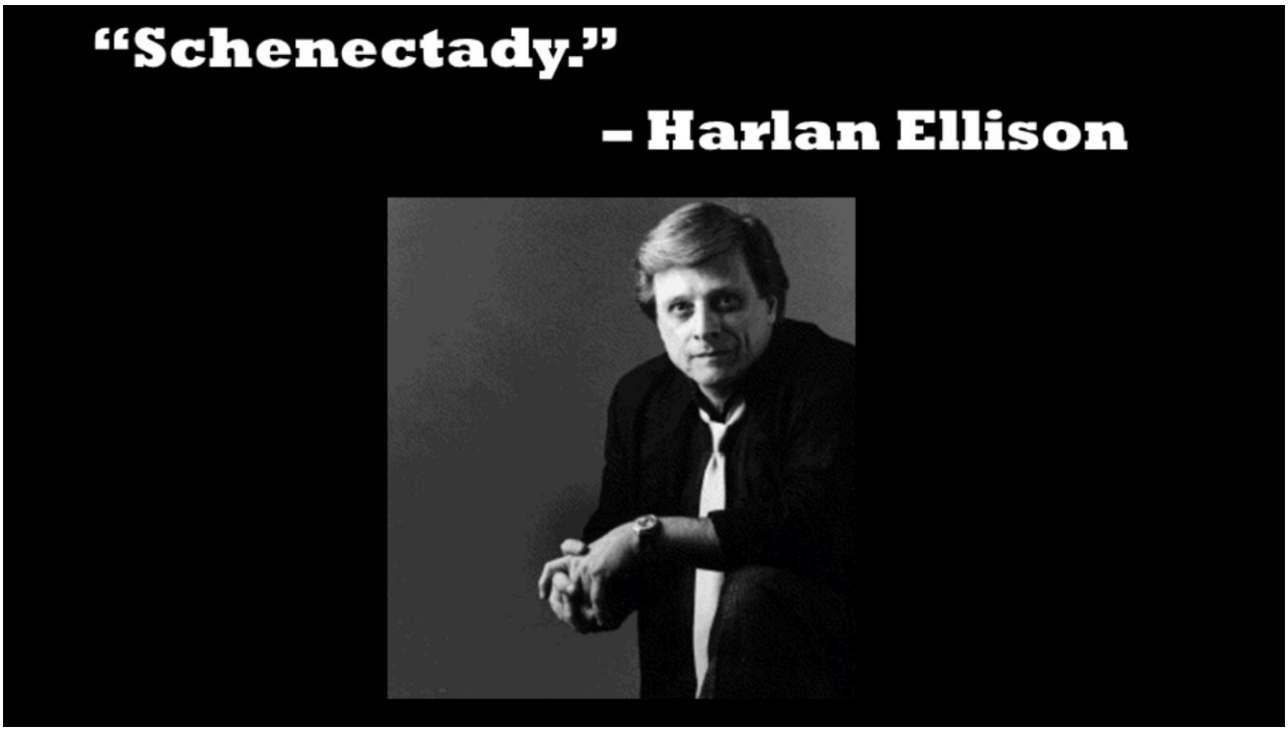
Welcome to the Rules of the Game session for 2016, I'm Richard and I'll be your host.

Isn't this the question every writer dreads?

As if there's some magical process to it.

Anyone who's been working in a creative field, like writing, like games, knows that ideas are actually a dime a dozen. Ideas are cheap, implementation is hard.

But don't get me wrong, good ideas do help.



So writers have come up with pithy answers…

“From the idea-of-the-month club.”

– Neil Gaiman

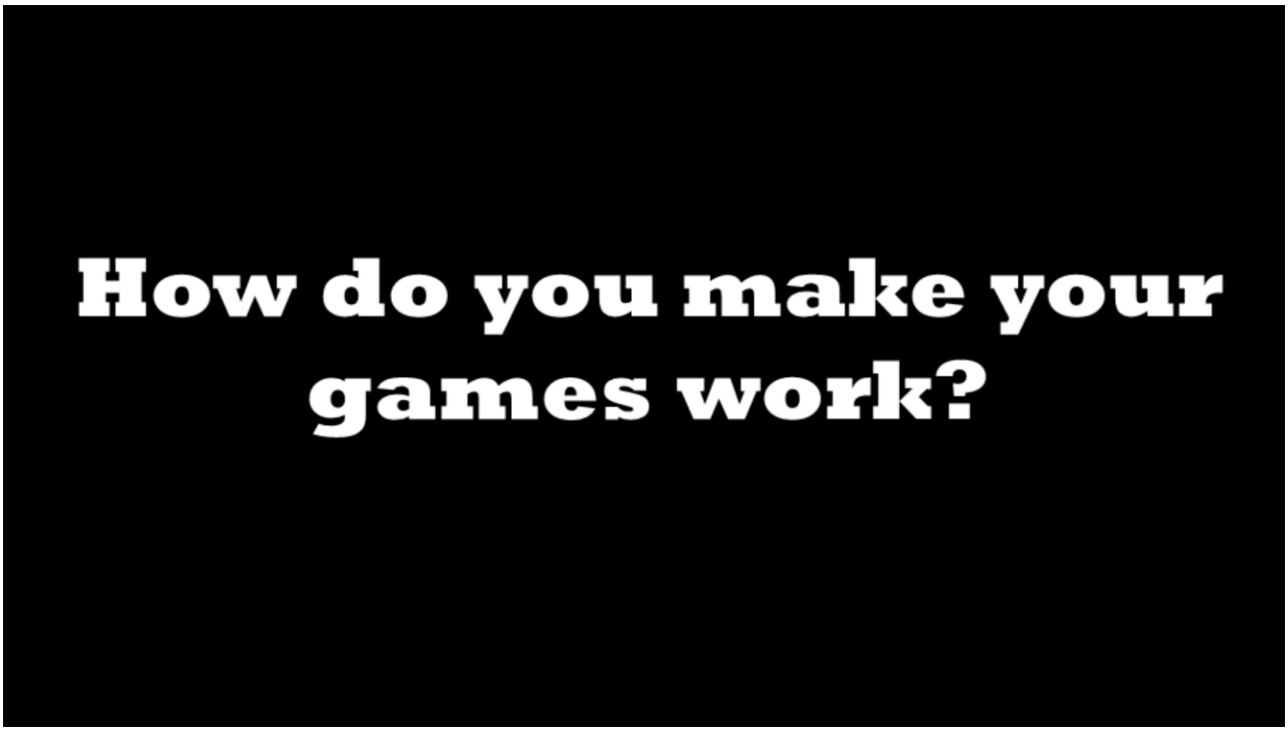
“I make them up. Out of my head.”

– Neil Gaiman

Where do you get your game ideas?

Protip for those of you maybe trying to get into games – avoid the word fun. Game designers don't like it, because it's too broad, too amorphous,

Of course we want people to have fun, but it's not a useful descriptor.



I think a far more interesting question to ask designers is this…

Because it’s going from the idea to the working game that’s the hard part.

How do you make your prototypes work?

How do you make your systems work?

How do you make your enemies work?

How do you make your teams work?

Welcome to the Rules of the Game session for 2016,

I'm Richard and I'll be your host.

I am the director/designer/writer at Paranoid Productions

where we are working on the action-infiltration game with the shifting narrative called The Church in the Darkness

Our format is simple – five designers get 10 minutes each to talk about a game design rule that is personal to them.

So last year we held the first edition of this session

We invited 5 accomplished designers to share one of the "rules" they work with and go into detail about it for 10 minutes.

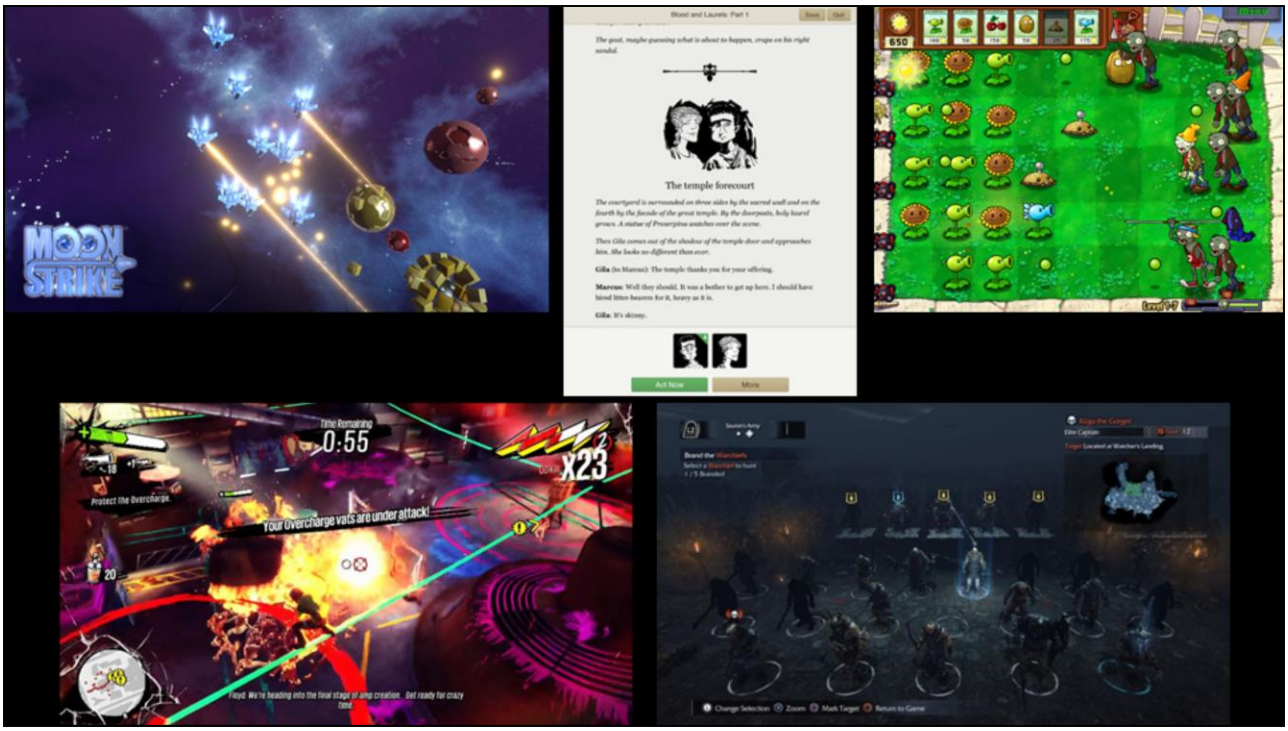
Last year's talk is up free on the GDC Vault, I encourage you to go see it!

That session went well, and I thought, hey, we should do this again, and here we are. So I recruited five new designers who I thought did great work and could talk about one of the rules behind that good work.

And as I said last year, game design rules are personal, not universal.

Just because you hear a rule here today doesn't mean it is the final word on a subject.

I find the rules we get out of this session fascinating because of what they tell us about their creators.



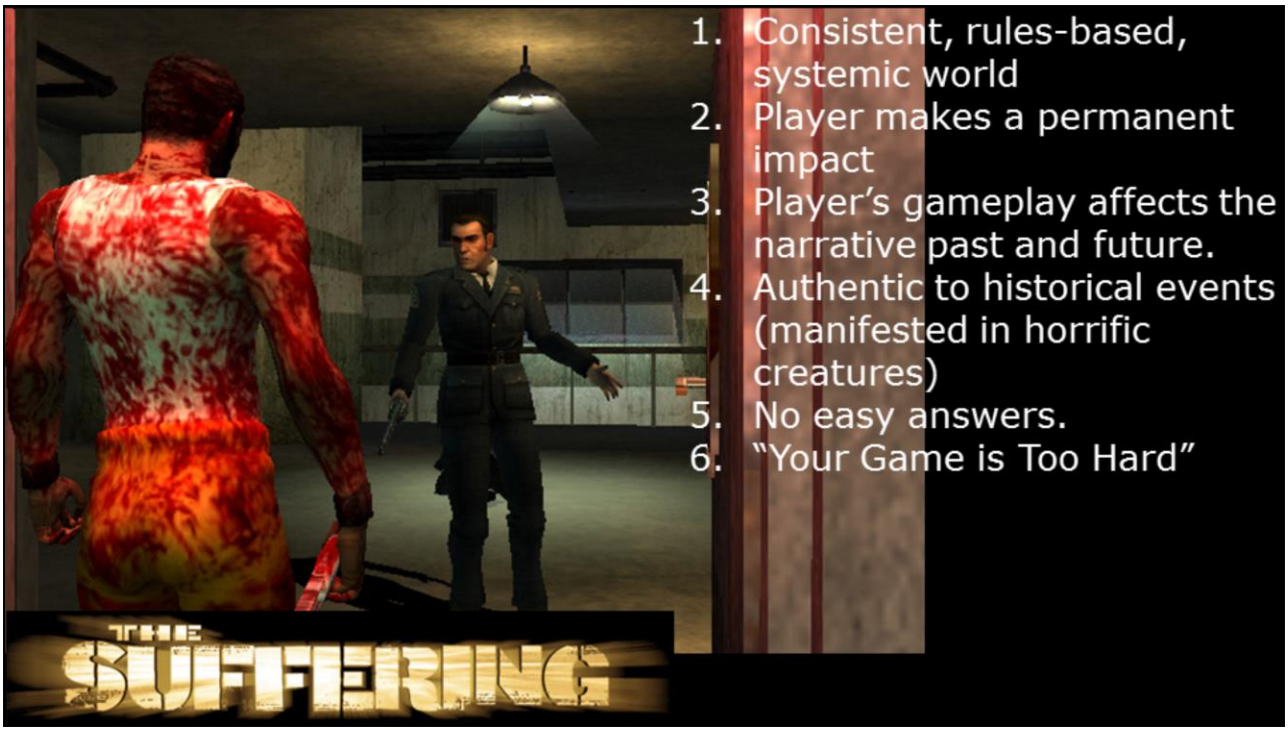
I want to know what rules the developers behind some of these fantastic games had going in their mind when they made them

Not so that I can copy them, but so that I can think of what my own versions of these rules may be.

I like to think of game design rules as a deck of cards that we as designers can choose to play when the time is right.

I’ve been working as a game designer for quite a while, so naturally I have my own deck of these rules

And sometimes I have rules specific to a project…



For example, the rules I used on The Suffering project looked something like this.

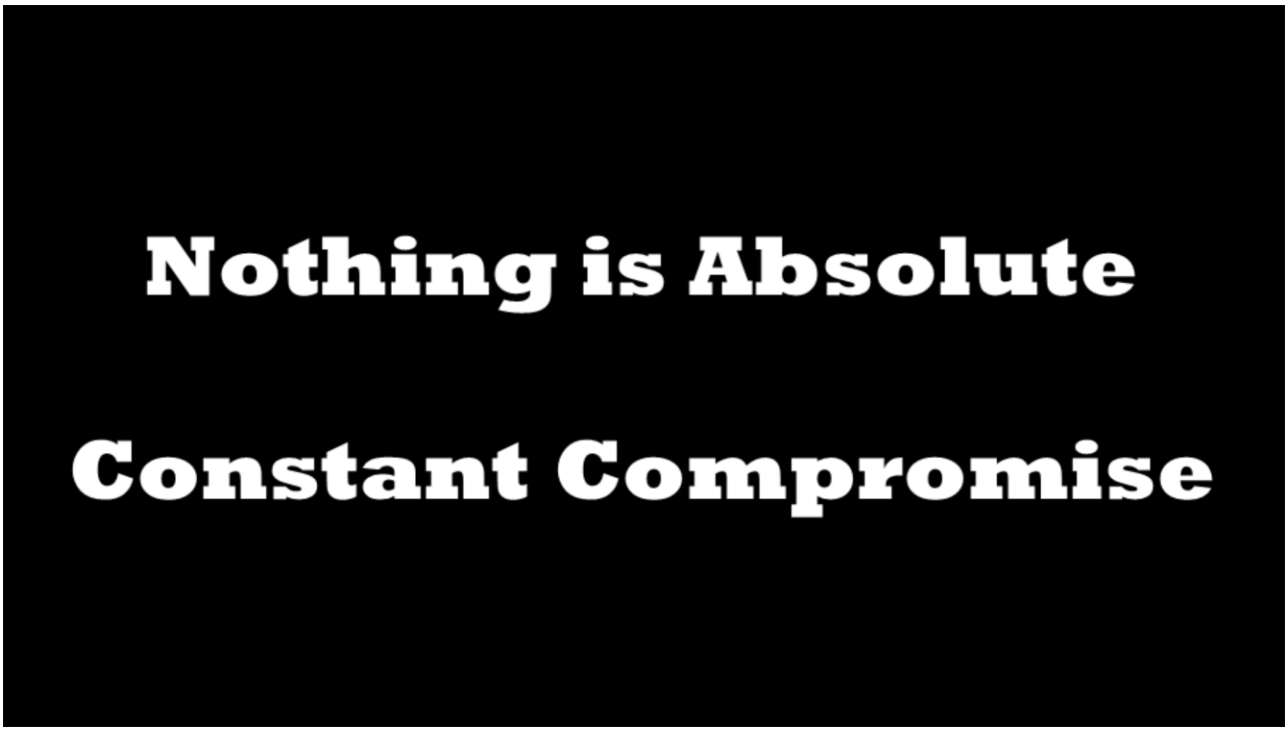
Now for the Church in the Darkness, a top-down action infiltration game set inside a religious cult…

Some of these rules changed, because of the game.

Some rules I realized were wrong and I changed them.

I used to think that all games were too hard, because of a team's necessary tendency to do that, but I realized you can overdo that – the Suffering turned out a bit too easy.

And of course, a new project probably brings new rules with it, that I may or may not continue to use in the future.



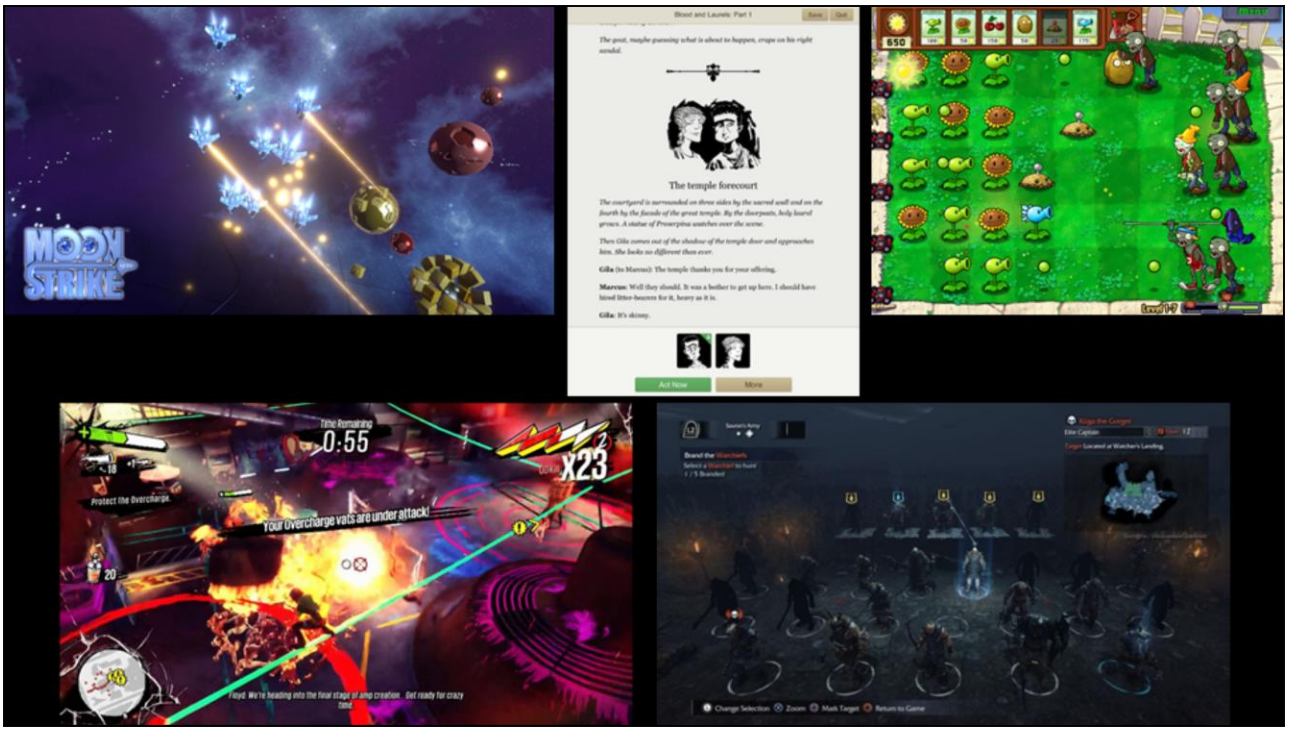
People tend to object to defining rules because there is a possibility for these things to be mis-interpreted.

So I should say, of course, that in the craft of game design nothing is absolute

And working with a team you need to be thinking of constant compromise.

BUT WHEN YOU ARE floating out to sea, into shark-infested waters…

I find that a strong personal goal or rule for yourself or your project can be that tool that pulls you out of shark territory and brings you back to the calm waters.



So coming up…

A little later on, Emily Short is going to tell us how she tests her game before it's ready for testing

George Fan is going to tell us about how your enemies are not as different as you think they are

Liz England is going to tell us who you should make your design documents for.

Michael de Plater is going to tell us how systems can bring human stories

But first! Lee Perry - Lee Perry is an experienced game designer, having worked on….

And he's going to tell us how everything you know is wrong about when to polish (and when to add pizazz to) your game.


Pizzazz First
Polish Later
Lee Perry - Big Dorks Entertainment

Starting out, like many, used to think games were about:
High concepts
Settings
Themes
THE CLASSIC BIG IDEAS!!!

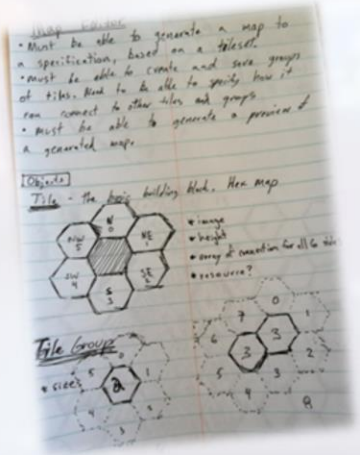
MAPS!
BIOS!

At some point I like to think most designers "level up" and realize that game design isn't ***JUST*** about the big concept…

Secret Sauce?

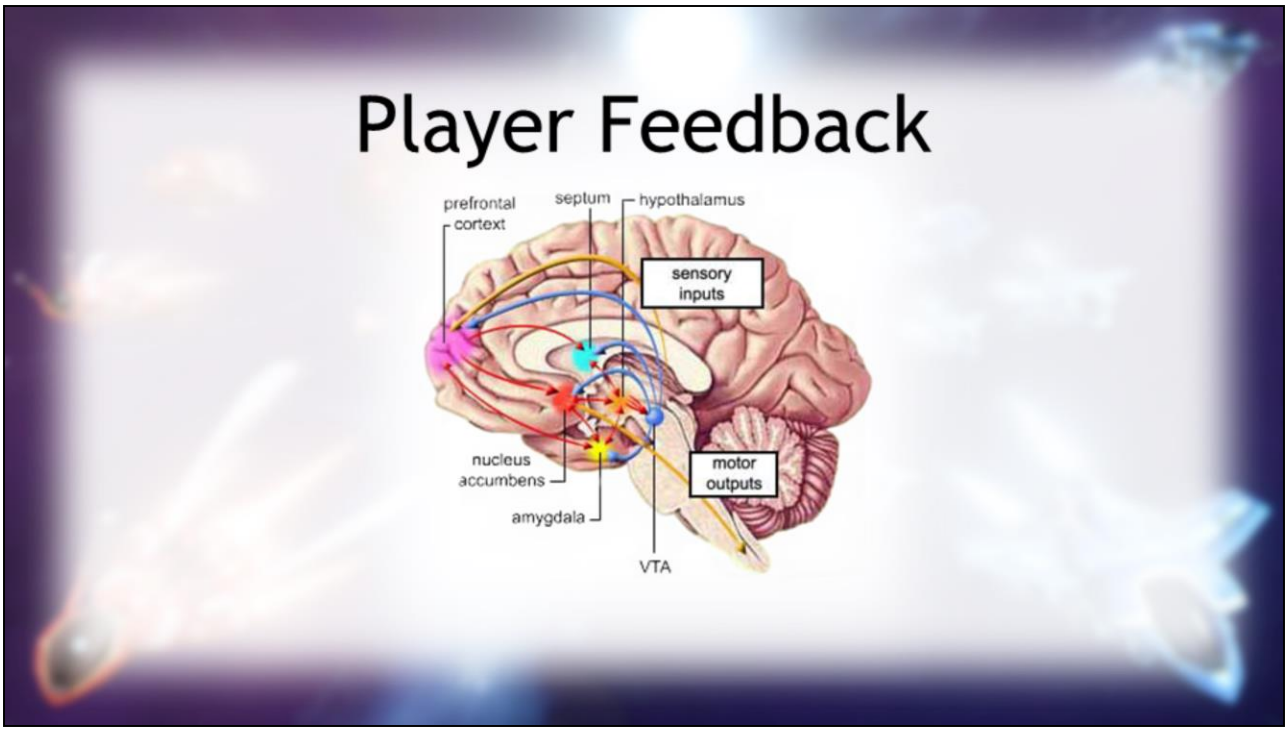
Great games have some secret sauce!

Secret Sauce?



NOT JUST ABOUT THIS…

IT'S ALSO ABOUT THIS



AND THIS

Interacting with the game, and having a really satisfying response!

MOST devs understand the importance of great player feedback

It's one **NOT SO SECRET SAUCE** that gives a game a sense of "character" and "soul".

It’s about the joy of Mario jumps! Fireballs! Bashing a block!

But PopCap is kind of among the Gods of Mt Olympus when it comes to making something *FEEL* awesome with amazing feedback.

1) They made the ball sound as it hits the pegs pitch shift up in this “ding diiing diiiiing” fashion.

2) Explosive Baskets!

3) End of level there’s that glorious **slow motion** effect and then the utter insanity of blaring “**ode to joy**” while **fireworks** start blasting.

Peggle is a master class in itself.

Player Feedback - Resources

"Juice it or lose it"

(Grapefrukt)

by Martin Jonasson & Petri Purho

Martin Jonasson and Petri Purho have an amazing video on YouTube where they take this very sterile Breakout clone and slowly add more and more feedback elements to it until it's this insane experience. Their enthusiasm is pretty infectious, I definitely recommend watching it.

