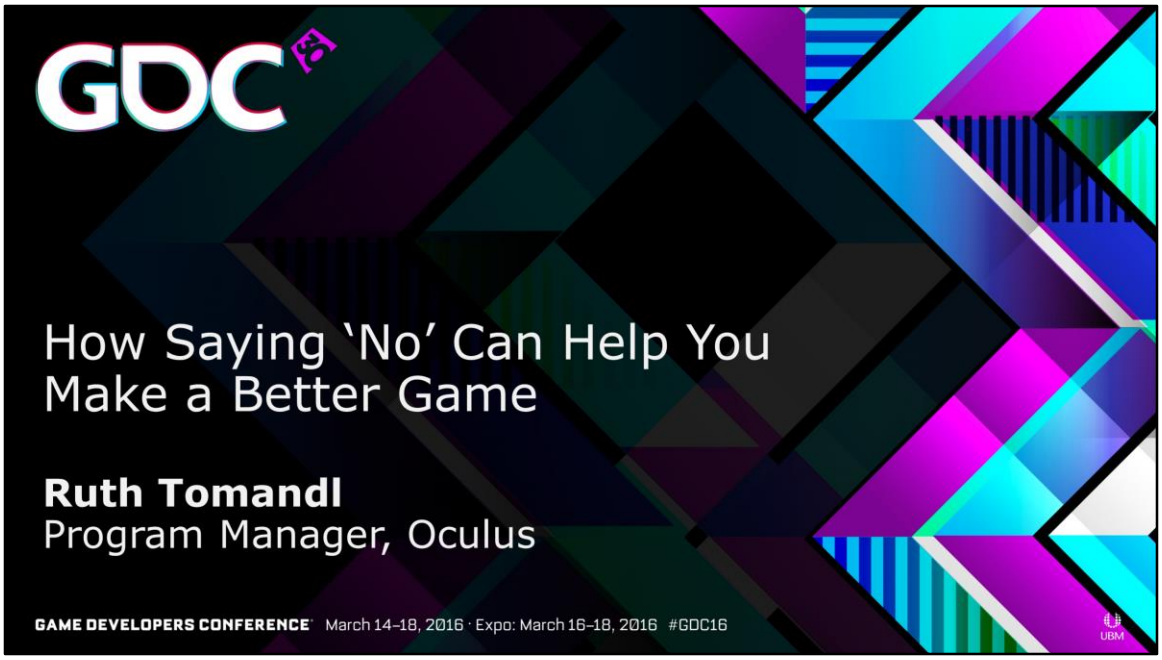


Good Morning! Silence phones, fill out surveys, email me instead of taking pictures of the slides.

I'm Ruth Tomandl, and today I'm going to go over: how saying 'no' can help you make a better game, some tools for saying 'no' the right way, and the producer's role in saying 'no'.

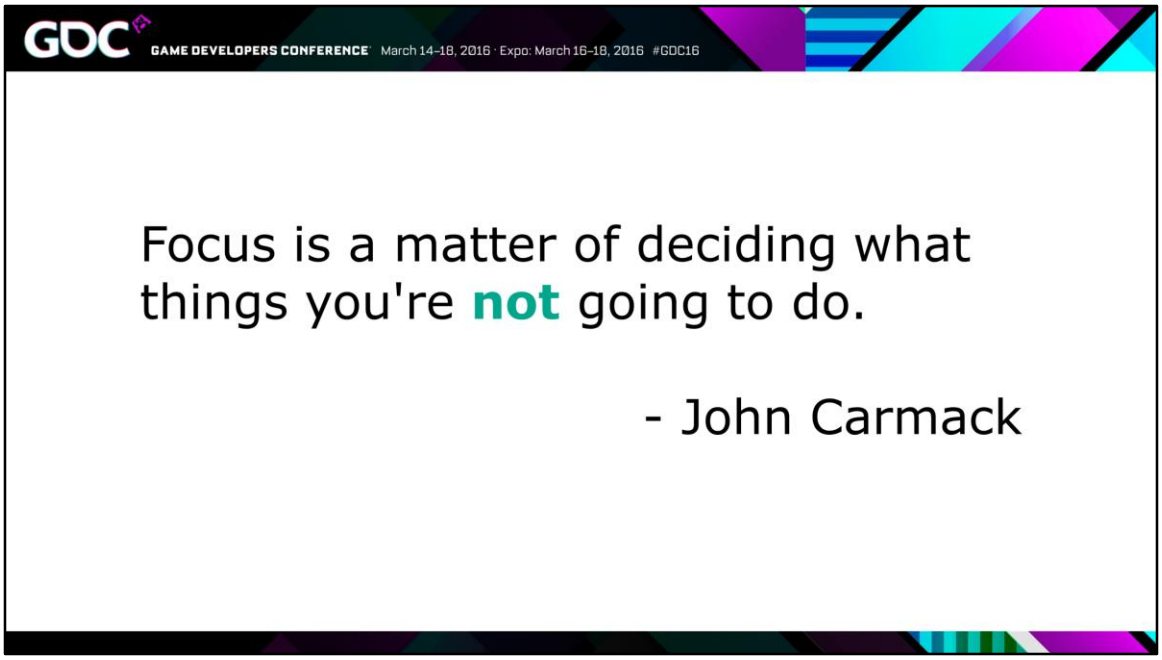
First, I'd like to know: how many of you are currently a producer on a video game?

How many of you want to get into production?

How many of you are game developers but not producers?

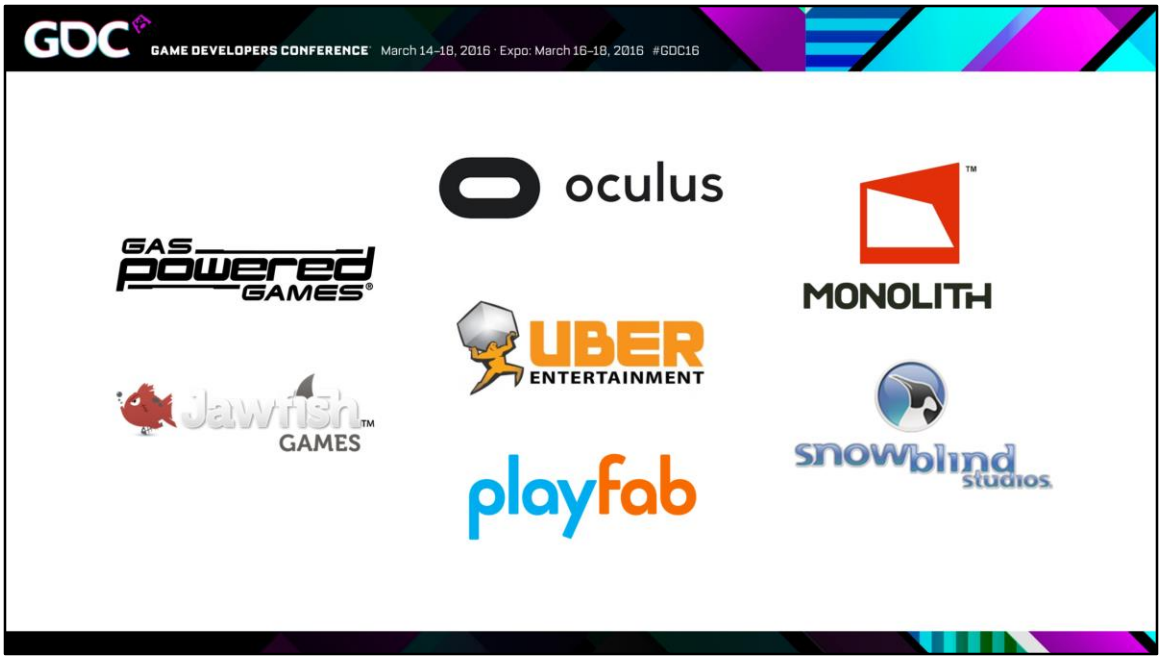
Students?

Thanks!



John Carmack says that Focus is a matter of deciding what things you're not going to do.

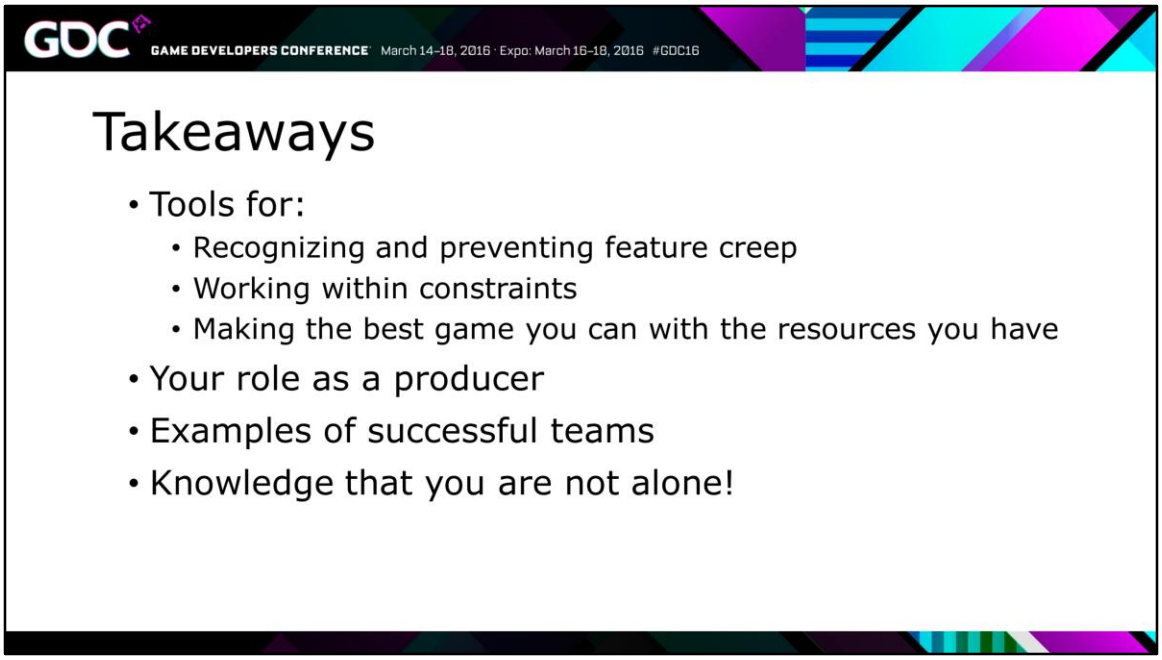
All dev teams are different, but almost all of them try to do more than is possible: either because they have a lot of great feature ideas, or because their stakeholders (that is, their publishers, their lead designers, their fans, or they themselves) ask for too much.



I've been a designer and producer on a lot of teams, and I've repeatedly seen that even teams that are great at coming up with a coherent, focused plan are rarely able to avoid doing work that's not on that plan. The studios that survive are the ones that learn how to say no to features or publisher requirements that they won't be able to get done.

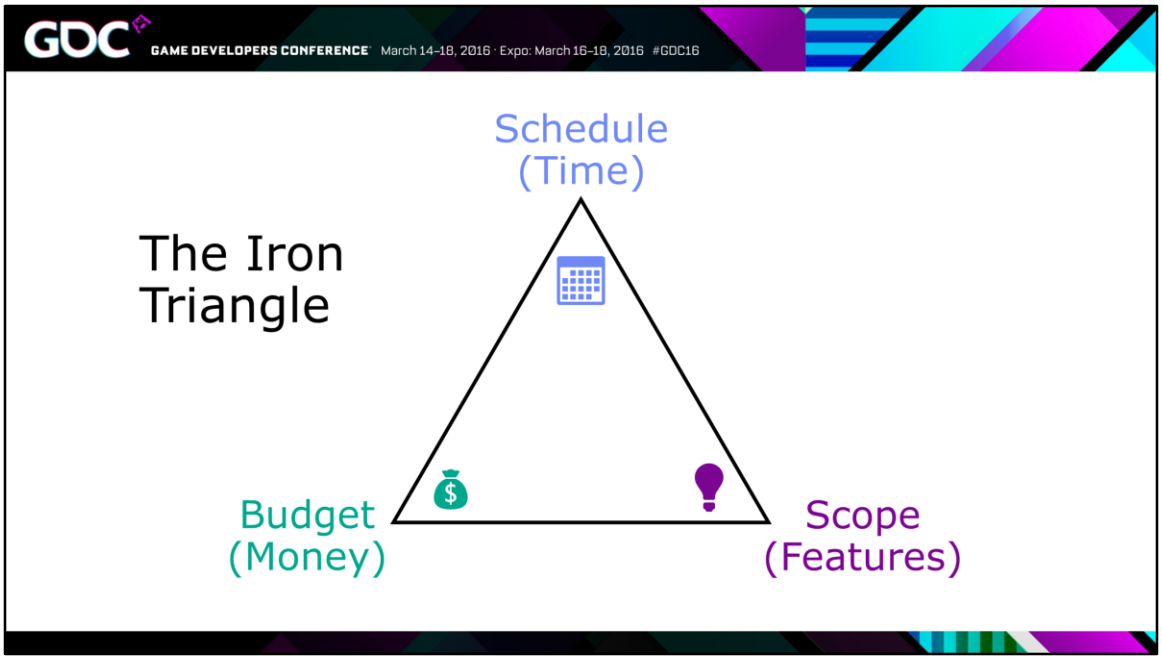
How many of you have shipped at least one game?
How many of you shipped a game that had all your good ideas in it?

