Game design is simple, actually – Raph Koster

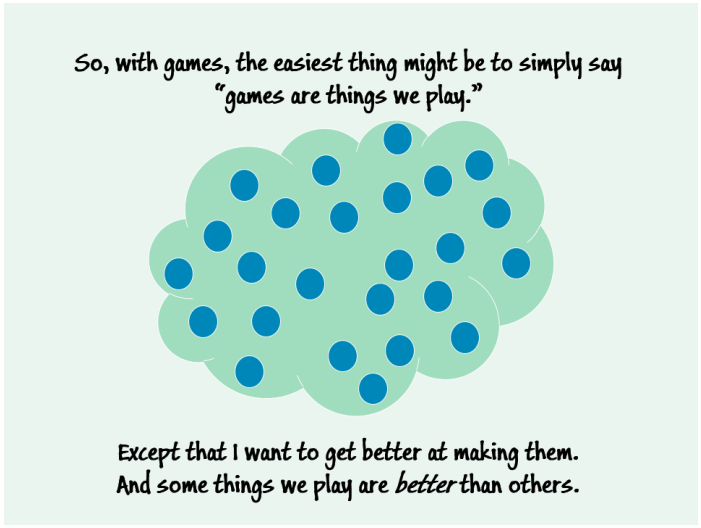
raphkoster.com/2025/11/03/game-design-is-simple-actually

November 3, 2025

So, let's just walk through the whole thing, end to end. Here's a twelve-step program for understanding game design.

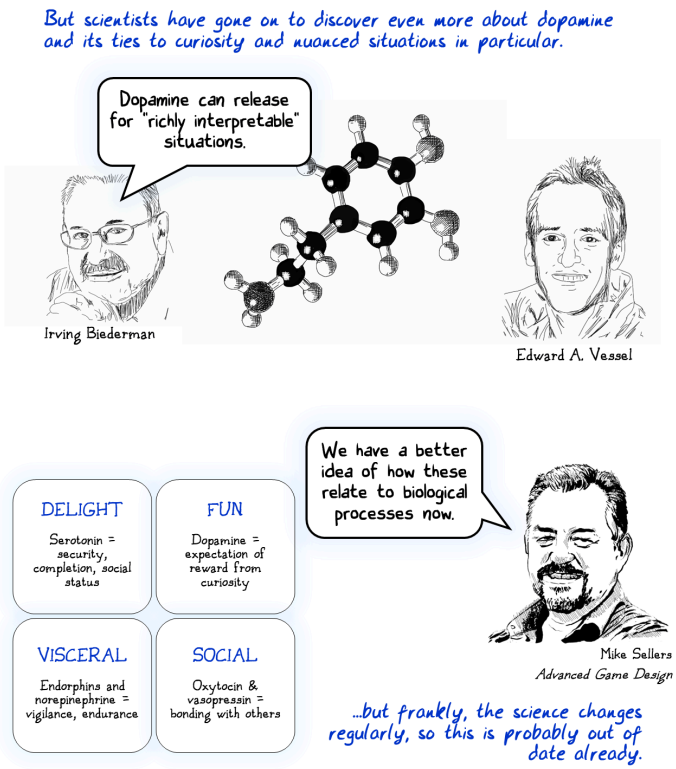
One: Fun

There are a lot of things people call "fun." But most of them are not useful for getting better at making games, which is usually why people read articles like this. The fun of a bit of confetti exploding in front of you, and the fun of excruciating pain and risk to life and limb as you free climb a cliff are just not usefully paired together.



In *Theory of Fun* I basically asserted that the useful bit for game designers was "mastery of problems." That means that free climbing a cliff is in bounds even though it is terrifying and painful. Which given what we already said, means that you may or may not find the activity fun at the time! Fun often shows up *after* an activity.

There's neuropsych and lots more to go with that, and you can go read up on it if you want.



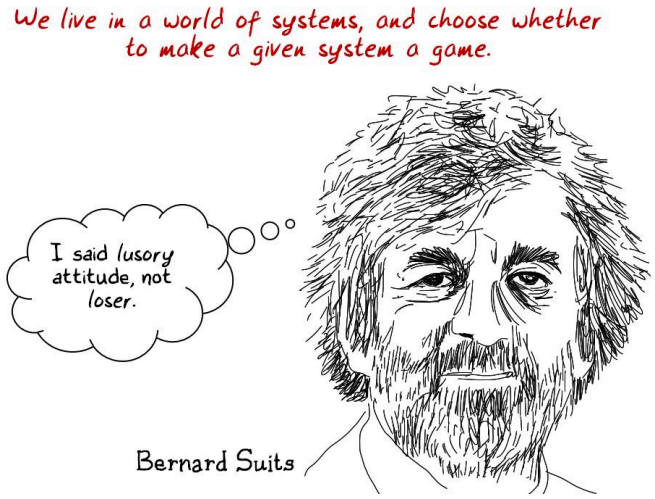
Anything that is *not* about a form of problem-solving is not going to be core to game *systems* design. That doesn't mean it's not useful to game *experience* design, or not useful in general.

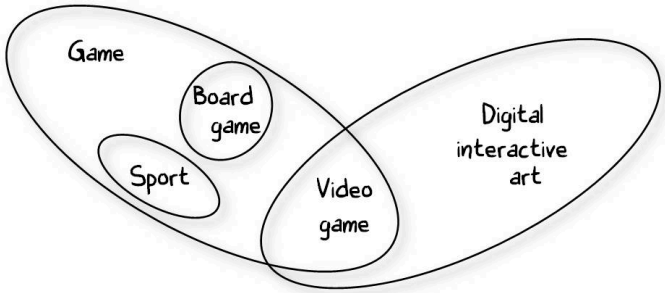
Also, in case it isn't obvious – you can make interactive entertainment that is not meant to be about fun. You can also just find stuff in the world and turn it into a game! You can also look at a game and *choose not to treat it as one*, and then it might turn into real work (this is often called "training").

This rules out the bit of confetti. A game being made of just throwing confetti around with nothing else palls pretty quick.

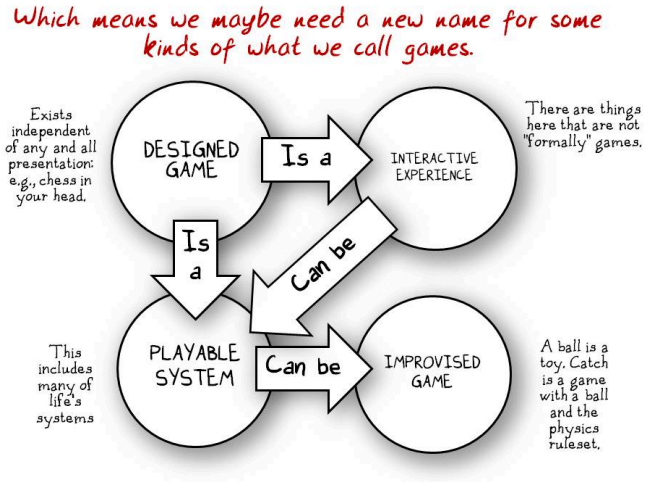
Bottom line: fun is basically about making progress on prediction.