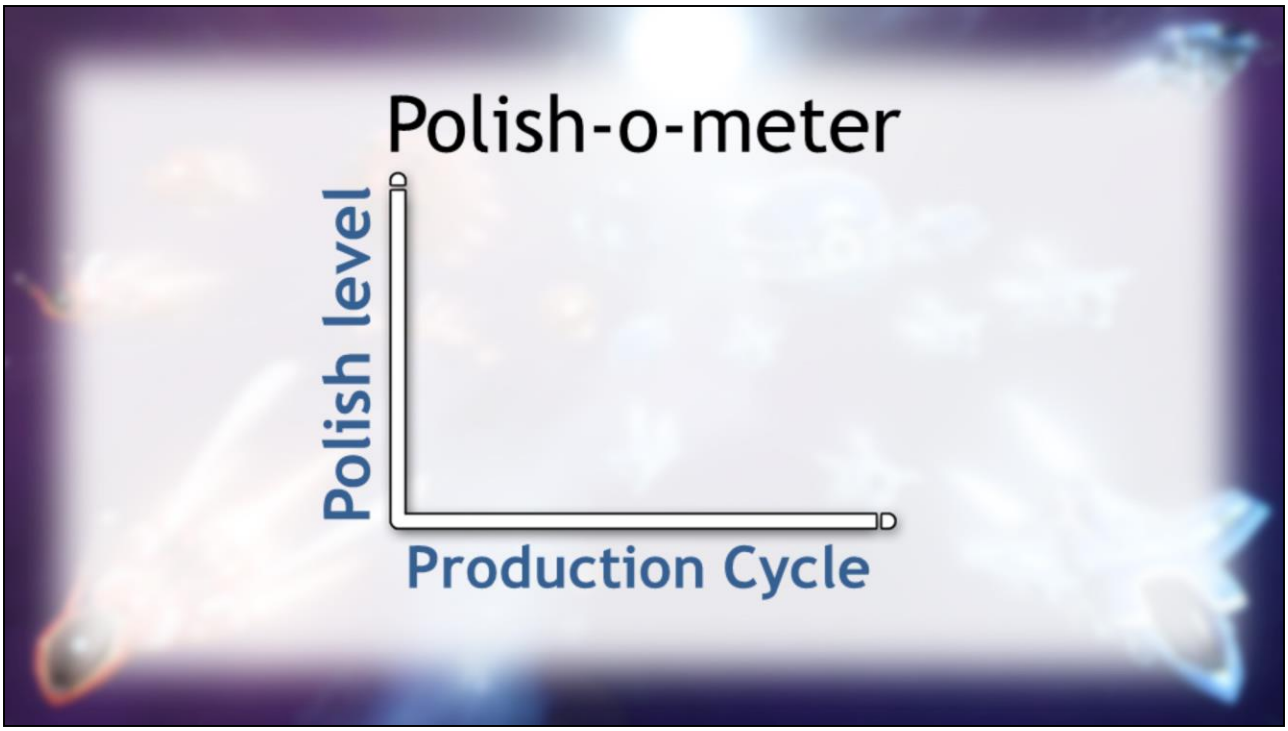
Player Feedback - Resources

"The art of Screenshake"

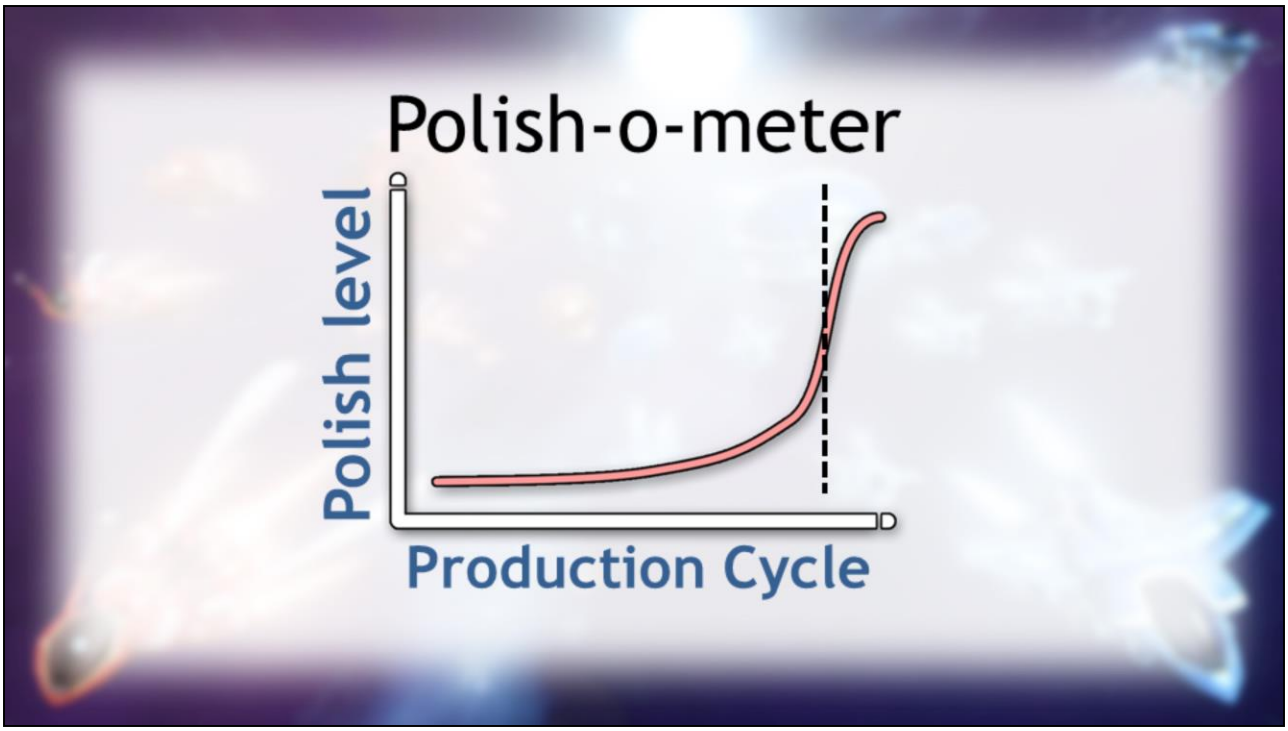
(*Vlambeer*)

by Jan Willem Nijman

One other excellent speech was Vlambeer's "The art of screenshake", where again he starts with a rather sterile platformer and using primarily different types of screenshakes makes the game feel like this fantastic experience. Again, definitely time well spent checking this out.



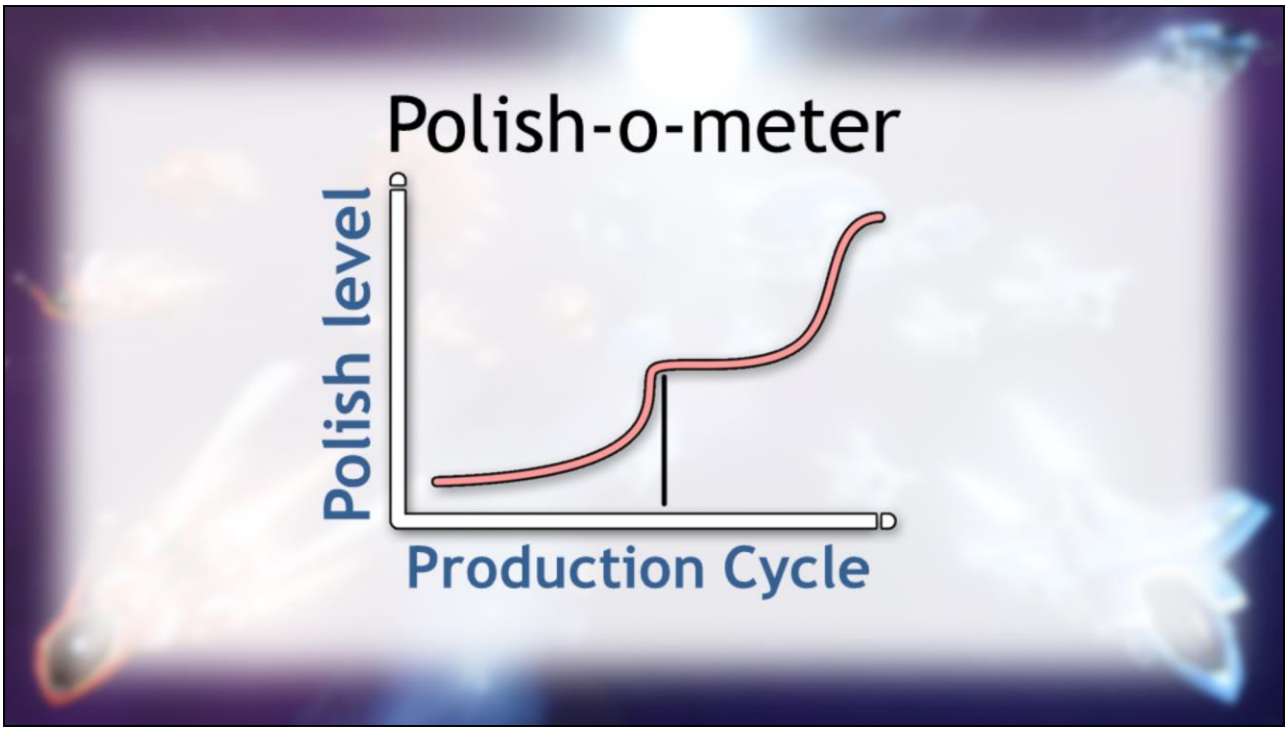
IS FEEDBACK PART OF THE "POLISH STAGE?"



To me though, I still thou

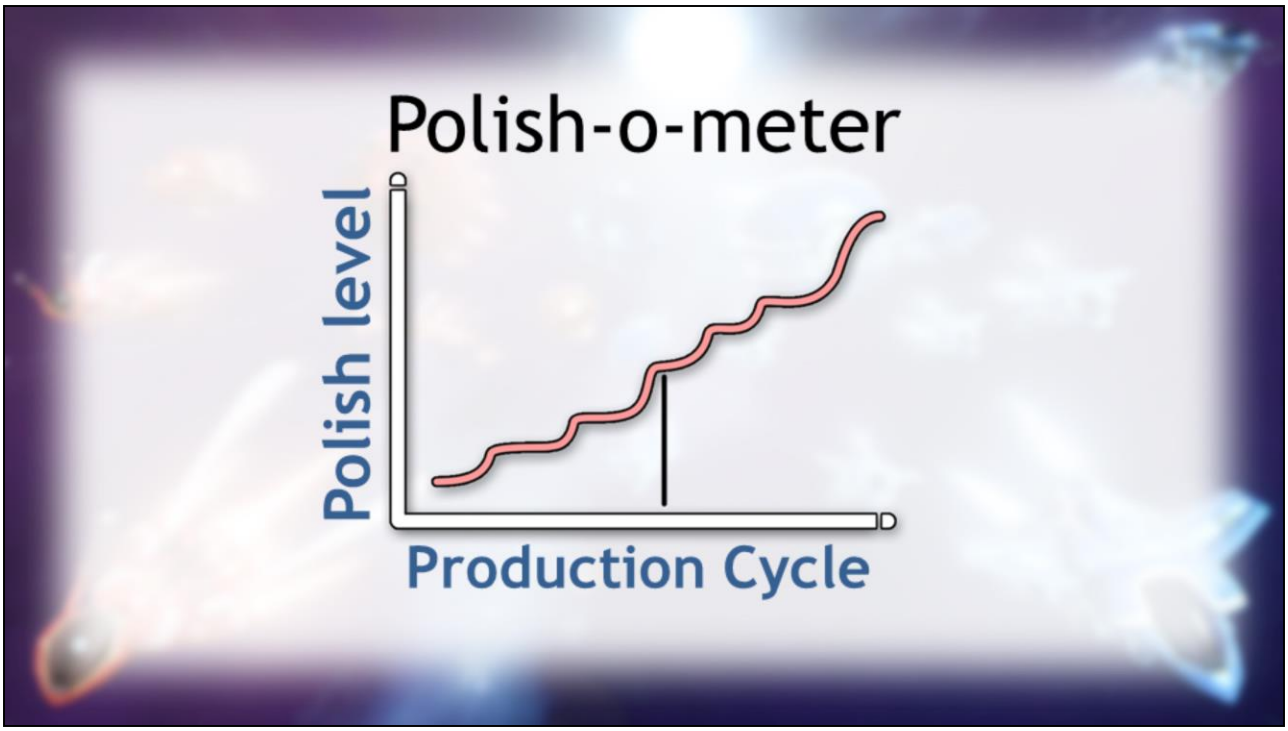
Problem is, we all know things get cut at the end of a game.

It’s hard to tell publishers or partners that the game needs to wait while you “add the fun” at then end.
ght polish simply needed to come online at SOME point before you shipped.



Or sometimes a key demo or press push would mean we bumped it up earlier.

The BIGGER issue is that it’s still not taking real advantage of what great player feedback has to offer.



During Gears of War 1, I made a TON of our prototypes for creatures, weapons, etc.

and the project looked more like this!

Why?

A constant pattern of pitching and debating…

But once we had a fun prototype everyone would get on board.

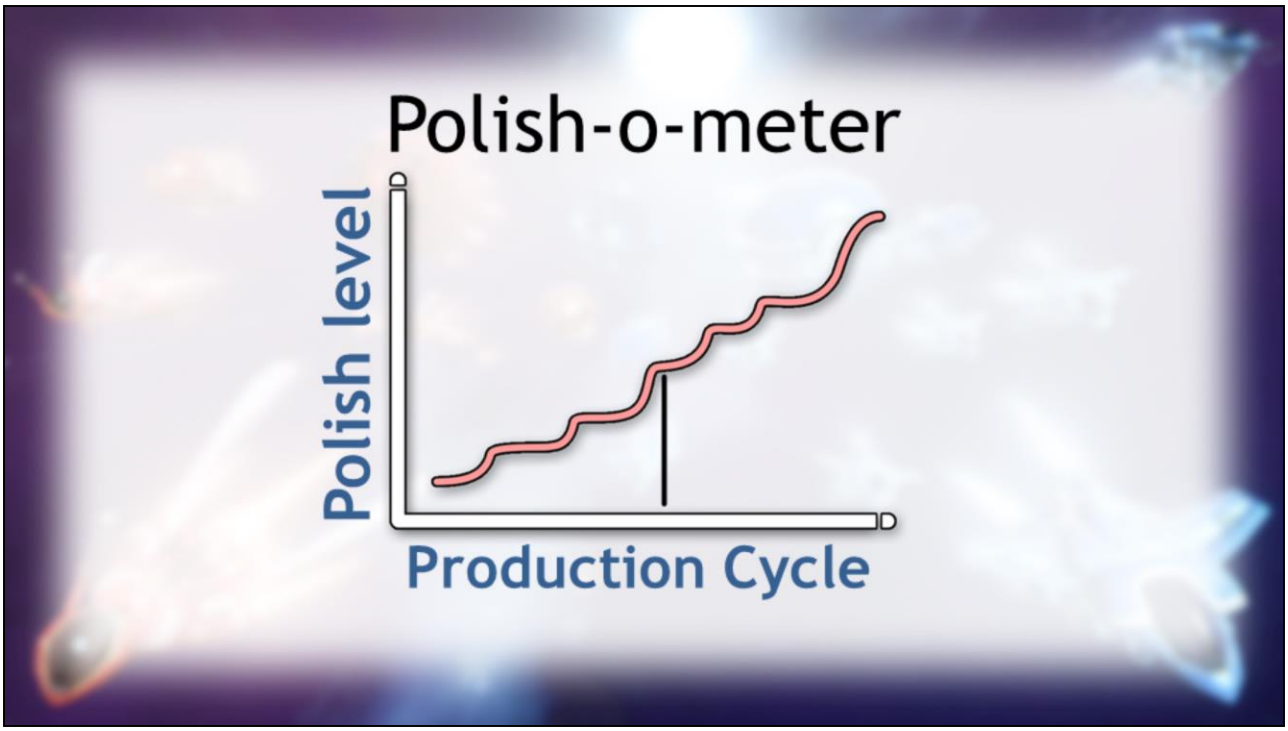
Saw it with the cover system early on.

Super debated, some HATED the idea

Then we added all kinds of awesome to it… SOLD!

NOW FLIP THROUGH OTHER EXAMPLES!

With the chainsaw on the lancer, having a shotgun at all, creatures, boss fights… you name it. We would have debates, until someone too a moment to make them "feel" great, and it suddenly became "real".

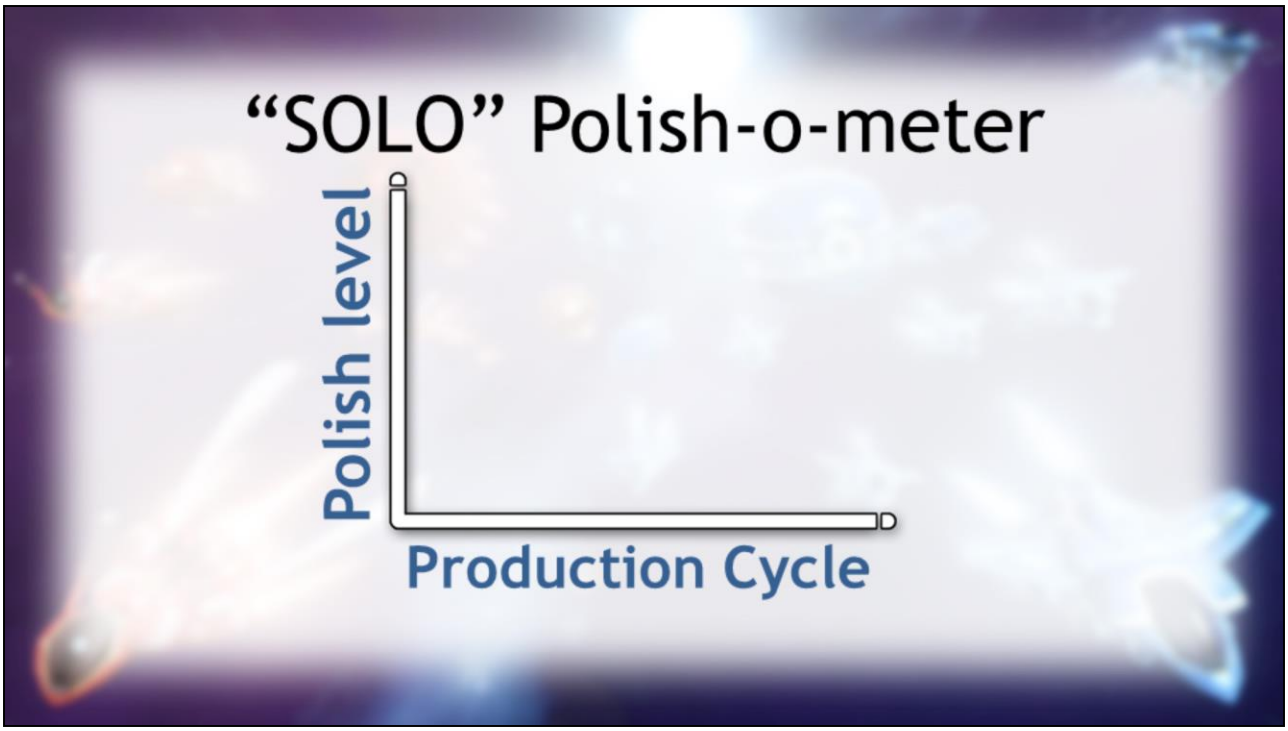


Internal salesmanship.

Adding polish was KEY, not to the player, but to the TEAM working on the game!

(publishers too!)

This is a dev tool!



Two years ago, started making games on my own.

AHHH! I didn't need to sell and pitch to anyone!

Soooo…

…what does it all mean now? When do I get all juicy?!

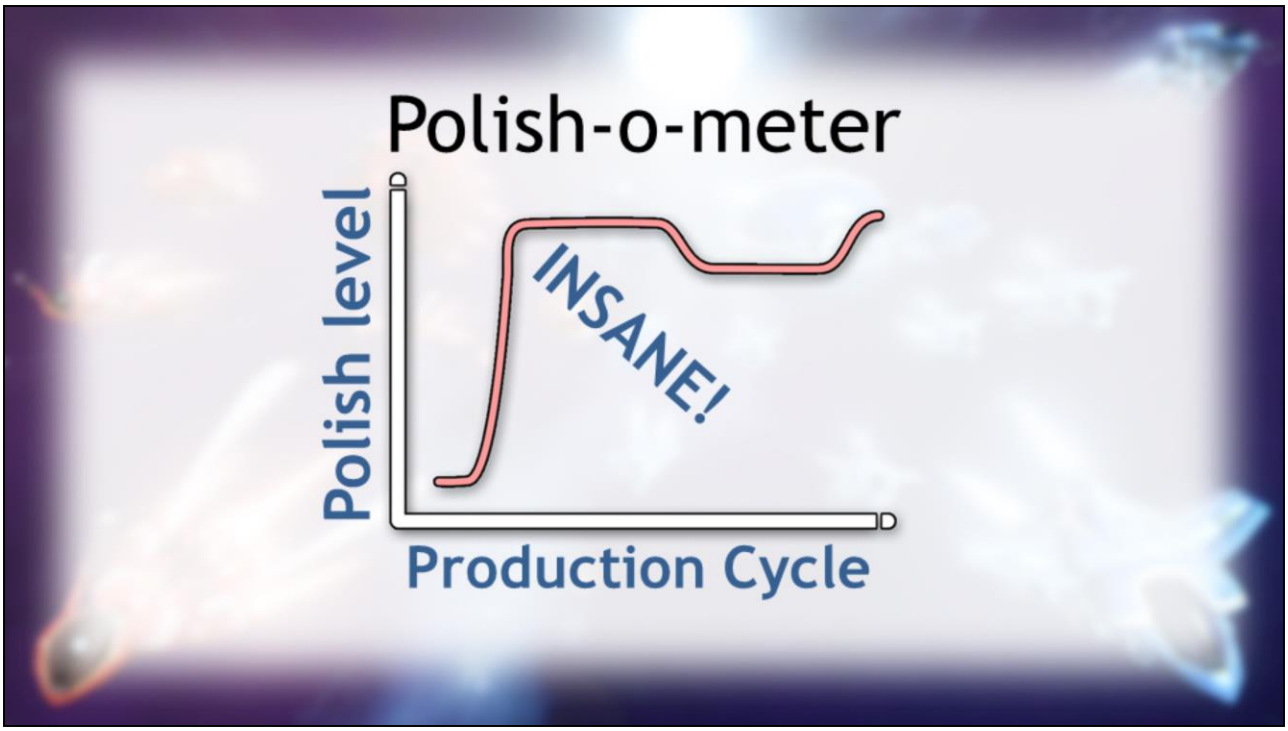
Just SOME time before I shipped? Revert to old snakey here?

One problem for tiny indie projects…

CRIPPLING SELF DOUBT!

Turns out:
Feedback isn't JUST for players!
Feedback isn't JUST for your team!
Feedback is for YOU yourself!

It's an amazing motivational tool!



IN AN IDEAL MAGICAL WORLD, WE WOULD HAVE THIS!

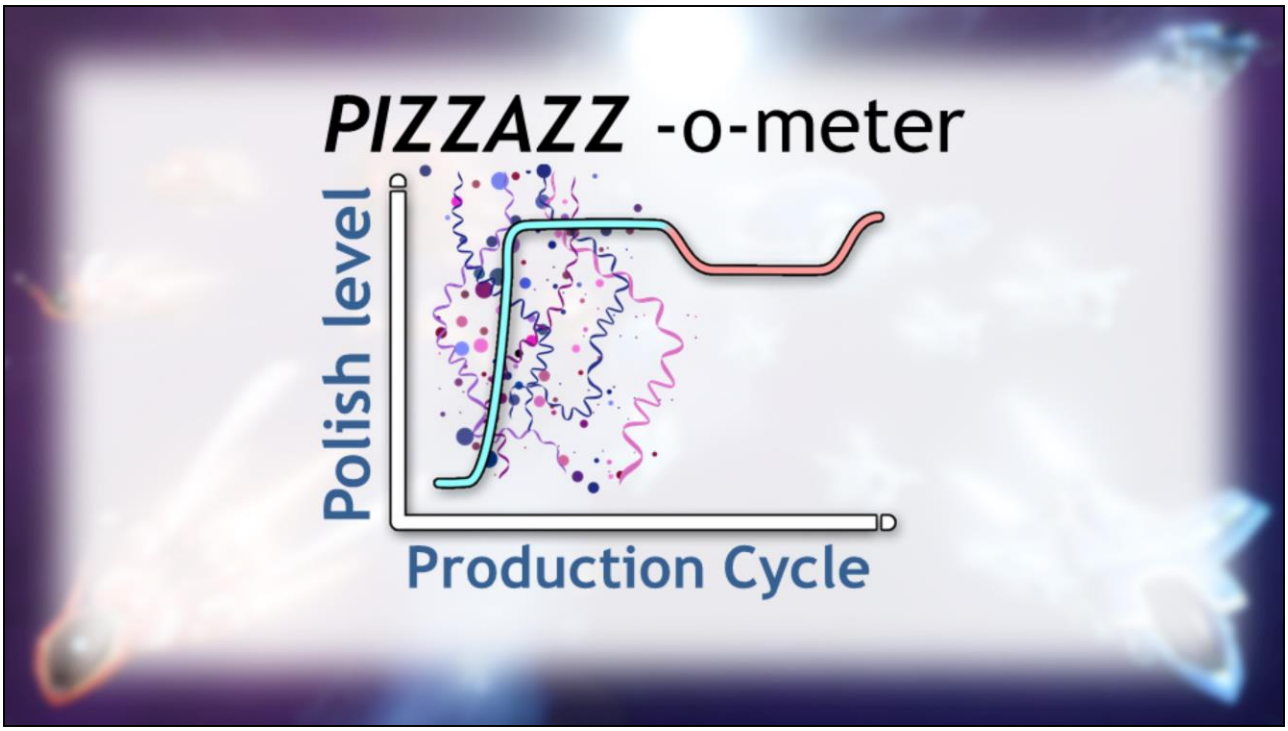
How awesome would it be to have all this polish UP FRONT?!

How cool to know your game's soul like that, so early on?!

INSANE!
THAT'S INSANE!!! You can't just start off polishing step 1!

You're right.

We have to **detach our concept of "feedback" from "polish"...**



It’s NOT not feedback!

It’s definitely not POLISH!

It’s ***PIZZAZZ*** instead! Weeeee!

WHAT?! You can’t just rename something and make it work?!

LEE!!! How is “Pizazz” different than “polish” or “feedback” or “**JUICE”**?!

Simple… It can be throw away, you’re not super attached to it.
It’s **punchy punchy placeholders**.

Early on, is when you **MOST need joy** to manifest in your game!
You MOST need to see the **promise**!

You MOST need that feeling like you're really "**on to something**"!

Later on, replace this stuff.

But, early on they're the **scaffolding of "fun**" for your game.

What is Pizazz!?

Pizzazz is our child level imaginations!

“WOOOOOOOOOOOOOOOOOOOOSH!
AKAKAKKAKAKAKAKAKKA!”
“BASH ABASHABASHHHH!” WEEE OOOOOOWEEEEOOOOO!”
“PBUBUBUBUBBUBUBUBUBU! RWARRRRRRRRRR!
KKUKUKUKUKUKUKU! PEOOOO! FOOOOOM!
BOOOSHH!HH!!!!”

The sooner your project can tap into at least SOME of that childlike joy from playing with a toy, the better you’re going to feel about your project as you progress.

CONCEPT ARTISTS CONSTANTLY DO THIS!

There are a handful of generally accepted player feedback techniques that work great also as Pizzazz because they're generally very simple to incorporate

Sounds - completely over the top is fine!

Particles – huge cheap libraries to drop in everywhere

Screen Shake – Everything!

Dynamic Lights - you might not be able to ship with some of these, but tying a crazy overbright light source to something can immediately make something feel intense and powerful.

Music – No sound guy? Go slap in anything you can get. Try soundrangers or stock music sites.

Post Processing – bloom, blur, whatever!

Physics – People love physics reactions, they're like free cool behaviors!

EVEN FOR SOLO GAMES…

BENEFITS FOR EARLY START ON THIS STUFF!

Lets you get a better establish your “30 seconds of fun” that you’re often trying to build the rest of the game around.

Unexpected aspects of your game can become really key features!

A game that has a lot of PIZZAZZ is one that you’re more likely to pass around and show people on short notice… you’ll get more feedback on it, and be more willing to take it out for a walk once in a while

The biggest benefit is one of confidence in your project. If you believe that when you hand the controls to another player, they’re going to truly enjoy it and see the potential, it colors EVERYTHING about that project. You get more excited about working on it, you get lost in your own game as you try out random builds, etc… it provides those moments that make

many of us so excited to breath life into games in the first place.

Working with VR lately, it has really became apparent how beneficial these techniques are.

Tutorials aren't needed so much, good feedback can do the trick. "**YES! Do more of that**!"

SO many unknowns… But many **old feedback techniques still work**

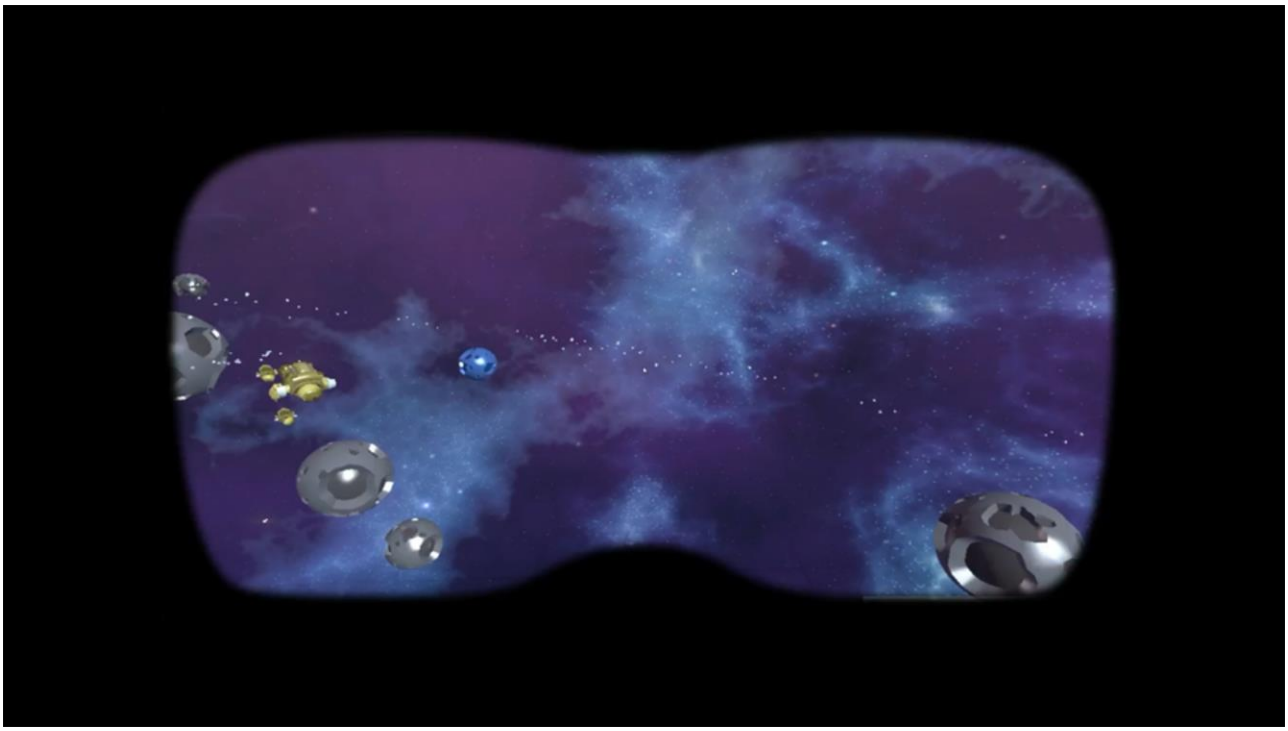
(not screen shakes ☺)

The **bar is relatively low right now**. Adding feedback can make your project "a thing"

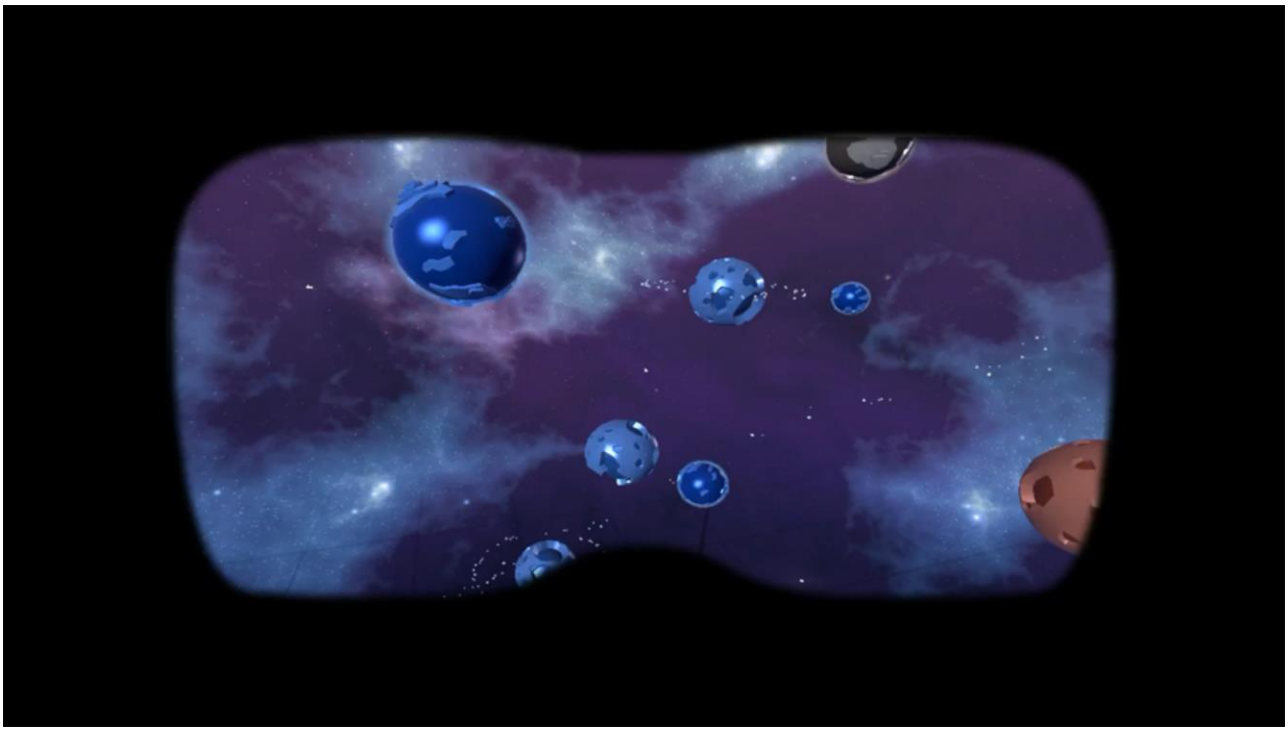
Prepare yourself!!!