Everyone cuts ideas; you have to if you're going to ship. The question is: how do you make the right cuts at the right time?



Here's what I hope you'll get out of this talk:

-Tools for recognizing and preventing feature creep, working within constraints, and making the best game you can with the resources you have
-Ways to manage the challenges of your role as a producer while helping your team finish what they start
-Examples of successful teams that stayed focused on their game's core features, and what they did to make that possible
-Knowledge that you are not alone! Other teams have faced the same problems you're facing right now, solved them, and shipped amazing games

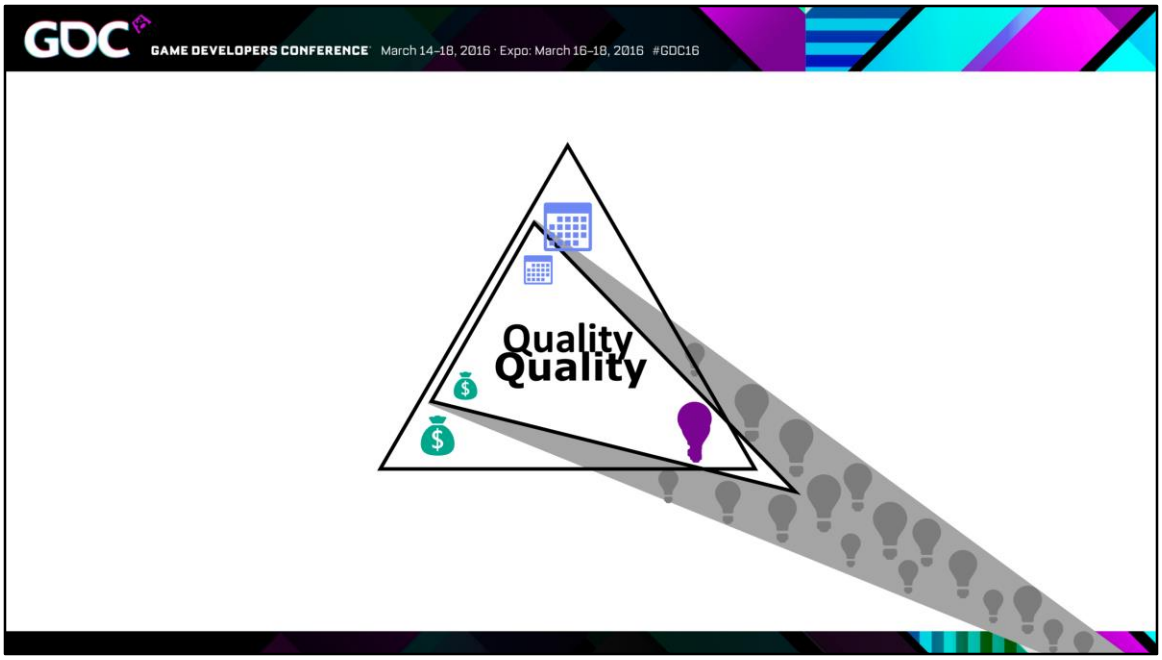


This is the iron triangle. How many of you have seen this before?

Most devs are familiar with it as “cheap, fast, or good: pick two”. But there’s another way to look at it.

Each corner of this triangle represents a constraint on one of your resources. Schedule is a constraint on how much time you have. Budget is a constraint on how much money you have. And Scope is a constraint on how many features you add to your game.

Time and money are pretty obvious; they’re limited resources; sometimes extremely limited. There’s not much you can do about that, most of the time. But scope is something you can definitely do something about.



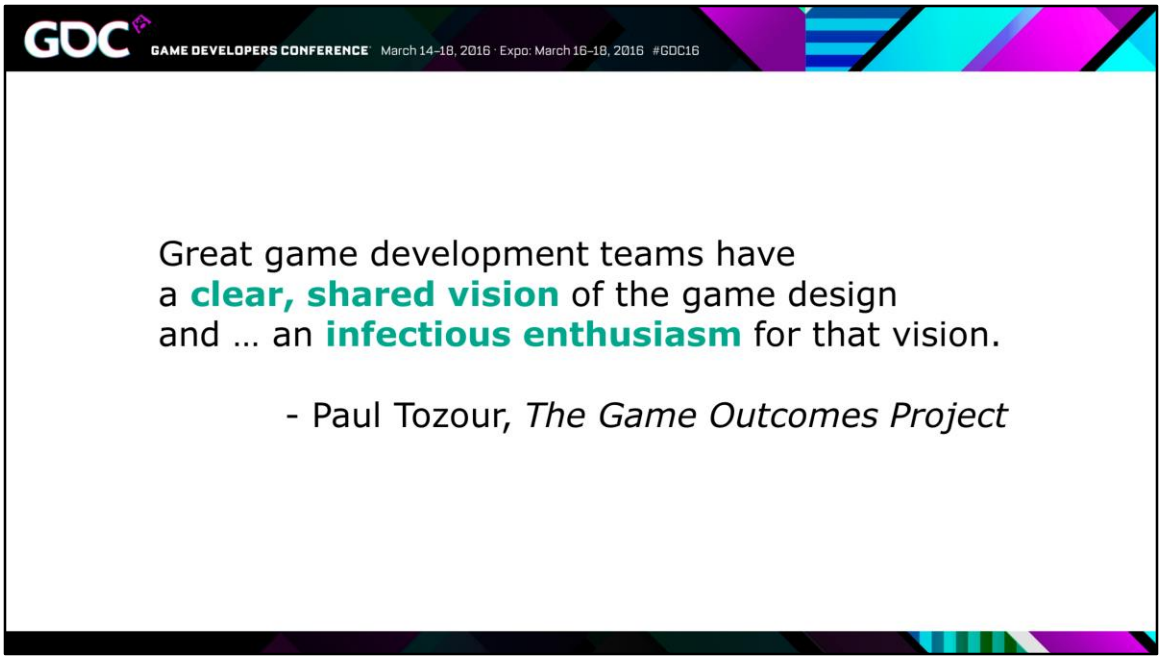
You can think of the area of the iron triangle as the overall potential quality of your game. The farther out any of those corners is from the center, the higher quality your game can be.

The problem is, when you're making a game, your schedule and budget probably look more like this.

So let's focus on that third corner: scope. That is, your game's feature set, or the design vision for your game. If this is really good, your game can be great even with a tight schedule and budget.

A lot of game developers, especially new developers, will try to add as many features as possible in the belief that it will make their game better. Push that corner out as far as it can go! The problem is that if your time and money are limited, you will not be able to finish all of those features. It's just not possible in a universe governed by the laws of physics. And if you ship a game with a bunch of unfinished features, your quality will be lower, not higher. Only finished features count for improving your game's quality.

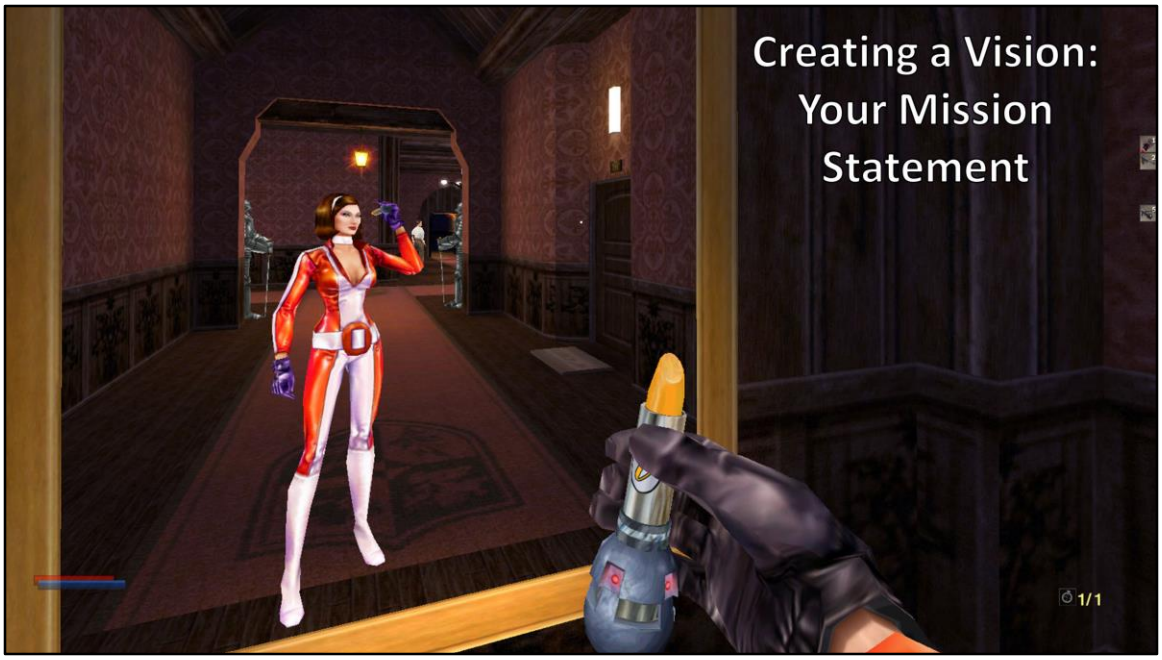
That means that you need to make sure the features you're choosing to spend your limited resources on are the right ones for your game. To maximize that corner of the triangle, you need to focus, not add.



This is from the Game Outcomes Project [http://intelligenceengine.blogspot.com/2014/12/the-game-outcomes-project-part-1-best.html], which was published about a year and a half ago. Paul Tozour and his team surveyed hundreds of game developers asking about their development process, teamwork, and culture, and comparing those responses to the success of the teams' finished games. The results are extremely interesting and often surprising, but their strongest result won't surprise you: teams that had a "viable, compelling, clear, and well-communicated shared vision" were the most likely to make a successful game: critically successful, financially successful, and meeting or exceeding the team's own expectations for what they were making.

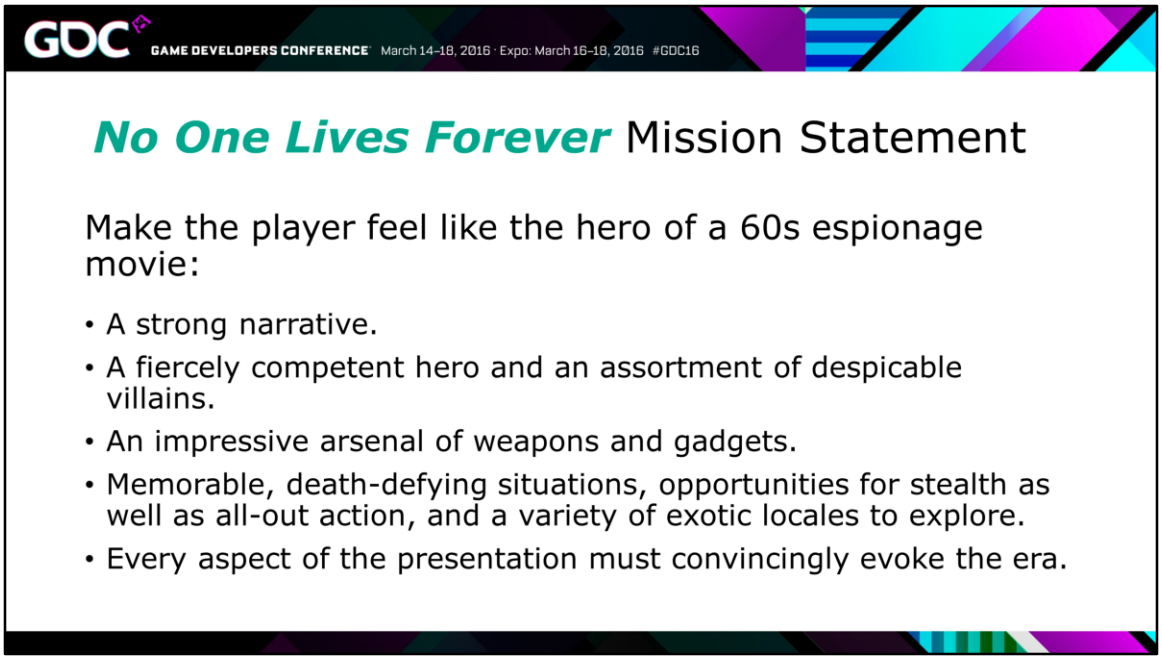
If you only take one thing away from this talk, it's this: have a clear design vision for your game, and make sure your whole team believes in it.

Let's talk about some ways you can create that focused vision, and how you can communicate it to your team in a way that creates that infectious enthusiasm.



