Hiring for Games

https://gerolds.github.io/textbook/textbook/posts/hiring/

Contents

Hiring for Games

A “good hire” is situational. It fills a gap that actually exists and augments whatever DNA (https://gerolds.github.io/posts/studio-dna/) the studio has developed. This article is about what that means in game development at scales of 4–40 people, where every person is structurally load-bearing and the wrong hire compounds into every decision the team makes. It assumes that making a widely appealing (https://gerolds.github.io/posts/wide-appeal/) game is very difficult, and that the wrong person in a small team is a structural problem, not a social one.

1. What this article covers

- **Hiring as design decision** — why adding a person changes what the game can become, not just how fast it ships.
- **Cost of a wrong hire** — coordination tax, taste dilution, morale erosion, commitment drift.
- **The pathologies, named** — title inflation, all talk, not getting it, complacency, wrong questions.
- **How hiring goes wrong** — comfort, referral traps, shallow search, proximity, fear of competence, lowballing, wishful thinking, self-rationalisation, reorganising, overlooking existing skills, not taking it seriously.
- **Functions, not job titles** — commitment keeper, eye, author, loop guardian, architect, skeptic’s advocate, bridge, unblocker.
- **Scale changes what breaks** — 4–8, 8–20, 20–40, and the checks-and-balances principle.
- **Taste** — what it actually means, how it manifests, why it scales inversely with headcount.
- **Extraordinary skill** — why process alone is insufficient, why extraordinary skill matters, and why it emerges from the right combination of people and circumstances, not just from a single individual.
- **Individual traits** — orientation, practice, opinions, alternatives, order, maturity, ambition.

- **When not to hire** — scope, process, and commitment problems that hiring cannot solve.
- **When to exit** — the signals, and how to act on them.
- **How to evaluate** — the nose, evidence trails, live tests, trials, and what to watch for.
- **The unifying principle** — the one question that governs every hiring decision.

---

2. Why hiring is a design decision

Most studios treat hiring as an operational problem: we're behind schedule, so we need more hands. Adding hands increases coordination cost, dilutes culture, and introduces someone who doesn't yet share the team's mental model of the game. If the reason for hiring is "we're behind," the real question is usually about scope, commitment, or process.

Hiring is a design decision. Every person you add changes what the team can perceive, what it values, what it will fight for, and what it will let slide. A programmer who thinks in systems will pull the game toward systemic design. An artist who thinks in spectacle will pull it toward set pieces. A designer who thinks in retention will pull it toward loops that measure engagement rather than delight. None of these pulls are wrong in isolation. All of them are wrong if they fight the governing commitment (https://gerolds.github.io/posts/making-the-thing/).

The useful question is not "is this person talented?" but *does this person's talent push the game toward its commitment, or away from it?*

Hiring against a generic rubric ("five years of Unity experience, shipped two titles") produces generic teams. The rubric measures capability in the abstract. It does not measure fit with the specific problem the studio is solving, the specific loop (https://gerolds.github.io/posts/prototyping-the-loop/) it is trying to prove, or the DNA it has accumulated.

3. **The cost of a wrong hire at small scale**

In a large organization, a wrong hire is absorbed. The team routes around them. HR processes eventually correct the mistake, months later, at a cost that is real but survivable.

At 4–40 people, a wrong hire is not absorbed. It metastasizes.

The coordination tax. Every person on a small team is in every conversation, or should be. A person who doesn't understand the commitment forces the team to re-explain it constantly. Meetings get longer. Decisions get relitigated. The team starts spending energy on alignment that should be spent on the game.

The taste dilution. Small teams rely on shared taste to make thousands of micro-decisions without explicit coordination. When someone on the team has different taste and lacks the self-awareness to recognize the mismatch, the game develops inconsistencies that compound. An artist who defaults to a different visual language. A designer who instinctively adds systems the game doesn't need. A programmer who over-engineers because that's what "good code" means to them. Each decision is locally defensible. The accumulation is incoherence.