BEHOLD! VIRTUAL… VIRTUAL REALITY!

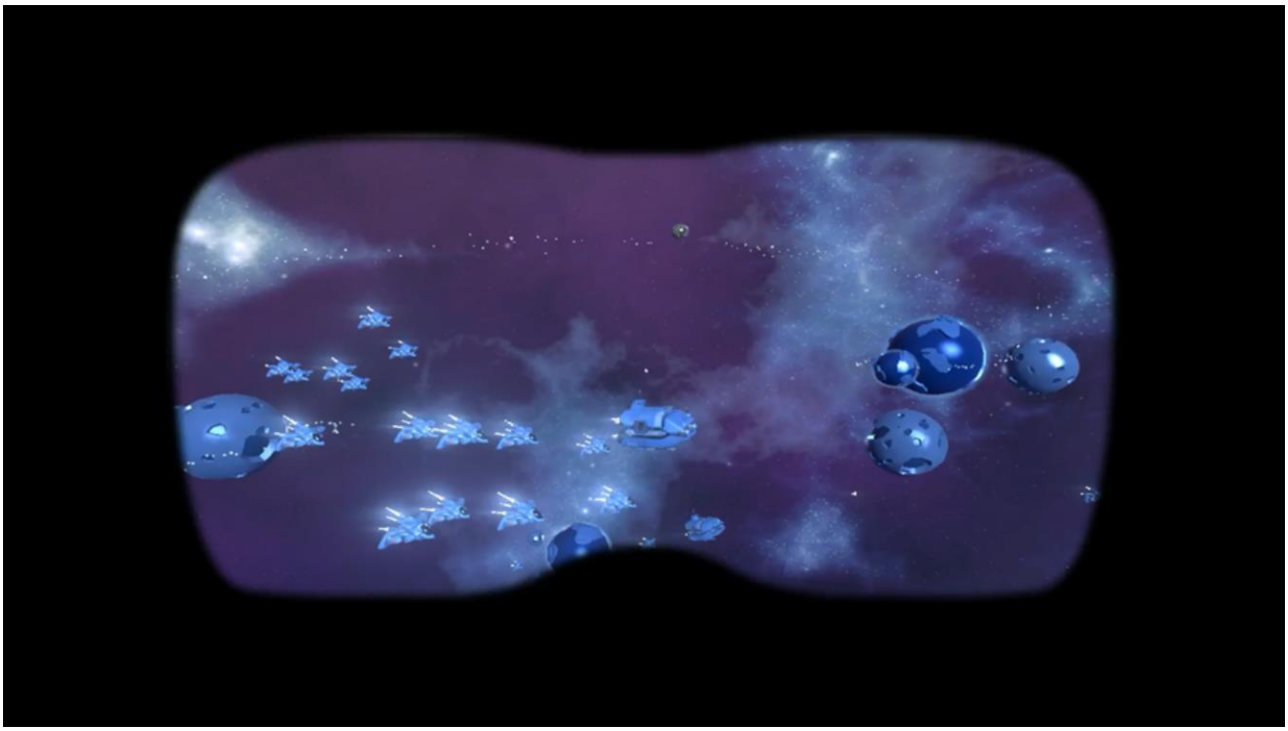
1



2

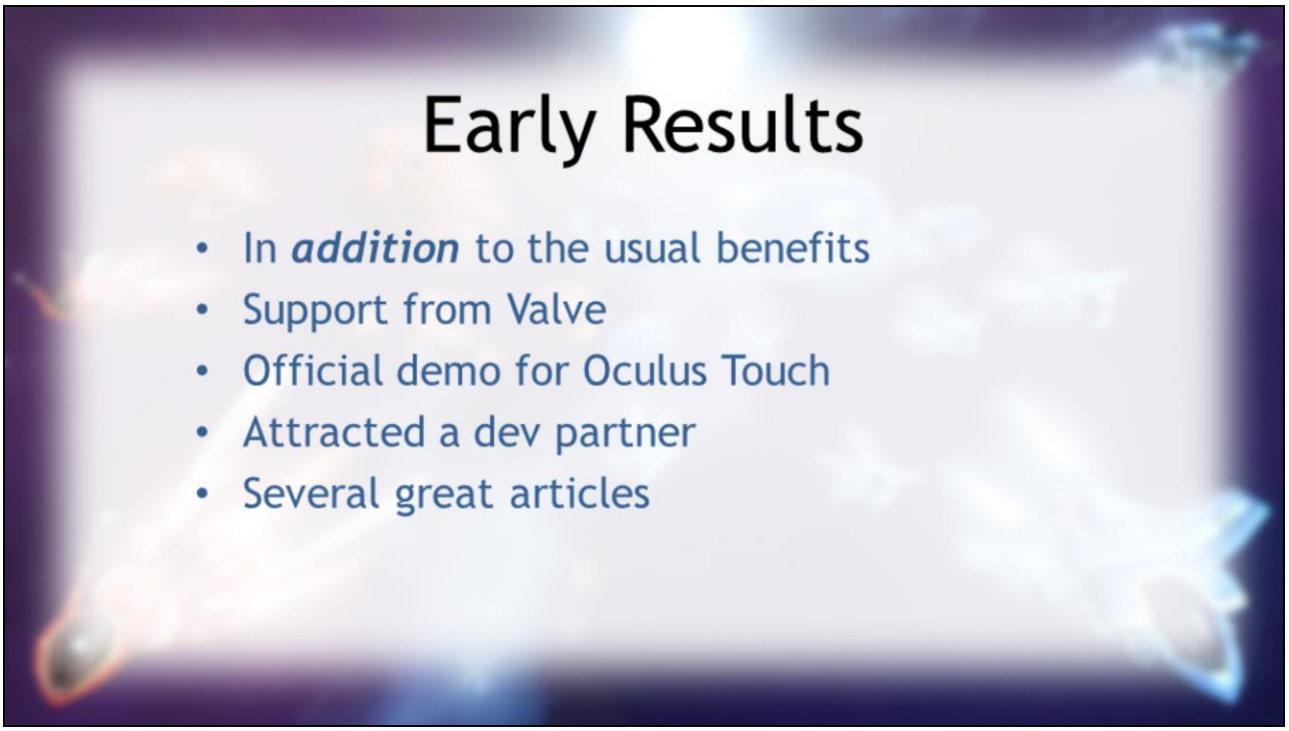


3



4


MOON
STRIKE



BY DOING JUST A FEW THINGS LIKE THE HOOP…

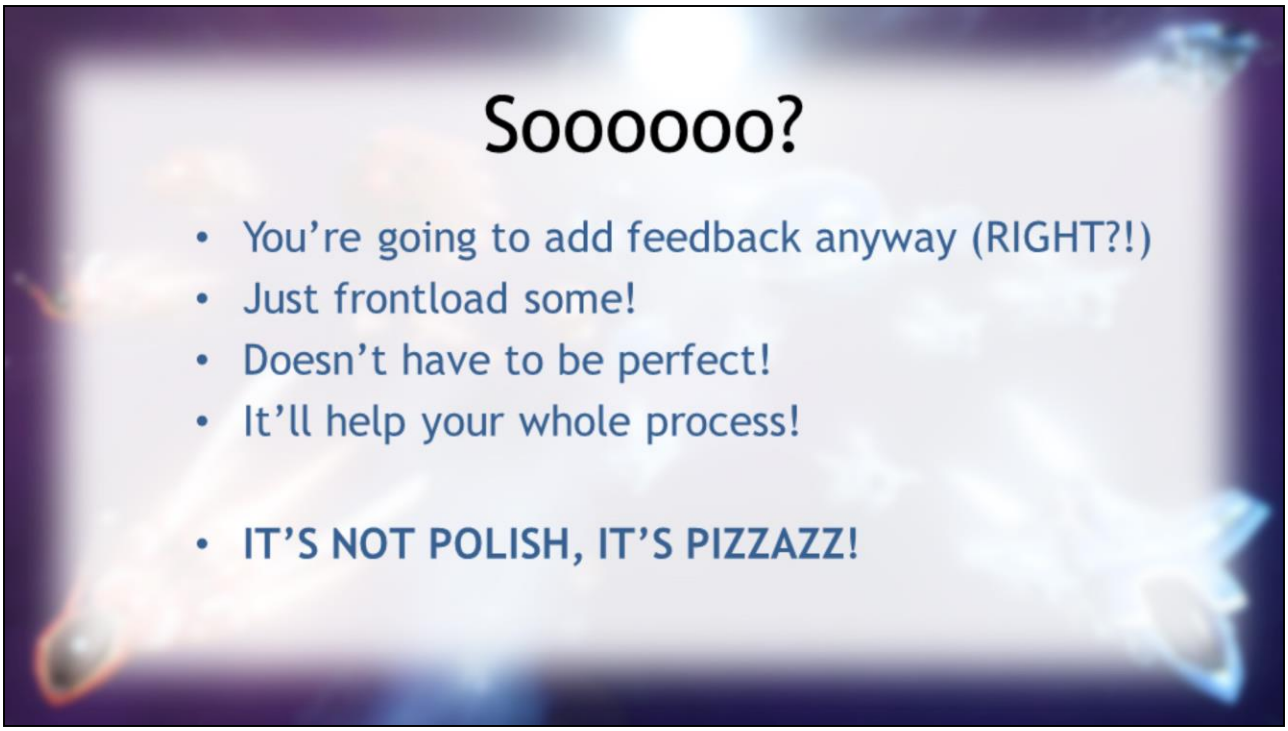
In addition to the usual benefits
- Feeling more legit
- Personal interest in playing the game
- MORE FUN DEVELOPMENT
- Etc…

Attention from Valve
- Early dev support

Official demo for Oculus Touch
- Oculus signed the project for a Touch Controller launch

Attracted a partner
- Programming /business partner sought me out

Several great articles
- Game informer, several VR publications

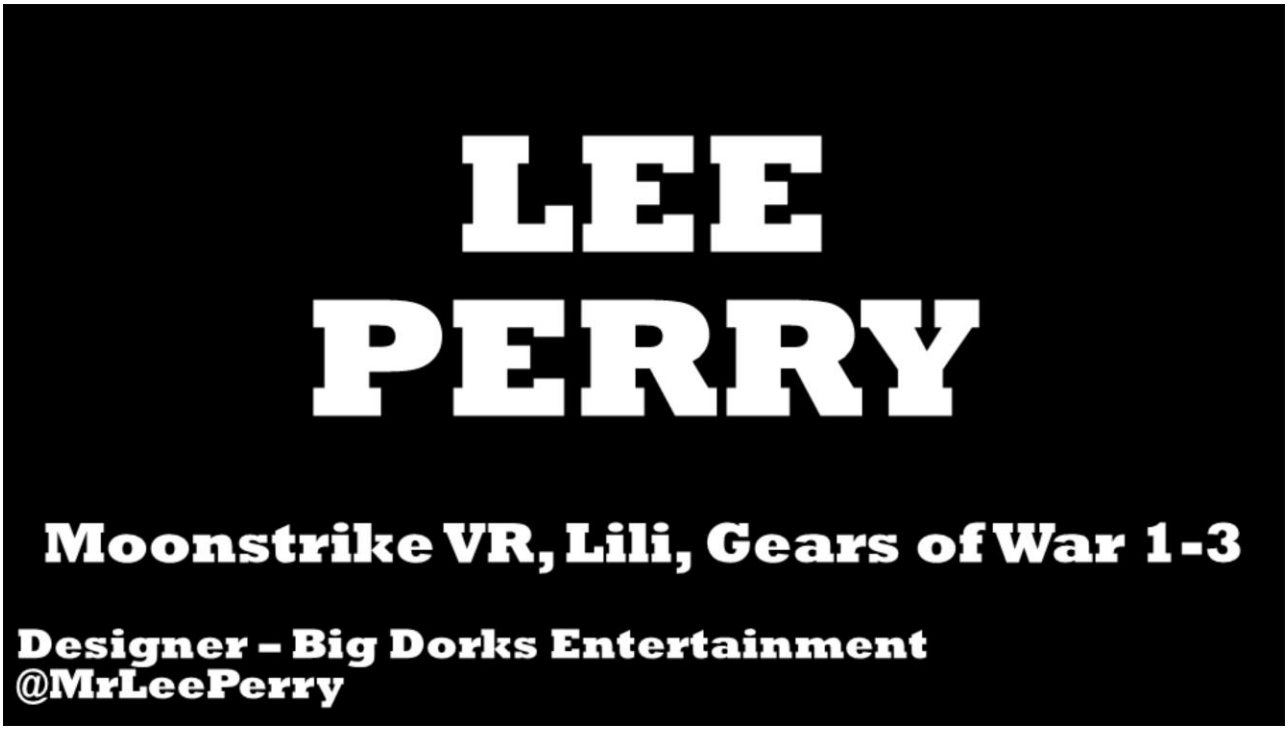


You're going to add feedback anyway
-UNLESS YOU'RE CRAZY!!!

Just frontload some!
- FOCUS ON YOUR CORE ACTIONS
- MAKE IT BIG
- ADD MORE THAN YOU THINK

It'll help your whole process!

Adios!



If anything, Lee's talk reminds us of how political the act of being a game designer on a big team can be.

You have to be constantly selling your ideas, not only to your bosses/publishers, but also to the team itself.

Next up, we have Emily Short!

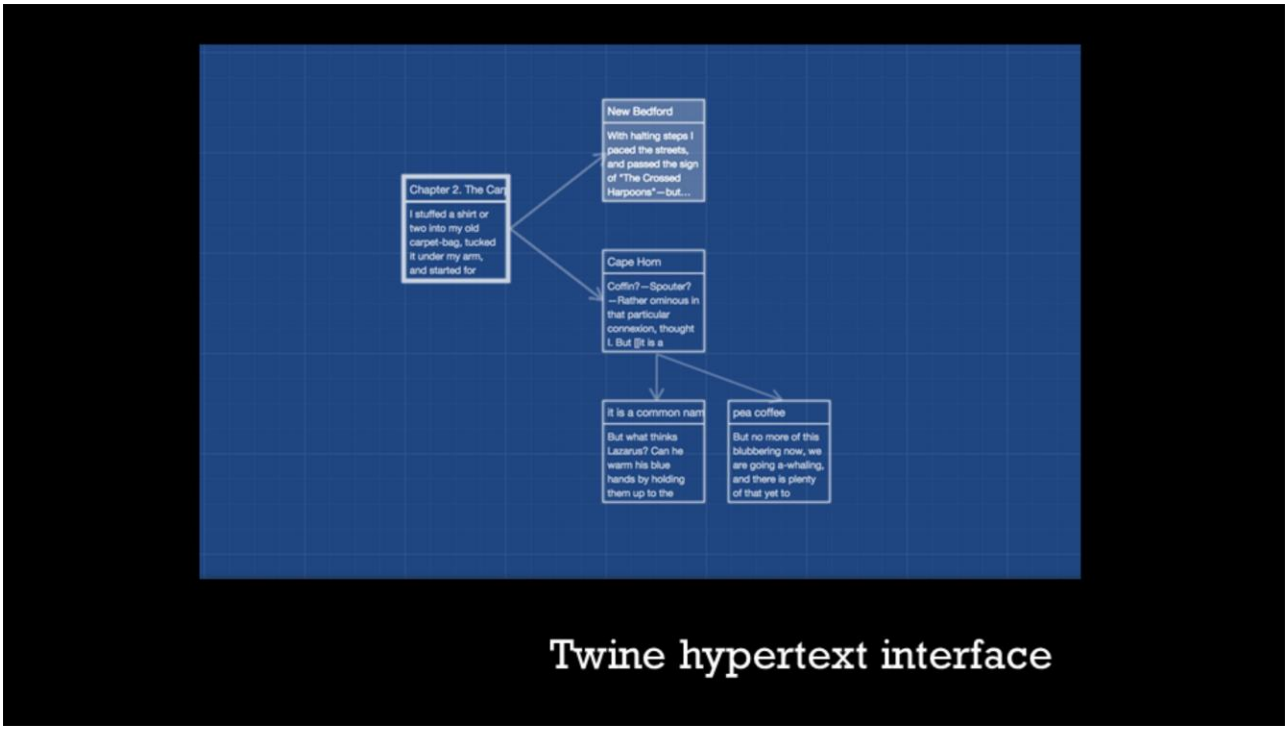
Richard Intro for Emily

She’s going to tell us how she tests her game before she even has testers.

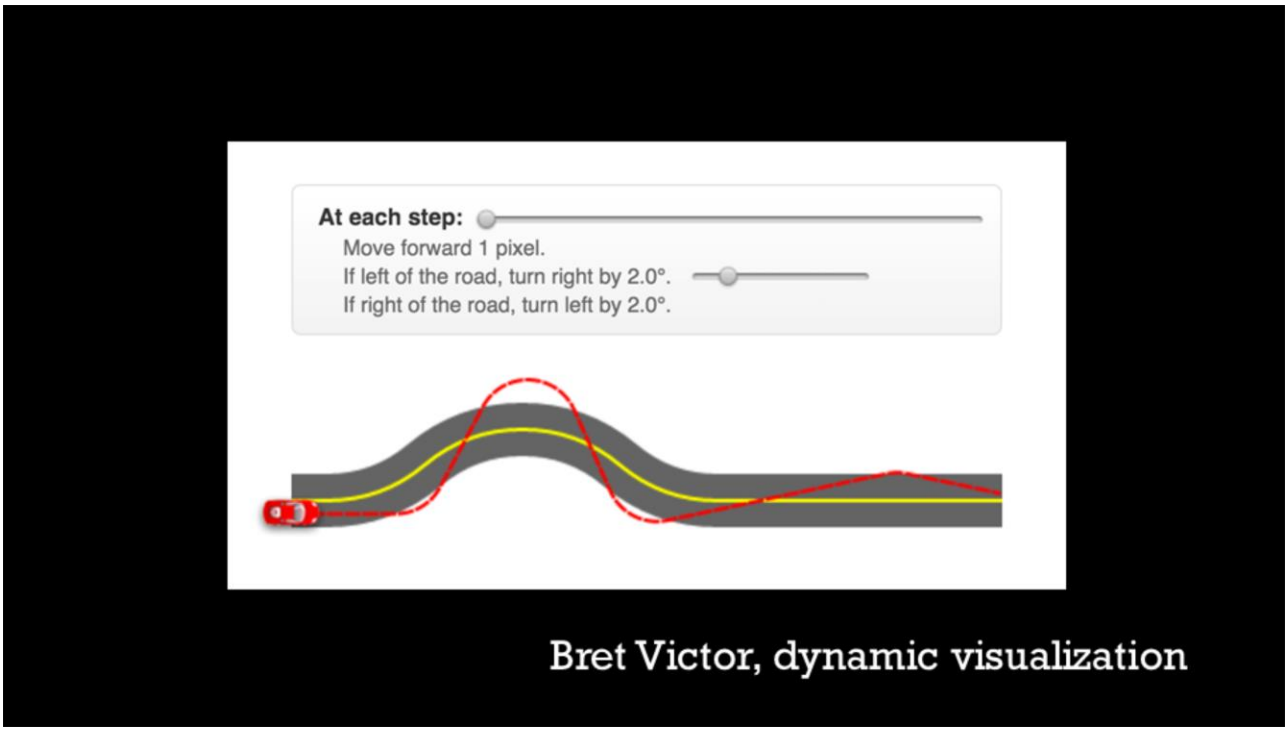
I'm a freelance consultant in interactive narrative, which means that I spend a lot of time with different toolsets for writing and managing content and dialogue. Sometimes I build my own tools, sometimes I work with other people's freeware tools, sometimes I work with proprietary tools that my client came up with. I've worked on a number of projects, including recently writing several of the island stories in Sunless Sea.

I'm here to argue for a pretty simple principle:

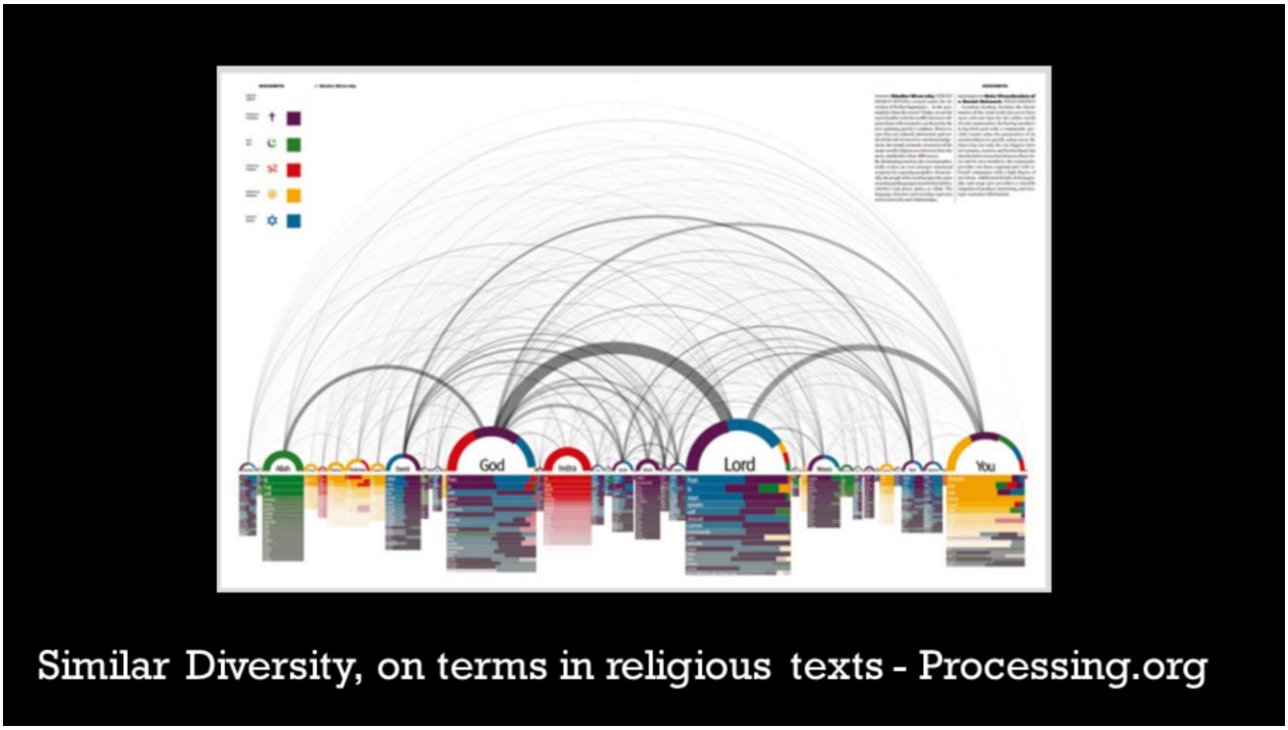
When you’re designing a new system, think about how you’re going to visualize its behavior.



If you're working on something with dialogue or narrative structure, that might mean visualizing how parts of the story or conversation feed into one another

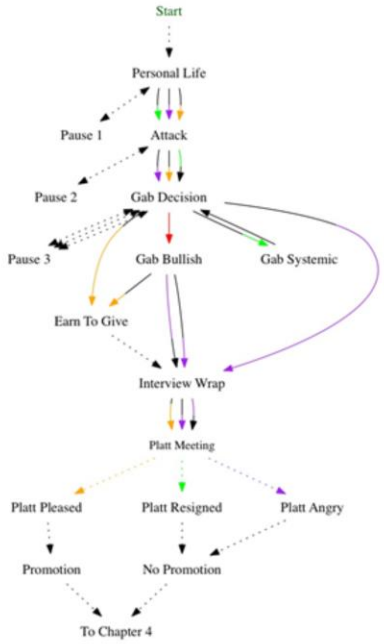


If you're working on dynamic system, that might mean heat maps. It might mean taking traces of many consecutive runs and laying them on top of one another. It might a dynamic visualization tool that projects movement in response to variables that you're changing.



If you're building a procedural system that uses lots of content data, it might mean coming up with ways to picture how much data you have, what categories it breaks into, and how you'll know when you have enough – here's an example that analyzed the words in English translations of major religious texts and represents how often particular terms appear in each text.

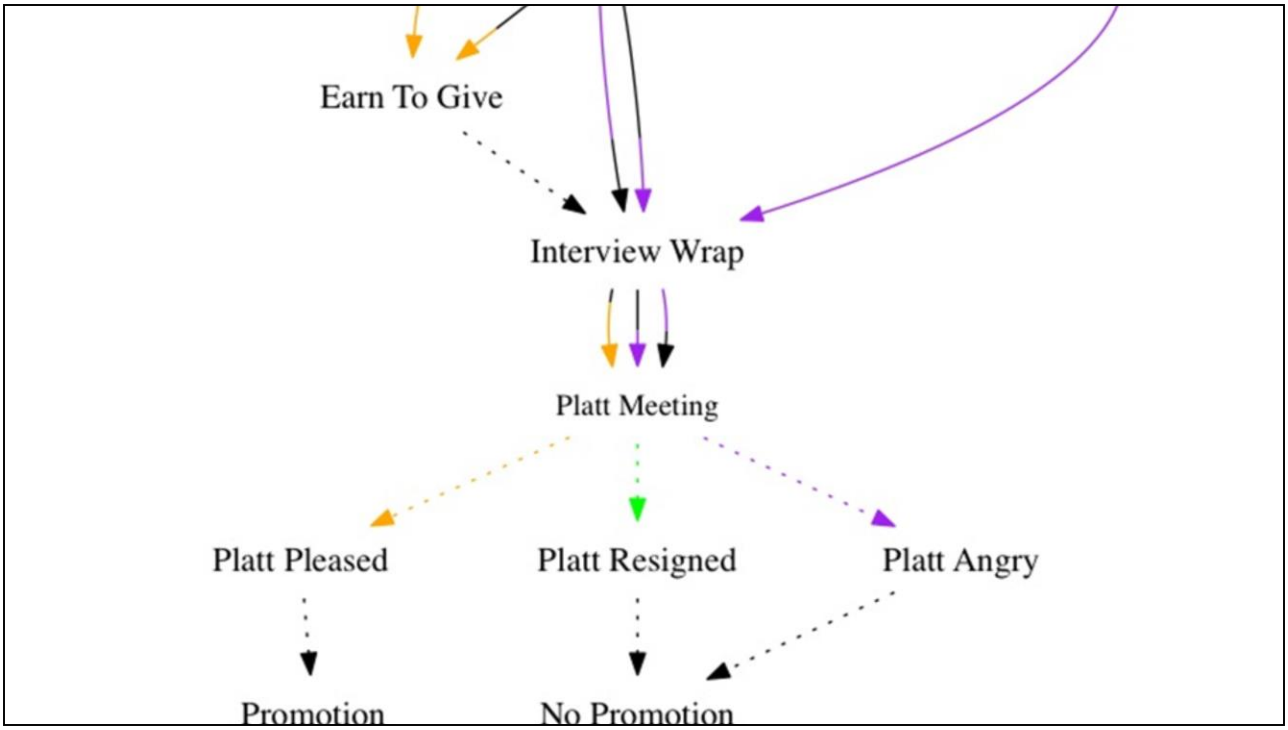
So I'd like to take us through how I've used early visualization in several of my own projects, and then expand to talk about how the same approach might apply to other kinds of work besides narrative-focused projects.