If you have strong design direction for your game, you can distill your design down into a clear, compelling mission statement.

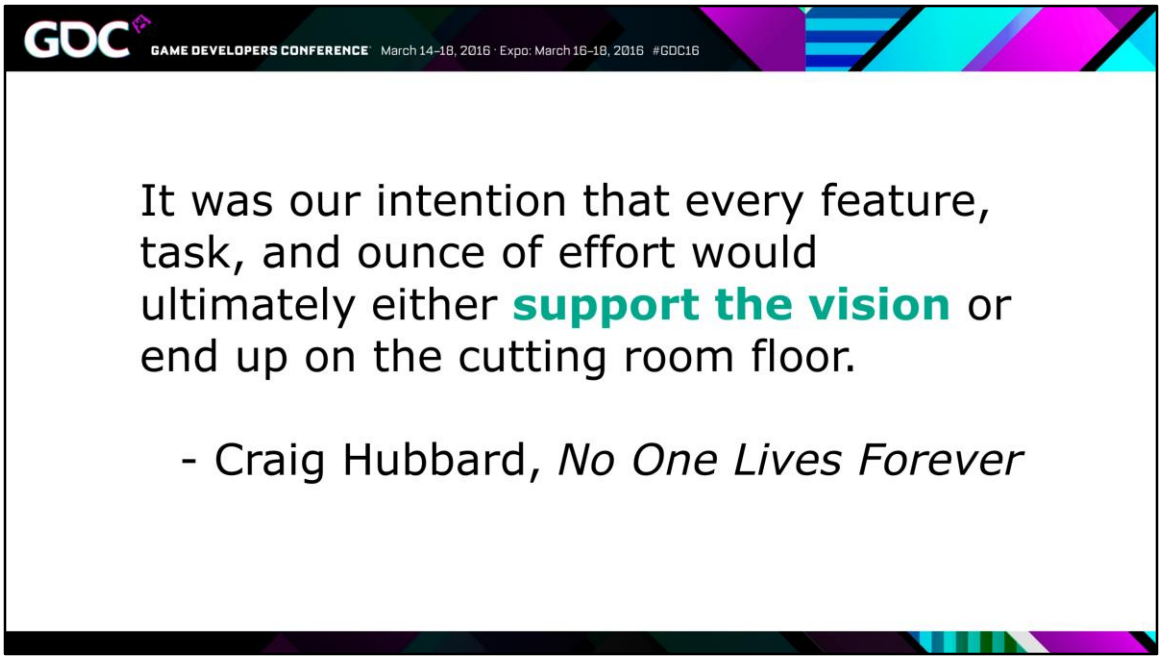
The No One Lives Forever development team had a concise mission statement in place when their publishing contract was signed, and they used it to guide their work throughout the game's development.



No One Lives Forever: [http://www.gamasutra.com/view/feature/3069/postmortem_monoliths_no_one_.php]
Our primary aim was to make the player feel like the hero of a 60s action/adventure/espionage movie.

- The game must have a strong narrative, with twists and turns in the spirit of *Charade* or *Where Eagles Dare*.
- It must feature a fiercely competent hero and an assortment of despicable villains.
- The hero must have access to an impressive arsenal of weapons and gadgets worthy of *Our Man Flint*, *Danger: Diabolik*, or *Get Smart*.
- There must be memorable, death-defying situations, opportunities for stealth as well as all-out action, and a variety of exotic locales to explore.
- Finally, every aspect of the presentation must convincingly evoke the era.

This covers a lot of ground; you could see how this mission statement could help a development team define everything in the game: character design, gun sound effects, level design tools, or lighting tech. It gives the team clear guidelines on how to prioritize their work. And again, it gave them a way to determine when to say no to a new idea that doesn't fit their design vision.

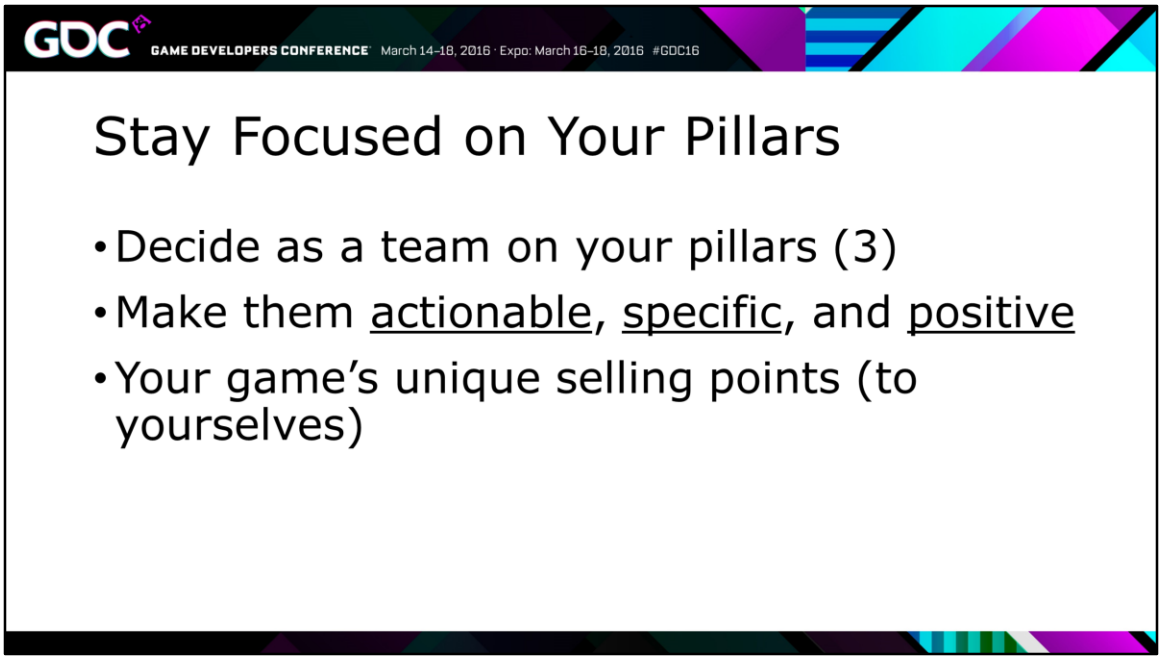


Limits on what you're able to do will always exist, so make sure that anything you are spending time on directly supports the things that make your game great.

[http://www.gamasutra.com/view/feature/3069/postmortem_monoliths_no_one_.php]

A mission statement is great if you have a strong design team or design director, and a really clear idea of the game you're making. But what if your game idea is more vague?

Pillars are a way to both identify and to communicate what your game vision is. Here's what they are and how to use them:



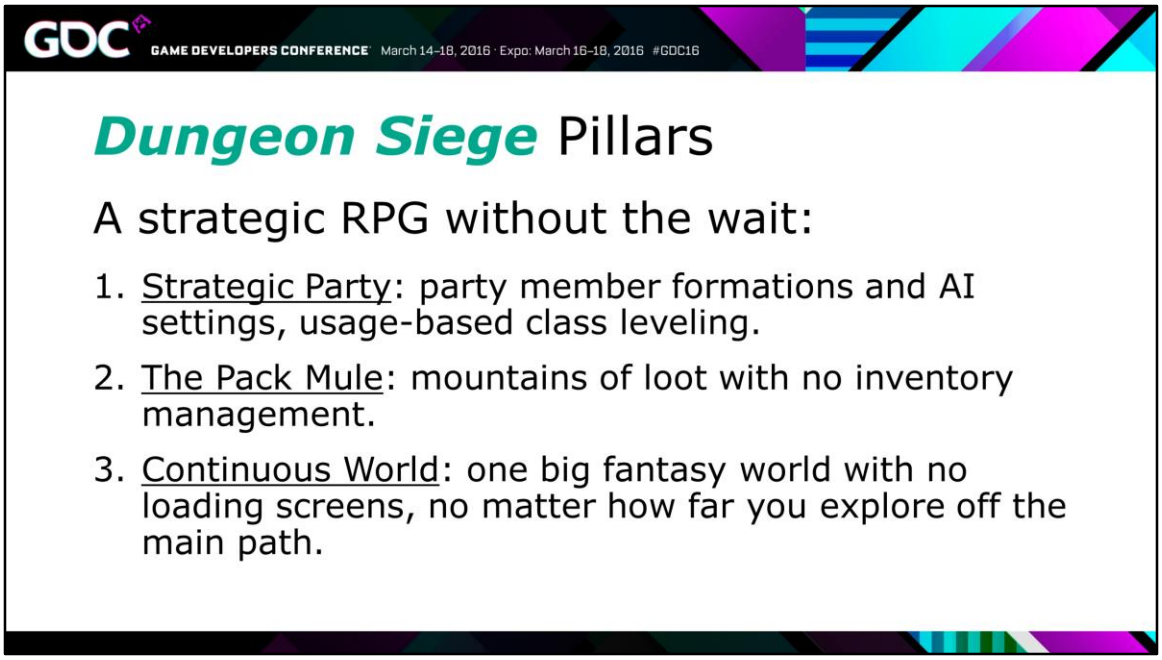
Pillars are major features, selling points, or unique aspects of your game that can be a strong foundation to build on. They are the core features that will make or break your game.

First, before you start development, or at least very early in development, decide as a team on three pillars for your game. (Three is a good number because it will require some discussion to get your team to agree on three, and any more than that can be hard to remember or can split your focus.)

Good pillars are actionable, specific, and positive. They should cover large areas of the project and make it easy for developers to know whether their work is aligning to the goals of the project.

Pillars are often very similar to a game's unique selling points, or marketing points. Which makes sense, because in a way pillars are how your team markets your game to yourselves. Like the mission statement, pillars are a way for your team to have context for what they're working on and how it supports the game vision.

Of all the games I've worked on, the ones that have strong pillars from the start were the ones where the development teams did their best work.



The first game I ever worked on was Dungeon Siege, and though we didn't have a specific enough idea of the design of the game to have a detailed mission statement, we did know the main selling points of the game we were making: strategic party-based gameplay, painless loot management, and a giant continuous world with no loading screens. Dungeon Siege 1 came out 14 years ago, and I still have those three pillars memorized.

These were great pillars; they got players and the development team excited about the game, they clearly illustrated what the game would be like to play, and they gave the team good guidelines for prioritizing their work. They were vaguer than a full mission statement, which left us with some room to try some different things as we built the game.

As a level designer I found these pillars really useful for my work, especially #3; it gave me a clear, strong idea of what kind of game we were building, and what to prioritize. For example, a smooth transition into a dungeon from the overland world was a really high priority, and we spent a lot of time making sure each dungeon entrance and exit was seamless. The game's story, on the other hand, was not as high of a priority (which was reflected in the reviews).

A developer or a player looking at this list of pillars will know what kind of game they're getting into: as a developer, I knew what I was building, and players knew what they were buying.

Pillars are a great tool, so spend time on them, and work to stick to them!

Here's an example of what happens when you don't do that:

Diablo III, during its development, had 7 design pillars.

Now there are two problems with these: first, some of them are extremely vague. Especially 'strong setting' and 'highly replayable'; neither of those are much more specific than 'the game is good'. As a developer, these wouldn't help me with my day-to-day decision making, and as a player they don't make me excited about the game.

Second, when Diablo 3 was released, the game actually didn't do a very good job of expressing the pillars here that are specific and actionable, as game director Jay Wilson discusses in his GDC 2013 talk on the game's design. In the original game, the loot progression was poorly paced, and character abilities weren't very well differentiated. And there's something else: I don't see anything about a real-money auction house in this list; in fact, that feature directly conflicts with at least two of these pillars. Yet it took up a huge amount of development time and effort, wasn't ready when the game launched, was strongly disliked by players, and was ultimately shut down.