The morale erosion. Strong people know when they're working alongside someone who isn't carrying weight or isn't pointed in the right direction. They won't always say it directly. They'll say it by disengaging, by quietly doing the other person's thinking for them, by losing the energy that comes from working with people who make you better. The best people leave first, because they have options.

The commitment drift. A person who doesn't fully understand or believe in the commitment will subtly redirect the project toward what they do understand. This is rarely malicious. It's gravitational. They advocate for what they know how to build. They ask questions that assume a different kind of game. They push back on cuts that serve the commitment because the commitment isn't real to them. Over months, the project drifts. By the time it's visible, the cost of correcting it is enormous.

At small scale, you cannot afford a person who is merely not making the game worse. Neutral is negative, because every seat

that isn’t generating leverage is consuming the attention and energy of people who could be.

4. **The pathologies, named**

Wrong hires in game teams tend to cluster into recognizable failure modes. Naming them is useful because they are often invisible until the damage is done, and because each one has a surface that looks professional or even impressive.

4.1. **Title inflation**

The person whose seniority exists on paper. They have the title (Lead, Director, Senior) but not the decision-making instinct, the speed, or the craft to back it up. They were promoted in a larger organization where the title meant “manages people” rather than “makes the hard calls that determine whether the game is good.” In a small team there is nowhere to hide. The title creates an authority the work doesn’t support, and the team either defers to bad judgment or spends energy working around it.

The tell: They talk about process more than product. They organize meetings about the game more than they improve the game. They describe their role in terms of oversight rather than output.

4.2. **All talk, no game**

The person who is articulate about game design, fluent in the vocabulary, well-read, opinionated in meetings. But when it comes to building, testing, iterating, and shipping, the output is thin. They can describe what a game should be. They cannot make it be that.

In game development, articulateness is seductive because the problems are genuinely hard to talk about. Someone who can name what’s wrong feels valuable. But naming is not solving. The gap between “I can see the problem” and “I can fix the problem in the build by Friday” is the gap that matters. Small teams need people who close that gap daily.

The tell: Their best work is in documents, decks, and conversations. The game does not visibly improve when they work

on it. Their critiques are accurate but they cannot demonstrate the alternative.

4.3. **Simply not getting it**

The person who is competent at their craft but does not understand *this* game. They can model, animate, code, or design to a professional standard, but the decisions they make don't serve the commitment. They produce work that would be correct in a different game. When given feedback, they adjust the surface but not the intent.

This is the hardest pathology to address because the person is not bad at their job. They are bad at *this* job. The mismatch is between their instincts and the project's needs, and no amount of feedback will fix it if the person's instincts are calcified.

The tell: Their work requires frequent and fundamental redirection, not polish notes. The team spends more time explaining *why* than *how*.

4.4. **Complacency**

The person who has shipped games before and believes that shipping is the hard part. They have a way of working that produced results in the past, and they see no reason to change it. They are resistant to new tools, new methods, and new information. They treat their experience as authority rather than as a prior that needs updating.

In a fast-moving medium where the audience, the tools, the platforms, and the competition change yearly, experience older than five years should be treated as hypothesis, not fact. The complacent hire treats it as armor.

The tell: "That's how we did it at [previous studio]." Resistance to prototyping. Discomfort with ambiguity. A preference for building what they already know how to build rather than what the game needs.

4.5. **Asking the wrong questions**

The person who is diligent, engaged, and well-intentioned, but whose mental model of the problem is misaligned. They ask questions about retention when the game hasn't proven its core loop. They worry about monetization when the promise isn't legible. They optimize for metrics when the game needs soul.

This pathology is insidious because the questions sound professional. In a different context, they'd be the right questions. But questions reveal priorities, and priorities shape the game. A team full of people asking the wrong questions will build a game that answers them, which is how you end up with a technically proficient, market-aware product that nobody loves.

The tell: Their concerns consistently skip ahead of the project's actual stage. They solve for problems the game doesn't have yet while ignoring the problems it does.