Two: Problems and toys





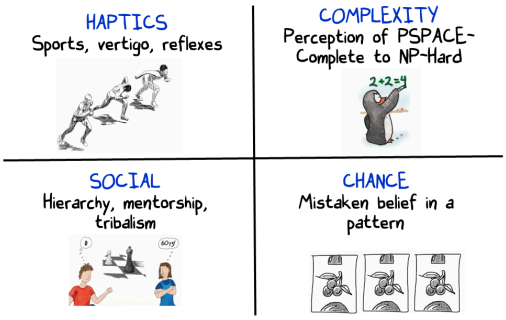
A consequence of all this was being led into approaching games themselves in a science-y way.



There are a *lot* of types of problems in the world. It is really important to understand that you have to think about problems games can pose as broadly as possible. A problem is *anything* you have to work to wrap your head around. A good movie poses problems too, that's why you end up

thinking about it long after.

You can go look at theorists as diverse as Nicole Lazzaro, Roger Caillois, or Mark LeBlanc for types of fun. You'll find they're mostly types of *problems*, not types of fun. "I enjoy the types of problems that come from chance" or "I enjoy the types of problems that come from interacting with others" or whatever.



Problem spaces overlap these sorts of fun, but are not identical.

This is not a bad thing. This is what makes these lists useful. Your game mechanics are about posing problems, so knowing there's clumps of problem types is very useful.

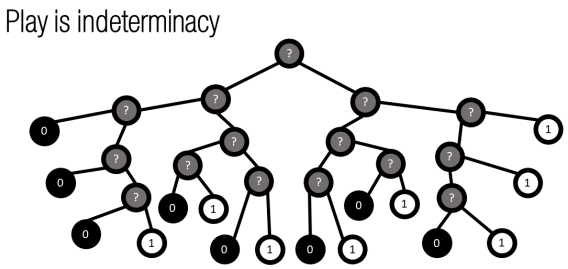
In the end, though, a problem is built out of a set of constraints. We call those rules, usually. It also, though, has a goal. Usually, if we come across a set of rules with no problem, we just play with it, and call it a toy.

Building toys is hard! Arriving at those rules and constraints to define a nice chewy problem is very challenging. You can think of a toy as a problematic object, a problem that invites you to play with it.

On the other hand, it's not hard to turn a toy into a game, and people do it all the time. All you have to do is invent a goal. We shouldn't forget that players do so routinely.

Building a toy is an excellent place to start designing a game.

Bottom line: we play with systems that have constraints and movement, and we stick goals on them to test ourselves.



Three: Prediction and uncertainty

Games are machines built around uncertainty. Almost all games end by turning an uncertain outcome into a certain one. There's a problem facing you, and you don't know if you can overcome it to reach that goal. Overcoming it is going to be about predicting the future.

If there's one thing that good games and good stories have in common, it's about being unpredictable as long as possible. (This is also where dopamine comes in, it's tied to prediction; but it's complicated and nuanced).

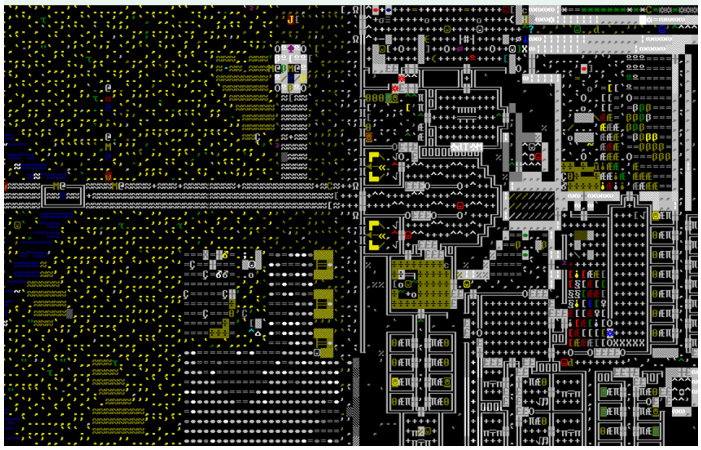
If a problem basically has one answer, we often call it a puzzle. There's not a lot of uncertainty built into a binary structure. You can stack a bunch of puzzles one on top of the other and build a game out of them (which then introduces uncertainty into the whole), but a singular puzzle isn't likely to be called that by most people.

It happens quite often that we used to think something was a game, and it turned out it was actually a puzzle. Mathematicians call that "solving the game." They did it to Connect Four – and you did it to tic-tac-toe, when you were little.

Good problems for games therefore all have the same characteristics:

- They need to have answers that evolve as you dig in more – so they need to have depth to them. Your first answer should only work for a while. There might be many paths to the solution, too. This is why so many games have a score – it helps indicate how big a spread of solutions there are!
- They need to have uncertain answers. (When you're little, this universe is a lot larger than it is when you're older – peek-a-boo is uncertain up to a certain point!).
- The problem should be something that can show up in a *lot* of situations.

Simulation, esp with dynamic systems

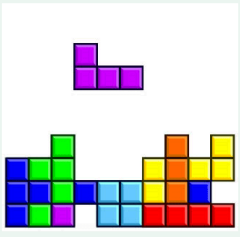


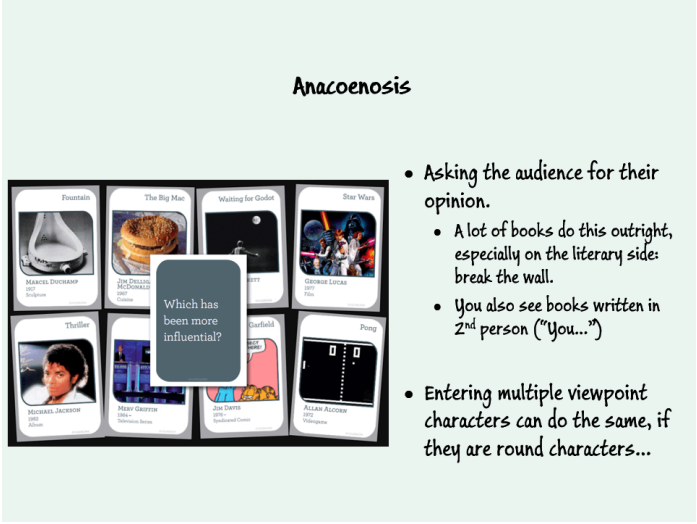
Signs as tokens

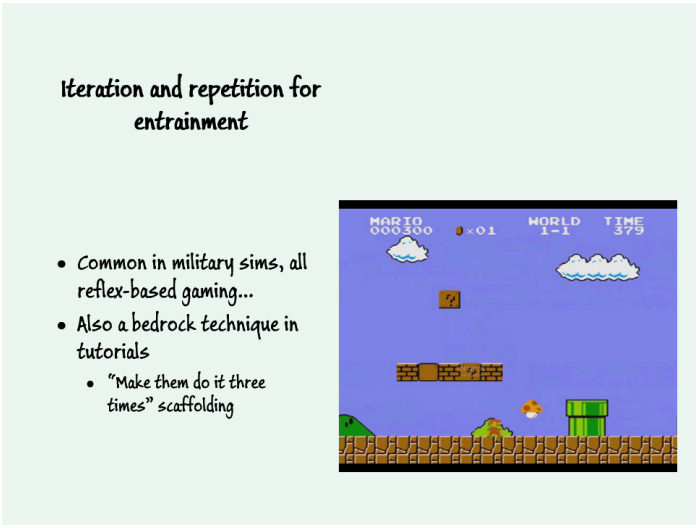
- Use non-quantifiable things as the tokens in the game
  - Dixit uses stories
  - The entire genre of RPGs makes use of collaborative storytelling this way

NP problems

- In a mathematical sense, humans tackle these via intuitive heuristics
  - This means they always think that they can find a better answer...
- There are handy lists of these types of problems online – just find some, and start designing games with these as mechanics!