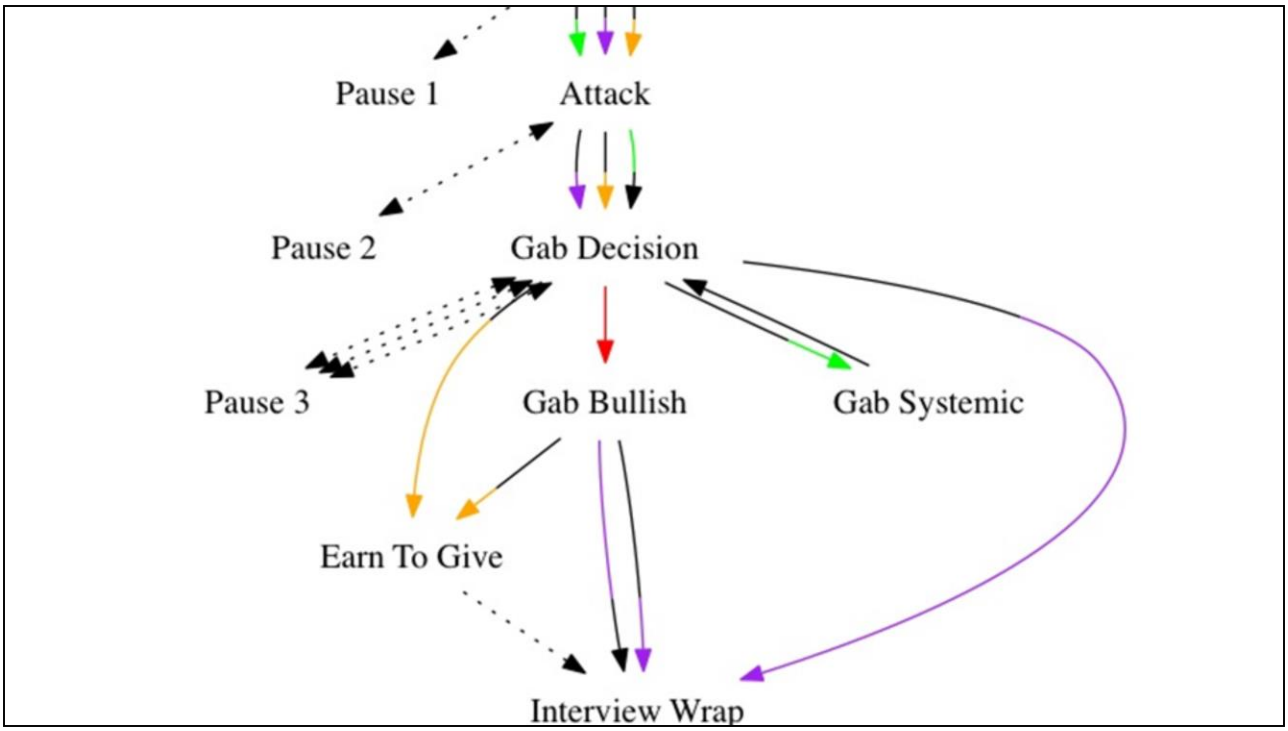
Platinum Package
(in progress)

CHOICE OF GAMES

Here's an in-progress visualization of a chapter in Platinum Package, a story I'm writing for Choice of Games' line of branching novels. It may look like it's a design document of some kind, but this is actually the result of running a Python script against the code for this story and automatically generating a dot format file that can be read into GraphViz. The visualization shows us the structure of part of one chapter.

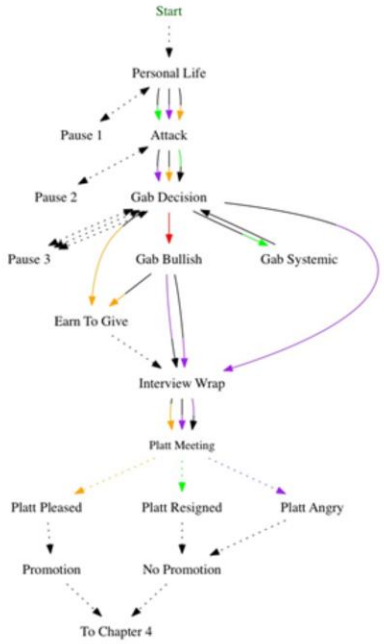


The visualization distinguishes player choices, which are represented by solid lines, from automatic transitions, which are dotted lines. I can immediately see that I have a mix of player choice and consequence, with the consequences clustered at the end of the chapter, which is what I want.



Meanwhile the colors of the lines show what kinds of stat changes are happening when the player makes these decisions – each color corresponds to a different stat that could be going up or down.

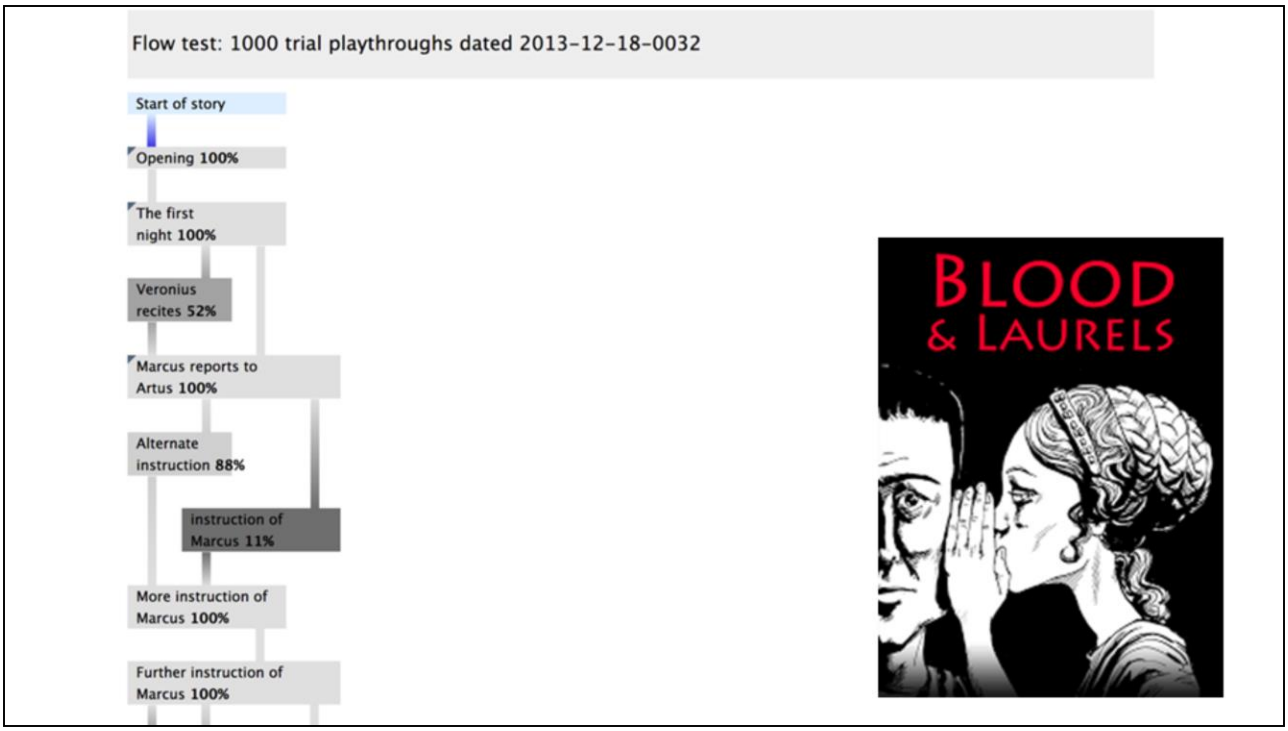
Transitions in red are ones that have no stat effects. When those are missing, that's a failure state that I need to be aware of.



Platinum Package
(in progress)

CHOICE OF GAMES

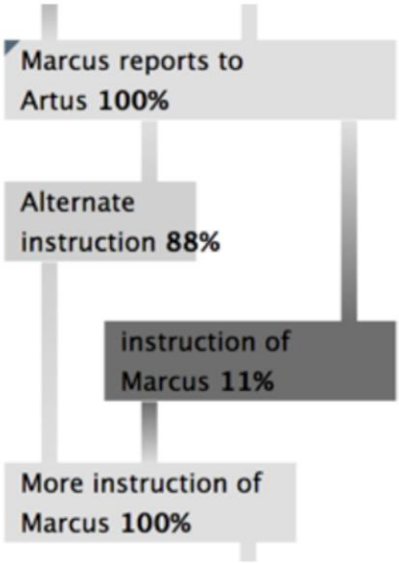
This chart keeps up with changes that I've made to the story, so unlike a hand-generated design document, it doesn't go out of date. That's important, because I'm responding to feedback from my editor as the story progresses, and over these iterations the game is diverging from that original document. I could do meticulous upkeep on that document, or I could just have a way of seeing directly into the code.

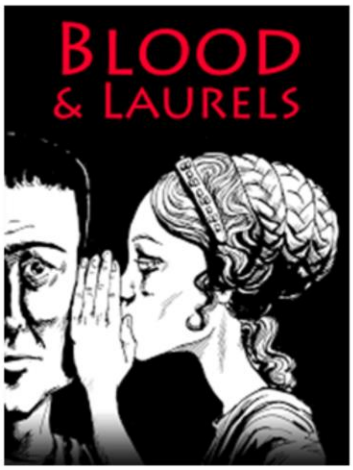


Now here's a case where the visualization I'm doing is further from a spec document and closer to a playtest report.

Versu is a project I worked on for several years that involved characters with an AI-driven approach to social interaction. That meant that there was an authored structure of a number of different scenes, but within each scene what happened was highly dynamic, depending on the moods and relationships of the characters. From a QA perspective, this meant we needed to go well beyond just having playtesters play the game a number of times. Instead, we would run thousands of trial playthroughs with an additional AI agent making the player's choices randomly.

This chart shows the way we then visualized the resulting information. This is just the beginning of the chart – it actually continues for a bunch more scenes. But even so it may be tough to read on this slide, so let's zoom in…



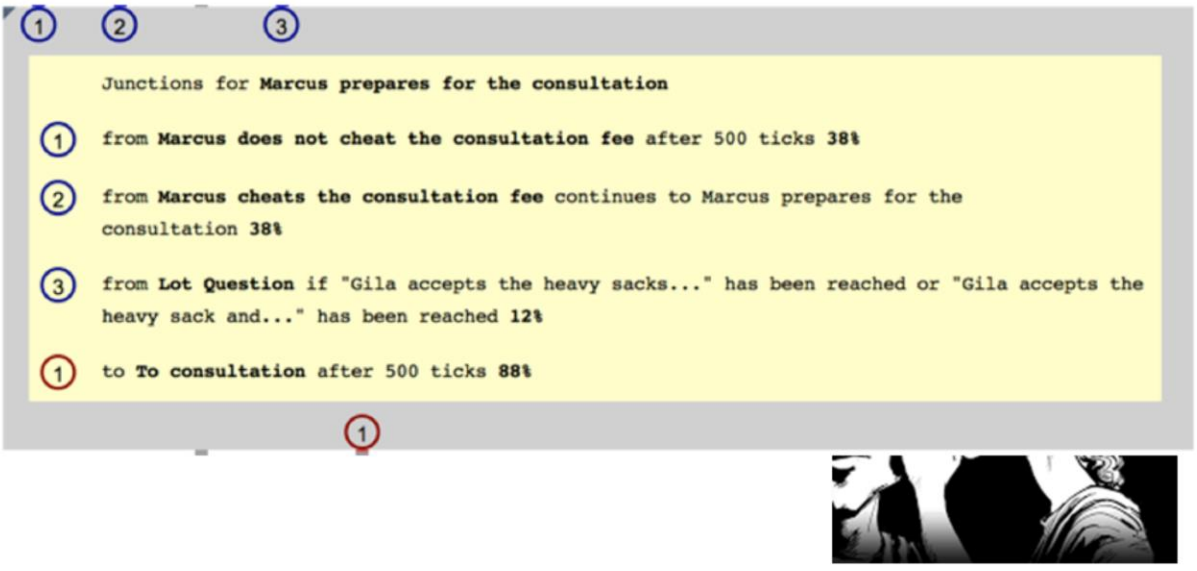


So here we're seeing just a few scenes of the game.

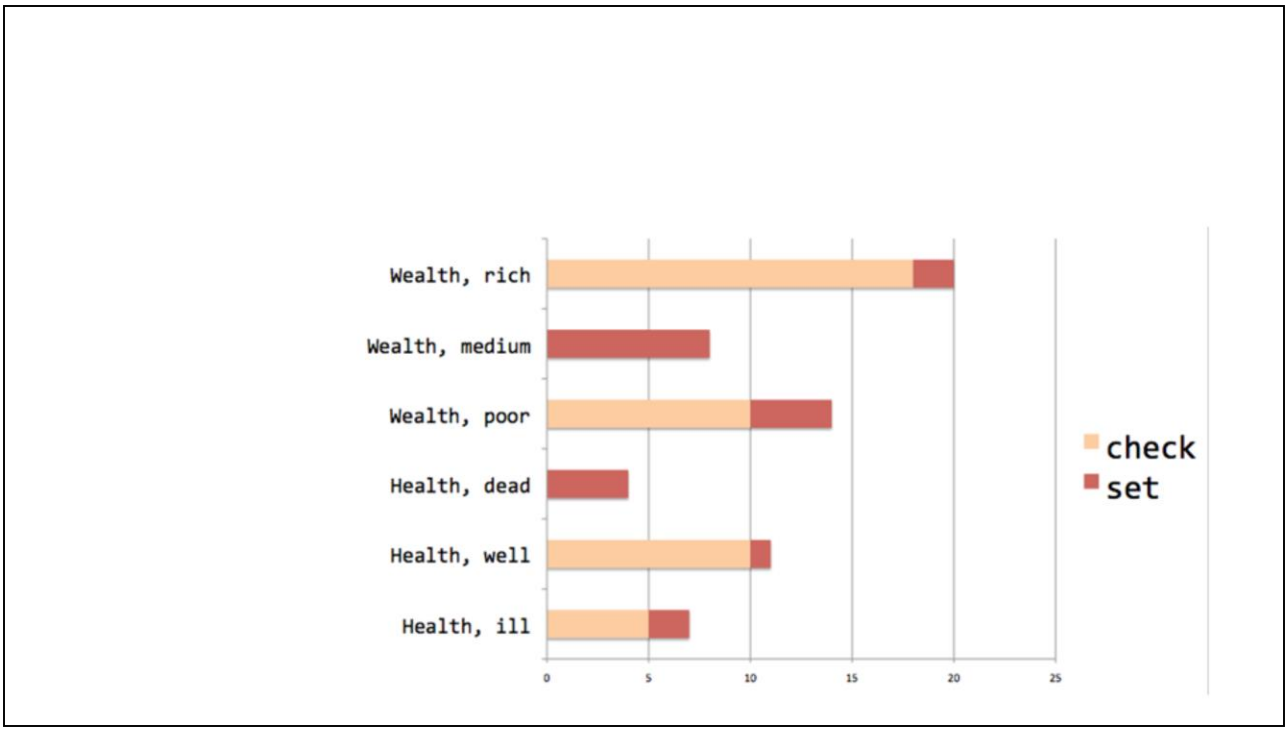
The first scene can end in one of two ways. Percentages indicate how many times the random player reaches each of the optional scenes. Scenes the AI reaches frequently are light grey; those it reaches more seldom are darker grey. Here we're seeing that the AI reaches the "alternate" scene 88% of the time and the "instruction" scene 11% of the time.

If there's a scene that the AI never reaches, that gets colored in red, to indicate that there is probably an implementation problem preventing access. Again, the visualization is designed to call out problem states.

In fact in this case because I had some help from the tool designer, we were able to build a visualization tool that was itself dynamic. So if I looked at one of those story nodes and wanted more information about what was happening during that node, I could click and open it up....



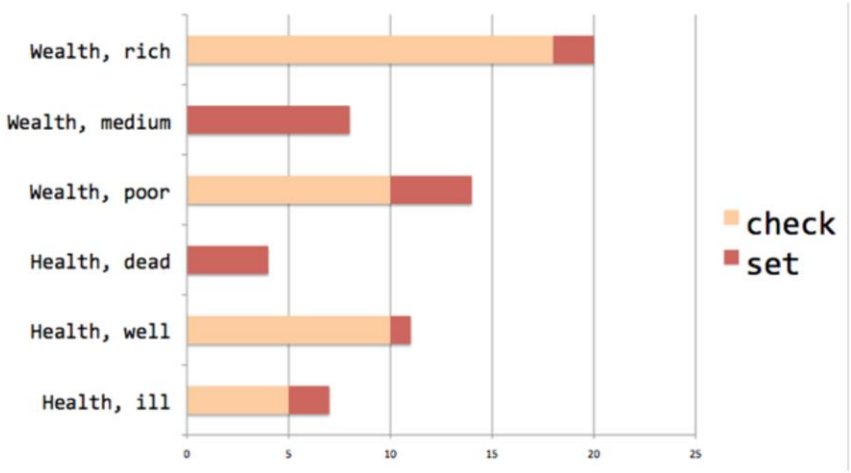
And now I can see all the possible ways to transition into that scene and what the percentages are -- what percentage of the time this scene is reached from each of the nodes that could lead into it, and where the simulation usually goes next.



Finally, I think it's important to make the point here that your visualization doesn't have to be attractive. I started the slide show with a bunch of pictures by people who are much better at visualization than I am, but it doesn't have to be like that to be functional.

(If this picture is unsexy…)

Yes, sometimes I'm just using Excel



If this picture is unsexy, that's on purpose – it still represents something very useful.

This is a simplified version of how I looked into an experimental procedural narrative system that used a lot of event data. We had written a bunch of events that could be selected to occur next depending on whether the protagonist had certain status features. And, of course, an event could also change the protagonist's status.

Running a simple script to count checking and setting instances for each quality and then charting the result in Excel helped get a visual sense of what was happening. For instance, in this representation, the wealth = rich attribute is being checked frequently but set infrequently, which is a warning that the system contains a lot of data that might rarely have a chance to fire.

Even conditional cell formatting counts as visualization

	Health, well	Health, ill	Religion, devo	Religion, heret	Religion, lapsed
Wealth, poor	5	3	5	5	1
Wealth, medium	5	4	0	1	1
Wealth, rich	2	10	10	12	0
Religion, devout	1	6	-1	-1	-1
Religion, heretical	0	5	-1	-1	-1
Religion, lapsed	1	1	-1	-1	-1

Another question I had about this data set was how specialized the events were. An event could be checking multiple prerequisite qualities at a time. To get a sense of whether I was getting good coverage for all the values of qualities, I threw some count data into a conditionally-formatted Excel sheet.

For instance, this would tell me quickly that I'd created a lot of events specific to being poor and sick, say, or rich and healthy, but no events tied to being well and medium-wealthy.

Obviously with the real data there are many more entries to the sheet, but dead zones are instantly visible.

Now it might seem like this is a really really open-ended piece of advice that would work out wildly differently for different kinds of projects. And it might also seem like I'm asking you to gain some new expertise that isn't necessarily in your wheelhouse as a designer.

I find that even asking myself the question, "what would a picture of this look like?" gives me a new angle on thinking about the quality of that design. A good visualization puts emphasis – like color or size – on things that are wrong or important. So what kind of information about your system is important? And what can go wrong?

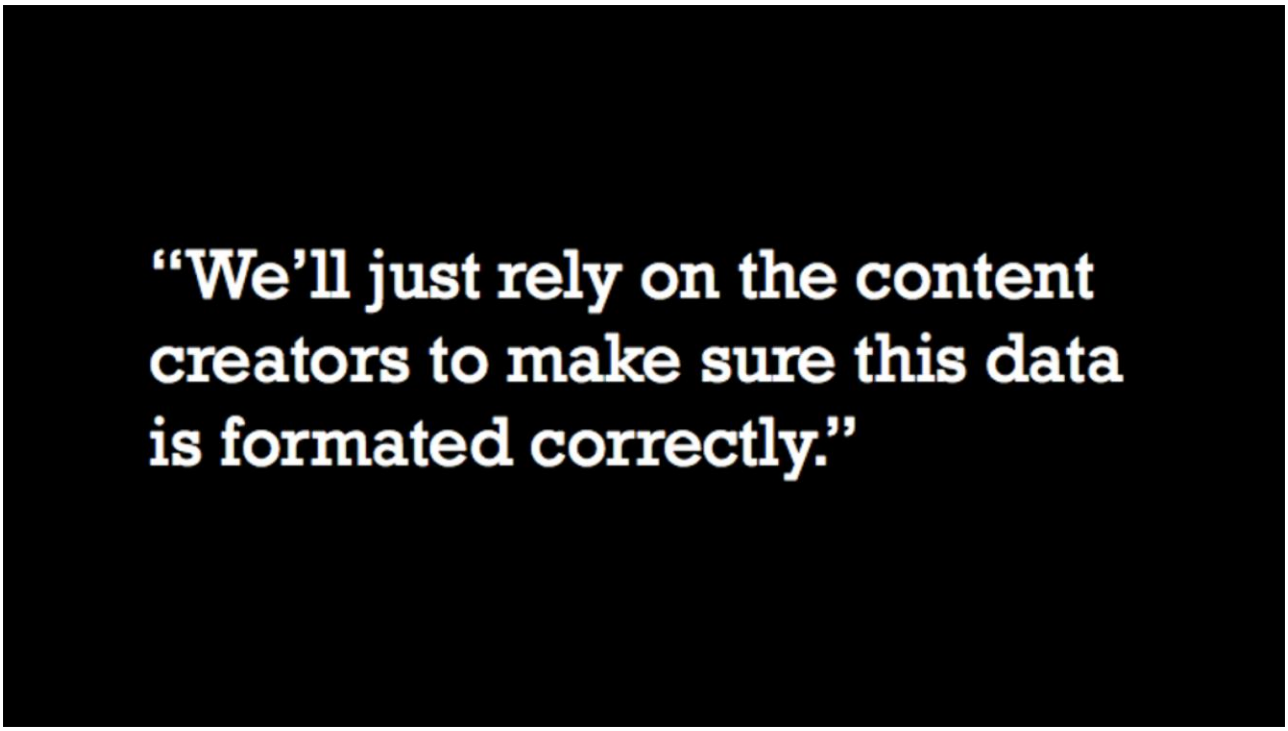
If you have a system with a large amount of data, thinking about visualizing that data means you're going to have to think about things like data types, what might characterize bad or good data, how much data you need, and whether you have adequate coverage of different aspects of your game world. And if you're not a programmer but you have one handy, you might want to talk to them at this stage about how they'd approach this.

Two: the sooner you have these ideas, the sooner you can build them into your tools. Or specify them so that your tool programmers can start building them into your tools.

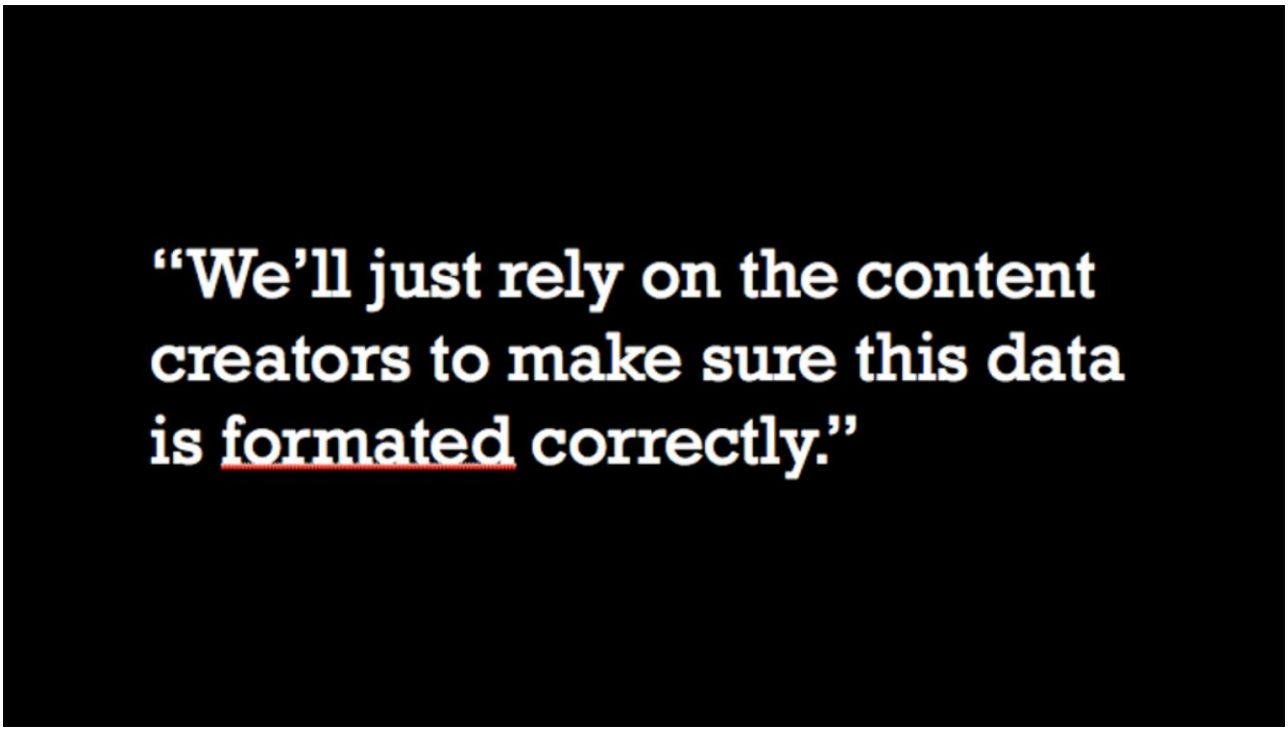
No, you're not going to anticipate everything that could possibly happen as the system evolves. But one of the useful things about a visualization is that it helps you instantly recognize issues in whatever you've just built. Anything you can do that highlights content problems *while they're being generated* will save you huge amounts of time later catching and debugging them.

Conversely, anything that makes content creators feel *confident* about what they've built while they're building it leads to faster, better content.

Here is how to make Future You really really hate Current You:

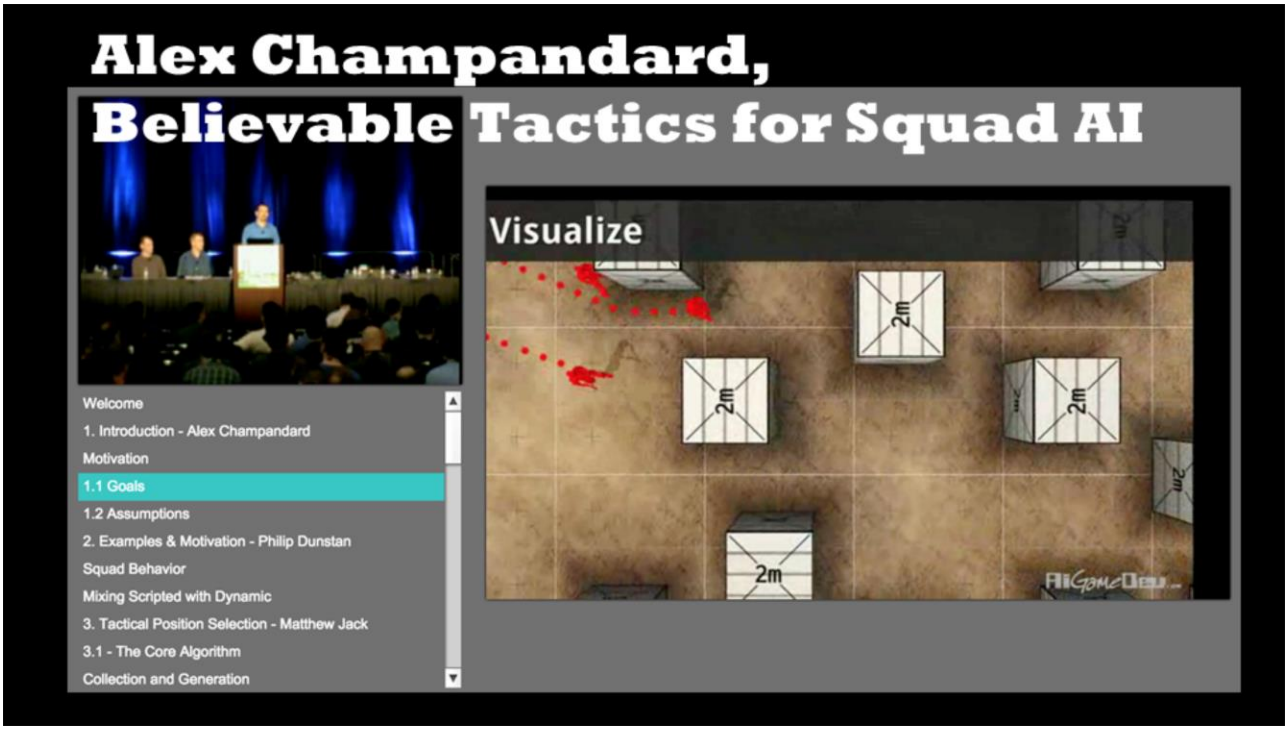


Your content team may be geniuses and they may be really detail oriented people, (but you want as many ways as you can to find their mistakes anyway.)