5. **How hiring goes wrong: management pathologies**

The previous section named what's wrong with the people you hire. This section names what's wrong with the people *doing* the hiring. These are failures of the process, the decision-makers, and the culture around hiring itself. They are at least as damaging as a bad hire, because they produce bad hires *systematically*.

5.6. **Hiring for comfort**

This is often the least visible failure. The person doing the hiring picks someone who feels safe: similar background, similar temperament, easy to talk to, unlikely to challenge the status quo. The interview feels smooth. Everyone gets along. The hire is pleasant, agreeable, and does not push the team to be better.

Comfort hiring is seductive because it produces a functional team. People communicate well. Meetings are cordial. The problem is that cordial teams with aligned blind spots will march in formation toward mediocrity and nobody will raise the alarm, because raising alarms is uncomfortable, and the team was built to avoid discomfort.

Pay attention when a candidate makes you slightly uneasy because they're sharper than you expected, or because they challenged something you said, or because their taste is strong enough to create friction. That unease is information. It might mean "wrong fit." It can also mean "better than what we have."

5.7. **The referral trap**

Someone on the team knows someone. The someone is available. The recommendation skips the evaluation process or biases it heavily, because saying no to a referral means saying "I don't trust your judgment" to a colleague.

Referrals are not inherently bad. The best hires often come through networks, because strong people tend to know strong people. The pathology is when the referral *replaces* evaluation rather than *initiating* it. The referred person must be held to exactly the same standard as a stranger. If the team cannot do that honestly, the referral process is a liability.

The deeper trap: referral networks are self-similar. People refer people like themselves. A team that hires primarily through referrals will converge on a type, and that type will have the same strengths and the same blind spots as the people who already exist. Diversity of thought, background, and instinct has to be *sought*, because it will not arrive through referrals.

5.8. **Not actually looking**

The search is perfunctory. The team posts a job listing, waits for applications, skims the top of the pile, and picks from whoever showed up. This feels like hiring. It is actually filtering a self-selected pool of people who happened to see the listing and felt confident enough to apply.

The people you most need, the ones with exceptional taste, deep craft, and strong options, are almost never in that pool. They are employed. They are not looking. They will not find your listing. If you want them, you have to go find them: through their work, through their writing, through their side projects, through people who have worked alongside them and can tell you what they're like when the deadline is tomorrow and the build is broken.

Active search is expensive. It is also the only way to hire above the median. If you are not spending meaningful time on sourcing, you are hiring from whatever washed up on the shore.

5.9. **Proximity hiring**

The person who once touched something vaguely related to what you need. They used Unity once, so they're a "Unity developer." They shipped a game that had multiplayer, so they "have multiplayer experience." They worked at a studio that made an RPG, so they "understand RPG design."

Proximity is not competence. Having been in the building where a thing happened is different from having done the thing. Ask them to *demonstrate*, not just describe. If their experience is real, the demonstration will be obvious. If it's proximity, the gap will be visible within minutes.

This pathology thrives when the hiring team itself lacks depth in the area they're hiring for. If nobody on the team can distinguish real multiplayer expertise from proximity to multiplayer, the proximity hire will look like an expert. This is the calibration problem from the excellence section, appearing in a different guise: you cannot evaluate what you have never seen done well.

5.10. **Fear of competence**

This is one of the most corrosive pathologies, and almost nobody admits to it. The person doing the hiring, consciously or not, avoids candidates who are clearly better than they are. The truly excellent candidate makes the interviewer feel inadequate. Their answers are sharp. Their portfolio is strong. Their questions expose gaps in the interviewer's own understanding. So the interviewer finds a reason to pass: "not a culture fit," "might be hard to manage," "overqualified."

This is how studios maintain a ceiling. The strongest person on the team determines the maximum quality of anyone they'll hire. If that person is threatened by excellence, the team can never exceed their level. The ceiling lowers with every hire, because each new person was chosen to be non-threatening to the person above them.

The fix has to be structural: the people making hiring decisions must be secure enough to hire people better than themselves, and the studio must *reward* them for doing so rather than treating it as a threat to the hierarchy. A lead who hires someone who outgrows the lead's role has done the studio a favor. The studio should treat it that way.