A lot of very good problems seem stupidly simple, but have depths to them. Math ones, like "what's the best path to cross this yard?" but also story ones like "For sale: baby shoes, never worn."

I recently watched a video that included the statement that "picking up sticks" is not a useful loop. Picture a screen with a single stick in the middle. The problem posed is to move the cursor over it and click it. Once you do it, you get to do it again.

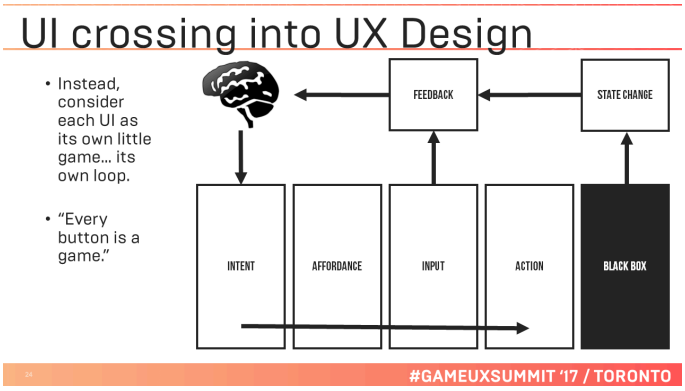
Guess what? The original Mac shipped with games that taught you how to move a mouse and click things. Once upon a time, mousing was a skill that was challenging; for all I know, you have grandparents who still have trouble with it. For them, it has uncertainty. For you, probably, it doesn't.

Bottom line: the more uncertainty, indeterminacy, ambiguity in your game, the more depth it will have.

Four: Loops

Now, imagine that the stick pops to a random location each time. Better, yes?

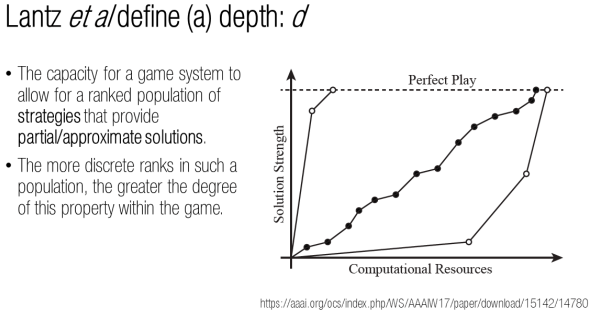
The core of a loop is a problem you encounter over and over again. “How do I get the next one?” But *something needs to be pushing back*, that’s what makes it an interesting problem and is usually what takes it past being a puzzle. I like to say “in every game, there is an opponent.” Even it’s just physics.



People talk about the core loop of a game. But there’s really two types of loops.

One is what we might think of as the *operational loop*. This is the loop between you and the problem, it is how you interact with it. You look at it. You form a hypothesis. You poke the problem. You see a result. Maybe it was success, and you grabbed the stick. Maybe it was failure. Maybe it was *partial* success. You update your hypothesis so you can decide what to do next.

The second loop is really your *progression loop* but is better thought of as a spiral. It’s what people usually mean when they say “a game loop.” They mean picking up the stick over and over. I say it’s a spiral, because clicking on the same stick in the middle of the screen over and over is not usually how we design games. That would actually be repeatedly doing the same puzzle.



Instead, we move the stick on the screen each time, and maybe give you a time limit. Now there's something you're pushing against, and there's a skill to exercise and patterns to try to recognize. Far more people will find this a diverting problem for a while. It's a better game. It'll get even better if there are reasons why the stick appears in one place versus another, and the player can figure them out over time.

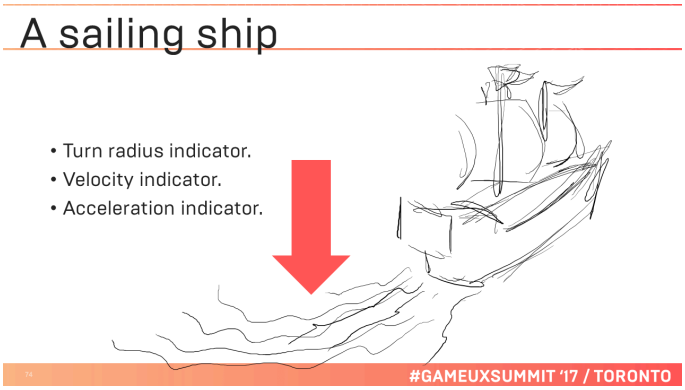
This matters: the verbs are in a loop. "Pick up," over and over. But the *situation* isn't. And you are learning how to reduce uncertainty of the outcome: move the mouse *here* and click, next move it *there*. That's why it is a spiral: it is spiraling to a conclusion. It'll be fun until it's predictable.

You can think of the operational loop as how you turn the wheel, and the situations as the road you roll over. A spot on the wheel makes a progression spiral as you move. One machine, many situations — we call these rules mechanics for a reason.

Bottom line: players need to understand how to use the machine, and the point is to gradually infer how it works by testing it against varied situations.

Five: Feedback

You can't learn and get better unless you get a whole host of information.

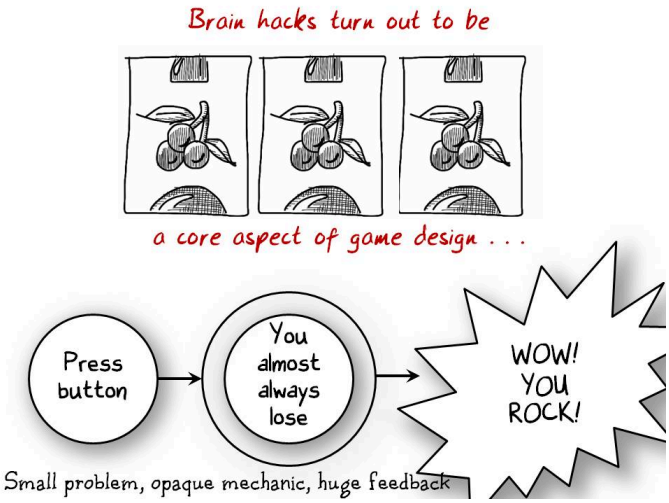


- You need to know what actions – we usually call them verbs — are even available to you. There's a gas pedal.
- You need to be able to tell you *used* a verb. You hear the engine growl as you press the pedal.
- You need to see that the use of the verb affected the state of the problem, and how it changed. The spedometer moved!
- You need to be told if the state of the problem is better for your goal, or worse. Did you mean to go this fast?