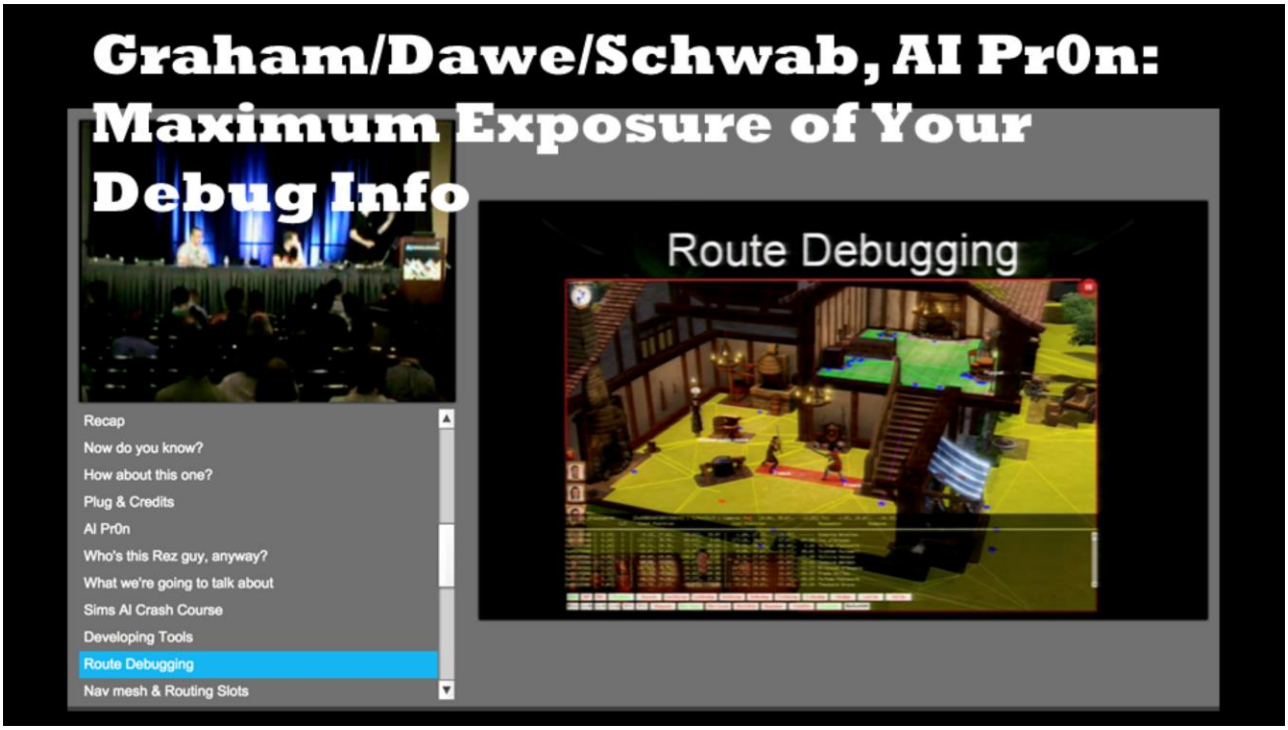


(but you want as many ways as you can to find their mistakes anyway.) We all know why spellcheckers are useful. Putting in more visualization options gives you more corrective tools.

I’m not by any means the first person to stand up here and talk about how to visualize particular aspects of what you might be building as a designer. Here’s Noah Falstein talking at the narrative summit a couple of years ago about puzzle dependency graphs to manage a puzzle based game >>

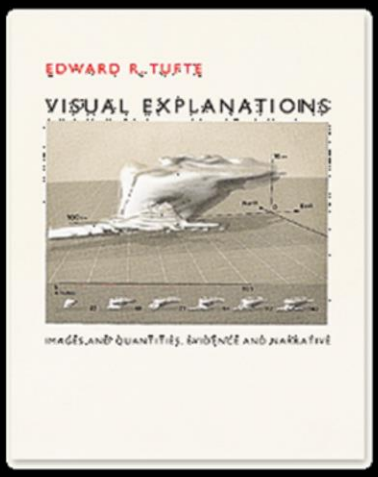


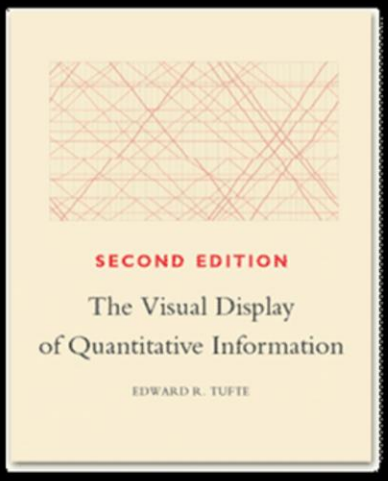
Here's a great talk from Alex Champandard at the 2012 AI summit which includes some discussion of how to visualize the behavior of squad AI



…and here's a whole panel from the AI Summit in 2011 – Rez Graham, Michael Dawe, and Brian Schwab talking about visualizing expected AI behavior.

Edward Tufte





Outside of GDC talks, you might also want to get some visualization ideas from Edward Tufte's books.

Or from Bret Victor's website…

Whatever you're trying to do, thinking about the visualization from the beginning does some useful things for your design process.

Visualization forces you to think about categories. What are the components of your system and how do they relate?

Visualization forces you to consider failure conditions so that you can draw visual attention to them.

Visualization also forces you to think about what success will look like. What is a finished system? How do you know when you have enough content and in the right places?

(And if you visualize early, you can either build)

And if you visualize early, you can either build – or better yet, make your tools programmer build – a better tool that will find your bugs faster and streamline your workflow.

Thank you Emily!

Of course, as you may know, we as designers love to say “But it’s not ready for testing!”

I think Emily has provided us with one more reason that we can’t use that excuse any more.

Next up we have George fan, who has been developing games for many years, one of his earlier titles being Insaniquarium, nominated for game design in the early days of the IGF

But you probably know him for a little game he made called Plants vs. Zombies.

George will be telling us how your enemies aren't as different as you think they are.

Fun fact about George, when you want to find a picture of him on the internet, you end up getting a bunch of pictures like this…

That’s OK George, you’re the only George *I’m* a fan of.

George Fan!

All right! Time to talk about *my* rule, which is: Make Your Enemies *Actually* Different.

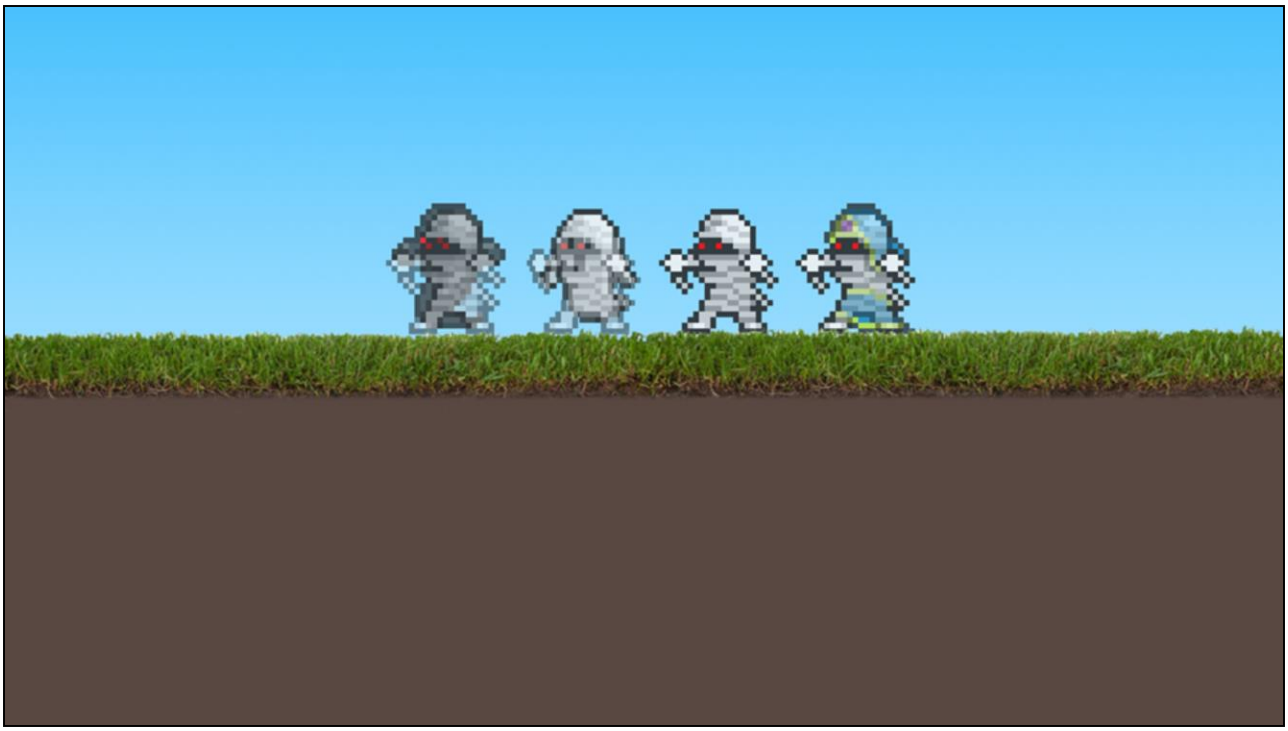
First, let's zoom in to this word "Actually". What do I mean by that?

In this case I mean making it so each new enemy is tackled in a different way.

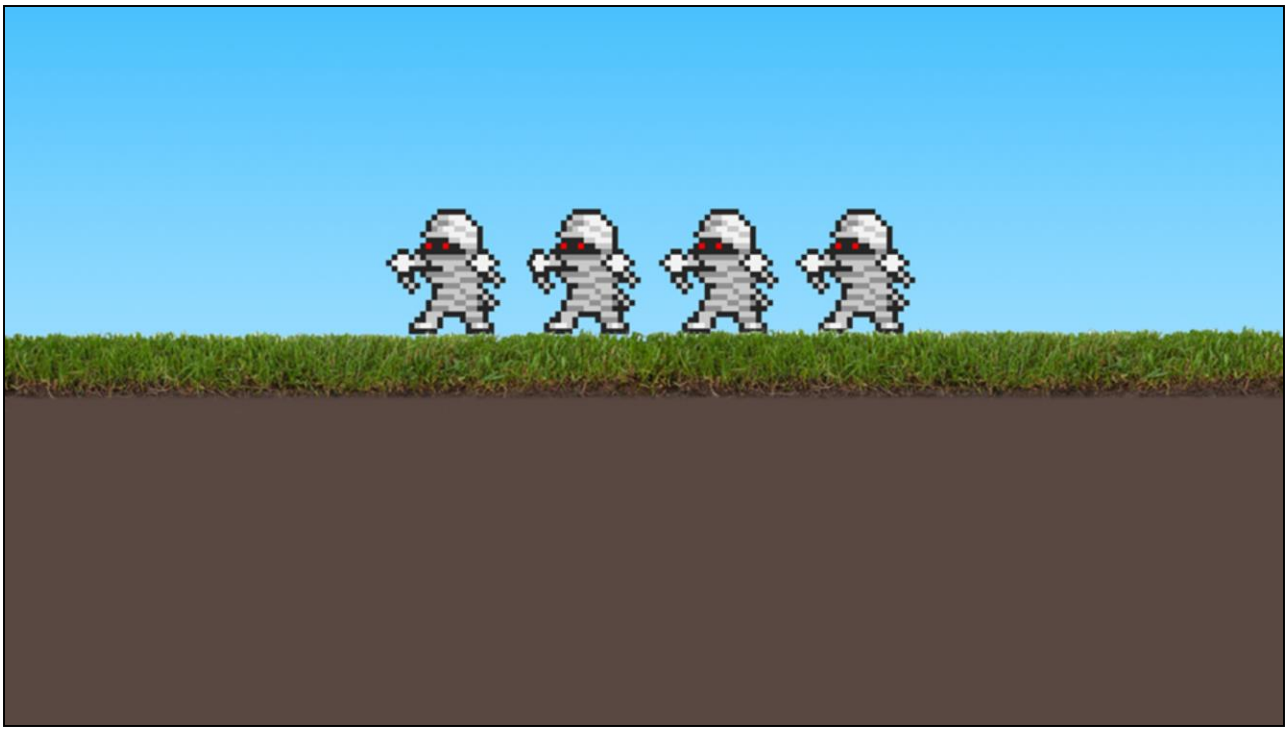
Think of some of the games you've played where the enemy design was exceptional. I bet in each case, there was a high density of.. enemies that you had to use different methods to defeat.

The reason I have Super Mario in this slide here is cause it did such a great job of meeting this criteria. Take note of the feeling you have right now looking at this set of bad guys, remembering all the different ways you defeated them. If you're ever unsure about enemy design, I recommend analyzing how the first Super Mario did things as a clean example of enemy design done right.

Sometimes we set out to make a game with a lot of enemy variety. On the surface, they look different enough and seem to do different things.



But to the player, fighting enemies gets monotonous, and each new encounter doesn’t bring enough uniqueness to the table.



We’re left with a game that feels like you just fought a bunch of the same thing. Today I wanna teach you how to avoid that.

To help us along, I'm gonna introduce a tool called Player Brain-O-Vision. It's a way for us remember to get into the player's brain, as this *is* the most important perspective.

Our player

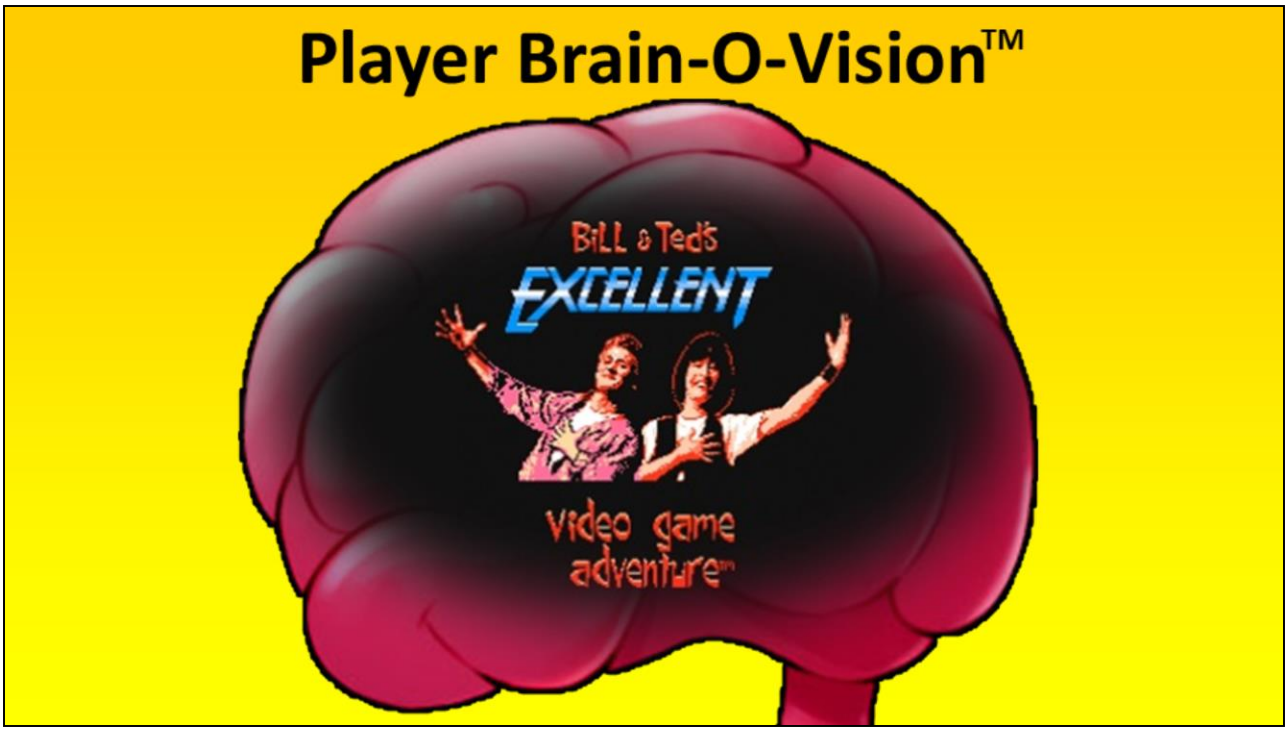
So let’s meet our player.

Our player has a brain

Naturally, our player has a brain.

Let’s try using Player Brain-O-Vision and imagine what this guy is thinking. Let’s try to “see” what his brain sees.

Based on this guy’s expression? I think.. He’s playing..

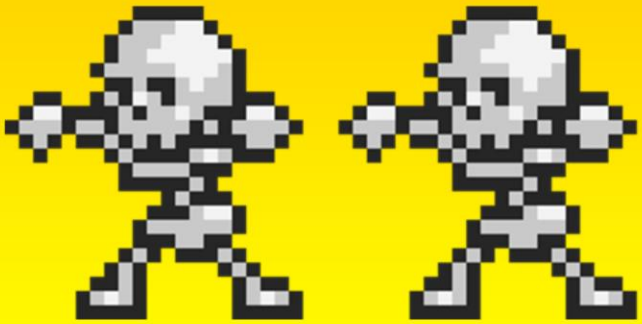


Bill & Ted's Excellent Video Game Adventure.

Making Enemies Actually Different:
DON'T DO's

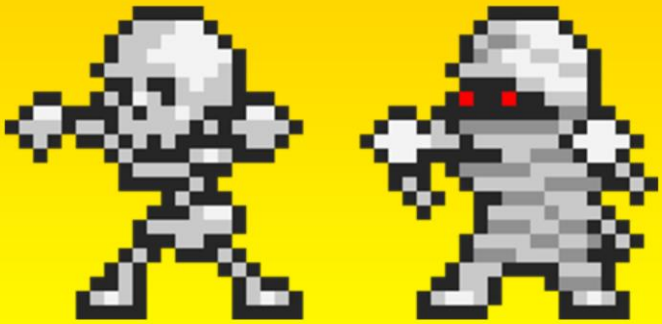
So let's get into this. We'll start with some DON'T DO's. Things you don't wanna do when your goal is to make your enemies *actually* different.

Don’t Do:
Different Look, Same Function

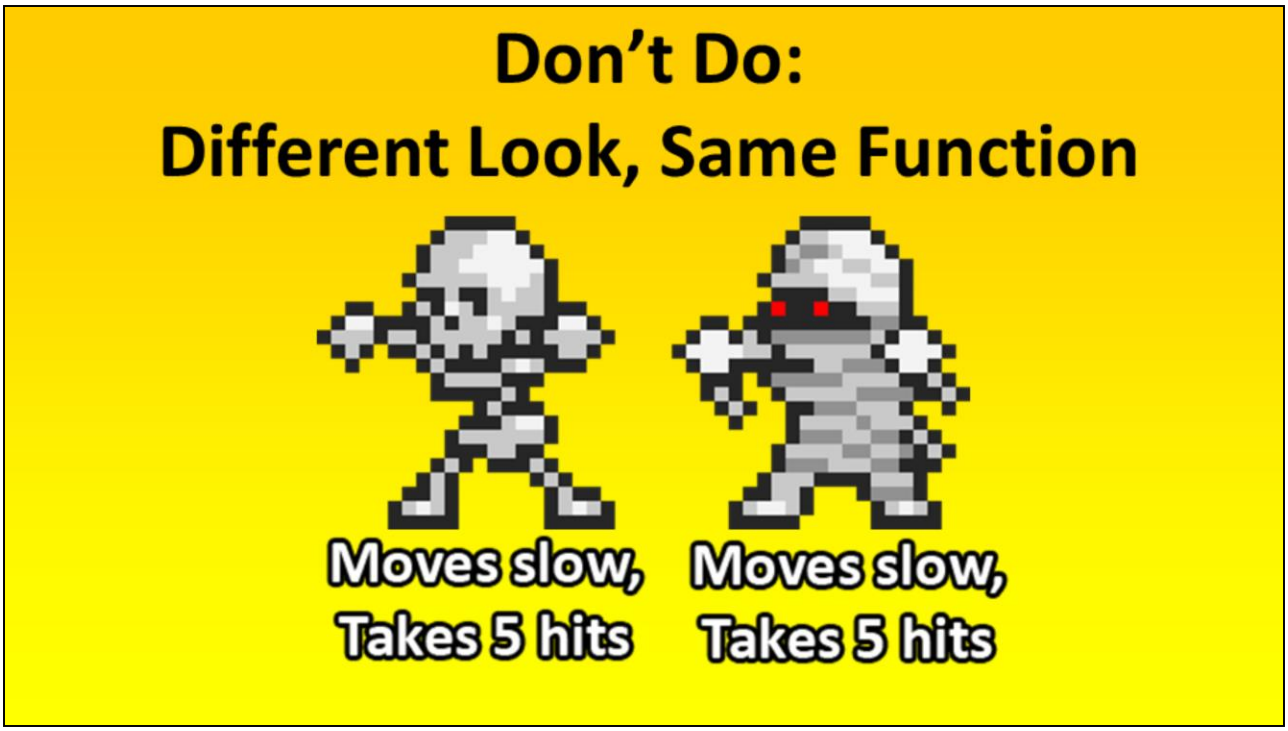


The first pitfall happens when we have two enemies that are identical, and think

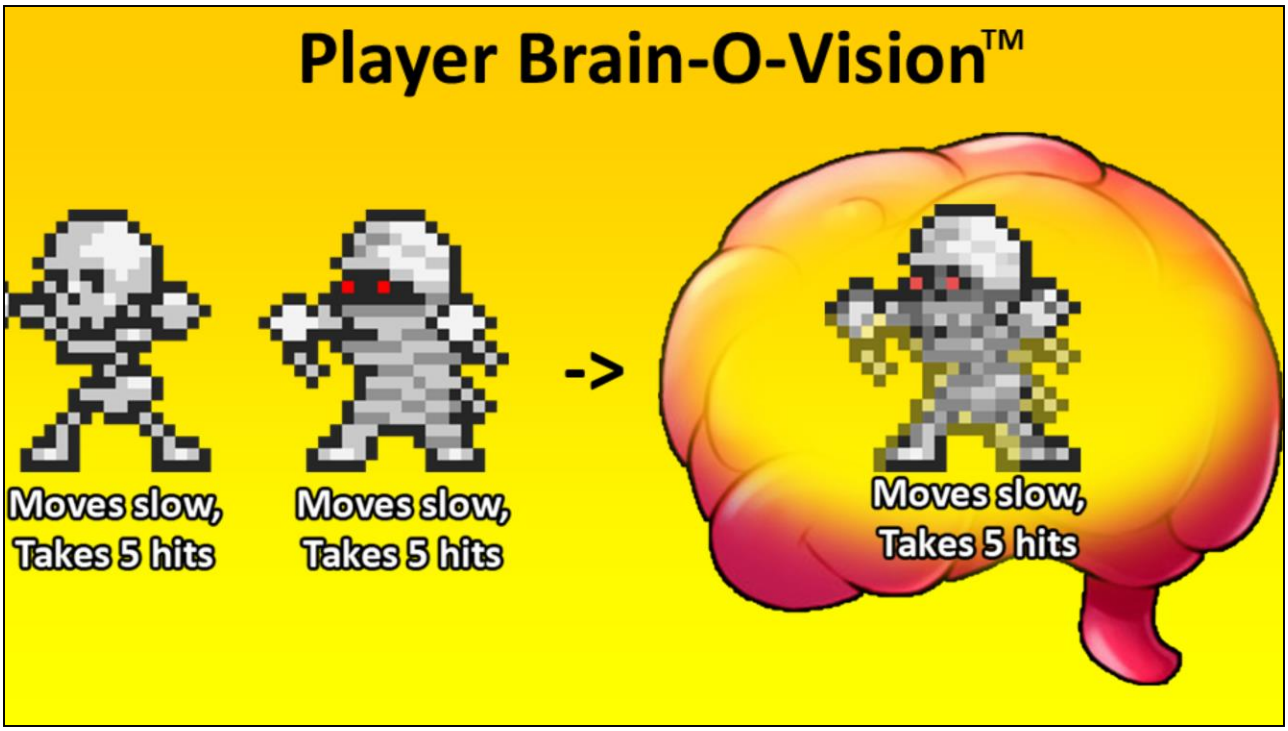
Don’t Do:
Different Look, Same Function



“Oh, we’ll just change the look of one of em, that’ll be enough” <buzzer sound> Do this over and over again and you’ll have the most boring game of all time.



The problem here is the two enemies are still doing the same thing. The player doesn’t handle the skeleton any different than the mummy..



so they'll store both of these in their brain, occupying the same space. Our brains tend to clump things together in order to keep up with all the information we need to process.

The player's goal is to beat the game. To that extent, the player doesn't care that the mummy *looks* different from the skeleton. In its efficiency, the brain will squeeze these two together into just one enemy. This is what we want to avoid, as we're not adding to true enemy variety here.

Don’t Do:
Just Palette Swap It

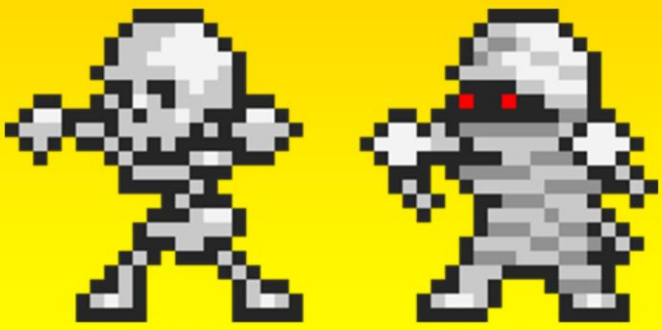
Even lazier than just changing the look is just changing one color to another.

I certainly didn’t think of this grey fish from Super Mario as a brand new enemy type. I’ll cut them some slack though cause they were up against some extreme memory limitations at the time. But nowadays, color shifts just don’t pass for “brand new enemies” anymore.

I don’t want to play this game

An abundance of em can often make your game feel cheap, and most of the time you’re better off with just 1 solid enemy instead of 2 palette swapped enemies.

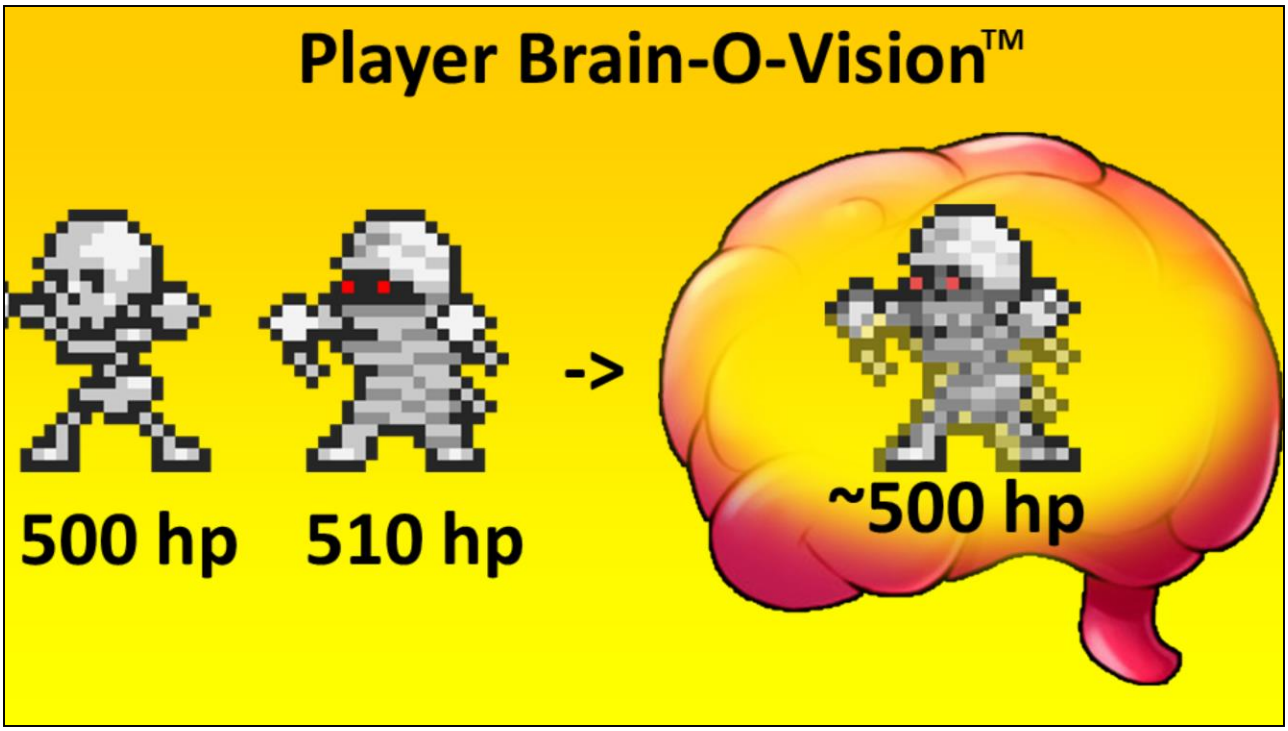
Don’t Do:
Small changes in hp, damage, speed



500 hp 510 hp

One more thing is.. to not think of small changes in hp, damage, and speed as brand new enemies.