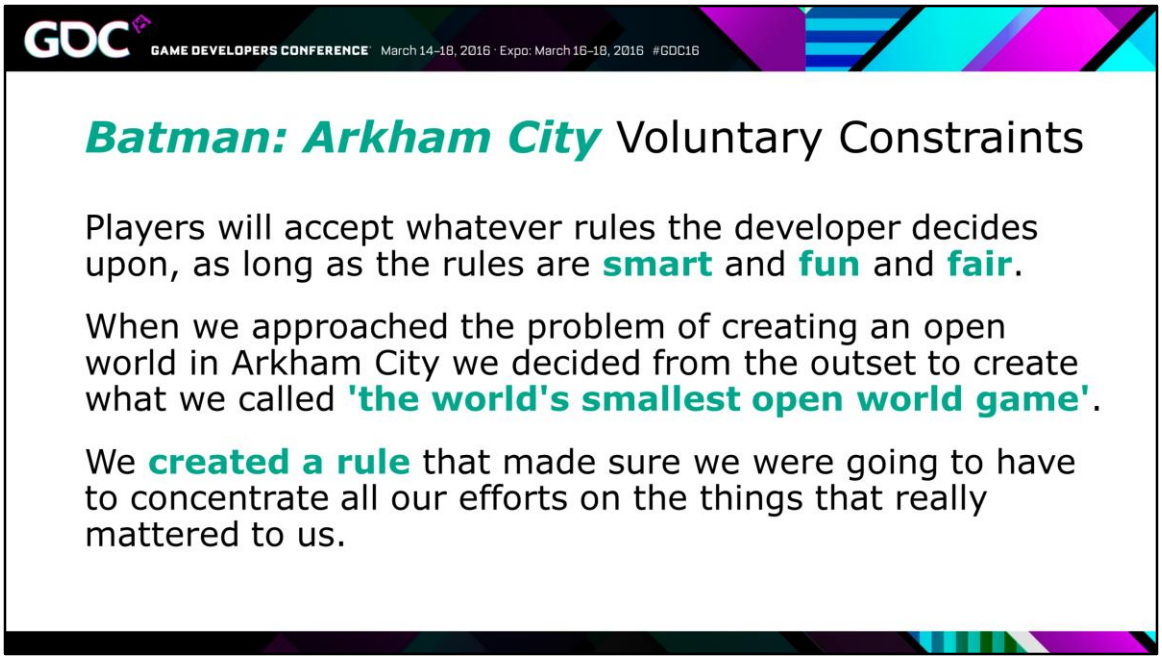
Now, I'm not saying any of this to criticize Diablo III; the game was successful by many metrics, even though the team didn't stay as well-focused on their design vision as they wanted to. I'm bringing this up because: even Blizzard, arguably the most successful development studio ever, screws this up sometimes. Creating a solid vision and then staying focused on it is really difficult, even for the experts.

[http://www.gamasutra.com/view/feature/170258/the_devils_workshop_an_interview_.php?page=1]
[http://www.gdcvault.com/play/1017813/Shout-at-the-Devil-The]

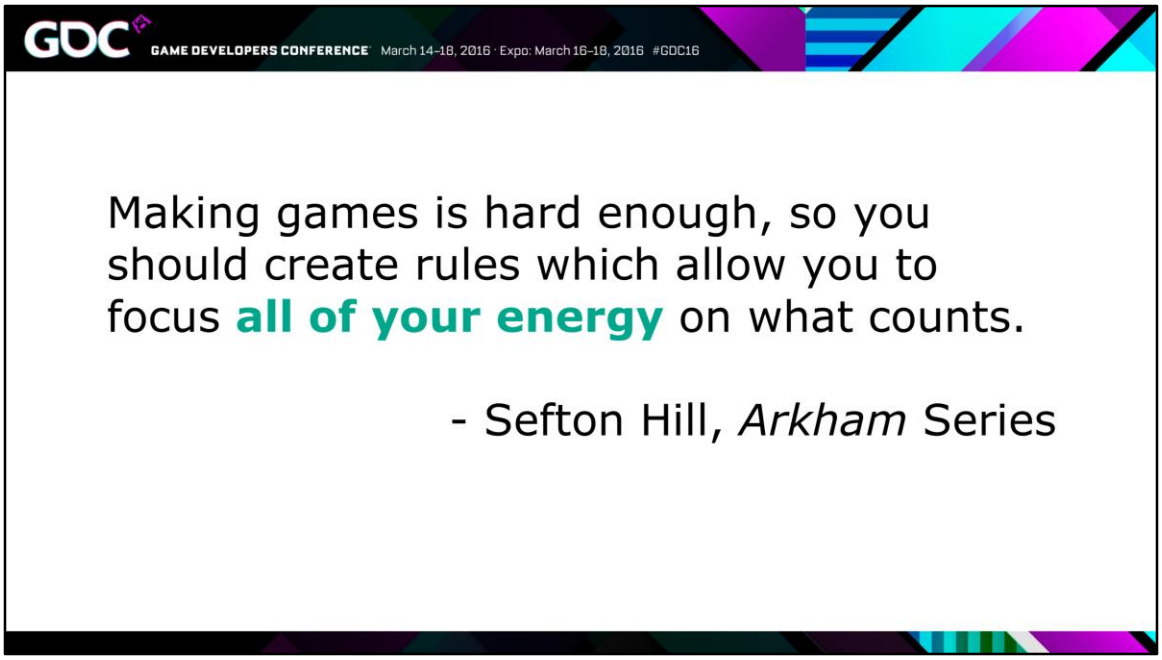
So here's a tool that can make it much more doable: voluntarily set constraints on what you're making. Limit your focus and your efforts to what you know will make your game great.

When the Rocksteady development team started work on Batman: Arkham City, they knew that they'd not only need to increase the scope of the game from Batman: Arkham Asylum, but also that they'd need to keep the design constrained to something their team could build to the same level of quality.

Here's how Sefton Hill, the Game Director for the Arkham series, described their decision:

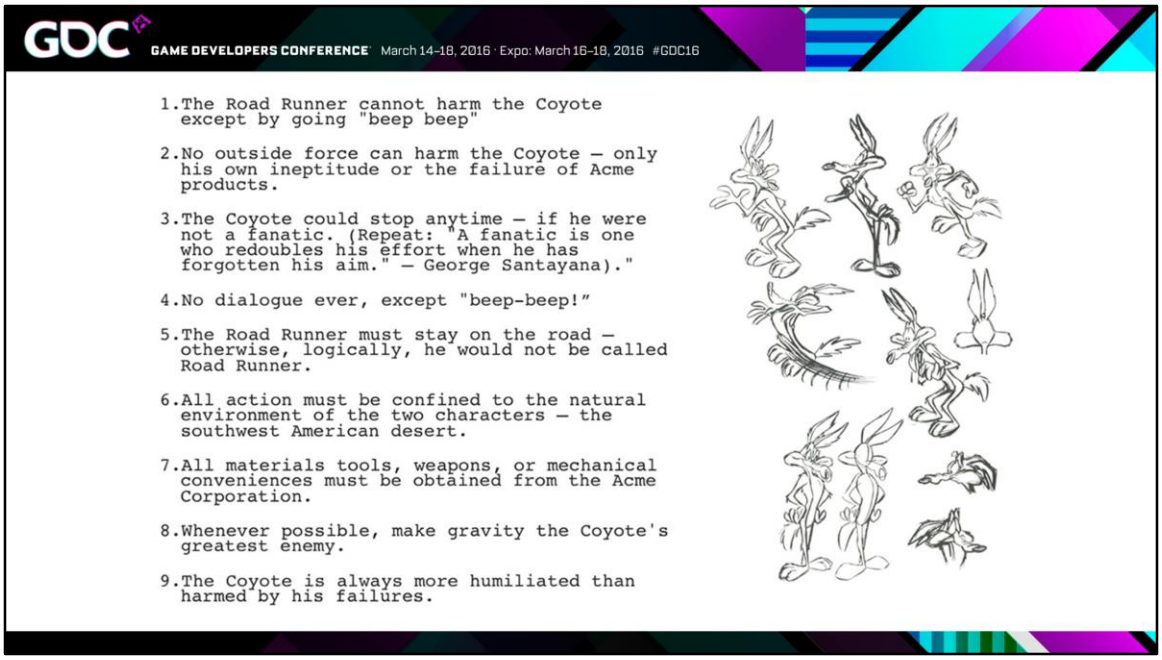


"The world's smallest open world game" is a great constraint. You could see how this would help a development team make smart decisions: everything from enemy density, to level size, to how Batman travels through the game world, is guided by this rule. It allowed the team to create a world that felt as dense and well-crafted as the levels in Arkham Asylum, but that felt much broader in scope and open to the player's decisions. Also very importantly, it empowered the team to say 'no' to ideas that didn't fit within this voluntary constraint.



If you have strong design rules that your whole team believes in, you can make sure that features that don't support the vision don't get worked on. You don't have to spend any of your precious, constrained resources on work outside of your vision.

[http://www.gamasutra.com/view/news/129578/DICE_2012_The_five_keys_to_Rocksteadys_Batman_success.php]

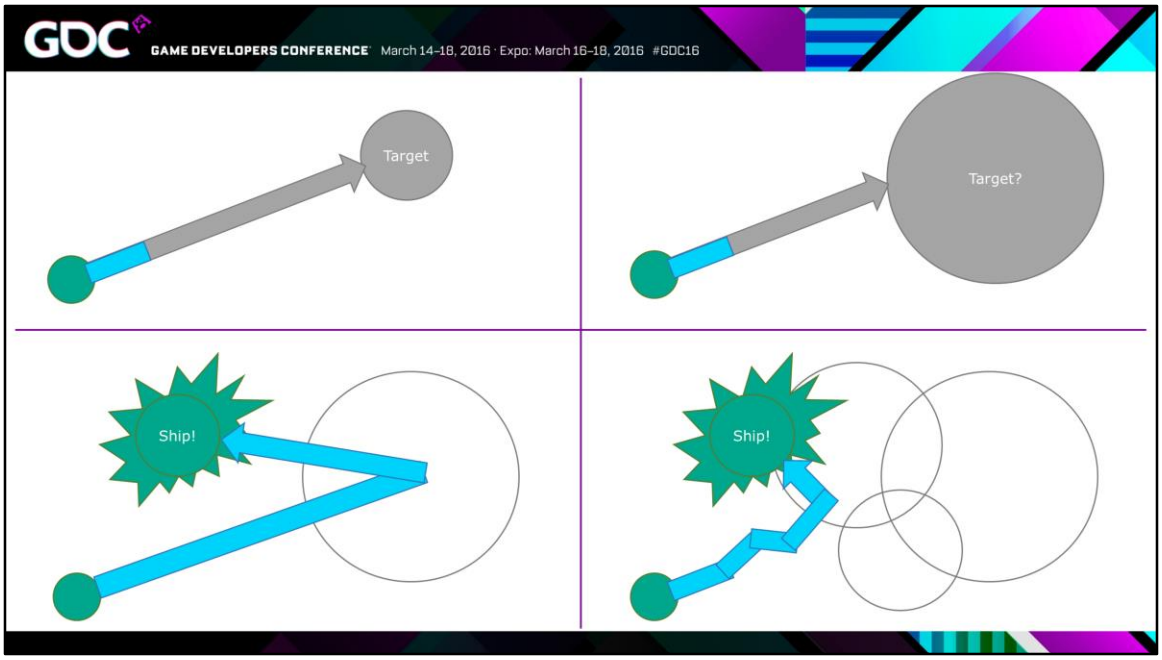


Here's another example: this list of Chuck Jones' Road Runner cartoon rules is a well-known example of intentional creative constraints; it's easy to see how this list of rules would make it really easy to say 'no' to ideas that don't fit the design vision. Chuck Jones was a strong creative director with a clear vision, and he's clearly communicating his vision to his team with these rules, in a way that makes it easy for them to evaluate whether their own work and ideas are a good fit.

Creating a Vision: Agile Exploration

But what if you don't know what your game should be yet? What if you just have a really vague idea of a direction, but you know it'll need a lot of exploration before you're able to find that clear vision? Don't worry; there are good ways to do that.

Octodad is a good example of this process. The team of student developers started with a basic idea for their game and then did a lot of prototyping, early playtesting, and major iterations to their design. Because their game was so unique, they didn't have a lot of other games to compare their design to; the possibility space was very large. Especially when you're building something very innovative, an agile exploration approach to your game can save you a lot of wasted work, and get you to a good game a lot faster.

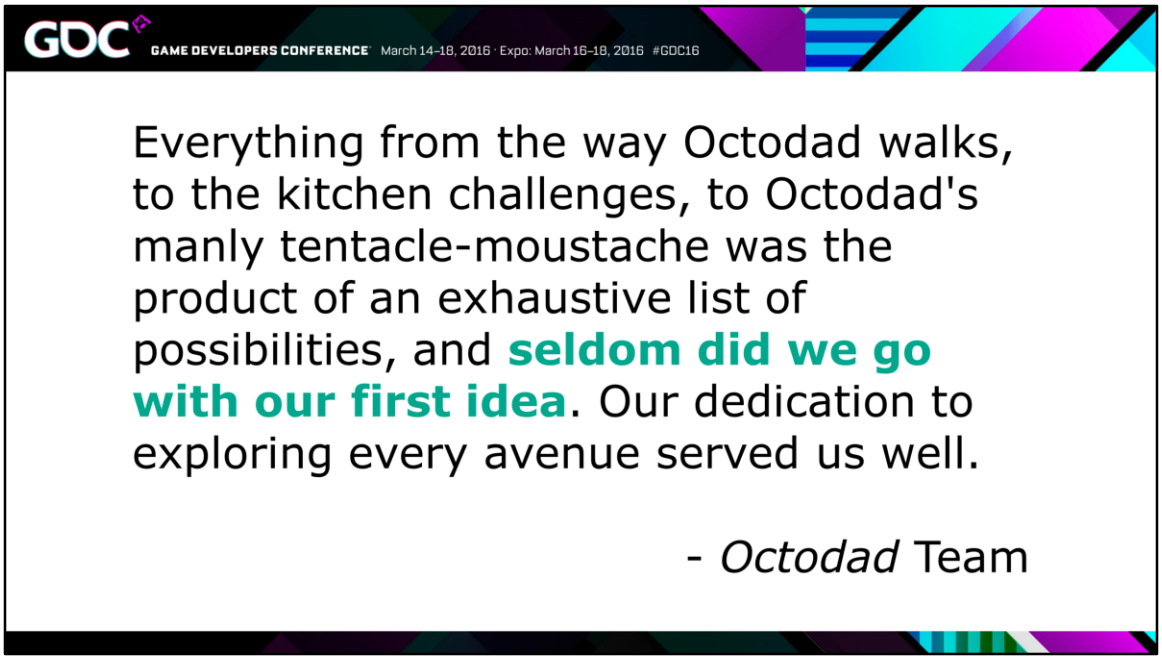


This is how games are traditionally designed: you start with a design document, spend a bunch of time building that, reach version 1, and at some point (unless you're working on a slam-dunk sequel), you realize that you need to make some big changes. So you make those changes to your design, do a bunch of work to get your game done under the new design, and ship.