There are fancy names for each of these, and you can go learn them all. Everything from "affordance" and "juice," to terms like "state space" and "perfect information" and very confusing contradictory uses of the words "positive" and "negative" paired with the word "feedback."

Feedback in general can, and should be, delightful. That means it's where you get to use all those forms of fun that I threw away at the beginning. It can be surprising. It can be a juicy multimedia extravaganza. It can be a deeply affecting tragic cutscene that advances the game story.

If you have too little feedback, players cannot go around the interaction loop. Picture *Tetris* if the piece you drop is invisible until it lands.



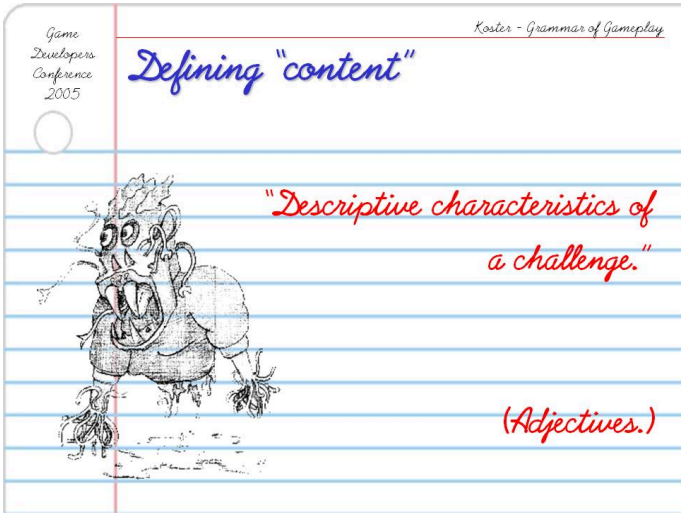
If you have bad feedback, players cannot go around the learning loop either. Picture *Tetris* if sometimes your score goes down when you complete a line and sometimes it goes up. You can't draw any conclusions about what the problem in the way of the goal actually *is*, in that crappy version of *Tetris*. Feedback needs to act as a reward to help you draw conclusions.

But there's a third mistake: you can supply a gorgeous and compelling set of feedback and not actually have a real problem under there. At minimum you're making shallow entertainment. At worst, you are building exploitative entertainment.

People will be willing to go along with pretty simple and pretty familiar problems as long as the feedback is great.

Bottom line: show what you can do, that you did it, what difference it made, and whether it helped.

Six: Variation and escalation

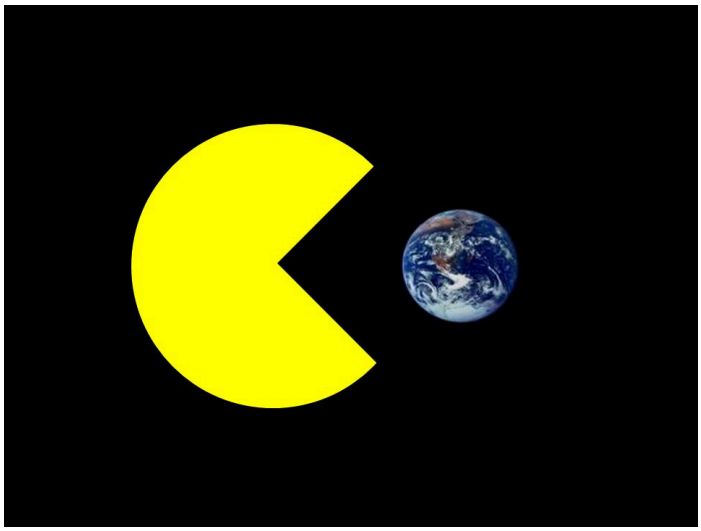


If you are trying to design and are thinking of a specific problem *scenario* you are not doing game systems design. You are doing level design. “How to multiply numbers” is a problem. [“What is 6 x 9” is not a problem, it’s *content*.]

Now consider the game of *Snake*, or *Pac-Man*. They are also games where the core loop is picking up a stick. The difference is that something is an obstacle to you picking up the stick: you get longer when you pick up the stick, and can crash into yourself. You have to avoid ghosts as you gather the stick.

How long you are in *Snake* is a different *situation*. Where the apple to eat is located is a different situation. To be specific, you have the same problem in different topology. Where you are relative to the ghosts, and which dots are left, and what directions you can go in the maze are different situations in *Pac-Man*.

You want the verbs you use in the loop to end up confronting many many situations. If your verb can’t, your core loop is probably bad. Your core problem (aka your core game mechanic) is probably shallow.



What you want is to be able to throw increasingly complex situations at the player. That's how they climb the learning ladder. Ideally, they should arrive at interim solutions (lots of words for that, too: heuristics, strategies) that later *stop working*.

Pac-Man actually got solved, by the way! That's why *Ms. Pac-Man* was invented. Sometimes, the way to escalate is to change the rules, and that's what *Ms. Pac-Man* did. It did it by adding randomness, and in fact using randomness is one of the biggest (and oldest) ways to create situation variation in games.

Bottom line: escalate the situations so that theories can be tested, refined, and abandoned.

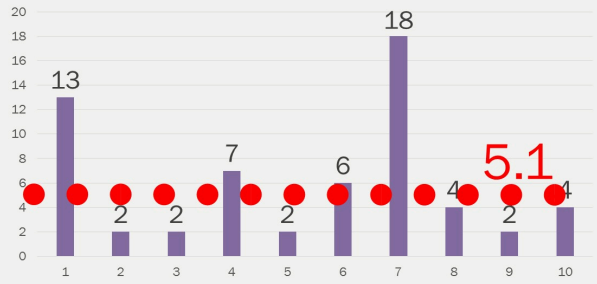
Seven: Pacing and balance

Since we can put all this this down very much to problem solving and learning and mastery, it means we can steal a whole bunch of knowledge from other fields.

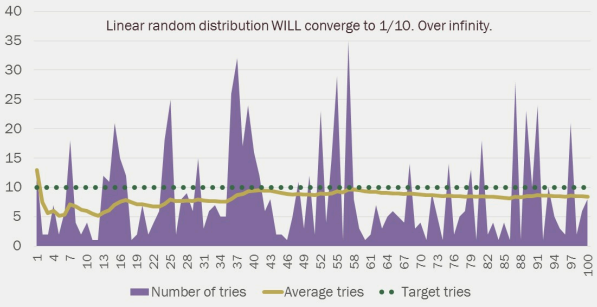
Roll, roll roll...