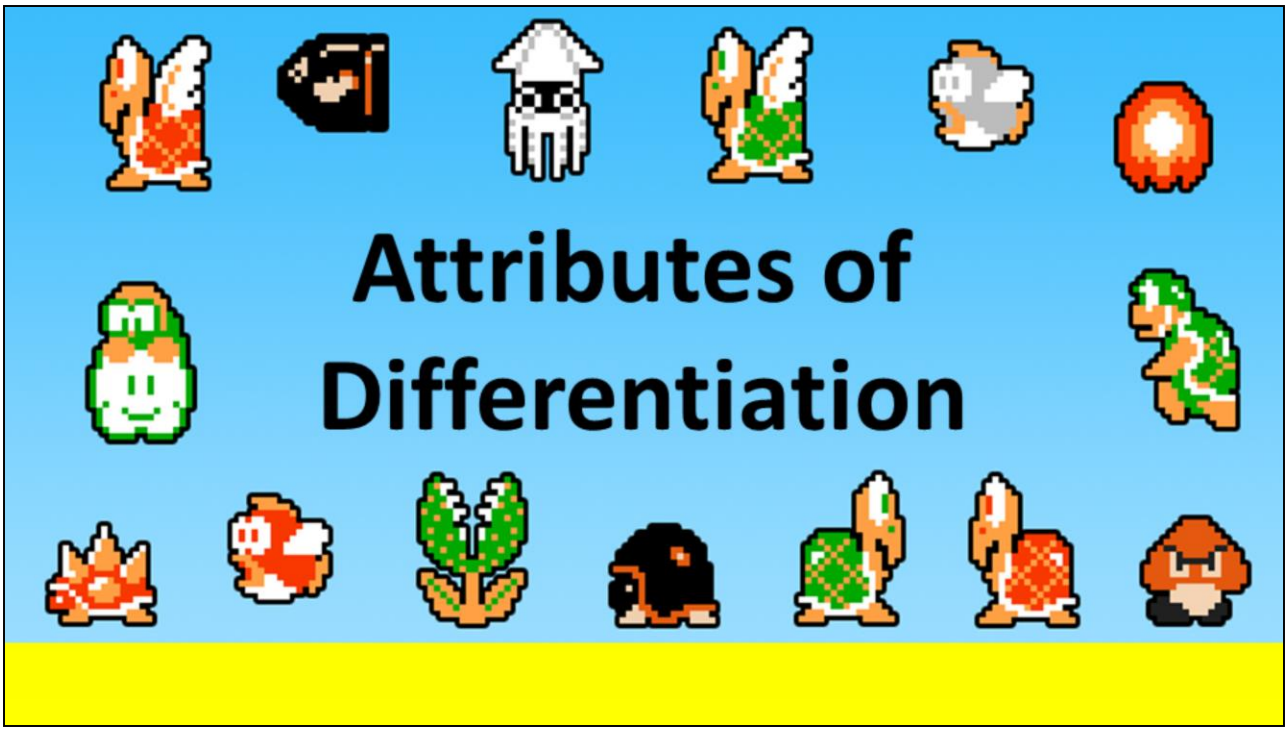
A little bit of this is ok, but if every bad guy in your game can be defined as just some degree of these three variables, then you’ve done something wrong.



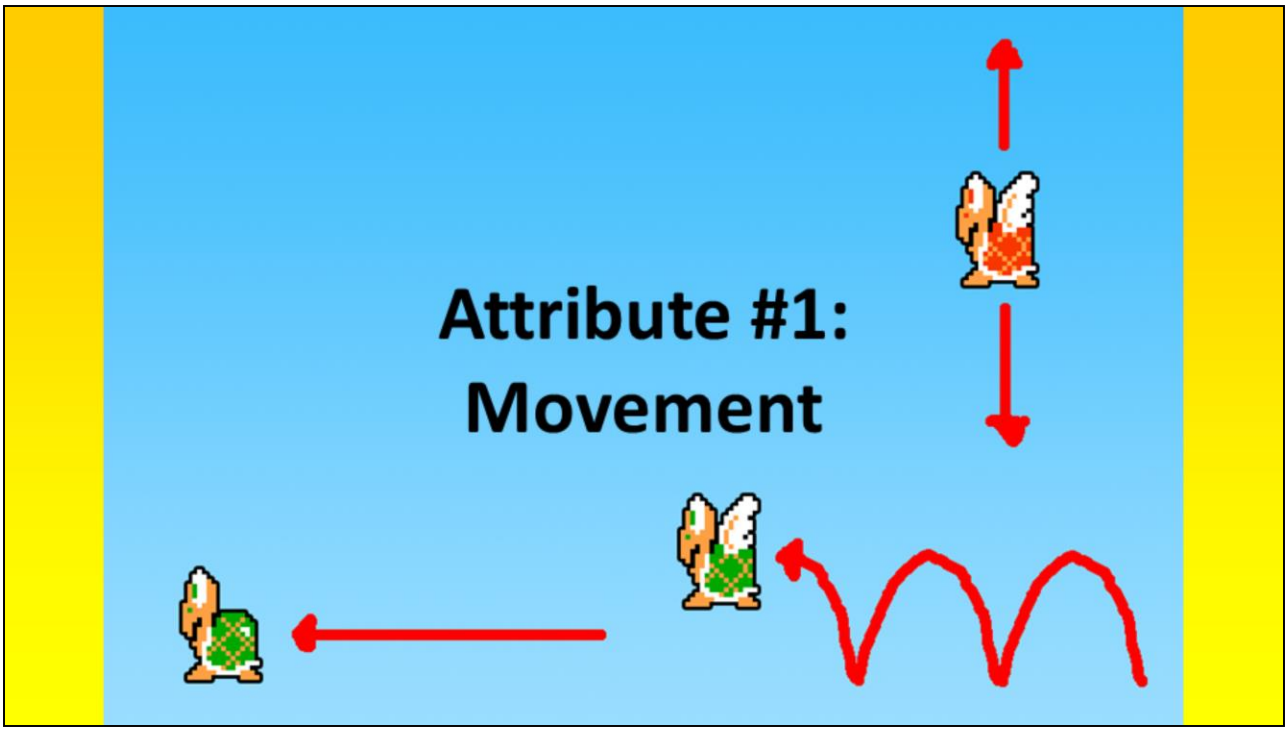
The difference between 500 and 510 is small enough that once again, the player's brain will meld the these two together into just one bad guy with about 500 hp.

All right, now that we've covered some Don't Do's…

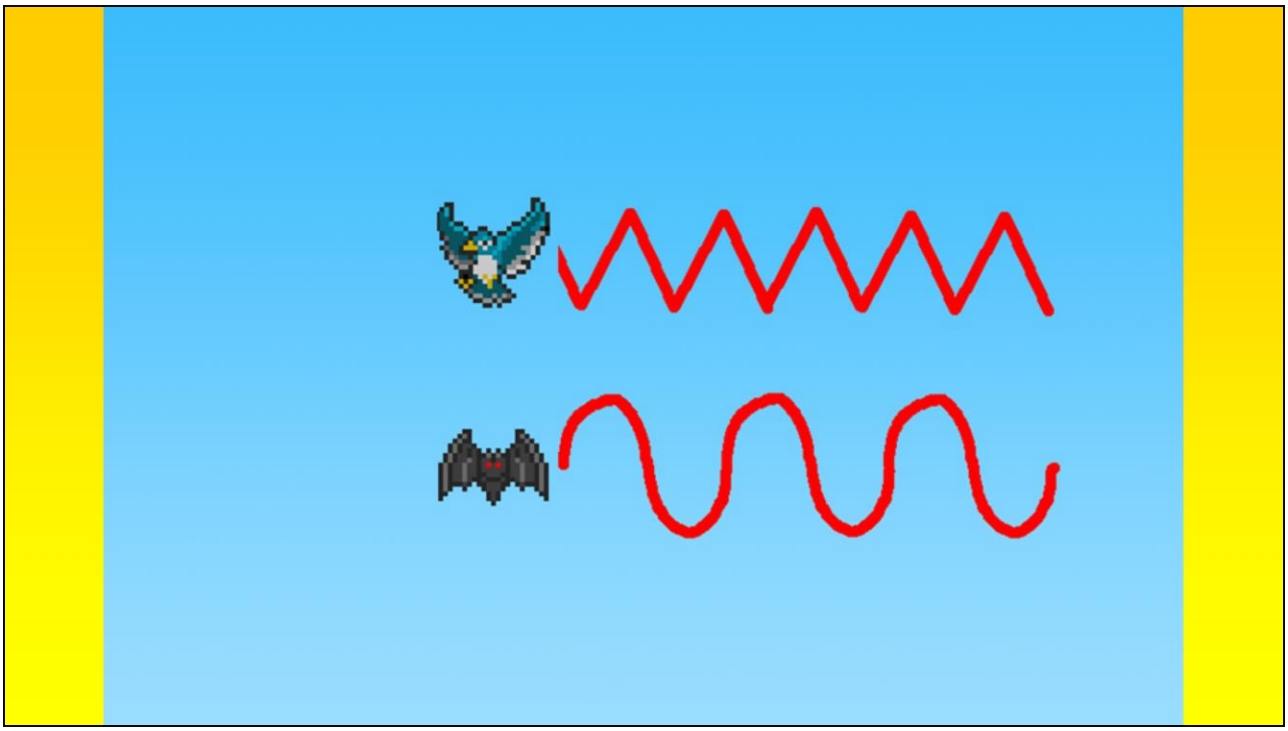
It's time to take a look at some DO DO's. I've got a big DO DO for you to take a look at..



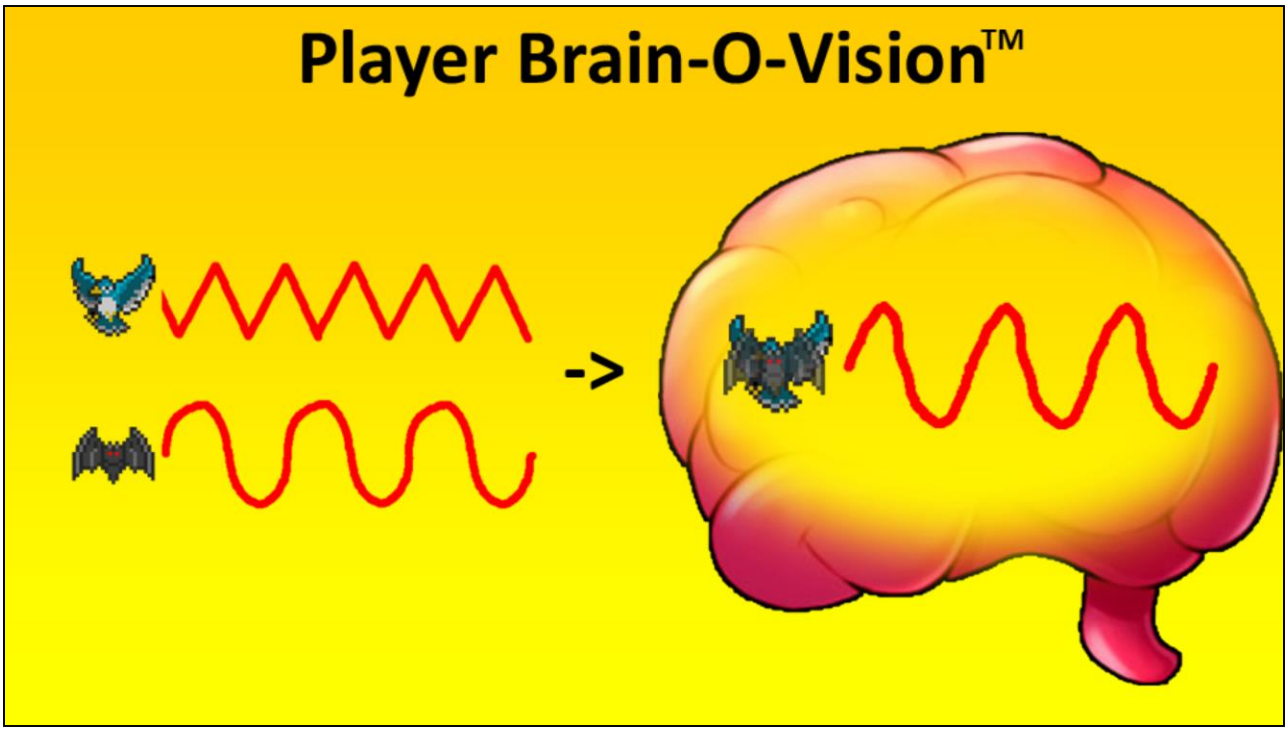
And I’m gonna call it “Attributes of Differentiation”. Basically, these are qualities enemies might have.. that would make em play differently than others. You can think of em as sort of avenues to explore while you’re trying to make your enemies actually different. In this section of the talk, I’ll provide you with some examples of attributes I find myself using again and again.



The first one of these is movement. We can often make the player handle the enemy differently by simply introducing a new movement pattern. In Super Mario, these enemies all have distinct styles of movement, and the player needs to account for each one differently.



But be careful you don’t spend time making things have special movement.. when in the end.. it gets handled by the player just like something else. In this hypothetical game, we’ve designed two enemies, a bird that moves in a zig-zag and a bat that moves in a sine wave.



There might cases where this difference *is* significant, but for the most part I see players handling these two movement patterns in the same way. Again, not good if our goal is to make enemies *actually* different. Stuff like this can be found through playtesting, but using Player Brain-O-Vision we can often catch it earlier.

Attribute #2: Priority

Another attribute I like to consider is priority, meaning, when seeing lots of enemies on screen, which order to defeat them in? How urgently do I need to defeat this enemy relative to others? An example of a high-priority enemy is the generator from Gauntlet. It doesn't matter how many ghosts you kill, if you don't kill the bone piles first, they'll keep spawning more ghosts.

High Priority

The UFO from Space Invaders is an interesting case. You prioritize it not because it's threatening, but because it's worth a bunch of points and only on screen for a short time.

Let’s think about priority some more. Suppose we have a group of basic grunts.

Now, let’s add a healer to that group. If we try killing the grunts first, it’ll take a long time cause the healer will keep em healed.

So the correct play is to focus on killing the healer first, and then take care of the grunts after the healer's been dealt with.

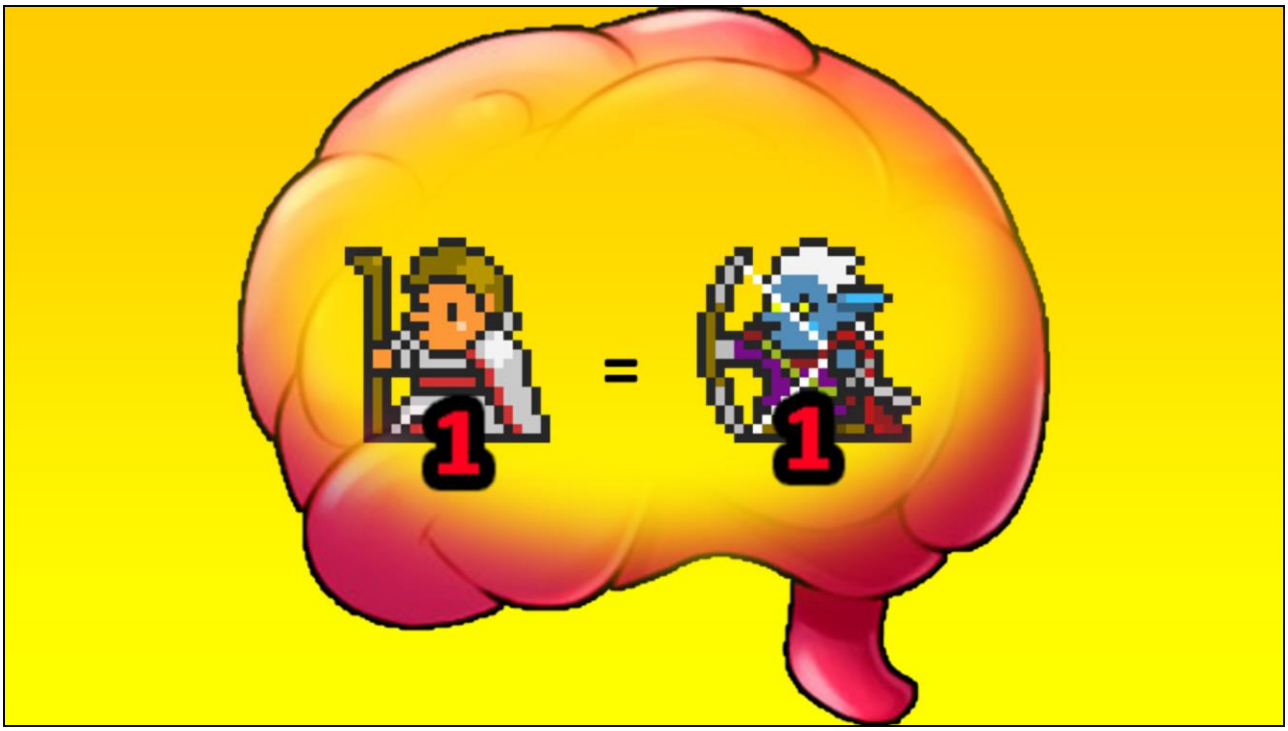
But what about this?

Let’s *instead* add a high damage archer to the group..

In this case, the *archer* becomes our number one priority because it's the more threatening unit. Again, it makes more sense to prioritize killing the archer over the grunts. Now, as long as our weird hypothetical game only presented you units in these two formations of..

Healers and grunts..

and Archers and grunts..



The healer and archer would be seen as exactly the same unit in terms of priority, even though they perform vastly different roles. Just something to keep in mind.

“Deal with me first”

Fact is, a lot of enemies we design tend to fall into this high priority, “Deal with me first” category

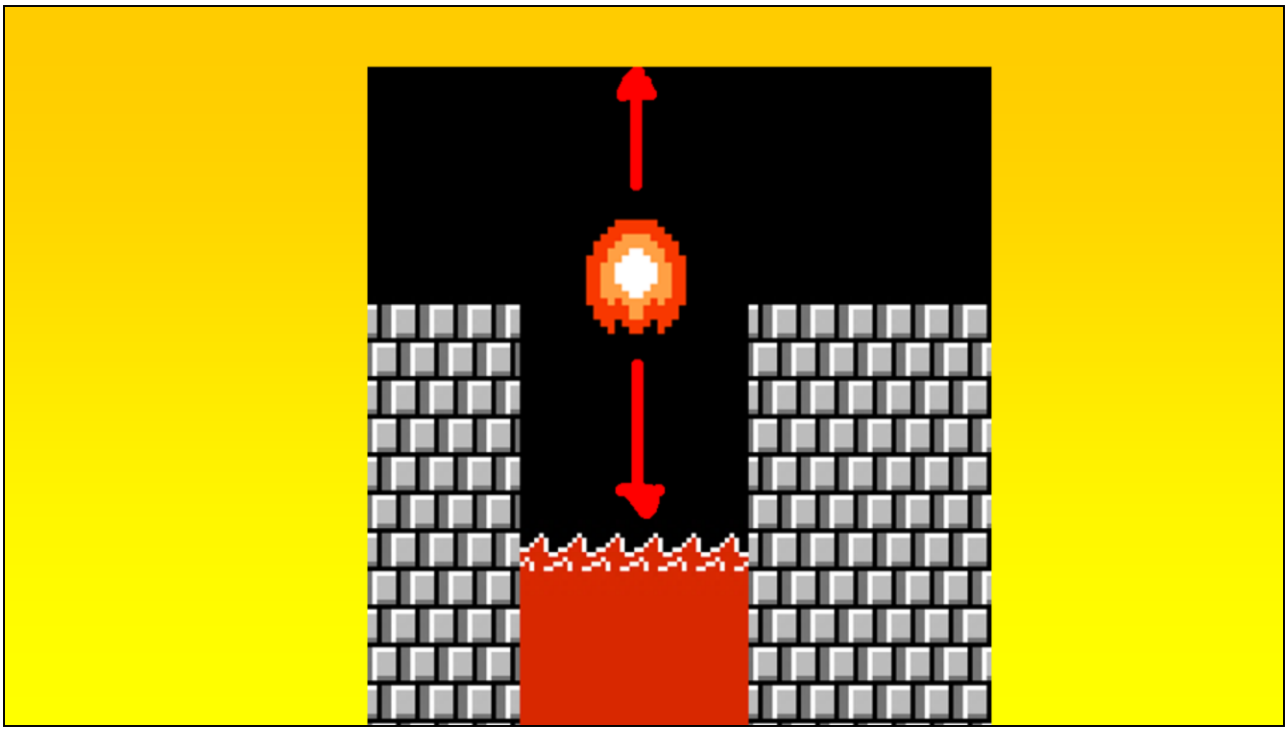
That's why it's good to explore the other end of the spectrum and design enemies that are best dealt with last. Both of these guys get mad and become more deadly after you damage em, so it's best to ignore em if you have other enemies to deal with.

Enemies that have lots of hp but do low amounts of damage also fit into this “Deal with me last” category.

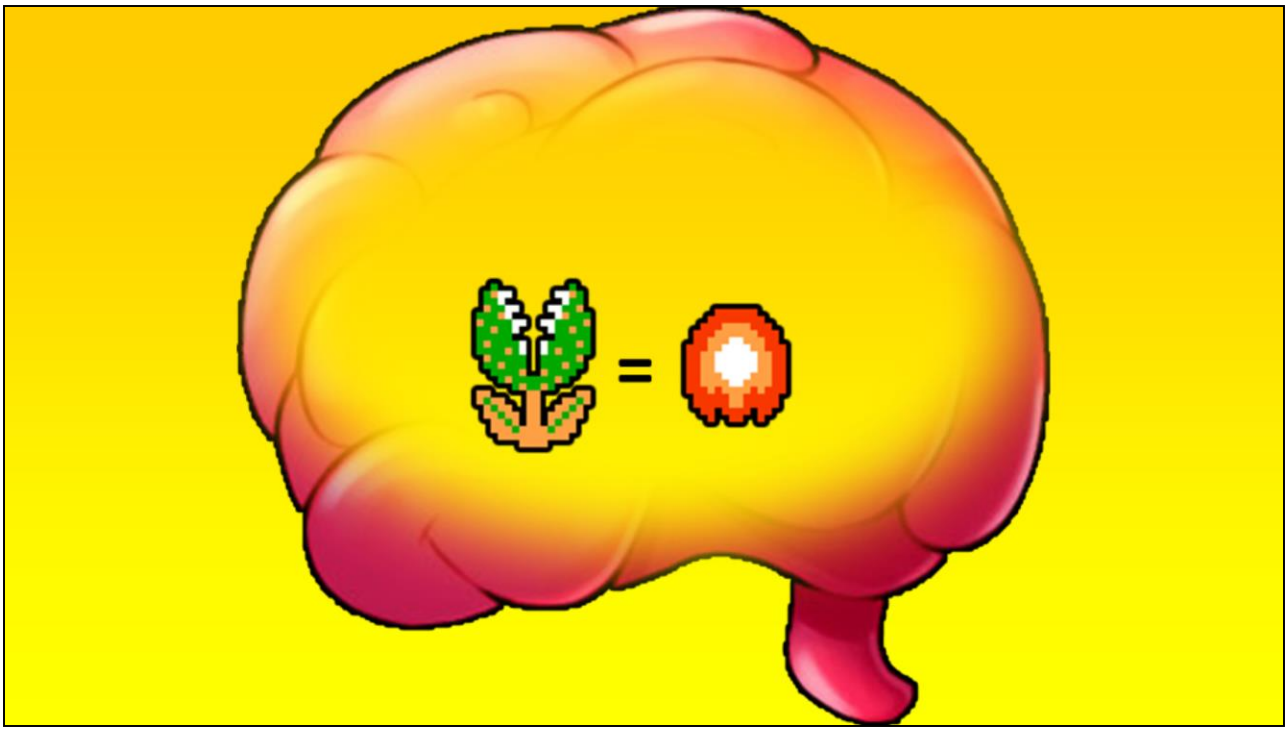
Attribute #3: Timing

Another attribute we can consider is timing. It often involves having an enemy that's more dangerous during certain times, or likewise more vulnerable during certain times.

A prime example of this is the Piranha Plant from Super Mario. It's safe to pass while it's retracted into the pipe, but dangerous otherwise.



The lava bubble is another bad guy from Super Mario that hops in and out of the lava and requires timing to avoid.



If we're observant and we use Player Brain-O-Vision, we might discover some similarities between these two units. I initially didn't think of these as being very similar, but in researching this talk I realized they're both timing-based enemies with up-and-down movement. The only differences are the lava bubble is immune to Mario's fireball, and only found in lava. If this were our own game, it would then be up to us to decide if these small differences were enough.

Attribute #4:
Range

On to the next attribute. When most of the enemies in your game do close-ranged attacks, you can shake things up by offering some enemies that are long-ranged.

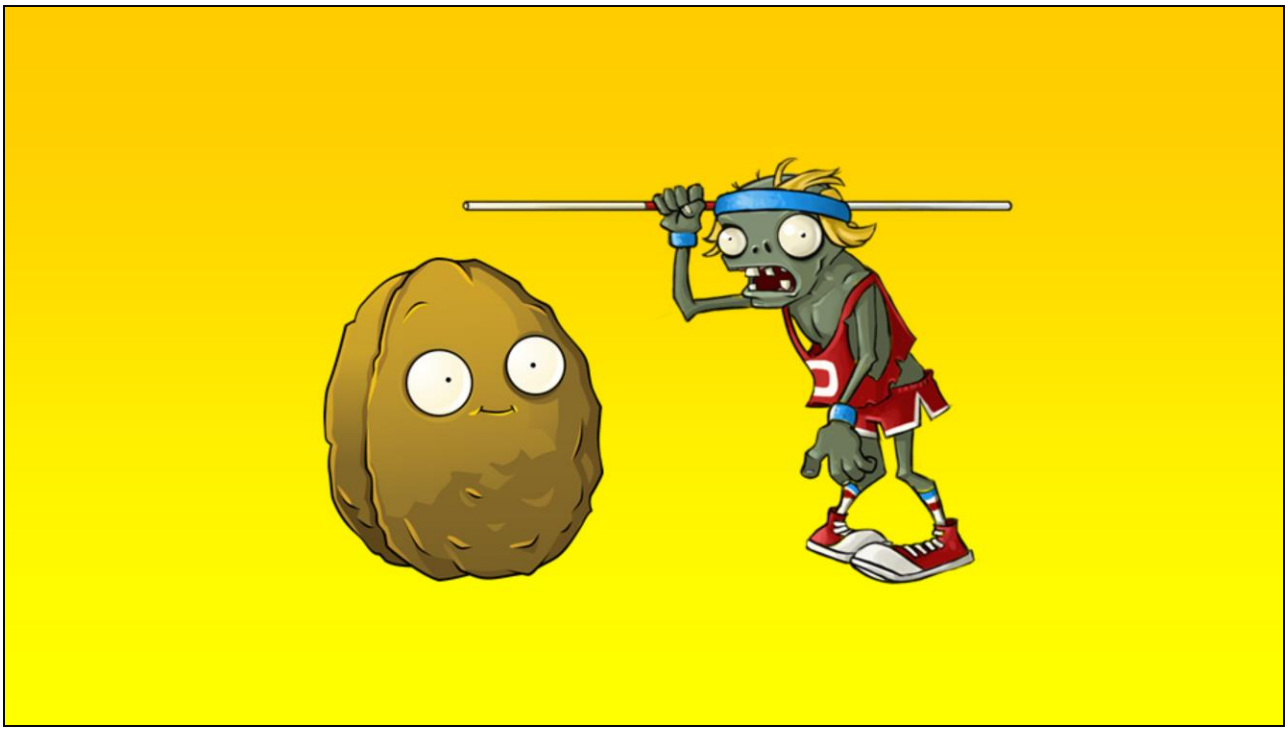
The catapult zombie and hammer brothers are examples of long-ranged enemies in games where most of the enemies are close-ranged.

Likewise, if most enemies in your game are long-ranged, consider throwing in a close ranged enemy just to liven things up. A lot of these attributes come down to.. being observant of what happens often in your game.. and designing things to disrupt that.

Attribute #5: Enemy-Counters-Player

Which brings us to our next attribute. A good way of coming up with new enemies.. is to think about what actions the player does most often in your game, then come up with an enemy that counters that action to shake things up.