5.11. **Lowballing**

Offering below-market compensation because the candidate seems eager, or because the studio has budget pressure, or because "the opportunity to work on a great game" is supposed to make up the difference.

Lowballing is a filter. It filters out everyone with options, which is to say everyone you actually want. The people who accept below-market offers are either desperate (and desperation is not a hiring signal you want to select for), early-career enough that they don't know their worth (which means you are exploiting a temporary information asymmetry), or bad at negotiating (which tells you nothing about whether they can make games).

Pay people what the work is worth. If you can't afford to pay market rate, hire fewer people and pay them properly. Two people at full rate will outperform three people who know they're being underpaid, because underpaid people are already looking for the exit.

5.12. **Wishful thinking**

"They haven't done this before, but I think they could." "The portfolio is weak, but the interview was great." "They don't have the experience we need, but they're a fast learner."

Wishful thinking is the gap between what the evidence shows and what the hiring manager *wants* to believe. It thrives under time pressure (we need someone now), scarcity (we can't find anyone better), and affinity (I liked them). Each of these is a real pressure. None of them change whether the person can do the job.

Separate what you *observed* from what you *inferred*. Write down the evidence. Read it without the narrative. If the evidence alone

does not support the hire, the narrative is wishful thinking, and the hire will likely fail in the ways the evidence predicted.

5.13. **Self-rationalising bad decisions**

The hire is made. The first month is rough. The work isn't at the level the team needs. The integration is awkward. And the person who championed the hire starts rationalising: "They're still ramping up." "The brief wasn't clear enough." "They just need more time."

Some of this is legitimate. People do ramp up. Briefs are sometimes unclear. But rationalisation and patience look identical from the outside, and the person who made the decision is the worst judge of which one they're engaged in, because admitting the hire was wrong means admitting their own judgment failed.

Make the evaluation of a new hire someone *else's* responsibility. The person who championed the hire should not be the one who decides whether it's working. They have too much invested in the answer being yes. Give the evaluation to someone with a calibrated nose, or to a lead who wasn't involved in the hiring decision, or to the team as a whole.

5.14. **Reorganising instead of recognising**

A new hire arrives and the studio reshuffles roles, processes, or reporting lines to accommodate them. The reasoning sounds sensible: "Now that we have a dedicated X, let's restructure how we handle Y." But the restructuring was not needed before the hire, and the disruption it causes, the lost momentum, the confused ownership, the politics of changing someone's role, can cost more than the new hire contributes.

The instinct to reorganise around a new person is a warning sign that the team didn't have a clear gap. They had a vague sense that something was missing, hired someone who seemed interesting, and are now trying to figure out where to put them. The hire should fit an existing gap, not create a new shape that the team then has to contort around.

The adjacent failure: changing the *process* to suit the new person's preferences. "At my last studio, we used X methodology." If X

methodology is genuinely better, that conversation should happen independently of who proposed it. If it’s happening because the new person is accustomed to it and the team is being accommodating, you are letting a new hire rewrite your operating system before they’ve proven they understand this game.

5.15. **Not seeing the skill that’s already there**

The team hires for a function that someone on the team already performs, just without the title. The existing person has been doing the work informally, effectively, and with deep context. The new hire arrives with the title and the mandate, and the existing person is quietly displaced.

This demoralises the person who was doing the work. It wastes the context they’ve accumulated. And it often produces worse results, because the new hire has the title but not the history, while the person who has the history no longer has the authority.

Before hiring for any function, ask: is anyone already doing this? If so, would formalising their role and giving them support be better than bringing in a stranger? The answer is often yes, and the cost is a fraction of a new hire.

5.16. **Not taking hiring seriously**

This is the pathology that enables most of the others. The team treats hiring as an interruption to the real work rather than as one of the highest-leverage activities the studio performs.

The symptoms: job descriptions written in ten minutes. Interviews conducted without preparation. Portfolios skimmed rather than studied. References not checked. Trials not done because “we don’t have time.” Decisions made in a single meeting based on gut feel.