Try number...	Rolled a...	Got a...	What I did:
1	3	Gerbil	Wasted ammo
2	3	Gerbil	Wasted ammo
3	3	Gerbil	Wasted ammo
4	10	ROUS	Died
5	8	Marmot	Fled
6	3	Gerbil	Wasted ammo
7	6	Guinea pig	Fled
8	10	ROUS	Died instantly
9	10	ROUS	Died on sight
10	4	Rat	Got pissed off
11	9	Capybara	Cussed at developer
12	5	Squirrel!	Thanked the gods

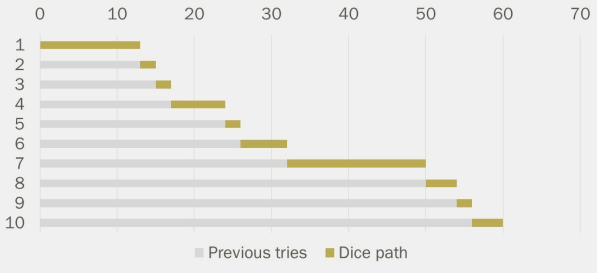
Tries it took to get ten squirrels

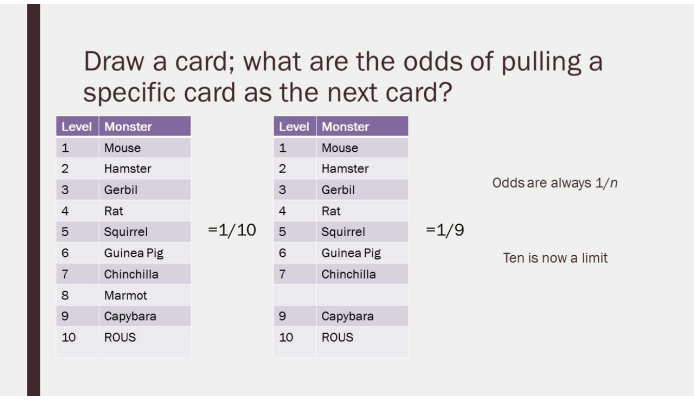


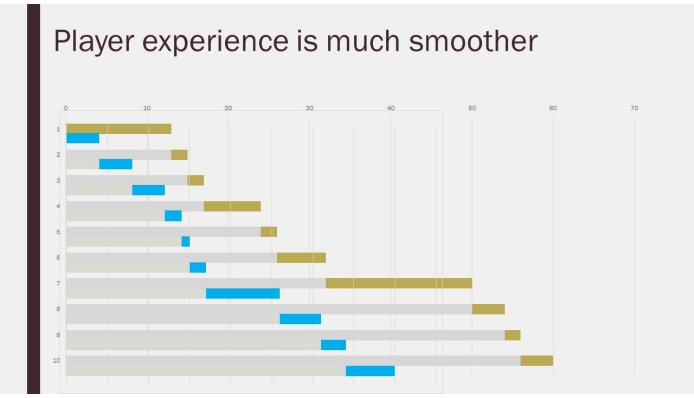
That didn't work



The player pacing was very unpredictable

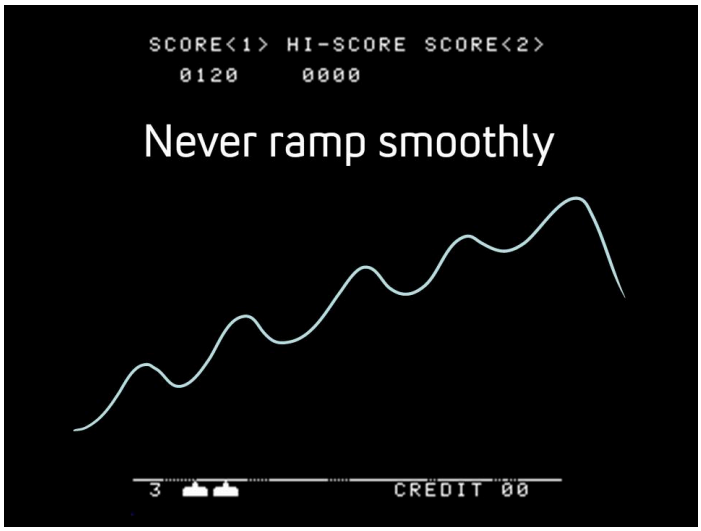






People learn best when they can experiment iteratively, which we also call “practicing.” That’s why loops make sense. There’s a lot of science out there about how to train, how to practice (and also a lot of educational theory that overlaps hugely), and your game will be better if it follows some of those guidelines.

People learn best when the problem they are tackling is right past the edge of what they can do. If it’s too far past that edge, they may not even be able to perceive the problem in the first place! And if the reverse is true and they see a solution instantly, they’ll either be bored, or they might just do that over and over again and never develop any new strategies and not progress.

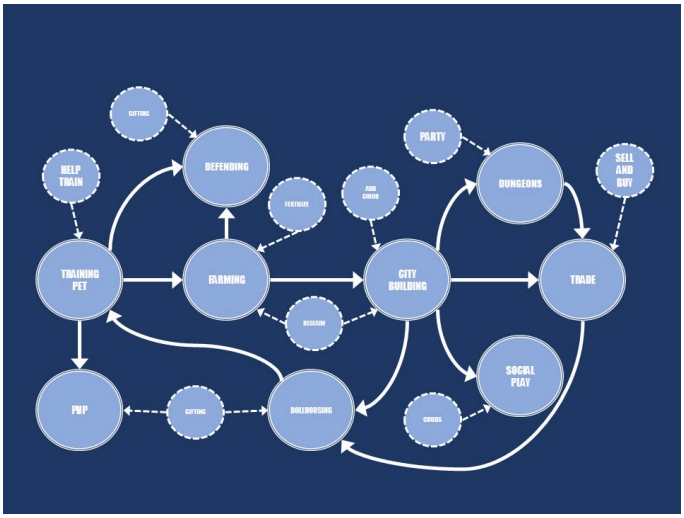


There's an optimal pacing shape. It looks just like what you see in your literature textbooks when they diagram tension, or whatever: sort of like a rising sine wave. You start slow, then speed up, hit a peak challenge, then back off a bit, give a breather that falls back but not all the way, then speed up… we have conventions for what to put at those peaks (bosses!). But what matters is the shape of the curve.

You need to structure your game so that you push players up. They might need to climb the curve at different paces, which is why you might also have difficulty sliders. They might not be capable of getting all the way to the top, and that's okay.

You also need to pace to allow room for everything that *isn't* mastering the problem — such as having fun with friends socially. But at the same time, things to do in the game need to come along at the right pace too!

Bottom line: Vary intensity and pressure, give players a chance to practice and moments to be tested.

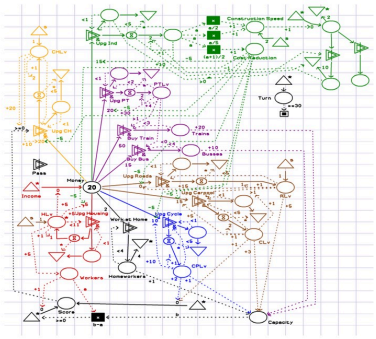


Eight: Games are made of games

Remember the game about clicking on a stick that appeared at a random location on screen? That's also a rail shooter. You move the mouse and click on a spot in 2d space. Which is also not that different from an FPS — only now you move the camera, not the cursor.

Almost no games are made of only one loop. Instead, we chain loops together – complete loop A, and it probably outputs *something* that may serve as a tool or constraint on a different loop.

An FPS has the problem of moving the camera (instead of the mouse) to click on the stick. It also has a loop around moving around in 3d space. Moving around is actually made of several loops, probably, because it may be made of running and jumping and spatial orientation. Those are all problem types!



IF IT'S MATH, IT CAN BE GRAPHED.

