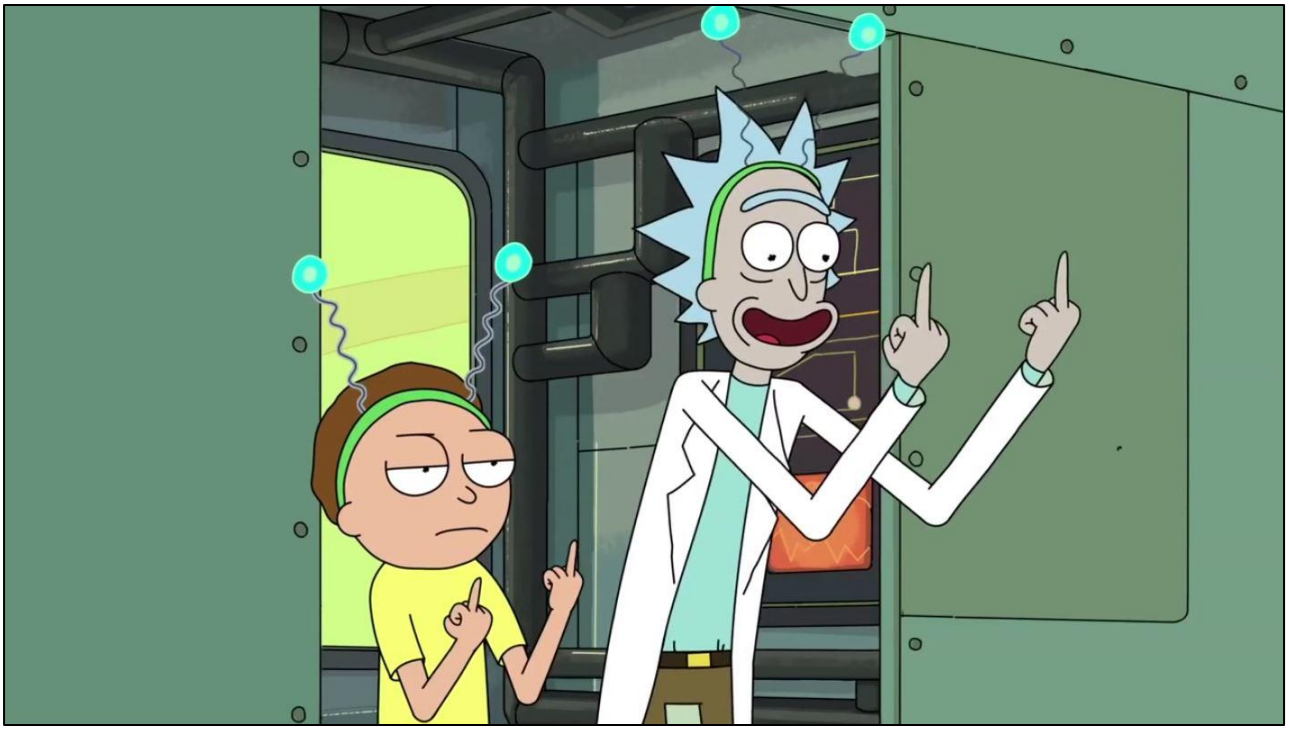
Seems like an **OBVIOUS** point, but it’s worth stating anyway.

Far from a problem with only juniors, but they can sometimes arrive **THINKING THEY KNOW MORE** than they actually do.

There’s always some coder who **TALKS DOWN** to other members on the team, particularly artists & designers.
They see themselves as the **SMARTEST** and so dismiss everyone else’s opinion.

It leads to an “**US vs. THEM**” attitude and is **TOXIC** to team spirit

The **WORST THING** you can do as a coder is to just say….



...**NO** to a request without bothering to give a valid reason beyond **TECHNOBABBLE**.
I've probably done it myself.
If you do, they'll probably just **FIND A WAY** to do what they want to do anyway.

Instead, **EXPLAIN** restrictions, and give **ALTERNATIVES**

Games need **EVERYONE** working together in the same direction.

Take any chance you get to **EXPLAIN HOW THINGS WORK**
Be approachable, **MAKE TIME** for people
Notable trait among the **BEST CODERS** I've worked with

Thankfully **KNOW-IT-ALL** juniors are a relatively small minority. Much more common is having **A CASE OF**….

IMPOSTOR SYNDROME: A feeling of somehow having stumbled into the job without really deserving it, and that you'll be found out at any minute.

This is compounded by the fact that everything seems to take **LONGER THAN YOU THINK IT SHOULD**.
This is usually only self-persecution - if you're worried about your perceived performance, talk to your lead.

The bad news: the impostor feeling will probably **NEVER GO AWAY**! I'm still feel like I'm making it up as I go along, and just hope people don't ask too many scrutinizing questions.

The good news: juniors are generally hired for their **POTENTIAL, NOT THEIR EXPERTISE**. Now is your chance to **LEARN ALL YOU CAN**, and grow into your new role!