Every one of these shortcuts is a bet that the hiring decision doesn’t matter enough to invest in. At 4–40 people, hiring is one of the few decisions that permanently changes the team’s capability, taste, and trajectory. Treating it as overhead is treating the team’s future as overhead.

The studios that hire well treat the process as seriously as they treat any other design problem. They invest time, thought, and

structure, not because they enjoy bureaucracy, but because they understand that the people in the room determine the game, and getting the wrong people in the room is the most expensive mistake the studio can make.

6. **What the team actually needs: functions, not job titles**

A common mistake is to think in job titles. “We need a level designer.” “We need a character artist.” “We need a UI programmer.” These are descriptions of craft skills, not descriptions of what the team is missing.

The right frame is *functions*: what does the team need to be able to do that it currently cannot? A function might map to a job title, or it might map to a capability that an existing person could develop, or it might map to a responsibility that needs to be explicit rather than assumed.

At the scale of 4–40, the following functions must exist. They do not all need to be separate people. In a team of four, one person might carry three of them. In a team of forty, some might be shared across two people. But every function must be *owned*, meaning someone is accountable for it and the team knows who.

6.17. **The commitment keeper**

Call it Game Director, Creative Director, whatever. The function is: *hold the governing commitment stable and make it the basis for every cut, every addition, every prioritization.* This person creates coherence by being the one who says no. Not from ego, but from an internalized understanding of what the game is trying to be that is deep enough to apply under pressure, ambiguity, and fatigue.

This function fails when the person is a visionary without judgment (they add but cannot subtract), or a manager without vision (they keep the trains running but can’t tell you where they’re going).

6.18. **The eye**

Art Director. The function is: *make the game readable and visually distinctive at every scale, from a screenshot to a five-second clip to*

an hour of play. Readability first, beauty second, because a beautiful game that players can't parse is a beautiful failure.

This person coaches artists not just on technique but on intent. They can see far enough into the production to know what the game will need to look like in six months, not just what it looks like today. They understand that visual identity is not a style guide but a continuous act of judgment applied to every frame the player sees.

6.19. **The author**

Writer, Narrative Designer. The function is: *create a world, a voice, and a context that makes the player want to explore and linger.* Not "write dialogue" or "produce lore documents." The function is authorship of the player's imaginative experience.

This person coaches the entire team on what the player's mind is learning, moment to moment. What is the player understanding? What are they anticipating? What are they misunderstanding productively? The author thinks in terms of the player's internal model of the game world, which is a design concern as much as a narrative one.

6.20. **The loop guardian**

Game Designer, Systems Designer. The function is: *protect the core loop, the invariant (https://gerolds.github.io/posts/prototyping-the-loop/), the moment-to-moment experience that is the game.* This person owns the question "is this actually enjoyable to do?" and has the authority and taste to answer it honestly.

They understand progression, pacing, difficulty, and reward, but they understand them as tools in service of the loop, not as systems to be optimized in isolation. They are the person who notices when the game is fun to describe but not fun to play.

6.21. **The architect**

Tech Lead, Technical Director. The function is: *build the leanest possible technology that realizes the vision, solves the team's*

workflow problems, and maintains velocity across the entire lifecycle.

This person directs the transition from prototype to production. They own pipelines, build systems, platform requirements, and the daily experience of “how long does it take to test a change.” They coach programmers. They can see far enough ahead to avoid architectural dead ends without over-engineering for futures that may never arrive.

A critical and underappreciated part of this function: *the architect determines the team’s iteration speed.* A team that can test a change in thirty seconds learns ten times faster than a team that waits five minutes for a build. The architect’s taste in tooling directly affects the quality of the game, because it determines how many cycles of Promise → Proof → Cut → Ship (https://gerolds.github.io/posts/studio-os/) the team can run before the money runs out.

6.22. **The skeptic’s advocate**

QA. The function is: *ensure there is nothing in the game that would cause a player to leave a negative review for a reason the team could have caught.* This goes beyond finding bugs. The function is adversarial empathy: playing the game as a stranger would, with no patience, no context, no loyalty, and reporting what breaks the contract.