More developers are starting to use a more agile approach; earlier prototyping and playtesting, earlier external input, and more frequent design changes. It means less work overall, with more of that work being spent on the right features, but it also means that you don't have a clear vision at the start of your project.

You can start with a vague idea of what you want to make if you're willing and able to spend the time to iterate on it and find your focus. If you can see that your game design has the skeleton of a good idea but you're not sure how to fill it in, give yourself the time and resources to try some things and discover what your game vision should be.

But it's risky; make sure that creating the core vision for your game is your top priority at all times, and that you don't get too distracted by cool ideas that don't get you closer to a finished game. Once you have your game vision, you'll still have plenty of work to do to ship the finished game.

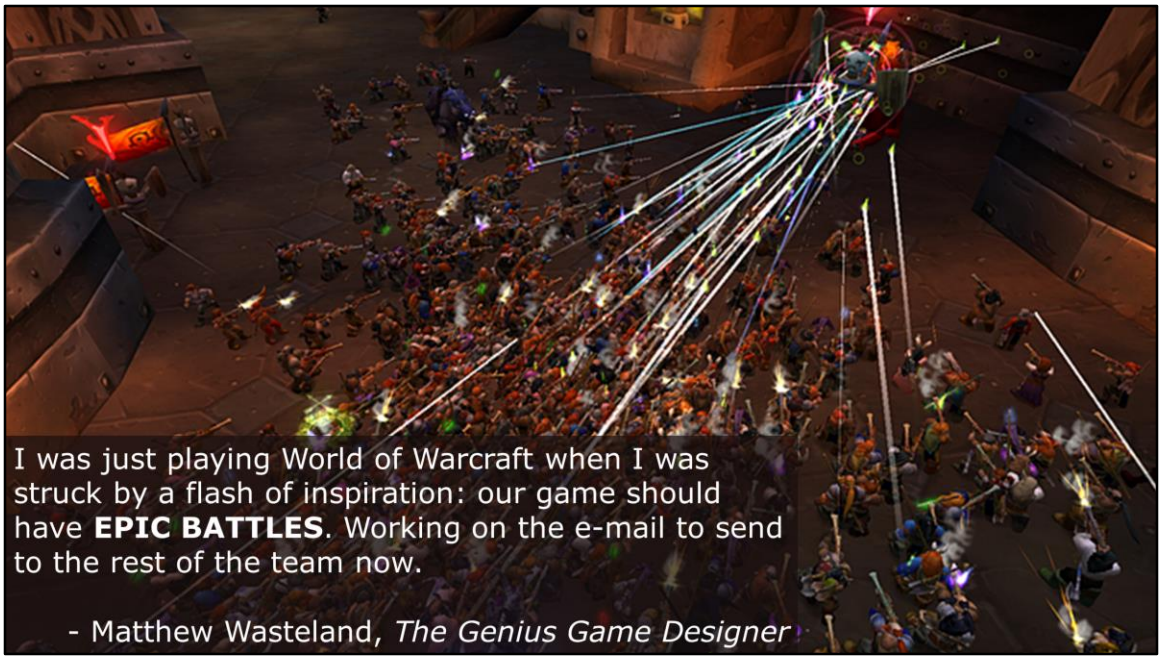


[http://www.gamecareerguide.com/features/1003/postmortem_.php?page=2]

The Octodad postmortem on Game Career Guide [linked in the notes] is a good read if you're planning on agile development: the Octodad Team represents the best and worst of agile design. They playtested early and iterated on their core idea, they shipped something very unusual and attention-grabbing, but they also got distracted coming up with more features than they could finish (and playing Minecraft) and ended up cutting big chunks of their game.

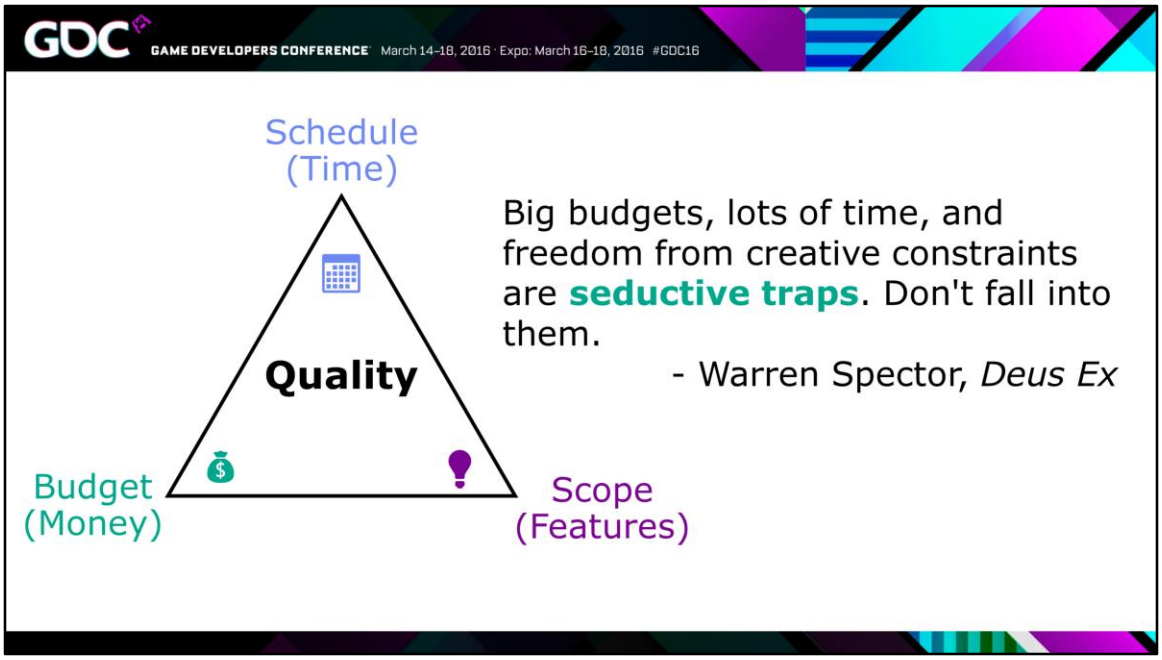
So you have a good design vision for your game, or you at least have a plan for how to develop one. Now you have to protect that vision by managing your scope.

Because sometimes your lead designer is this guy.



I think we've all worked with this guy. His heart is in the right place, but this is why scope management is so important. Inspiration is everywhere. Your team is made up of a bunch of smart people who decided to be game developers on purpose: they're never going to run out of ideas. Ideas are an infinite resource. But finishing Ideas requires resources that are not infinite.

[http://www.gamasutra.com/view/news/200153/Beware_the_Genius_Game_Designer.php]

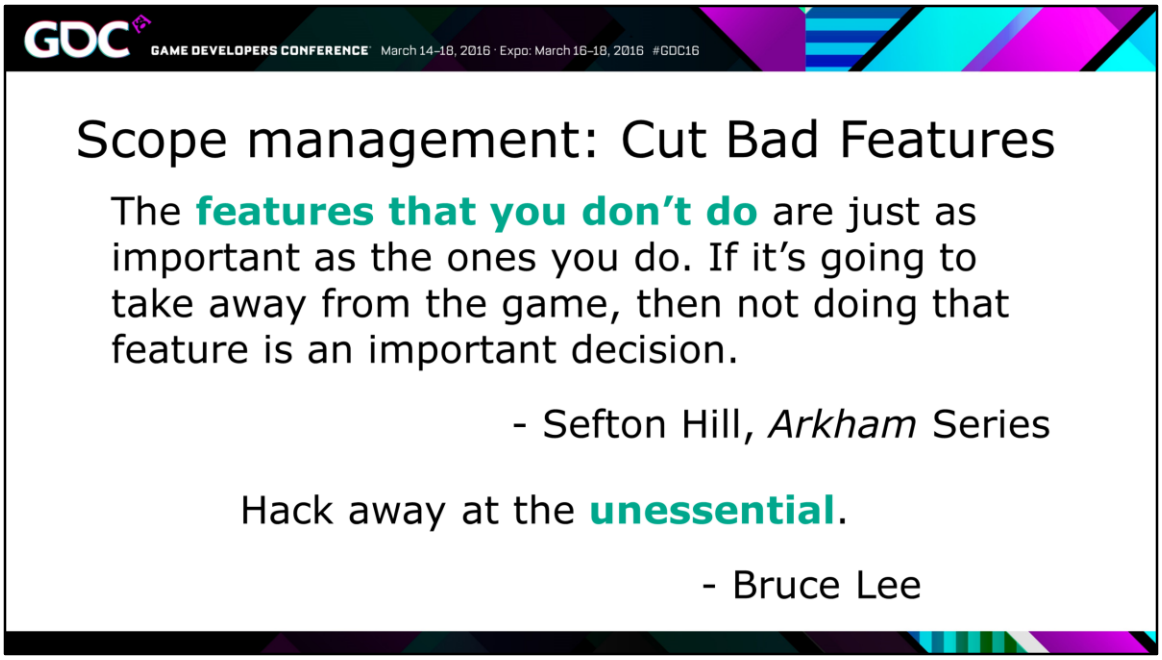


Back to the iron triangle. Constraints feel frustrating, especially when you're trying to build something really amazing. We often get into the habit of thinking that if our publisher would just give us more money, or if we just didn't have to ship by Christmas (or got the team to work through Christmas break), or if your team would just get on board with that great idea for EPIC BATTLES, our game would be awesome.

But that's not true. [Warren Spector quote] Remember how having a clear shared design vision was the #1 correlating factor to game success? Number 2 was protecting that vision by managing design risk: that is, protecting against building the wrong things. Statistically, the biggest risk to your game is not lack of funding, or lack of time, or someone else stealing your ideas. The biggest risk is wasting your precious development resources building the wrong features.

Embracing your constraints will help you guard against that. Say no to ideas that don't fit your time and budget constraints, or that don't fit your game's vision. Make the iron triangle your ally and use it to protect your game.

[http://www.gamasutra.com/view/feature/131523/postmortem_ion_storms_deus_ex.php]



This is hard to believe during development: often when you're working on a game, your vision of the completed game includes all the ideas you started with, and easily expands to include any new ideas you have.

When you have to cut big features, it's hard to imagine a player seeing the finished product and not noticing the big holes left by your cuts. But if the rest of your game embodies your clear, focused vision, they won't see what you had to cut away. Incomplete and badly implemented features stick out like a sore thumb; but missing features are almost always invisible if the rest of the game is executed well.