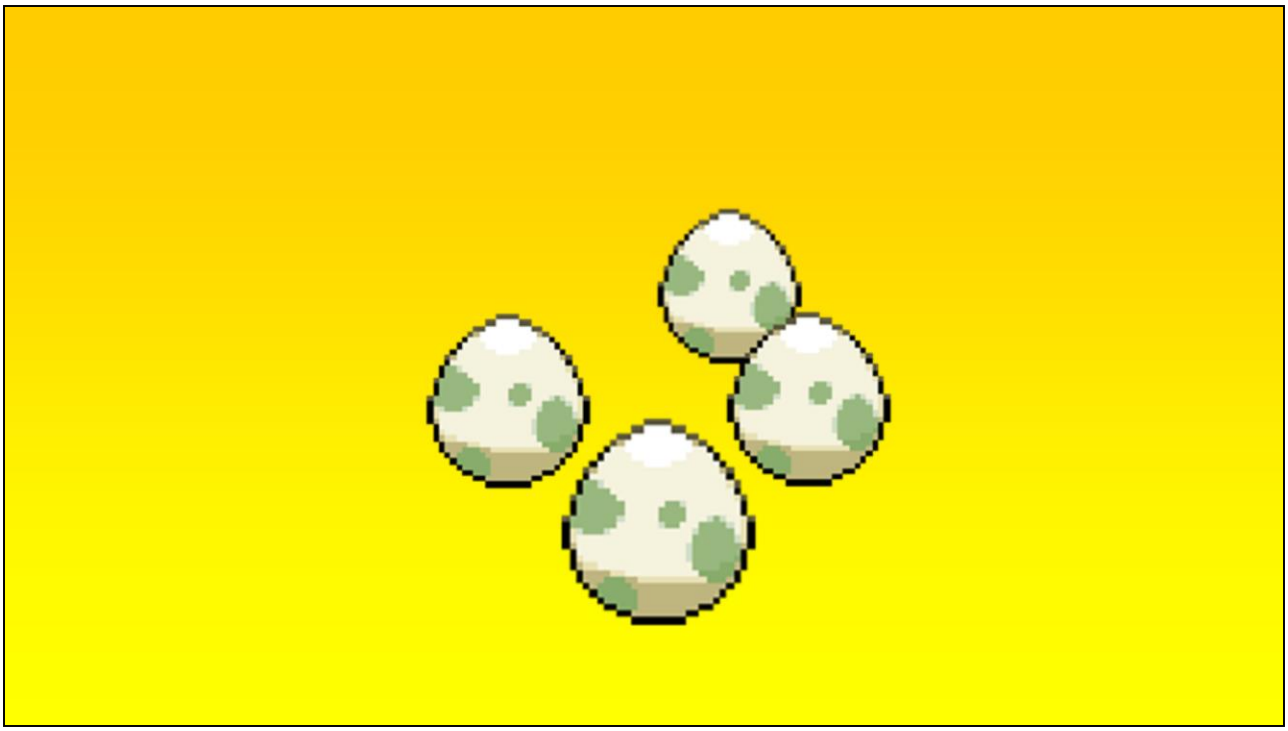
In Super Mario, the most common action used to defeat enemies is to stomp on em. So halfway through the game, the Spiny enemy is introduced, which you obviously can't stomp on. Along the same lines, the buzzy beetle is immune to fireballs, another common action of Mario's. Both enemies do their job of keeping Mario on his toes and not letting him perform just the same actions over and over.



In Plants vs Zombies, I had the early issue of Wall-nuts being really good, and noticed players falling into a rut of using wall-nuts to solve everything. So I designed these Pole-Vaulting zombies that would leap over the first plant they ran into, thus making wall-nuts a little less universally good and mixing things up for the player.

Think about what your players do often, then come up with ways for your enemies to disrupt that.

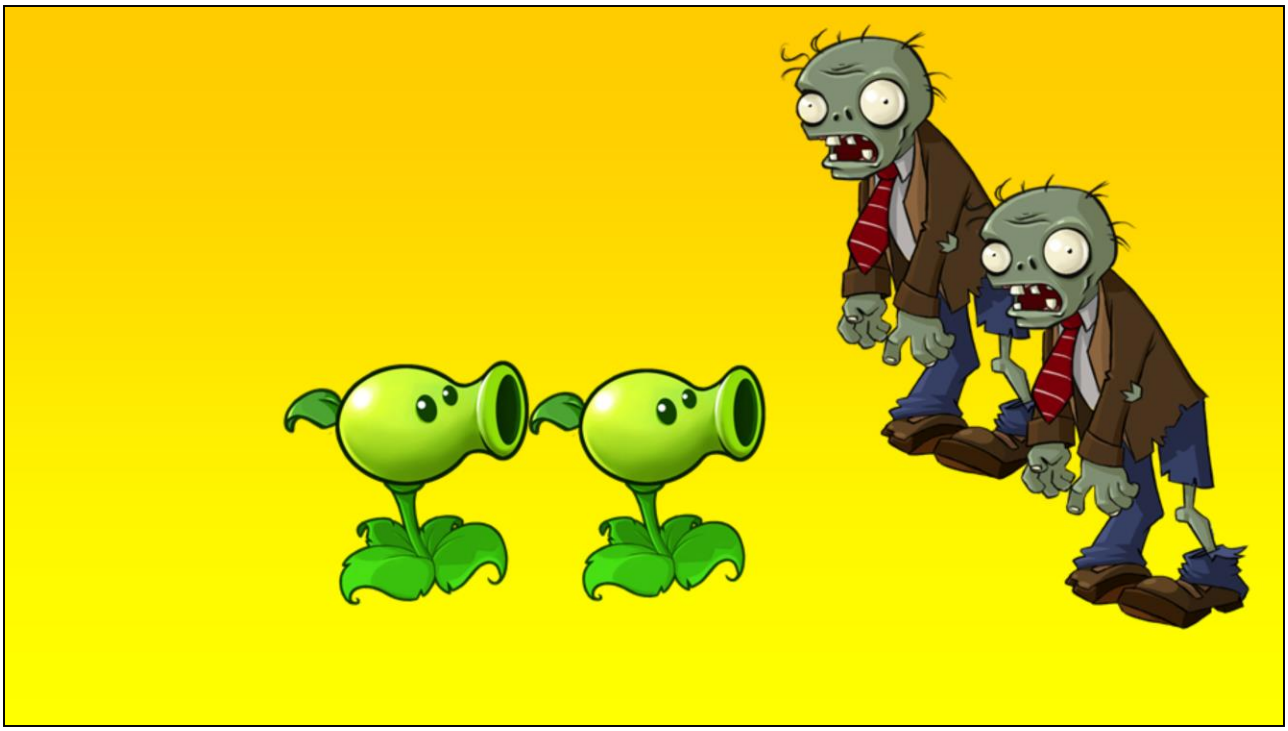
A good exercise is to think about what your players do often, then come up with ways for your enemies to disrupt that.



For example if you notice in your game your players are just constantly spamming AOE attacks, you could design a unit that counters AOE. These eggs, when damaged,

release extremely powerful demons, punishing the player who just goes around AOE attacking everything willy-nilly. Unleashing all the demons at once is a bad move as the demons become too overwhelming for the player to deal with.

Instead, the correct play is to break the eggs one at a time, dealing with each demon individually before breaking the next egg. With this one enemy type, we've added a nice beat to our game where players play more carefully for a bit, and then they go back to constantly spamming AOE.



In PvZ, I knew players had gotten used to zombies coming in from the right side of the screen,

So I threw em a curveball and introduced the Digger Zombie, which surprise-attacks your plants from the left!

Attribute #6:
Player Counters Enemy

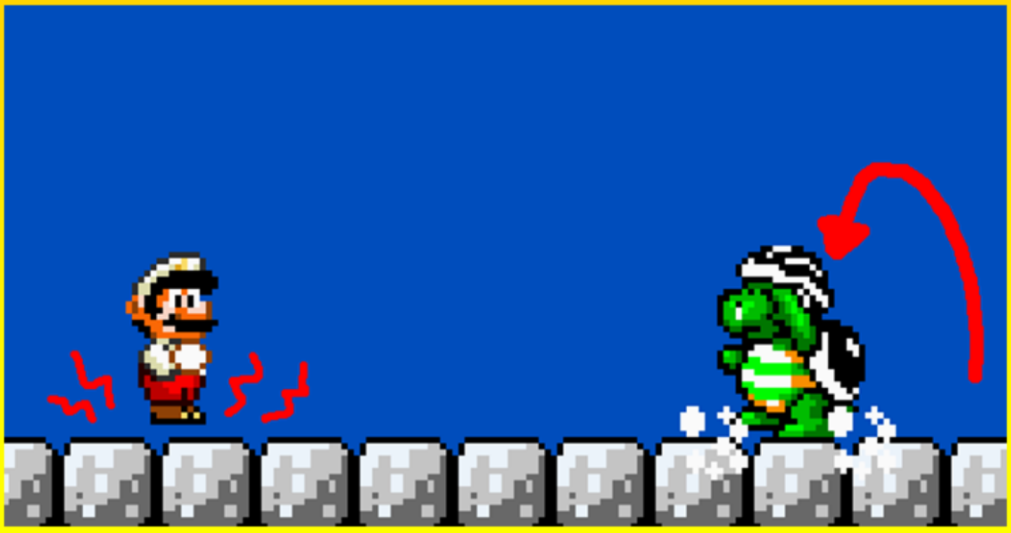



On the flip side, we can also design enemies by giving em a weakness, then offering a power to the player that exploits that weakness

Feels good to defeat enemies you couldn't before

It often feels satisfying to face down an enemy that's tough to deal with initially, but then you're given a power that helps you defeat it with ease.

Attribute #7: Attention (telegraphed attacks)



Finally, we have the attribute of player attention, which mostly comes in the form of telegraphed attacks. This involves designing enemies whose attacks deal a ton of damage if not avoided, but give you ample warning.. so you have the opportunity to avoid them. How they function in groups of enemies is to demand your attention for some amount of time.

An example of this is from Super Mario 3, where this sledge brother will jump up into the air, forecasting a ground slam. If you're still on the ground when they slam into it, you'll be stunned for some time. This forces you to pay attention and react by jumping.

To recap…

Ok, to sum things up, here's an effective way to make your enemies *actually* different.

To recap…

1. Use Attributes of Differentiation

First, try to design each new enemy with an Attribute of Differentiation in mind.

To recap…

1. Use Attributes of Differentiation

a) Movement b) Priority
c) Timing d) Range
e) Player-Counters-Enemy
f) Enemy-Counters-Player
g) Attention (Telegraphed Attacks)

I've given you some examples of attributes you can explore.

To recap…

1. Use Attributes of Differentiation

a) Movement b) Priority
c) Timing d) Range
e) Player-Counters-Enemy
f) Enemy-Counters-Player
g) Attention (Telegraphed Attacks)

2. Then use Player Brain-O-Vision

Once you've done that, pass the enemy through Player Brain-O-Vision.. to see if it *actually* makes the player DO something different.

To recap…

1. Use Attributes of Differentiation
    a) Movement
    b) Priority
    c) Timing
    d) Range
    e) Player-Counters-Enemy
    f) Enemy-Counters-Player
    g) Attention (Telegraphed Attacks)
2. Then use Player Brain-O-Vision
3. Your enemies are now awesome!

Do this, and your enemies will be awesome. I'm looking forward to playing your upcoming games with truly diverse sets enemies.

If you wanna talk some more about this, my info's right here. I'll be coming out with a game later this year that has its own share of enemies that're *actually* different. Follow me on twitter for more updates on that.

All right, thanks!

Thank you George!

I feel like right now people in the audience have cracked open their lap tops are ripping out their palette swapping code

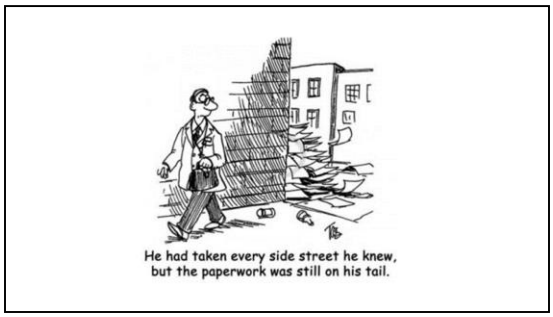
Next up, we have Liz England!

She’s a game design veteran having worked on everything to Scribblenauts to Resistance to…

Sunset Overdrive – a game we both worked on but as is the case in modern development, we didn’t meet until this week

Now, for another modern reality, if you’ve been doing game design in a big team for a while, you’ll know that no one reads your precious game design documents.

Well, good news, Liz has the solution!



Today I'll share a trick - more like a philosophy - with you that I call "Make actionable documentation".

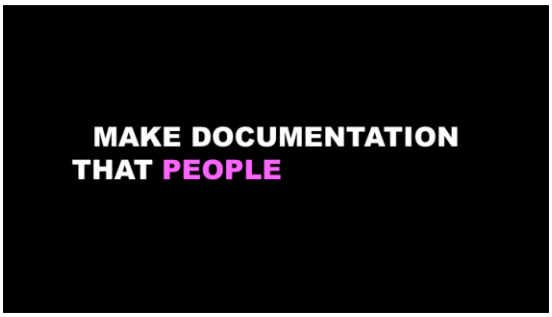
This talk is kind of specific and more advanced. It assumes you already know many of the trials and tribulations of making good design documentation.

(Will replace this pic with one of GDDs around the office)

For some further reading I recommend this prior GDC talks on design docs because I'll be spending my 10 minutes on a specific subtopic.

For some further reading I recommend this prior GDC talks on design docs because I'll be spending my 10 minutes on a specific subtopic.

For some further reading I recommend this prior GDC talks on design docs because I'll be spending my 10 minutes on a specific subtopic.

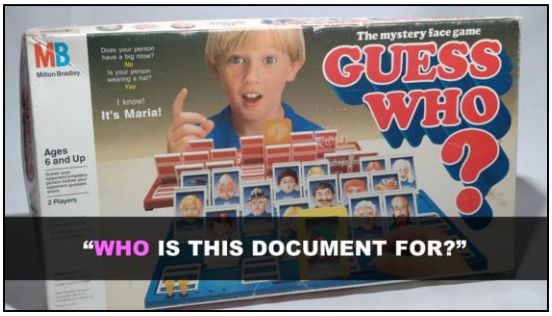


For some further reading I recommend this prior GDC talks on design docs because I'll be spending my 10 minutes on a specific subtopic.

For some further reading I recommend this prior GDC talks on design docs because I'll be spending my 10 minutes on a specific subtopic.

For some further reading I recommend this prior GDC talks on design docs because I'll be spending my 10 minutes on a specific subtopic.

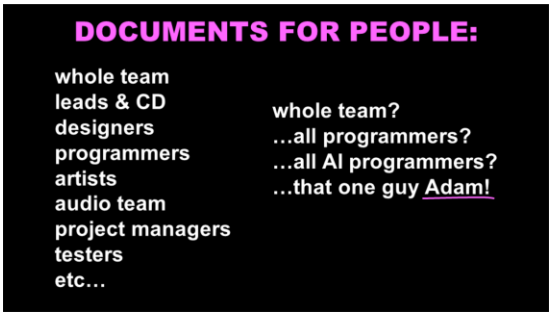
For some further reading I recommend this prior GDC talks on design docs because I'll be spending my 10 minutes on a specific subtopic.



Instead of asking "what feature am I documenting?" ask yourself this: "Who is this document for?" A game can't read your documentation, it doesn't care if you document it or not. People can read it. People care. So write docs for people, not for games. If you can design a game for players, you can design documentation for readers.

When I talk about people I'm obviously talking about members of your team. When you spec out a system, don't make a doc for the entire team. Make a document for an artist, or for designers, or for programmers.

When I talk about people I'm obviously talking about members of your team. When you spec out a system, don't make a doc for the entire team. Make a document for an artist, or for designers, or for programmers.



I encourage you to get more specific, though. I try to write my documentation for an INDIVIDUAL as often as possible. So that means you make documentation for Adam the AI Programmer and ideally you sit down and talk to him about what he wants to see in that doc. Different people work differently, so some people want extensive use cases, others want flowcharts, and others might work best if you kind of narrate what the player is doing.

The important thing here is: the needs of the reader are most important. It doesn't matter whether you detail everything out, but rather you detail what your reader needs to know.

SUBJECTIVELY describe the system instead of OBJECTIVELY describing it

The next question I ask myself is, "What do I expect them to do with this?" Well what do you expect them to do? You expect them to read it, right? Wrong! Documentation needs to be USED somehow, not just read. It needs to serve a purpose.

So my second rule is "Make documentation with a purpose".

So my second rule is "Make documentation with a purpose".

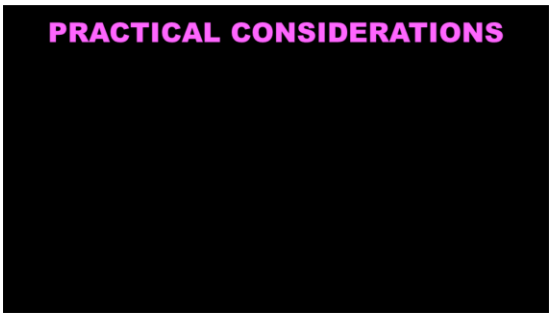
Game designers like to talk about verbs (I know I do) like run, jump, shoot, interact, etc. I like to try applying those **verbs to my documentation**. This is just identifying what action you want people to take when they read it.

Maybe you are writing a doc for someone to implement a system for you, which requires a large amount of detail. Or this doc is a blueprint that tells fellow designers how to implement something according to a set of guidelines. Maybe it's just a high level spec for a system that you need to shop around to the leads and creative director for approval. Maybe it's a big poster meant to inform the rest of the team and get everyone on the same page.

The important lesson here is that form will follow function.

MAKE ACTIONABLE DOCUMENTATION


MAKE
ACTIONABLE
DOCUMENTATION



As a summary, some of the practical implications of writing documents for people with a purpose is:

I write docs when they are needed, and not before. I try not to anticipate what documents we'll need.

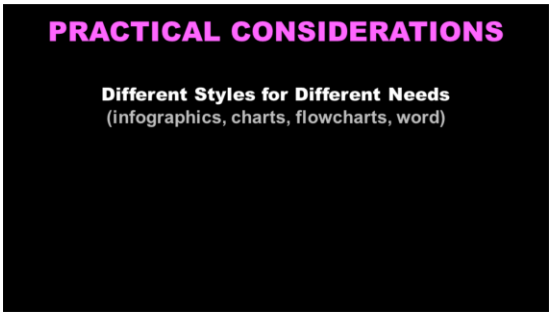
Each doc tends to be smaller and more concise, with its own specified level of detail. That also means a single feature may have multiple documents for it with some, but not tons, of overlap.

Actionable docs require different styles of documents for different needs. I find word docs are still the most common, but least effect, way of sharing information. Don't be afraid to use infographics or flowcharts or excel files, and keep in mind the level fo detail you actually need and don't try to be everything for everyone.

It's also okay to not keep docs up to date if they've served

their purpose. Sometimes the purpose in ongoing throughout development, and other times we should be comfortable with abandoning documentation.

Lastly, ‘actionable’ documentation (and I’ll get into examples in a moment) doesn’t have to replace every document you make. It replaces about half of my documentation, and the other half still tend to follow more traditional standards.



As a summary, some of the practical implications of writing documents for people with a purpose is:

I write docs when they are needed, and not before. I try not to anticipate what documents we'll need.

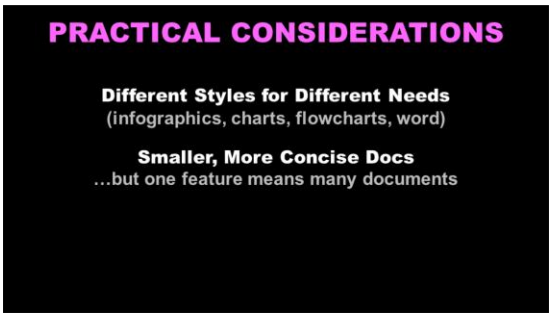
Each doc tends to be smaller and more concise, with its own specified level of detail. That also means a single feature may have multiple documents for it with some, but not tons, of overlap.

Actionable docs require different styles of documents for different needs. I find word docs are still the most common, but least effect, way of sharing information. Don't be afraid to use infographics or flowcharts or excel files, and keep in mind the level fo detail you actually need and don't try to be everything for everyone.

It's also okay to not keep docs up to date if they've served

their purpose. Sometimes the purpose in ongoing throughout development, and other times we should be comfortable with abandoning documentation.

Lastly, ‘actionable’ documentation (and I’ll get into examples in a moment) doesn’t have to replace every document you make. It replaces about half of my documentation, and the other half still tend to follow more traditional standards.



As a summary, some of the practical implications of writing documents for people with a purpose is:

I write docs when they are needed, and not before. I try not to anticipate what documents we'll need.

Each doc tends to be smaller and more concise, with its own specified level of detail. That also means a single feature may have multiple documents for it with some, but not tons, of overlap.

Actionable docs require different styles of documents for different needs. I find word docs are still the most common, but least effect, way of sharing information. Don't be afraid to use infographics or flowcharts or excel files, and keep in mind the level fo detail you actually need and don't try to be everything for everyone.

It's also okay to not keep docs up to date if they've served

their purpose. Sometimes the purpose in ongoing throughout development, and other times we should be comfortable with abandoning documentation.

Lastly, ‘actionable’ documentation (and I’ll get into examples in a moment) doesn’t have to replace every document you make. It replaces about half of my documentation, and the other half still tend to follow more traditional standards.

Let’s compare that to another visual document that has very few words.

This is a map of Sunset Overdrive we used throughout production. It’s visual, but relatively easy to keep up to date, and almost everyone on the team used it to keep track of the state of the city. Being used a lot, that meant there was high demand to keep it up to date – which led to the doc’s form (photoshop file, not as pretty, rough around the edges).

The downside is that it requires some memorization of in-house terminology, so when new people rolled onto the team they needed to be brought up to speed on color coding or what “C2” or “Q3” meant. But that was not the intended reader for this document.

Let’s compare that to another visual document that has very few words.

This is a map of Sunset Overdrive we used throughout production. It’s visual, but relatively easy to keep up to date, and almost everyone on the team used it to keep track of the state of the city. Being used a lot, that meant there was high demand to keep it up to date – which led to the doc’s form (photoshop file, not as pretty, rough around the edges).

The downside is that it requires some memorization of in-house terminology, so when new people rolled onto the team they needed to be brought up to speed on color coding or what “C2” or “Q3” meant. But that was not the intended reader for this document.

I want to compare that to another map, though. One of the side effects of actionable documentation is that you often need multiple docs for one feature.

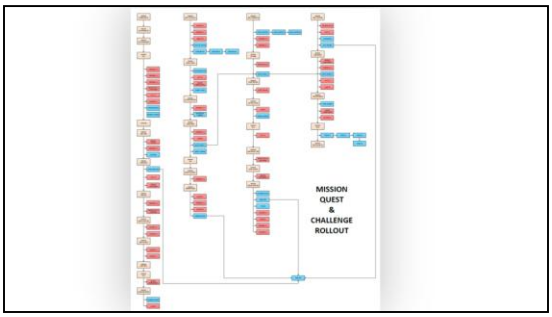
This map is a literal poster printout of the prior map with post-it notes. It was used by the challenge and quest team to help identify where to populate the game with content. This came about because sitting in a meeting trying to edit a photoshop file on a computer was really awkward compared to piling around a real life map and just moving post-it notes around.

(I promise to take a higher quality photo for final draft)

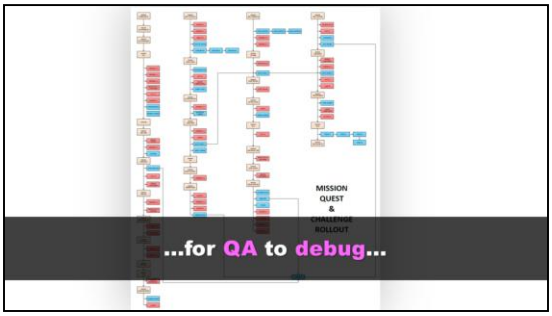
I want to compare that to another map, though. One of the side effects of actionable documentation is that you often need multiple docs for one feature.

This map is a literal poster printout of the prior map with post-it notes. It was used by the challenge and quest team to help identify where to populate the game with content. This came about because sitting in a meeting trying to edit a photoshop file on a computer was really awkward compared to piling around a real life map and just moving post-it notes around.

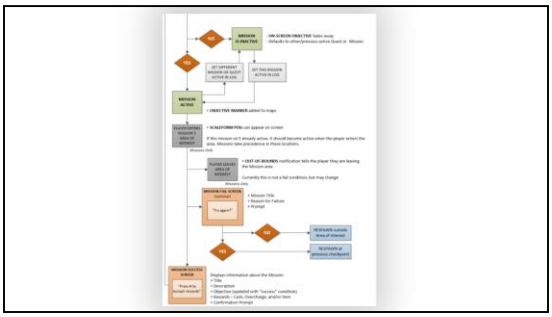
(I promise to take a higher quality photo for final draft)



How content unlocks over time. Tons of information, not many words. Giant flowchart image was much more successful than excel tracker. This doc was written in a way that meant everyone on the team could use it to find in-game content and grok our rollout plan.



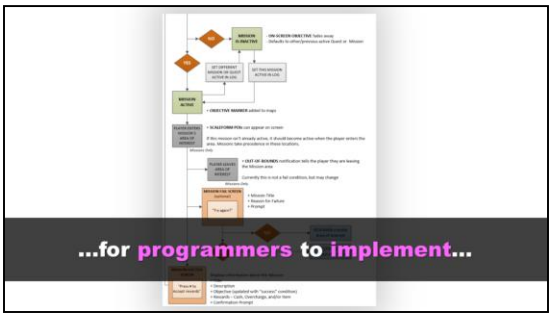
How content unlocks over time. Tons of information, not many words. Giant flowchart image was much more successful than excel tracker. This doc was written in a way that meant everyone on the team could use it to find in-game content and grok our rollout plan.



Here’s another compare and contrast.

This is the mission flow when a player accepts a mission, and what happens when they fail, or die and respawn, or quit the game, or change to a quest, and so on. This annotated flowchart worked great for the programmer working on that system.

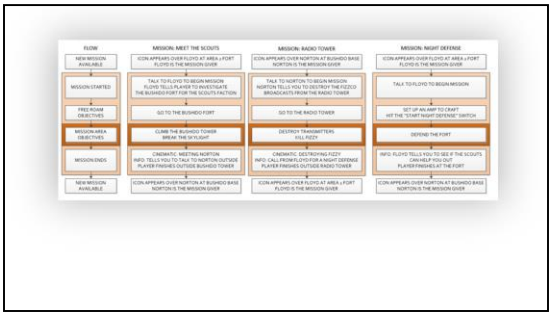
However, when I handed it to the mission designers to implement, their eyes just glazed over. As a result…



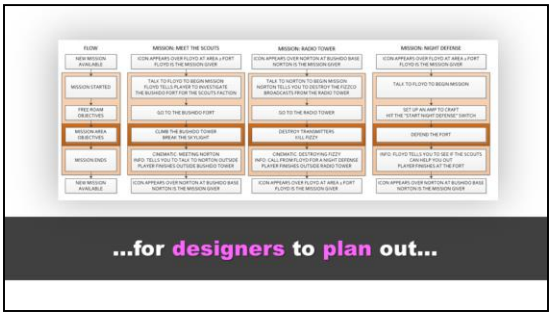
Here’s another compare and contrast.

This is the mission flow when a player accepts a mission, and what happens when they fail, or die and respawn, or quit the game, or change to a quest, and so on. This annotated flowchart worked great for the programmer working on that system.

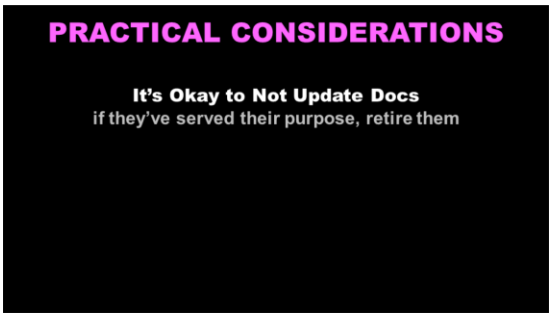
However, when I handed it to the mission designers to implement, their eyes just glazed over. As a result…



I made this document. It still tells the reader about how a mission flows, but it has much less level of detail and uses explicit examples from within the game. In this case, designers can look at examples and make intuitive leaps about how that affects their mission structure, without being bogged down with details they don't really need.



I made this document. It still tells the reader about how a mission flows, but it has much less level of detail and uses explicit examples from within the game. In this case, designers can look at examples and make intuitive leaps about how that affects their mission structure, without being bogged down with details they don't really need.



As a summary, some of the practical implications of writing documents for people with a purpose is:

I write docs when they are needed, and not before. I try not to anticipate what documents we'll need.

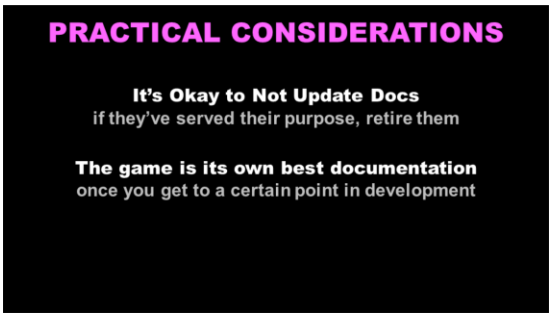
Each doc tends to be smaller and more concise, with its own specified level of detail. That also means a single feature may have multiple documents for it with some, but not tons, of overlap.

Actionable docs require different styles of documents for different needs. I find word docs are still the most common, but least effect, way of sharing information. Don't be afraid to use infographics or flowcharts or excel files, and keep in mind the level fo detail you actually need and don't try to be everything for everyone.

It's also okay to not keep docs up to date if they've served

their purpose. Sometimes the purpose in ongoing throughout development, and other times we should be comfortable with abandoning documentation.

Lastly, ‘actionable’ documentation (and I’ll get into examples in a moment) doesn’t have to replace every document you make. It replaces about half of my documentation, and the other half still tend to follow more traditional standards.



As a summary, some of the practical implications of writing documents for people with a purpose is:

I write docs when they are needed, and not before. I try not to anticipate what documents we'll need.