6.23. **The bridge**

Community Manager. The function is: *maintain a genuine dialogue between the team and its audience.* This person translates player language into design-actionable information and translates team decisions into language players can respect, even when the news is unwelcome.

6.24. **The unblocker**

Producer. The function is: *remove whatever prevents other people from succeeding in their roles.* Schedules are one tool. The broader function is identifying and eliminating friction: unclear priorities, missing information, broken tools, unresolved decisions, interpersonal conflicts that drain energy from the work.

The best producers are invisible when things are working and decisive when things aren’t. The worst producers add process that creates the illusion of control without improving outcomes.

7. **Scale changes what breaks**

The functions above are constant. What changes with team size is *how they fail.*

7.25. **At 4–8 people**

Every person carries multiple functions. The game director is also the designer. The artist is also the UI person. The programmer is also QA.

What breaks: Gaps. Not “we need more people” gaps, but “nobody is actually responsible for this” gaps. The most common: nobody owns the player’s first-time experience. Nobody owns legibility. Nobody owns the question “would a stranger understand this?” Everyone is so deep in building that they’ve lost the ability to see the game as a newcomer.

What to hire for: Breadth over depth. At this scale, you need people who can think across disciplines, who notice problems outside their specialty, who can do passable work in two adjacent areas while excelling in one. A programmer who can do placeholder art. An artist who understands game feel. A designer who can script gameplay without waiting for an engineer.

The critical trait: Self-direction. There is no management layer. If a person needs to be told what to do next, they are consuming leadership bandwidth that doesn’t exist.

7.26. **At 8–20 people**

Functions begin to separate into dedicated roles. Specialization becomes possible. Communication overhead becomes real.

What breaks: Silos. The art team starts making art that is beautiful but doesn’t serve the design. The engineering team starts building systems that are elegant but don’t solve the right problem. The designer writes documents that nobody reads

because the document isn’t the game. People start optimizing for their craft rather than for the game.

What to hire for: Depth with integration. You can now afford specialists, but they must be specialists who understand how their work connects to everything else. An animator who thinks about game feel, not just motion quality. A systems programmer who thinks about designer workflow, not just code architecture. A level designer who thinks about narrative pacing, not just spatial composition.

The critical trait: Communication. At this scale, people can no longer rely on overhearing everything. Information must be actively shared. A person who does excellent work in isolation but doesn’t surface their reasoning, their blockers, or their discoveries is creating invisible risk.

7.27. **At 20–40 people**

Hierarchy becomes necessary. Leads emerge. Sub-teams form. The commitment is now mediated through layers.

What breaks: Coherence. The game director cannot personally review every decision. The commitment must be internalized by leads, who must internalize it deeply enough to apply it independently. If the leads don’t carry the commitment in their bones, the sub-teams will each build a slightly different game, and the seams will show.

What to hire for: Leadership taste. The leads you hire at this scale determine the quality of fifty downstream decisions per day that the director will never see. A lead with the wrong instincts is not one bad decision but a hundred bad micro-decisions, compounding daily, invisible until integration reveals the damage.

The critical trait: Judgment under autonomy. At this scale, the lead must be able to make decisions that the director would agree with, without asking. This requires not just competence but a shared mental model of what the game is trying to be. That shared model is the real thing you’re hiring for.

7.28. **At every scale: checks and balances**

There is a structural problem that cuts across all team sizes, and it gets worse as the team gets smaller: **when one person holds two functions that should be in productive tension, honest antagonism becomes impossible.**

Functions check each other. The loop guardian pushes for what feels good to play; the architect pushes back on what is feasible to build. The commitment keeper says “cut this”; the author says “this serves the world.” The skeptic’s advocate says “a player will hate this”; the designer says “we’ll teach them.” These tensions are not friction. They are the immune system. When the tension is real, when two separate people with separate accountability genuinely disagree, the outcome is better than either would produce alone.