[http://www.mtv.com/news/2464057/batman-arkham-city-creator-sefton-hill-on-the-secrets-of-successful-game-design-2/]

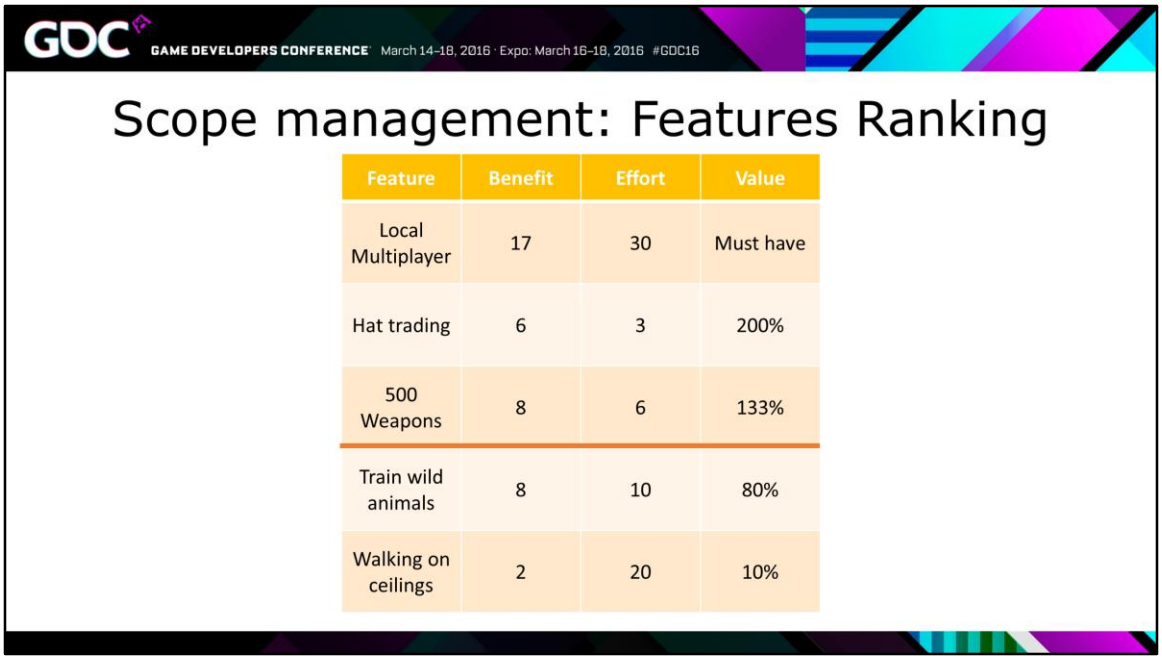
The last big game I worked on was Middle-earth: Shadow of Mordor. This game started and ended with a strong design vision, but during production a lot of features were added that were later cut. And not minor features: major game systems, levels, and awesome features that some players would have really loved. But the final game was successful without those features, by every metric, and the team was proud of the final product. The nemesis system, where individual enemies that you fight often come back with more abilities and more personality, was a big hit with players.

Cutting big features did leave holes, but players, as far as I can tell, didn't notice them. I didn't see a single review or online discussion that said, "Yeah, the nemesis system was ok. But it would have been so much better if there were two different factions of orcs in a civil war." Or, "the combat is pretty cool, but without a house-sized great beast to climb and parkour around on, it just felt kind of empty". No one felt that way: the vision for the game was solid, and the absence of those cut features didn't hurt it. When those features that didn't match the vision were cut, it made the game feel more finished, not less.

Features that aren't necessary to your game's vision feel tacked-on and unnecessary if you leave them in, and they won't leave holes if you cut them. And cutting them early enough will save you tons of time that you can spend on features that do match your vision.

So how do you decide which features are the right ones to cut, and which ones are the right ones to spend time on? It can help to have a process for making decisions on new features or additions, and to make sure everyone on your team is on board with it. This is often called 'change management' or 'scope management'. Let's go over some tools that can help your team have good scope management and stay focused on the right features.

[http://gamasutra.com/view/news/234421/Postmortem_Monolith_Productions_Middleearth_Shadow_of_Mordor.php]



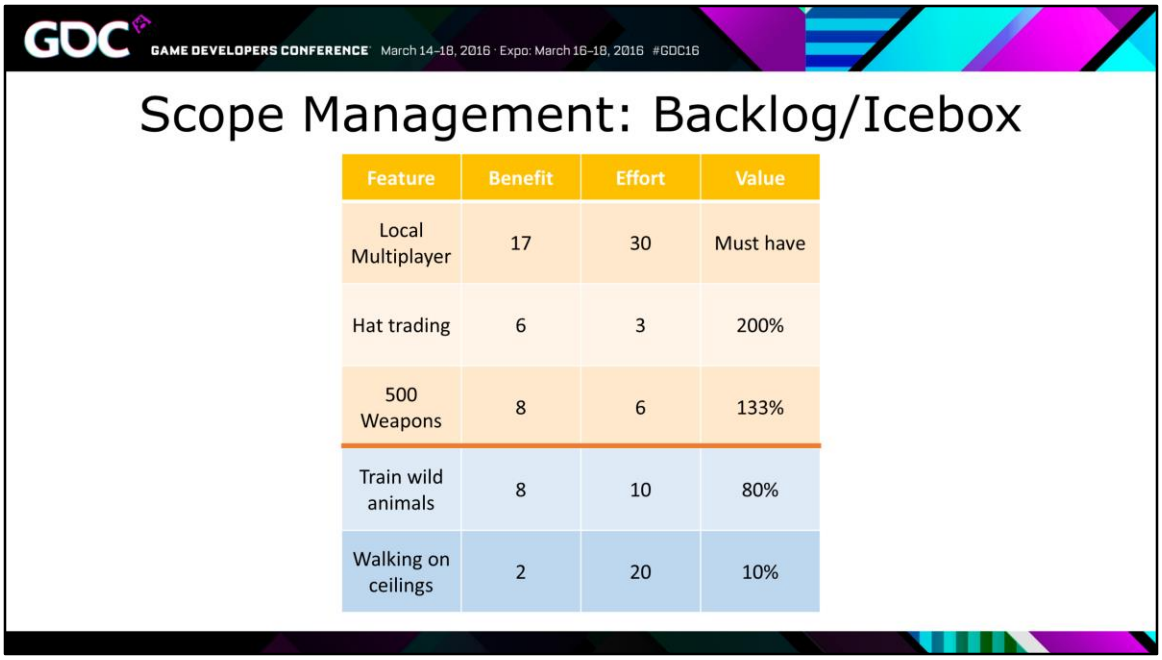
The first is objective feature ranking.

There are two big problems with new ideas: first, someone on your team is going to be convinced that, no matter how expensive, risky, or hard to implement the feature is, it's going to be the best thing in the game. Second, they might be right; you never really know which feature is going to be the thing a player falls in love with and tells all his friends about.

Find an objective way to rank your planned features. Like the Arkham City team, the Witcher 3 development team knew that their new project was a lot more ambitious than Witcher 1 and 2. So they used a numerical grading system similar to the one in this table. Team members voted on the features they liked best, and the features were evaluated for level of effort to implement. Those numbers were combined into a value score that they used to prioritize their backlog and decide what to spend time on.

A lot of teams hear about techniques like this and don't even try them, because they know that the numbers won't be perfect. And they won't! Of course taking time during pre-production to guess how much a feature will benefit your final game will never give you totally accurate predictions. But what's important here is the act of looking as objectively as possible at your ideas and really considering whether they're worth working on. Assigning an actual number to a feature's value can really help you evaluate them more logically, instead of based on who on your team expresses the strongest emotions about a feature.

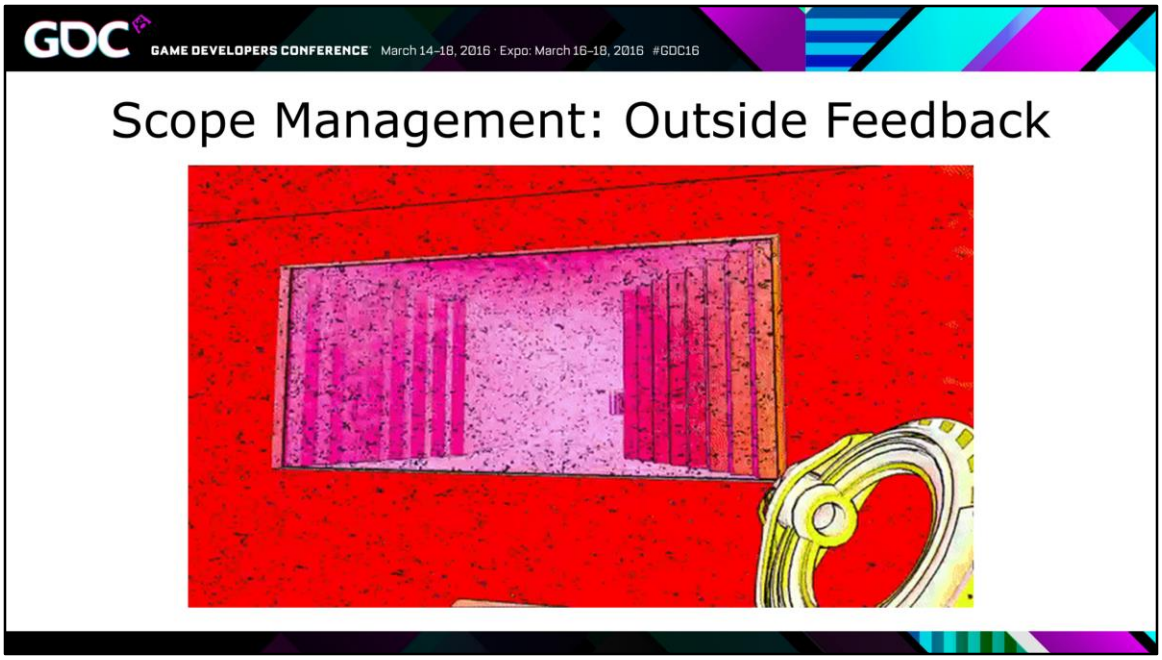
[http://www.gdcvault.com/play/1017938/From-Great-Ideas-to-Game]



A second way to manage scope is to have a feature backlog.

If a proposed feature isn't very high-value, or you're just not sure if it fits the vision for your game, you can put it in a feature backlog. Let it sit around for a while; it's possible that as you work on other features, you'll realize that it's higher-value than you thought, or that it fits really well with the design vision for your game. Most of the time, as you work on high-value features, you'll find plenty of things about them that you want to expand and build on and you'll realize you didn't need those backlog items after all, and you'll be glad you didn't spend time on them.

Make sure to do this early in development! You'll get close to shipping much sooner than you think, and once you've committed a bunch of work to a feature it's much harder to cut it, or even to put it on the back burner.



Finally, external feedback is invaluable for scope management. It's very difficult, as a game developer, to be as objective about your game as you need to be.

As early as you can, do regular playtests with people who aren't on your team. Do they find your game satisfying? Do they want to play more of it? Do they see your vision in it? Even before you have anything playable: when you explain your game vision, mission statement, or pillars to people who aren't on your team, do they 'get it'? Are they excited?

Knowing what feedback to take from playtesters and what feedback to ignore is a learned skill. Be as objective as you can; find a way to measure player feedback with numbers if at all possible. Alexander Bruce, the creator of Antichamber, tracked the play times of everyone who tried his game demo at conventions over years of iteration, and used that number as his metric: how long will the player play the game voluntarily before they walk away? He increased that number from an average of 5 minutes to over 40 minutes. (And the result is pretty amazing; Antichamber is also a great example of agile development and iterating until you zero in on your game's design. Check out Alexander Bruce's GDC 2014 talk; it was rated best talk that year.)

There was a great talk yesterday by Ben Kane on designing Keep Talking and Nobody Explodes, where he also discussed using this process. They eavesdropped on players trying out their bomb-defusing game at conventions, and used the data they gathered to iterate on the game and focus in on their core design: a game about communication between players. Check that talk out as well if you're interested in agile exploration as a design strategy.

And remember that if you aren't measuring something, it's very hard to improve it, and it's even harder to tell whether you've improved it. Have objective, numerical measurements for the things in your game that you want to be great, so that you know for sure when you're succeeding.

[http://www.gdcvault.com/play/1020776/Antichamber-An-Overnight-Success-Seven]

Warner Bros. hired a Immersyve, a firm that studies player behavior, to conduct play tests for Shadow of Mordor and the Arkham games to see if the games were satisfying for players. Immersyve's research has shown that whether someone is having fun is a very poor indicator of whether they will keep playing a game. But they've also shown that players have specific, measurable needs which, if fulfilled by your game, will not only keep them coming back to it but that will also make them more likely to tell their friends about it!

Those needs are:
- Compentence, or feeling that they're effective and skilled when playing
- Autonomy, the feeling that they're in control of their own decisions
- Relatedness, feeling a strong connection to other players, game characters, or the game world.

Troy Skinner and Scott Rigby gave a very good talk on Immersyve's process in 2015 that's in the GDC Vault, and Jason VandenBerghe is giving a talk Thursday afternoon about player motivation that discusses these metrics.