THE MOST EXTENSIVELY DEVELOPED METHOD IS MACHINATIONS, WHICH IS USED FOR PROBLEMS EXPRESSIBLE AS ECONOMIES AND CALCULUS (BASICALLY, MANAGEMENT OF RATES OF CHANGE).

We speak sometimes of *value chains*: that's where one loop outputs something to the next loop. We speak also of *game economies*, which is what happens when loops connect in non-linear ways, more like a web. This is not the sort of economy where you are simulating money or commerce. Instead it's a metaphor for stocks and flows and other aspects of actual system dynamics science. In this view, your hit points is a "stock" or, if you like, a "currency" you spend in a fight.

Games nest fractally, they web into complex economies, and they unroll chains of linked loops. That's why they can be diagrammed in a multitude of ways.

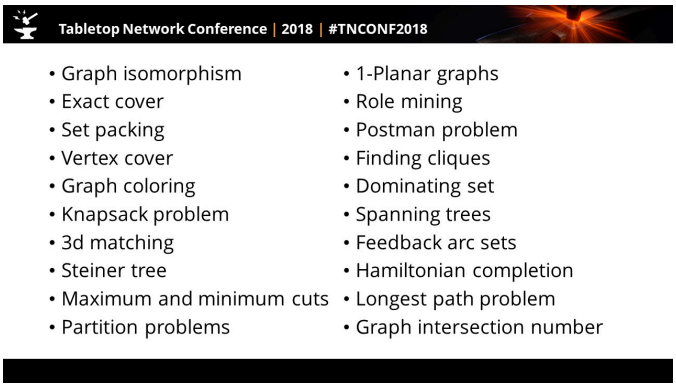
At heart though, you can decompose them all into those elemental small problems, each with an interaction loop and a learning loop centered on that problem.

Bottom line: build small problems into larger webs, and map them so you understand how they connect.

Nine: Actual systems design

The common question is "okay, so how do I design a problem like that?" And that is indeed the unique bit in games, because the other items here are common to lots of other fields.

The list of possible problems is, as mentioned, enormous. This is a big rabbit hole. And once you consider that you can stack, web, and otherwise interlink problems, it means that there's a giant composable universe of games (and game variants) to create.



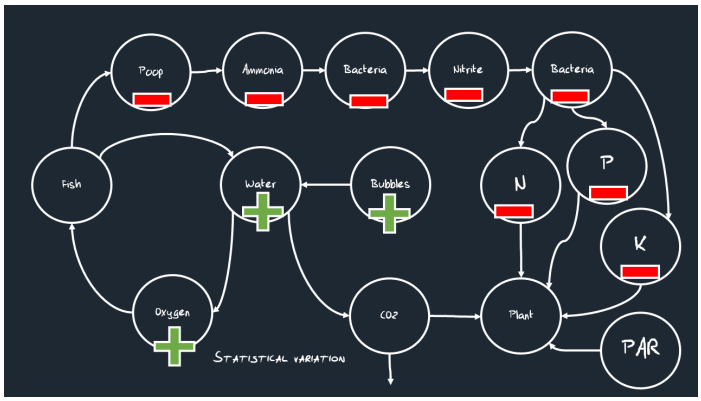
Just bear in mind that because of varied tastes and experience, the diversity of the set of problems you pose is going to affect who wants to play your game.

There are basically a set of categories of problems that we know work, and this is the absolute simplest version of them:

- Mathematically complex puzzles
- Figuring out how other humans think
- Mastering your body and brain

These break down into a ton of sub-problems, but there are less than you think, and you can actually find lists of them. The hard part is that often they each seem so small and trivial that we don't think of them as actually being worth looking at!

They are also often in disguise: the problem behind where a tossed ball will land, and the problem of how much fuel you have left in your car if you keep driving at this speed, and the problem of when your hit points will run out given you have a poison status effect on you *are the same thing*.

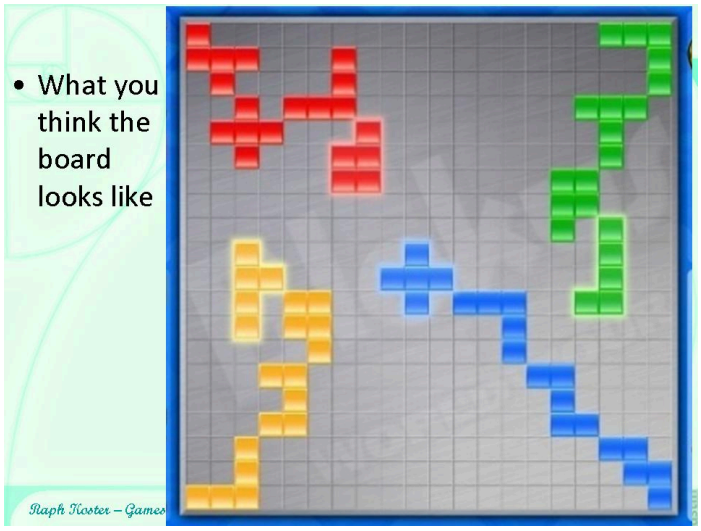


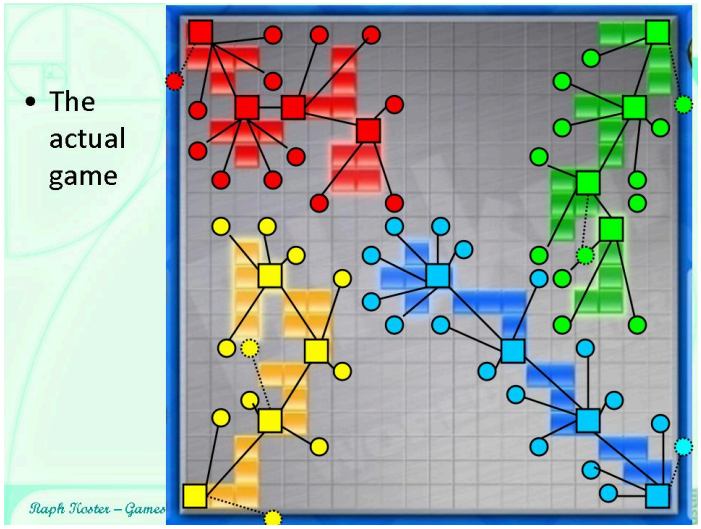
But the more of them you as a designer have wrapped your head around, the more you can combine. And you'll find them very plastic and malleable. In fact, you could almost make a YouTube video about *each one*.

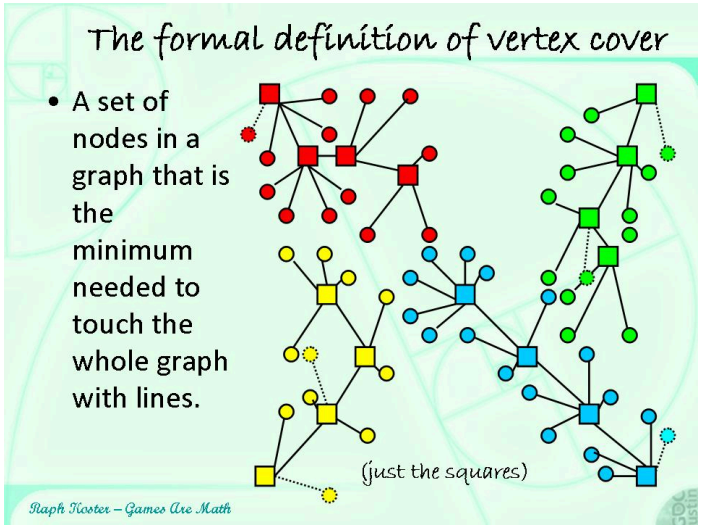
So where do you get them? Steal them. Other games, sure, but also, the world is full of systems that pose tough problems. You can grab them and reskin them.