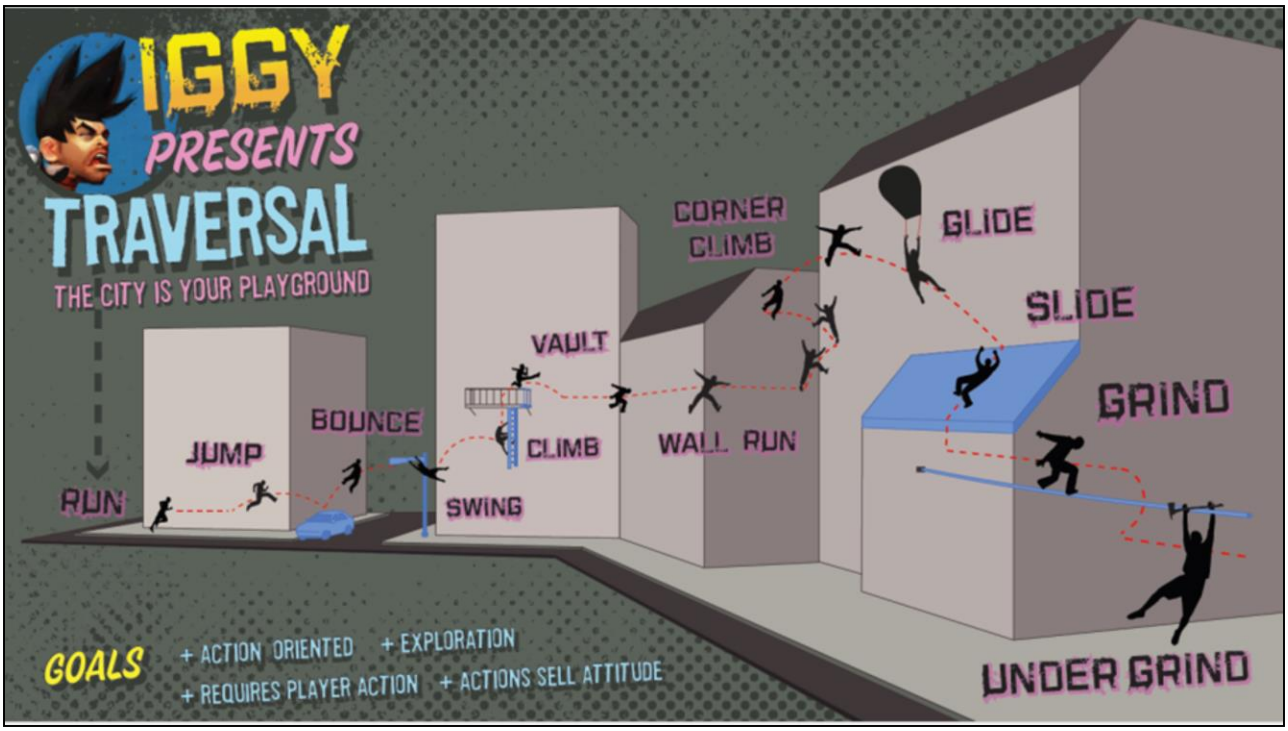
Each doc tends to be smaller and more concise, with its own specified level of detail. That also means a single feature may have multiple documents for it with some, but not tons, of overlap.

Actionable docs require different styles of documents for different needs. I find word docs are still the most common, but least effect, way of sharing information. Don't be afraid to use infographics or flowcharts or excel files, and keep in mind the level fo detail you actually need and don't try to be everything for everyone.

It's also okay to not keep docs up to date if they've served

their purpose. Sometimes the purpose in ongoing throughout development, and other times we should be comfortable with abandoning documentation.

Lastly, ‘actionable’ documentation (and I’ll get into examples in a moment) doesn’t have to replace every document you make. It replaces about half of my documentation, and the other half still tend to follow more traditional standards.

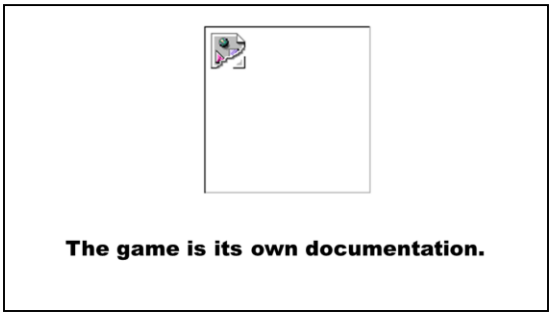


This is a visual design document early on Sunset Overdrive. It worked exceptionally well to inform the whole team about the traversal system we had planned for the game. It has a lot level of detail and a lot of visuals, and shows the promise of traversal. There's very little text.

There's no way you can implement off of this document. Visual docs also look great but are practically impossible to keep up to date. But that's okay! If you're making actionable docs – for people and with a purpose – then when it fulfills that purpose, don't be afraid to retire the doc and archive it. You do not need to try to keep every doc up to date.

This is a visual design document early on Sunset Overdrive. It worked exceptionally well to inform the whole team about the traversal system we had planned for the game. It has a lot level of detail and a lot of visuals, and shows the promise of traversal. There's very little text.

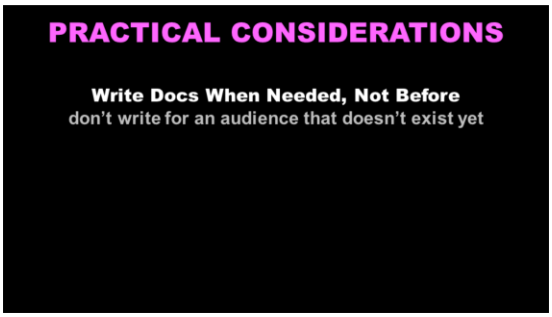
There's no way you can implement off of this document. Visual docs also look great but are practically impossible to keep up to date. But that's okay! If you're making actionable docs – for people and with a purpose – then when it fulfills that purpose, don't be afraid to retire the doc and archive it. You do not need to try to keep every doc up to date.



…but sometimes it does! In the case of our vanity system, by the end of the project we did not have a holistic doc that explained vanity to the whole team. People who worked directly on it had their own documentation.

With a system like player vanity, which had little affect on your work if you weren't on the system directly, we found that just pointing to other games, or to the initial preproduction proposal, or just letting people use the vanity closet that existed in game was enough to communicate the system to the team.

The game itself was better documentation than anything we would've written. A doc here would have been wasted time.



As a summary, some of the practical implications of writing documents for people with a purpose is:

I write docs when they are needed, and not before. I try not to anticipate what documents we'll need.

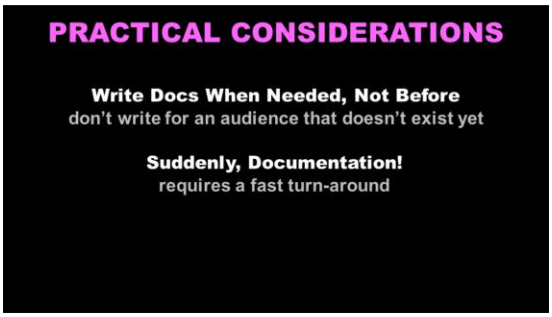
Each doc tends to be smaller and more concise, with its own specified level of detail. That also means a single feature may have multiple documents for it with some, but not tons, of overlap.

Actionable docs require different styles of documents for different needs. I find word docs are still the most common, but least effect, way of sharing information. Don't be afraid to use infographics or flowcharts or excel files, and keep in mind the level fo detail you actually need and don't try to be everything for everyone.

It's also okay to not keep docs up to date if they've served

their purpose. Sometimes the purpose in ongoing throughout development, and other times we should be comfortable with abandoning documentation.

Lastly, ‘actionable’ documentation (and I’ll get into examples in a moment) doesn’t have to replace every document you make. It replaces about half of my documentation, and the other half still tend to follow more traditional standards.



As a summary, some of the practical implications of writing documents for people with a purpose is:

I write docs when they are needed, and not before. I try not to anticipate what documents we'll need.

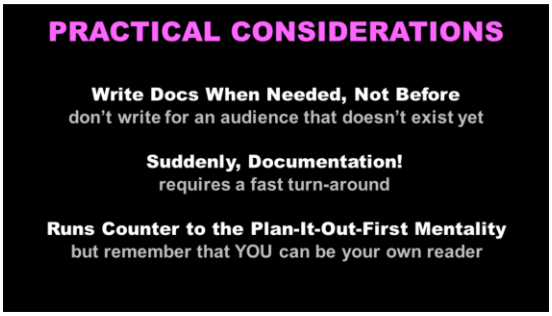
Each doc tends to be smaller and more concise, with its own specified level of detail. That also means a single feature may have multiple documents for it with some, but not tons, of overlap.

Actionable docs require different styles of documents for different needs. I find word docs are still the most common, but least effect, way of sharing information. Don't be afraid to use infographics or flowcharts or excel files, and keep in mind the level fo detail you actually need and don't try to be everything for everyone.

It's also okay to not keep docs up to date if they've served

their purpose. Sometimes the purpose in ongoing throughout development, and other times we should be comfortable with abandoning documentation.

Lastly, ‘actionable’ documentation (and I’ll get into examples in a moment) doesn’t have to replace every document you make. It replaces about half of my documentation, and the other half still tend to follow more traditional standards.



As a summary, some of the practical implications of writing documents for people with a purpose is:

I write docs when they are needed, and not before. I try not to anticipate what documents we'll need.

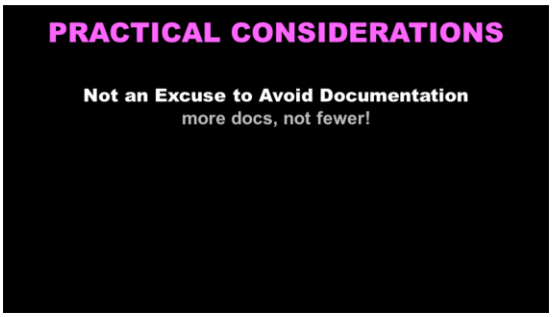
Each doc tends to be smaller and more concise, with its own specified level of detail. That also means a single feature may have multiple documents for it with some, but not tons, of overlap.

Actionable docs require different styles of documents for different needs. I find word docs are still the most common, but least effect, way of sharing information. Don't be afraid to use infographics or flowcharts or excel files, and keep in mind the level fo detail you actually need and don't try to be everything for everyone.

It's also okay to not keep docs up to date if they've served

their purpose. Sometimes the purpose in ongoing throughout development, and other times we should be comfortable with abandoning documentation.

Lastly, ‘actionable’ documentation (and I’ll get into examples in a moment) doesn’t have to replace every document you make. It replaces about half of my documentation, and the other half still tend to follow more traditional standards.



However there's some caveats…

It really depends on the size and structure of the team. A team of two people probably already do something like this, and

It can be hard to get new people up to speed if you don't have comprehensive documentation. However, comprehensive but totally out of date documentation is bad too, and that's what we normally deal with.

Some people work best with planning every detail out first before you start implementing, but actionable documentation . On the flip side, don't be afraid to write documentation for YOURSELF as the reader, so if planning on paper works great for you then keep doing it.

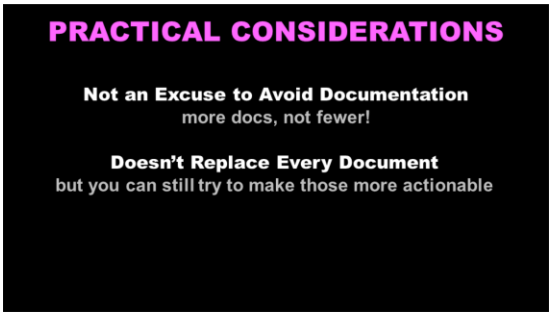
Suddenly, documentation! – if you want until someone asks for documentation, that might mean you have to drop

everything and write it immediately.

Like I said, this means more docs and not less.

And it’s not an excuse to avoid documentation. This is simply an attempt to make less wasteful documentation.

So now I am going to dive into some practical examples of docs I think were exceptionally “actionable”. I’ve made most of these, but did



However there's some caveats…

It really depends on the size and structure of the team. A team of two people probably already do something like this, and

It can be hard to get new people up to speed if you don't have comprehensive documentation. However, comprehensive but totally out of date documentation is bad too, and that's what we normally deal with.

Some people work best with planning every detail out first before you start implementing, but actionable documentation . On the flip side, don't be afraid to write documentation for YOURSELF as the reader, so if planning on paper works great for you then keep doing it.

Suddenly, documentation! – if you want until someone asks for documentation, that might mean you have to drop

everything and write it immediately.

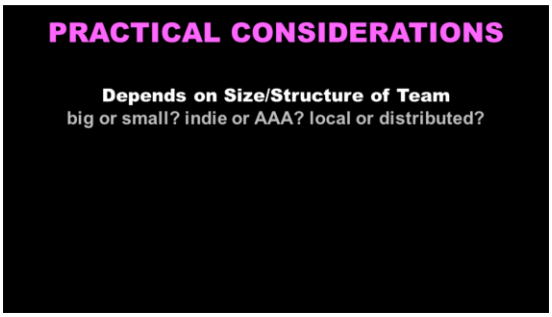
Like I said, this means more docs and not less.

And it’s not an excuse to avoid documentation. This is simply an attempt to make less wasteful documentation.

So now I am going to dive into some practical examples of docs I think were exceptionally “actionable”. I’ve made most of these, but did

Now, sometimes you still need to spec out a large system on paper and that leads you to the infamous 20+ page design doc.

A trick I use to get those more usable is to try to keep one subject per page. This is what the design doc for the object editor in the third Scribblenauts looked like. Part of my approach is the idea that if someone needs to know about a specific part of the system, I could just send them THAT PAGE and nothing else in the doc to help inform them. In a way, each page in this doc is it's own document.



However there's some caveats…

It really depends on the size and structure of the team. A team of two people probably already do something like this, and

It can be hard to get new people up to speed if you don't have comprehensive documentation. However, comprehensive but totally out of date documentation is bad too, and that's what we normally deal with.

Some people work best with planning every detail out first before you start implementing, but actionable documentation . On the flip side, don't be afraid to write documentation for YOURSELF as the reader, so if planning on paper works great for you then keep doing it.

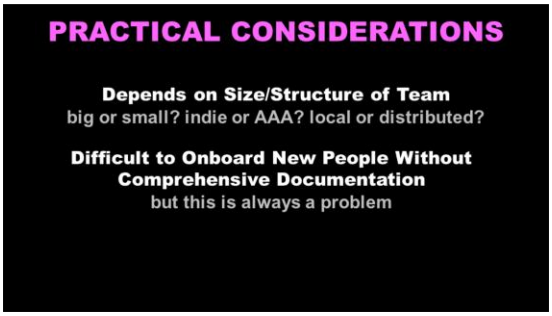
Suddenly, documentation! – if you want until someone asks for documentation, that might mean you have to drop

everything and write it immediately.

Like I said, this means more docs and not less.

And it’s not an excuse to avoid documentation. This is simply an attempt to make less wasteful documentation.

So now I am going to dive into some practical examples of docs I think were exceptionally “actionable”. I’ve made most of these, but did



However there's some caveats…

It really depends on the size and structure of the team. A team of two people probably already do something like this, and

It can be hard to get new people up to speed if you don't have comprehensive documentation. However, comprehensive but totally out of date documentation is bad too, and that's what we normally deal with.

Some people work best with planning every detail out first before you start implementing, but actionable documentation . On the flip side, don't be afraid to write documentation for YOURSELF as the reader, so if planning on paper works great for you then keep doing it.

Suddenly, documentation! – if you want until someone asks for documentation, that might mean you have to drop

everything and write it immediately.

Like I said, this means more docs and not less.

And it’s not an excuse to avoid documentation. This is simply an attempt to make less wasteful documentation.

So now I am going to dive into some practical examples of docs I think were exceptionally “actionable”. I’ve made most of these, but did

So that brings me to the end of my talk on “Make Actionable Documentation”.

Thank you Liz!

The one type of document Liz didn’t go into is the one you write to impress the publisher.

Ah, to live in a world where those weren’t necessary.

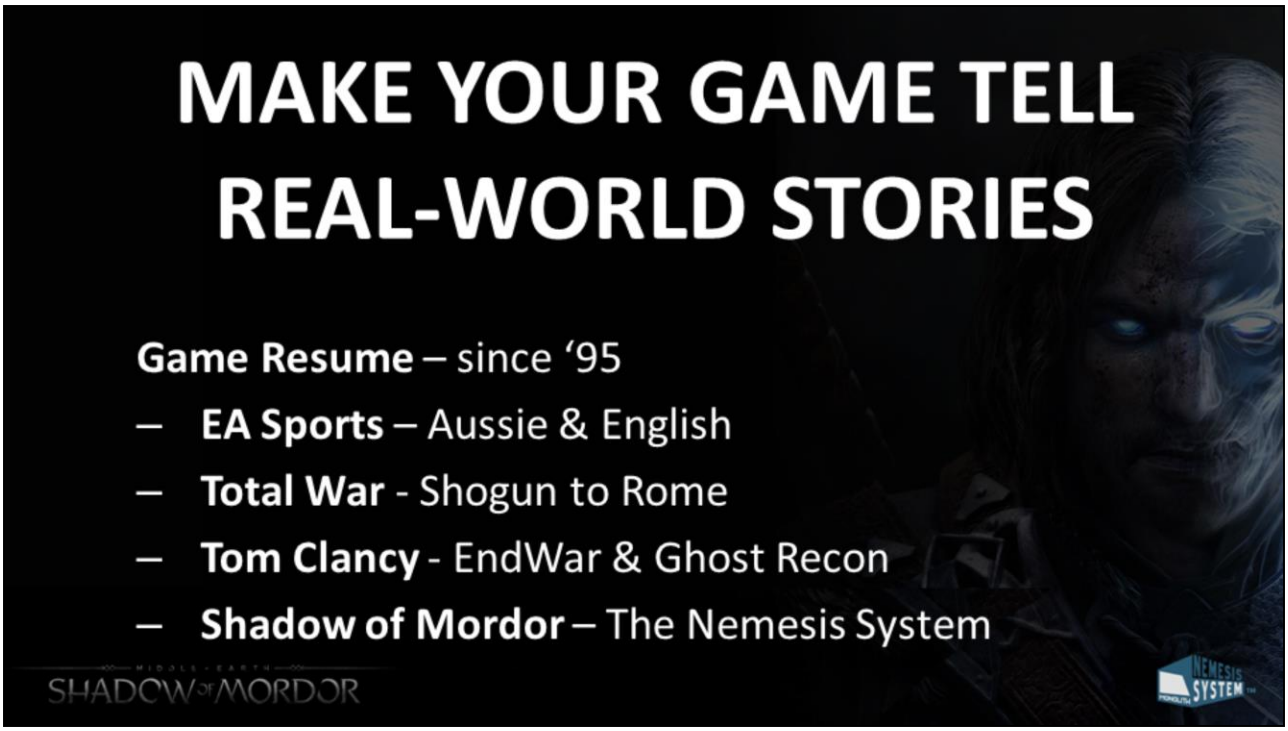
And finally, our last speaker is here to end us on a subject dear to my heart.

Often we think about game systems as intuitive or sticky or challenging or good for flow.

But what about systems that produce humanity?

Our next speaker has worked on everything from sports games to Total War to 2014's game of the year Shadow or Mordor…

Michael de Plater!



Todays talk is going to be about what connects all of these games I've worked on over the last 20 years. They are all systems which produce stories.

These three may seem very different genres, but the underlying designs have a lot in common in terms of how they handle conflict, drama, time and the nesting of the personal within the epic.

There's a reason they all have Movie Genres and TV Tropes pages

SPORT SEASONS

The Sports Season as Interactive Narrative

- Time always moves forward
- Failure creates drama
- Escalation to a dramatic climax
- Neither linear or branching narrative

MIDDLE-EARTH SHADOW OF MORDOR

NEMESIS SYSTEM

Sports Seasons ARE Game Design – and you can see examples of different Designs across different Sports and Cultures.

Including of course E-Sports

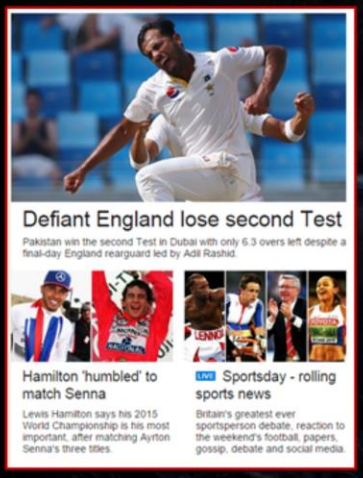
Confession and Caveat – I don't really like Sports. I've always just looking at them from the perspective of a Game Designer.

SPORT SYSTEMS

Sports Systems dynamically generate Personal Narratives

- Champions
- Heroes & Villains
- Underdogs
- Grudges & Revenge
- Rematches

Within a structure of Epic Conflict



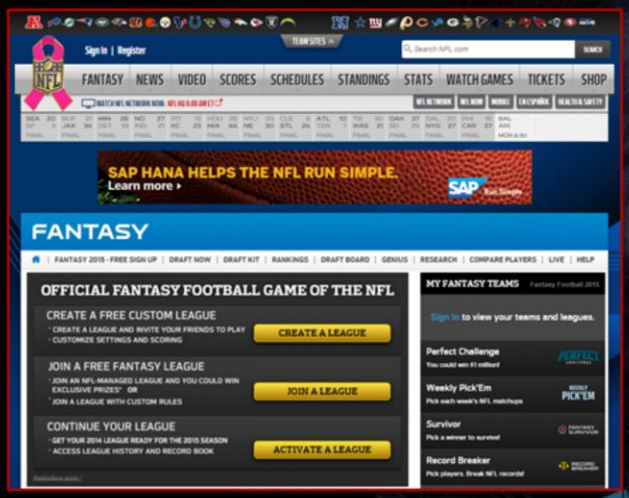
As well as high level narrative structures Sports System Design also generates personal / individual stories implicitly through the structure of its design - Daily.

[Will update this to the most recent Sports News]

Heroes and Villains emerge implicitly out of the Design of Winners, Losers, Statistics, Injuries, Grudges

CREATING STORIES FROM DATA

Sports Systems even generate narratives from Statistical Analysis



SHADOW of MORDOR

NEMESIS SYSTEM

Statistics can create Stories within the System as well as Meta-Narratives outside of the Game – via commentators and players.

People may follow these stories without even particularly caring for the sports or even watching them.

War scenarios are similar to sports in many structural ways.

Time always moves forward to create dramatic situations – no "restarting" the moment something goes wrong.

Just as you may lose the play but win the game, you may lose the battles but winning the war, escalating towards an inevitable climax

WAR STORIES

Death & Drama

- Inheritance – Traits in Rome: Total War

Specific example of embracing Failure but moving forward – in Rome: Total War characters would die (similar to injuries in Sports) but leave heirs who would inherit Traits.

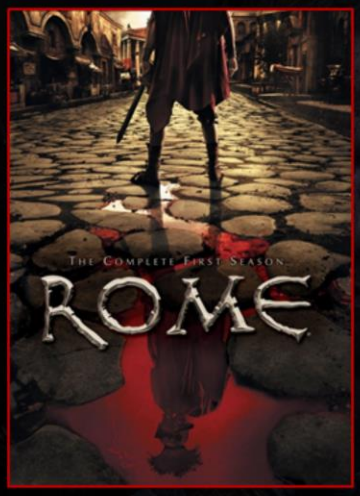
Creates strong attachment to characters and emergent drama.

And incidentally, looking forward, the Nemesis System deals with this in a different way through Cheating Death and Scars – literally writing stories on the flesh of the characters.

WAR STORIES

Civil Wars in **Rome: Total War**

- Balancing as a Story Mechanic



MIDDLE-EARTH SHADOW OF MORDOR

NEMESIS SYSTEM

Often in Strategy Games the late game is dull because beyond a tipping point you know who's going to win and the rest is mopping up

We dealt with this in Rome: Total War with the Civil War Mechanic – once the Roman Faction is close to winning it splits up and becomes multiple factions – generating drama and a climatic end game.

And this system was derived from the real world reference.

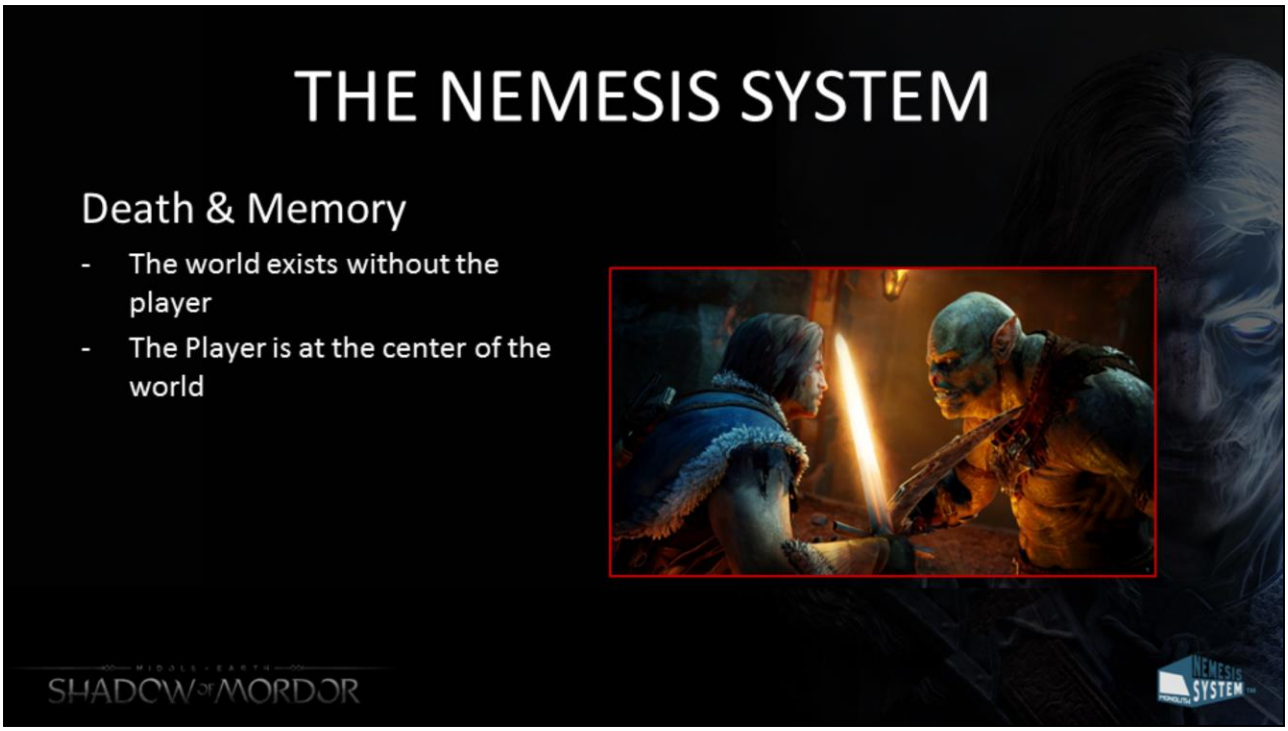
Though it may be not obvious at first, I hope see now how this built to Shadow or Mordor's Nemesis System.

Same ingredients as Sports and War – but focused on the Personal, which is appropriate for an Action Game. The ideas are similar but the scale is different.

Once again there is an emergent procedural narrative that's neither linear nor branching. Player has tremendous choice and personal investment in a story that is "theirs" yet we're not building bespoke "branches".

Humans are hard-wired to understand shifting social structures and hierarchies – a great deal of human storytelling and drama is based on this model. Makes it a great mechanic for generating stories.

The Nemesis System enabled Players to intervene in a Social Hierarchy through Death (their own death moved up their enemies), Revenge and ultimately Domination and Power – players loved being able to mess with social structures.



Balancing two superficially conflicting ideas is key

Note this is very similar to Play by Play Commentary in Sports Games – but again the difference / emphasis is on making it Personal and embedding it into the dynamic narrative.

I spoke about my experience across 3 genres – Sports, Wargames and Action-Adventure.

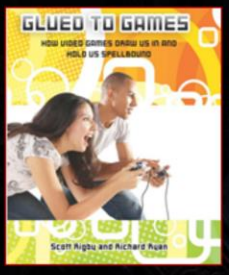
But these ideas can be applied across many genres yet explored.

For instance, think about how all of these ideas are implicit to Multiplayer Experiences and there are huge opportunities to narrativise those systems – like the fertile grounds of Minecraft & Day-Z – where the stories could be made strong while still keeping the game systems-driven and open-ended.

REFERENCES

Highly recommended for thinking about Stories as Systems

- **Playing at the World** – The History of D&D
- **Play Unsafe** – Improve Techniques in Role Playing
- **Glued to Games** – Player Engagement through Needs Satisfaction




20
MONOLITH
YEARS
SHADOW OF MORDOR
MAKE YOUR GAME TELL
REAL-WORLD STORIES
Michael de Plater
www.lith.com
Monolith is hiring!
NEMESIS SYSTEM

Thanks

Thank you Mike!

That's our final session – now that we've seen all these, think about how you may want to add these rules to your own deck of design rules

Like building your own personal Magic deck, the cards you choose to use will effect how you develop games.

Different decks will work better or worse in different situations.

But most important is that your deck of game design rules feels good to you and is one that you enjoy developing with.

That's a wrap!

Slides are up right now on my web site if you want to go check them out – in the writings section!

We're not going to do Q&A but we'll be hanging around up here for as long as they'll let us if you want to come ask us some questions.

Please remember to fill out your surveys. The comments are really useful for helping us know how we did and what you'd like to see in the future.

Thanks to all the speakers and thanks to everyone for coming!