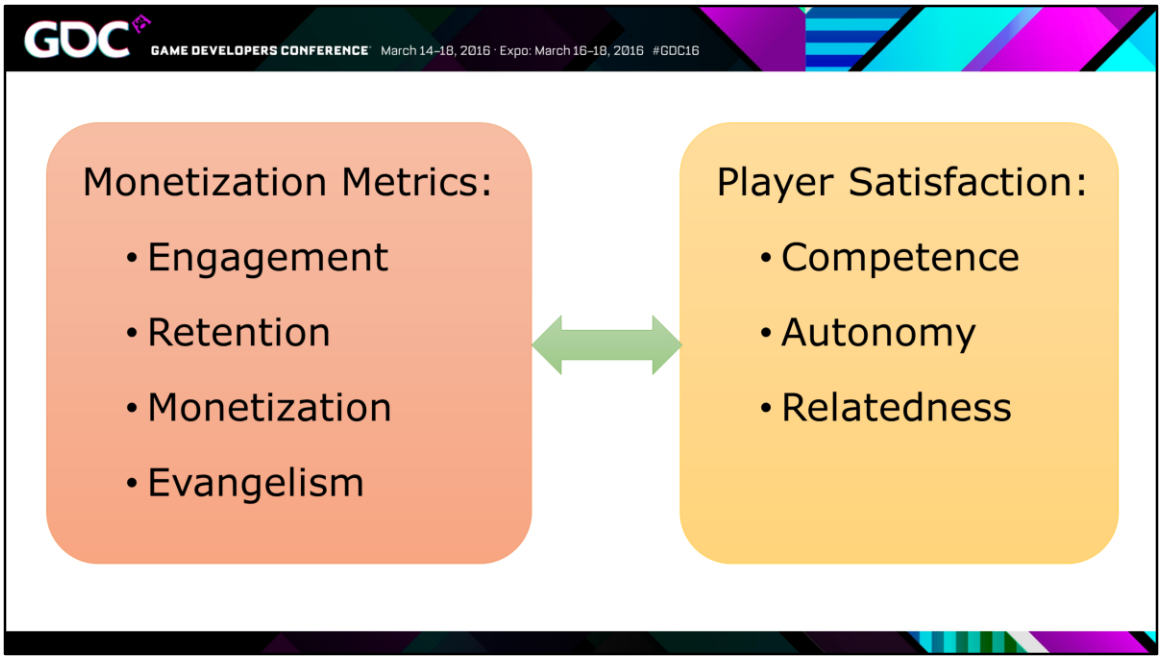
I'm not saying that you have to use these metrics, but they're a very good set. I personally believe that they were a critical part of why Shadow of Mordor was able to re-focus, and they have a solid foundation in psychology outside of games: if you think about what you find satisfying in your job, it's probably going to be a very similar list.

But the important thing is to identify something measurable that you want your players to do, and then measure whether they're doing it. Alexander Bruce wanted players to play Antichamber longer. Warner Bros. wanted players to be drawn to their games in the ways that make people tell their friends how awesome a game is. If you want your players to really be drawn into your game's story, ask playtesters how absorbing they think your game's story is, on a scale of 1-10. Then work on your story some more, and ask a new batch of playtesters the same question. If that number goes up, you're getting closer to your goal.

Remember that having measurable goals requires you to know what you want your game to be. It always comes back to having a clear, strong design vision for your game.

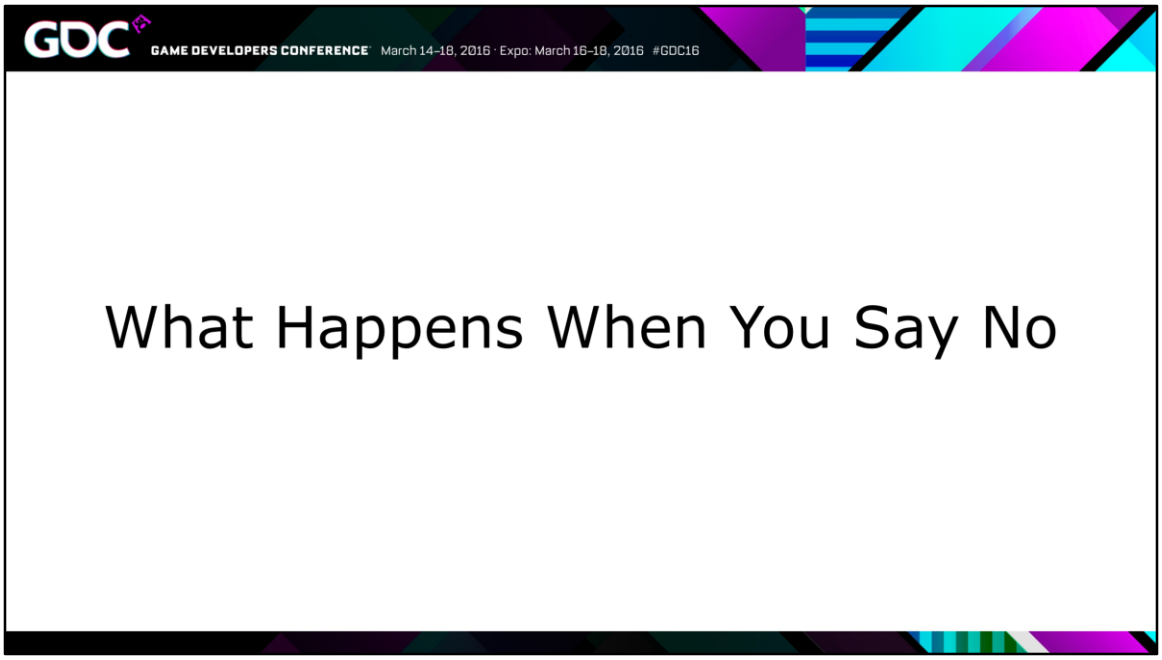
Immersyve: PENS metrics [http://www.gdcvault.com/play/1022369/Why-Did-Players-Buy-That]

[http://www.gamasutra.com/view/news/235777/Designing_Shadow_of_Mordors_Nemesis_system.php]



Many of us have worked, or are working, on free-to-play games that monetize through in-app purchases. For those games, you already have sheaves of metrics to maximize: player acquisition, engagement, install rate, player retention (that is, do they keep coming back), player monetization, and evangelism, or how much your players invite their friends or tweet about your game or give gifts to other players or whatever else you can measure about them.

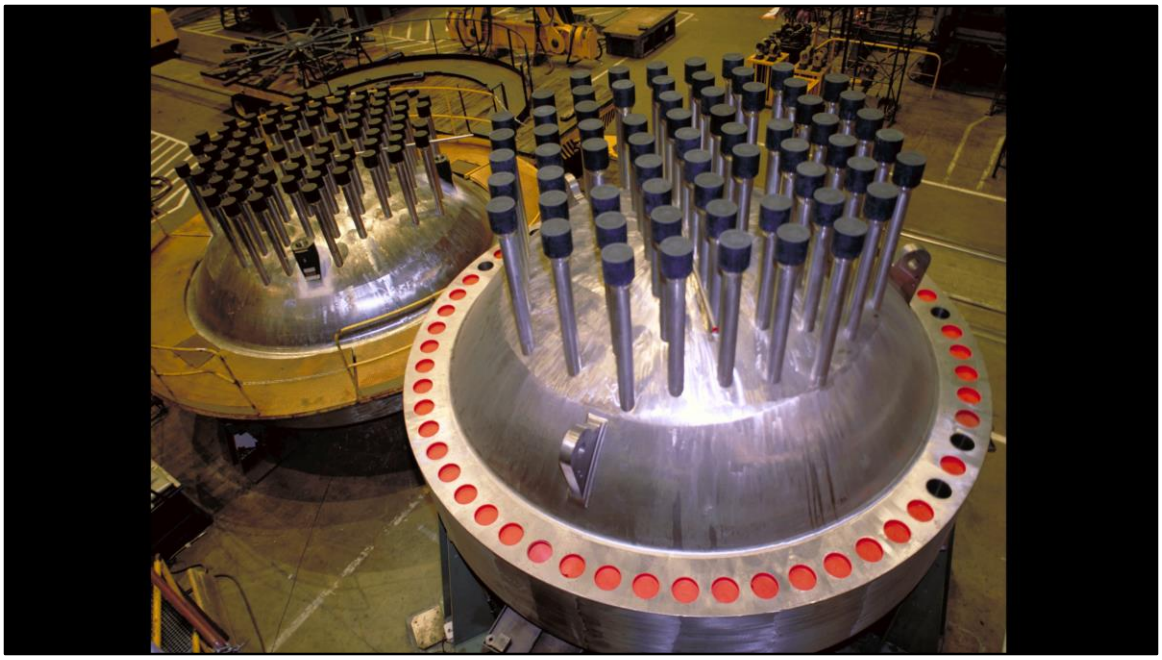
What's interested is that these two sets of metrics are strongly correlated: if players find a game satisfying they'll keep coming back to it and be more inclined to pay. Again, find ways to measure whether you're giving players what they need, and make decisions based on those numbers.



So: you have a solid vision for your game that your team is excited about. You have a process for deciding which features to work on, you have objective numerical measurements for your design goals, and your team understands both the importance of scope discipline and what the biggest risks to your project are.

But then someone on your team has a cool idea for a feature, but it doesn't fit your design vision. You have to say no. What happens next?

I once had an interview for a producer position at a major game studio where I kept getting asked the same question: "How do you deal with people hating you when you tell them they can't have something?" That question really worried me, and indicated to me that the team had some major problems.



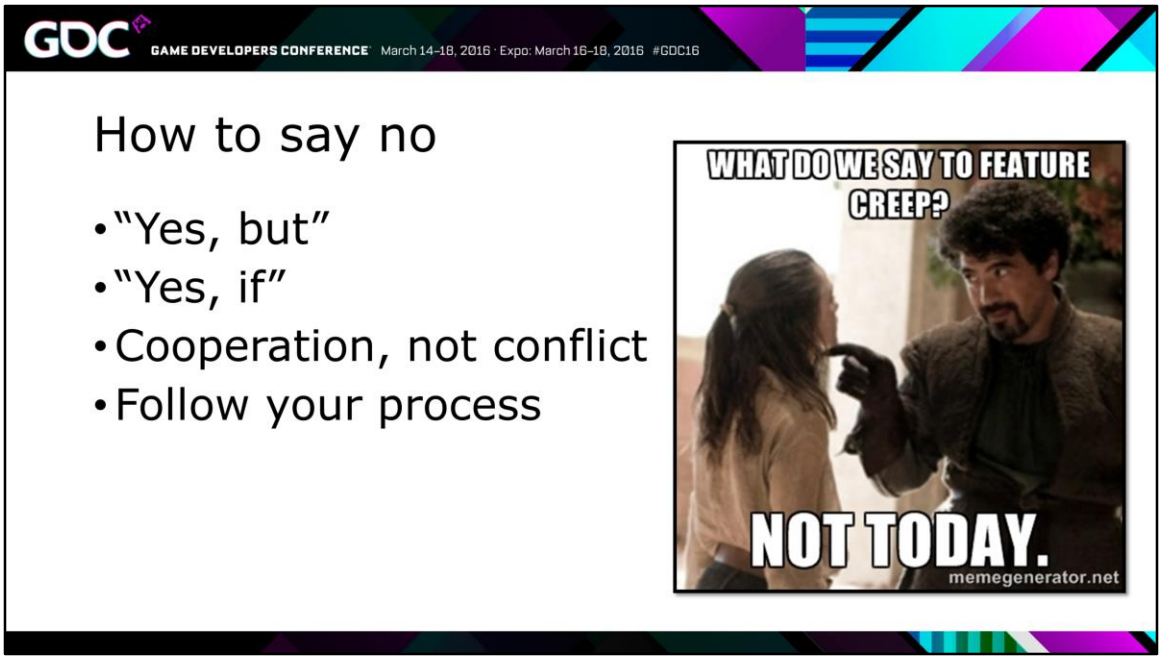
People who aren't willing to work within constraints are like a nuclear reactor with no control rods; they cause a lot of damage but don't produce anything useful.

Right now on your team, you probably have some people who act as your power source: the ones coming up with the awesome ideas, building prototypes, feeding energy into the team, and staying excited about the cool game you're building. And you probably have people who act as the control rods: the pragmatists who keep an eye on your priorities, your budget, and your schedule, and who make sure that everyone is staying focused and working on the right things. You need both of those types of people; they're both vital to your success. If you're a solo developer, you need to *be* both of those people. If you don't have the right balance between passion and pragmatism on your team, and *especially* if, like on that team I interviewed for, one or the other of those types of people are seen as 'the problem', you have a major fundamental team issue that needs to be fixed.

Both your optimists and your pragmatists are precious resources; unless you have both, you won't ship a great game.

A lot of game developers are afraid of saying no to cool ideas because they don't want to dampen their team's enthusiasm or slow down their momentum. But enthusiasm is only effective when it's directed towards the right things, and people do their best work when they know they're working towards an exciting goal. Give them that goal, by setting a clear design vision.

There are potentially bad consequences for saying no to features: you might upset someone, you might accidentally say no to the coolest idea in the game, or you might be overly cautious. Let's go over some tools you can use when saying no to minimize those negative effects.