Bottom line: not every mechanic has been invented, but a ton have. Build your catalog and workbench.

Ten: Dressing and experience

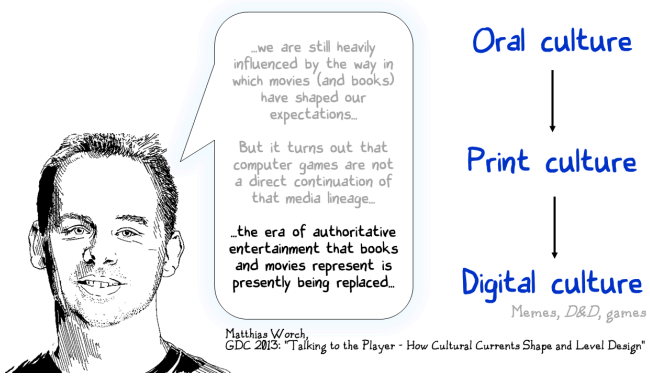






In the end, the feedback layer of a game is *everything* about how you present it. The setting, the lore, the audio, the story, the art…

How you dress up the problems can change everything about how the player learns from it, and how they perceive the problem. The exact same underlying problem can be as different as picking up sticks or shooting someone in the face, or as mentioned, the calculus problem of estimating the trajectory of a variable in a system of rates of change (the ball, the car and its gas, the hit points and poison) might be the same but dressed extraordinarily differently.



When you think about how you dress up the problems, you are in the realm of metaphor. You are engaging in painting, poetry, and music composition, and rhetoric, and the bardic tradition, and all that other humanities stuff.

This is a giant and deep universe for you as a designer to dive into. A lot of this stuff gets called "game design," but then again, we also often say that a given game designer is a frustrated moviemaker, too.

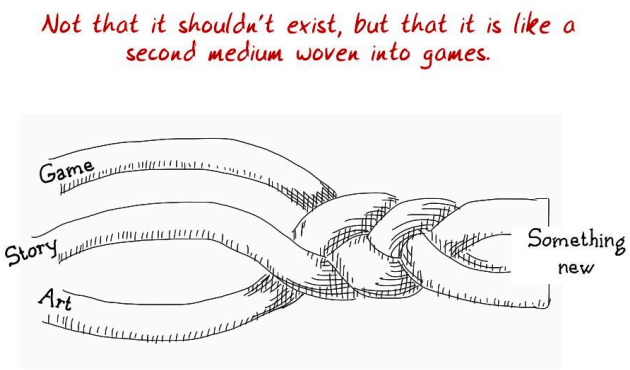
It is really easy to create an experience that clashes with the underlying problems it is teaching. There are fancy critical terms for this. You also need to be very conscious about whether you are building your game so that you are telling the player a story, or so that the player can tell stories with your game.

So the takeaway should be: this stuff is deeply, deeply synergistic with the "game system" stuff that this article is about, but *they are not the same thing.* And games is *not the best place to learn how to do these things.*

Those other fields have much longer traditions and loads of expertise and lessons. They won't all apply to the issue of "how do I best dress up this collection of problems" but most of them will.

It does not frickin' matter if you start out wanting to make interesting problems, or if you start out wanting to provide a cool experience. You are going to need to do both to make the game really good.

Bottom line: game development is a compound art form. You can go learn those individual arts *and* the part unique to games.

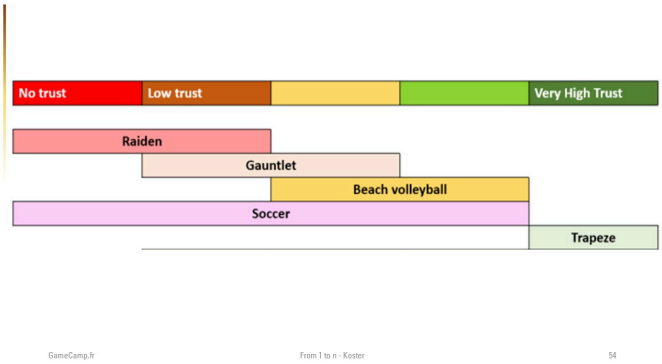


Eleven: Motivations

Researchers have done a ton of studying “why people play games.” This gets called “motivations.”

Motivations are basically about people’s personal taste for groups of problems and how those problems are presented, and characteristics of those problems and the situations in which you find them. Some people like problems where you destroy stuff. Others like problems where you bond with others. Some have trouble trusting other people. Others want to cooperate.

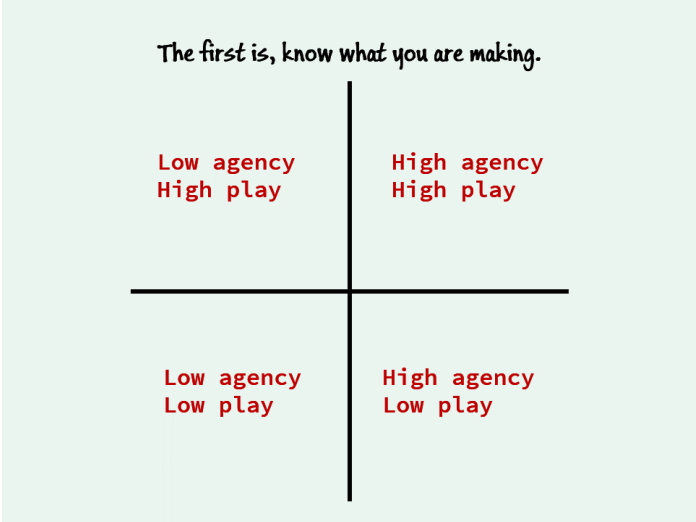
Not everyone likes the same sorts of problems or the same sorts of dressings. Some of this is down to personality types, some of it is down to social dynamics, how they were raised, what their local culture is like, what trauma they have had, and countless other psychological things. That’s why one fancy term for this is *psychographics*.