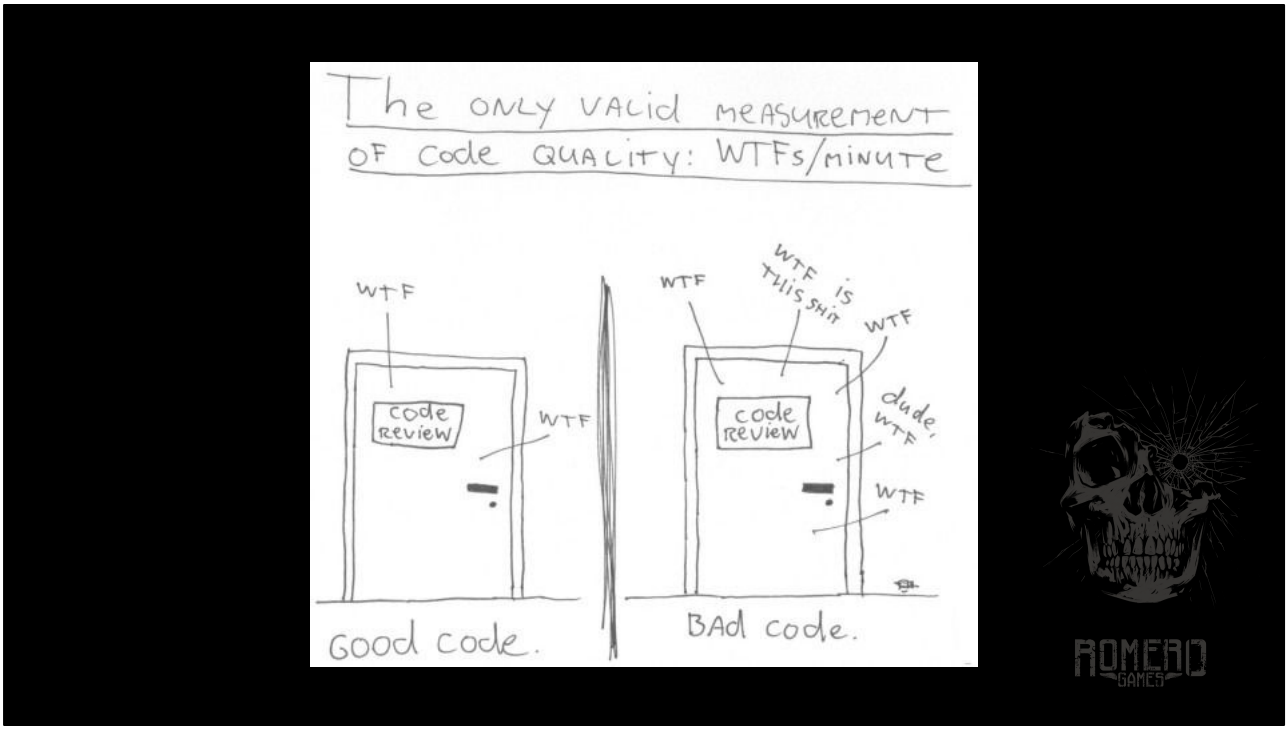
Being able to **LEARN FROM YOUR TEAMMATES** is part of what makes gamedev teams so awesome

Try and **FIGURE THINGS OUT YOURSELF**, but don't get stuck for too long - **ASK FOR HELP,** even if it's just to talk something through out loud

The **BEST WAY TO LEARN** is seeing how other coders solve problems, and what they think about
Even the **MOST SENIOR** coder still has things to learn from talking through with **OTHER CODERS**

If possible, find a **MENTOR** - a lot of coders are generous with their time and happy to help enthusiastic students and juniors

[http://stephaniehurlburt.com/blog/2016/11/14/list-of-engineers-willing-to-mentor-you]



A more structured and invaluable way to learn is to **ASK FOR REVIEWS**

Code reviews are awesome, but the earlier the better - **CODE DESIGN** reviews can be as productive if not more

Seniors will have a different **PERSPECTIVE** on your code & design, and the problem you're solving
They've **SEEN IT BEFORE**
Or they can think of **POTENTIAL PROBLEMS** that you might not
Or they have a better idea of the **BIG PICTURE** beyond your work
Or know some **PART OF THE ENGINE** that has an impact on your work
Or they know **SOMEONE ELSE** who is working on something similar or has related expertise
And so on

Don't be **DEFENSIVE** - learn!

One **LAST POINT** because I've seen it with many juniors...

As you settle in and **FIND YOUR FEET** on a game team, you probably hope to get straight to work on some **SEXY FEATURES**… AI, or advanced shaders, or core gameplay….

...only to find yourself **WORKING ON UI** or something else that might not really interest you that much.

A **LARGE AMOUNT** of game coding is **NOT SEXY** at all - build pipelines, localization, save games, asset exporters, etc. - but still **VITALLY IMPORTANT** to the overall game.

Don't be **DISCOURAGED** - as time passes you'll gain **EXPERIENCE**, confidence, and the trust of the team, and so work on more **HIGH-PROFILE** features.

No matter how seemingly minor the work is, there are always **INTERESTING PROBLEMS** to solve - find them, and **LEARN FROM THEM**.

And **KEEP LEARNING**. There are tons of great resources out there.

Having lunch at your desk? Pull up **GDC VAULT**!
Don't understand something or never even heard of it? **LOOK IT UP**!

Learn about **OLD TECHNIQUES** and keep up on new developments.
Keep **STRETCHING** yourself.

You'll **NEVER RUN OUT** of things to learn, and that's exciting - right?!

@keithoconor

keith@romerogames.com

http://www.fragmentbuffer.com