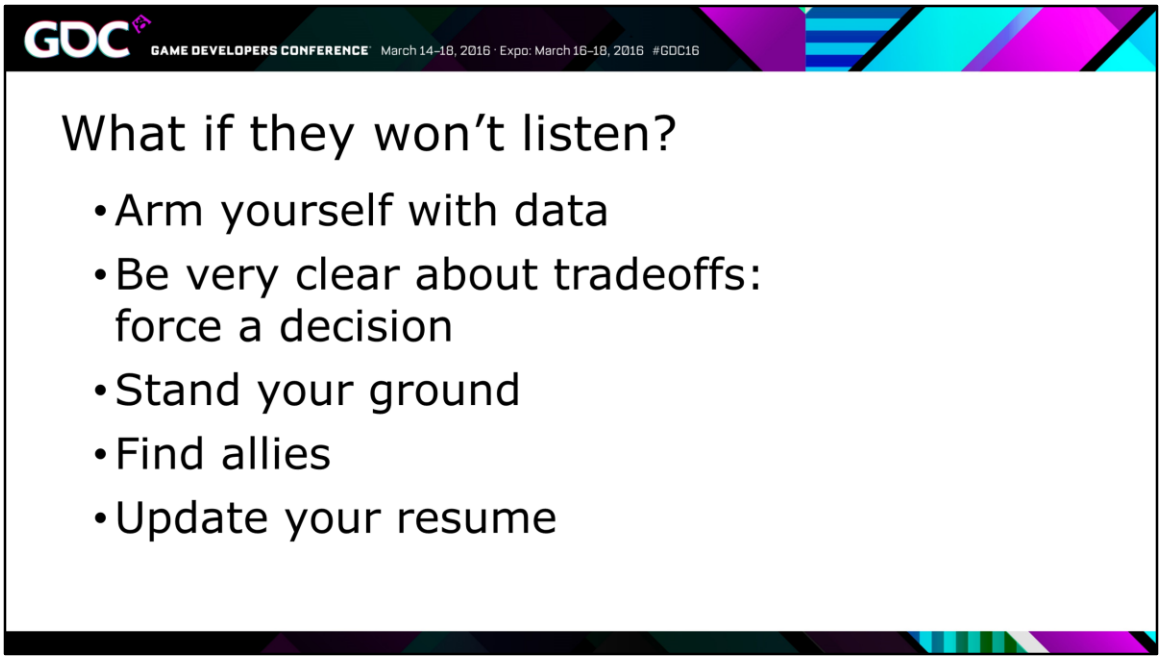
"Yes, but": Before you say yes, make sure that the stakeholder fully understands the consequences of what they're asking for. Remind them of the risk they're adding and make it their responsibility. Remind them of the design vision, and make sure they understand it and believe in it. Protect your vision as well as you can. "Yes, we can do another demo for E3, but it will delay the next milestone."

"Yes, if:" sometimes, well, all the time, your team will get new information that will mean you need to do something different; sometimes a major pivot in the project. That will happen, and it's fine when it does. "Yes, if" is figuring out how to do the necessary new thing. "Yes, we can do another demo for E3, if we cut level 14 in half and use it for the demo." Or, "Yes, we can add that feature, if we cut something equally big. What do you think we should cut?" This is where your ranked feature list can come in very handy.

Cooperation, not conflict: When a stakeholder wants something new added, and you know that it will add risk to the project, do not let it become you versus them. That's not productive, and it isn't a position you want to be in; do not let yourself be the only person opposed to the design director's awesome new idea. It's exhausting, and it makes it easy for people to dismiss you as someone who hates fun. Instead, make it you *and* them versus the facts. "The priorities we agreed to at the beginning of the milestone put the demo last. The top priorities were X, Y, and Z, and all three of those are vital for the design of the game. What can we change in our plan to be able to do a demo too? What can we do to create a great game within our constraints, and also do this new thing you want?"

Finally, follow your process. Even if you don't have strict numerical rankings or metrics, have some

objective way that will let you be confident that a proposed feature will or won't support your game vision. You won't have to worry about accidentally cutting the best thing in your game if you have a way to objectively measure what the best thing in your game is.



None of what I've said up until now will do you any good if the ambitious, optimistic people on your team just won't listen to your pragmatists, or if your team's prioritization decisions get overridden by someone higher up the chain of command. So what happens if, like that team I interviewed with, you have people in authority who won't take 'no' for an answer?

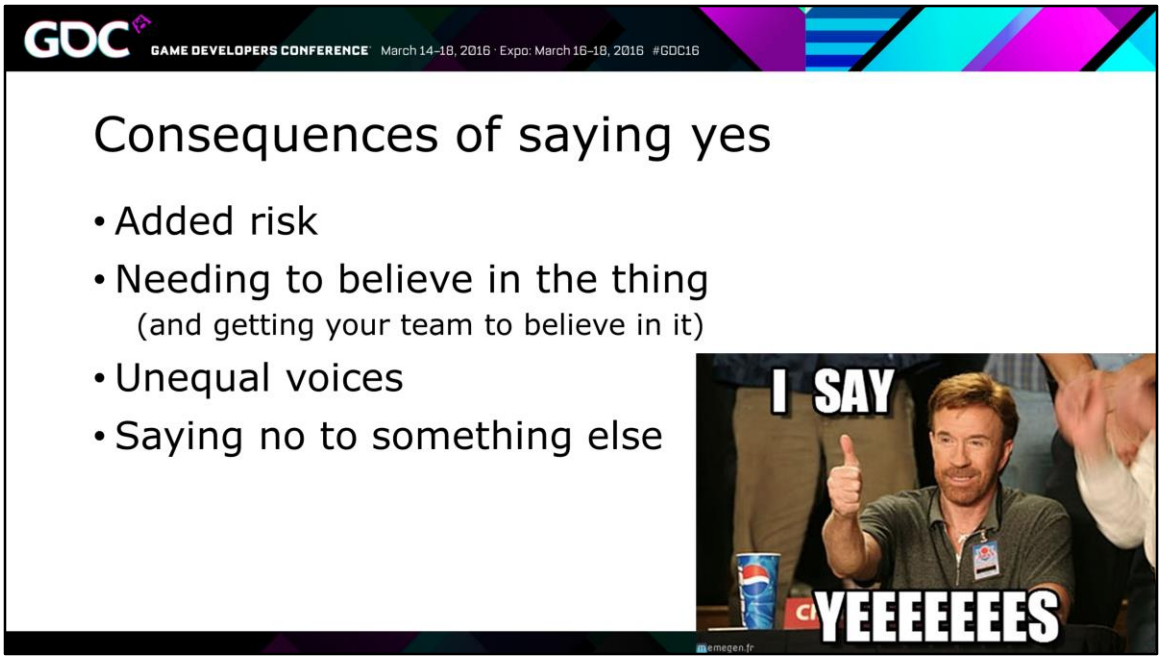
Arm yourself with data: remember all those metrics you're gathering on whether players are doing what you want? Keep those ready to present. I know really good producers who, when finding themselves in a situation like this, keep a powerpoint always updated with the latest data, ready to convince whoever needs convincing that the team is making the right decisions. Show them that data, tell them what your team has decided, and make them show you where you're wrong.

Be very clear about tradeoffs: Present a menu of options, and make them choose. If the choice is realistically between feature A and feature B, show them in the data why that choice is necessary, and make them make a choice.

Stand your ground; if you're working with someone who stubbornly refuses to reign in their ideas or requirements or refuses to cut features, if you're not just as stubborn as they are, your game will not ship.

Find allies on your team who may know more about how to get a specific person to listen, or who might have more authority than you and who want to know the real situation with what's doable for the game. People who have been around for a while and who have learned how to make great games often want to hear the truth about what your team is capable of: if you've been gathering and keeping track of that data, they'll see you as a valuable resource and want to hear what you have to say.

And last, if none of those help, don't stay in a situation where you aren't able to be effective, or where you'll be beating your head against a wall until you ship a bad game that was the best you could do under the circumstances. Game development is hard enough without working under people who are dead set on making it even harder.



Remember that saying yes is never free; it always costs you something.

For one thing, every feature you add adds risk: maybe a little, maybe a lot, but always some. Anything you add to your game's design requires more assets that need to be created, more gameplay that needs to be tested, and more bugs that will have to be fixed. Keep your eyes open to that cost up front and make sure your team can handle it.

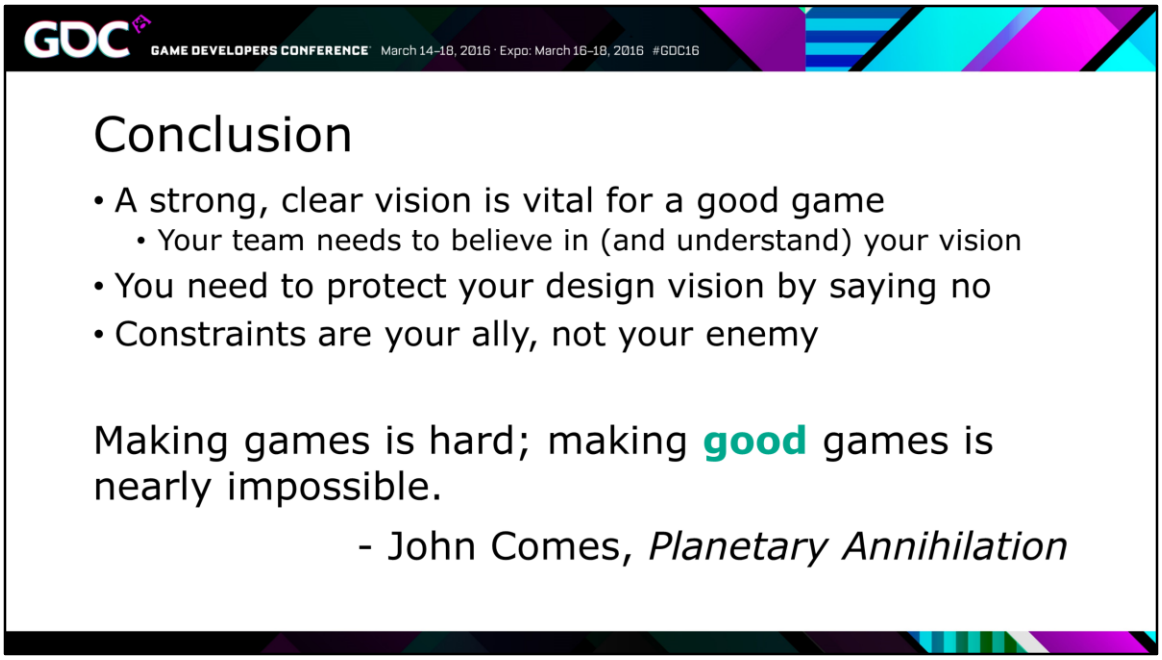
If you say yes to something, you're agreeing to really believe in it. And you have to get your team to believe in it too. If your publisher mandates that you add a My Little Pony champion to your MOBA, you have two choices (and neither of them is saying no): You can say yes and half-ass it, and your players will hate it and you'll hate it and it'll be a dirty smudge on your otherwise great game, or you can say yes and believe in it and have fun with it and make the best damn My Little Pony champion that you can.

A great example, again from Shadow of Mordor: the game's publisher, like they do, required regular demos for conventions and press. Instead of reluctantly saying 'yes' and viewing those demos as yet another unreasonable publisher demand, the team used the demos as a tool to make the game more polished and to get a better understanding of how long it took them to finish a feature. They took something that could have been a burden, and made it something that benefited them and that they believed in. Ideally, anything you say yes to needs to add value to your game; if it doesn't, see if you can figure out a way to get value out of it, if you're going to be doing it anyway.

Whenever you say yes or no to anyone, you're setting a precedent for how much of a voice that person has; every 'yes' makes it harder to say 'no', and every 'no' makes it easier to say 'no' in the future. The people involved in your game have different amounts of authority; your publisher has power over your budget, but your audio designer doesn't. This is another reason it's so important to have a clearly defined design vision and an objective way to measure new ideas: if you don't have those things, you're more in danger of getting into the habit of saying 'yes' to the ideas coming from those with more authority, and saying 'no' to ideas from people with less authority, regardless of how good for your game those ideas actually are.

When you say yes, you're taking your limited time and money resources away from something else; you just don't know what is yet. Every yes is also a no. You will find things through playtesting, later in your project, that will make your game much better. Saying yes to low-value features early in your project means you'll have fewer resources to spend on those great found ideas later on.

[http://www.gamasutra.com/view/news/234421/Postmortem_Monolith_Productions_Middleearth_Shadow_of_Mordor.php#tophead]



A good vision is vital for a good game
If you can't say no, you can't control your scope or your risk
Constraints are your ally, not your enemy. Make friends with the iron triangle.

GDC GAME DEVELOPERS CONFERENCE March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16

Thanks!

rtomandl@gmail.com

@slideruth

(feedback welcome!)

Resources

- Online:
  - The Game Outcomes Project
  - Gamasutra Postmortems
  - The GDC Vault
    - Antichamber 2014 talk
- Books:
  - The Mythical Man-month
  - The Goal
- Personal:
  - 9 years of Project Management
  - Cussedness

The Game Outcomes Project:
http://intelligenceengine.blogspot.com/2014/12/the-game-outcomes-project-part-1-best.html

Gamasutra Postmortems:
http://www.gamasutra.com/features/postmortem/

GDC Vault:
http://www.gdcvault.com/
Especially “Antichamber: An Overnight Success, Seven Years in the Making”:
http://www.gdcvault.com/play/1020776/Antichamber-An-Overnight-Success-Seven

The Mythical Man-Month:
http://www.amazon.com/Mythical-Man-Month-Software-Engineering-Anniversary/dp/0201835959

The Goal:
http://www.amazon.com/Goal-Process-Ongoing-Improvement/dp/0884271951