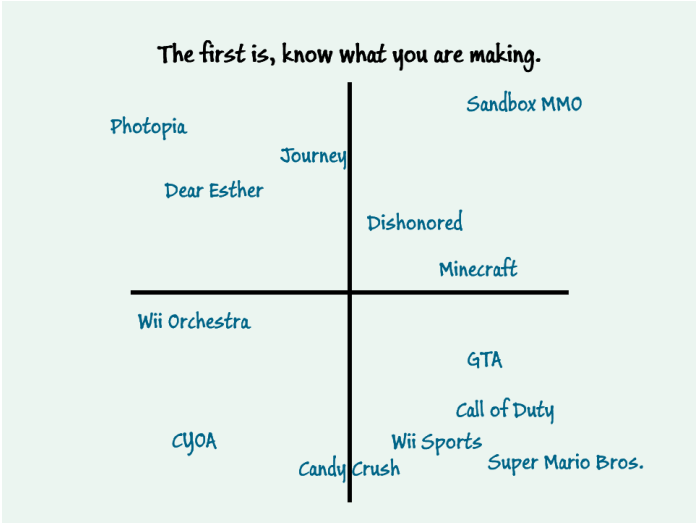


The big thing is, it's not enough that the problems need to not be obvious to you, and also not be baffling to you. They also have to be *interesting* to you. What problems fit in that range is going to depend entirely on who you are, what your life experiences have been, what skills you have, and even what mood you are in.

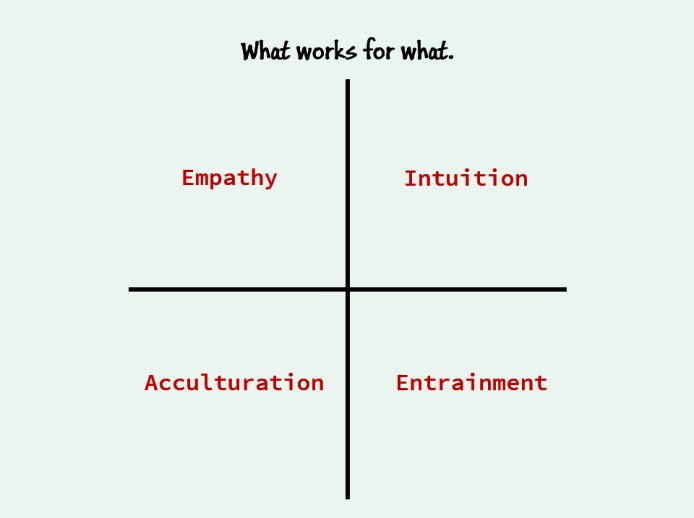
Picking motivations and selecting problems based on them is a great way to design. But motivations are not the same thing as fun. They're a filter, useful in marketing exercises and in building your game pillars (which is an exercise in focus and scope).

Scientists have spent a bunch of time surveying tons of people and have arrived at all sorts of conclusions that map people onto reasons to play and from there onto particular problems.


The first is, know what you are making.
Low agency
High play
High agency
High play
Low agency
Low play
High agency
Low play


The first is, know what you are making.
Sandbox MMO
Photopia
Journey
Dear Esther
Dishonored
Minecraft
Wii Orchestra
GTA
Call of Duty
CYOA
Wii Sports
Candy Crush
Super Mario Bros.


These are marketing categories too.
Sandbox MMO
Photopia
Art game
Journey
Sandbox
Dear Esther
Dishonored
Immersive game
Story game
Minecraft
Wii Orchestra
GTA
Kids' game
Call of Duty
Arcade game
CYOA
Wii Sports
Candy Crush
Super Mario Bros.



If you start with motivations, then you can go from there to types of problems, types of experience, and even player demographics. And then, if you want problems that are about interacting with people, well, there's lists of those. If you want problems that are about managing resources, or solving math issues, there's lists of those too.

Bottom line: no game is for everyone, so you will make better games if you know who you are posing problems for.

Twelve: It's simple, but not

I run into game developers who do not understand the above eleven steps *all the time*. And understanding all eleven is *more valuable* than building expertise in just one, because they depend on one another. This is because getting any one of the eleven wrong can break your game. The real issue is that each of these eleven things is often multiple fields of study. And yeah, you do need to become expert in at least one.

To pick one example, some of us have been working out the rule set for how you can link loops into a larger network of problems for literally over twenty years.

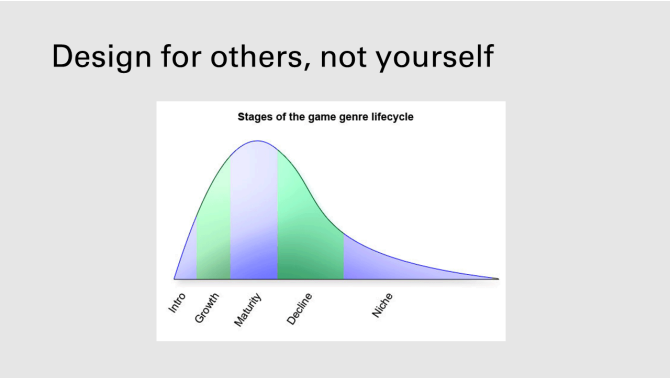
Others have spent their entire career doing nothing but figuring out how best to provide *just the affordances part* of feedback.

So game design is pretty simple. But the devil is in details that are not very far below the surface. It's fairly easy to explain why something is fun for an given audience. It is much harder to build something new that is fun for an arbitrary person. That said, every single one of those fields has best practices, and they are mostly already written down. It's just a lot to learn.

Put another way — *every single paragraph in this essay could be a book.* Actually, probably already is several.

Bottom line: each of these topics is deep, but you want a smattering of all of them.

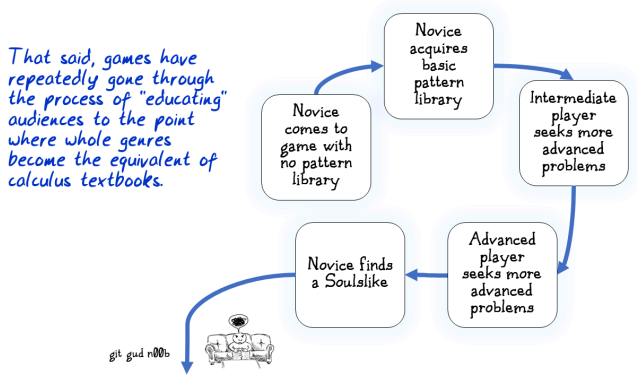
Some of you may not like this deconstructive view on how games are designed. That's okay. Personally, I find it best to poke and prod at a problem, like "how do I get better at making games?" and treat it as a game. And that's what I have done my whole career. The above is just my strategy guide. Someone else will have different strategies, I guarantee it.



But I also guarantee that if you get better at the above twelve things, you will get better at making games. This is a *pragmatic* list. And it will be helpful for making narrative games, puzzle games, boardgames, action games, RPGs, whatever. I breezed through it, but there are very specific tools you can pick up underneath each of these twelve things. It really is that simple, but also that hard, because that's a frickin' long list if you want to actually dive into each of the twelve.

What that also means is that people designing games *fail a lot at it*. You might say, "can't they just do the part they know how to do, and therefore predictably make good games?"

No, because *players learn along with the designers.* If you just make the same game, the one you know how to make, the players get bored because it's nothing but problems they have seen before and already have their answers to. Sometimes, they get so bored that an entire genre dies.



And if you instead make it super-complicated by adding more problems, it might dissolve into noise for most people. Then nobody plays it. And then the genre dies too!

Game designers will routinely fail at making something fun. **When the game of making games is played right, it is always right outside the edge of what the designers know how to do.**

That's where the fun lives, not just for the designer, but also for their audience.

That’s it, the whole cheat sheet. That’s it.

Hope it helps.