When one person holds both sides of that tension, the check vanishes. The programmer who is also QA will not honestly adversarially test their own code, because the adversarial instinct and the builder’s instinct cannot coexist in the same moment. The director who is also the designer cannot ruthlessly cut the design, because the design is their work, and cutting it feels like self-harm rather than service to the commitment. The producer who is also the creative lead cannot prioritize the schedule over the vision, because both belong to them and the tradeoff becomes internal rather than structural.

This is a structural impossibility, not a character flaw. No person, however disciplined, can be their own honest opposition.

At 4–8 people, function-merging is unavoidable. There aren’t enough seats to separate every tension. But the team must recognize *which* merges are dangerous and hire specifically to restore the missing antagonism. If the lead programmer is also handling QA, the next hire should probably be someone whose function is adversarial testing, not another programmer. If the creative director is also the lead designer, the team needs someone with the taste and authority to challenge design decisions, not just execute them.

At 8–20, function separation becomes possible. This is the scale where you can start designing the team as a system of checks: the

art director and the tech lead should be in regular, healthy conflict about visual ambition versus pipeline cost. The designer and the QA person should be in regular, healthy conflict about intended challenge versus unintended frustration.

At 20–40, the checks must be formalized into review structures. Playtests, build reviews, cross-discipline critiques. The hierarchy that emerges at this scale will naturally suppress dissent unless the studio deliberately builds mechanisms that protect it.

The hiring implication is concrete: when you look at the team and see functions merged in a single person, ask whether those functions should be in tension. If they should, that tension is currently dead. Reviving it is a higher priority than filling whatever role feels most urgent on the production schedule.

8. Taste: the word that needs defining

Every section above invokes “taste” without defining it. Taste in game development is hard to define and easy to fake, but it is the most important quality in a small-team hire, and its absence is the root cause of most of the pathologies listed above.

Taste in game development is the ability to distinguish between “this works” and “this is right for this game.”

A programmer with taste doesn’t just write code that functions. They write code whose structure reflects the game’s priorities. If the game is about rapid iteration, the code is built for rapid iteration, not for theoretical extensibility. If the game is about feel, the code gives designers direct control over the parameters that affect feel, rather than burying them in abstraction layers.

An artist with taste doesn’t just produce beautiful assets. They produce assets that serve the game’s mood, readability, and visual contract. They know when to add detail and when to subtract it. They can look at a screenshot and tell you what’s fighting for the player’s eye and why that’s a problem.

A designer with taste doesn’t just create systems that function. They create systems that *feel* like they belong in this specific game. They can tell you why a mechanic that would be excellent in

a different game is wrong here. They have the judgment to cut their own best work when it doesn't serve the commitment.

Taste scales inversely with team size. In a team of four, every person's taste is the game's taste. A single person with bad taste (or taste misaligned with the project) will drag the game toward mediocrity because there aren't enough other people to counteract the pull. In a team of forty, a lead's taste shapes a department. Individual contributors can be less taste-aligned if the lead is strong.

This is why the smallest teams need the most extraordinary people. Not the most experienced or the most technically skilled, but the most *aligned in taste with what the game needs to be*, and the most capable of exercising that taste independently, quickly, and under pressure.

How taste manifests in practice:

- The person looks at the game and can name what's wrong without being told.
- The person's instinctive first attempt is in the right direction, even if it needs refinement.
- The person's questions reveal that they're thinking about the player's experience, not just their own deliverable.
- The person pushes back on requests that would weaken the game, with alternatives, not just objections.
- The person's work makes adjacent work better. Other people's contributions improve in the presence of their taste.

How taste-absence manifests:

- The person's work is technically correct but doesn't feel like it belongs.
- The person requires constant redirection on intent, not just execution.
- The person cannot distinguish between their own preferences and the game's needs.
- The person's work makes the game feel like a collection of parts rather than a whole.

9. **The irreplaceable: why games need more than process**

This article has argued for functions, structure, checks and balances, and process. All of that is necessary. None of it is sufficient.

Games that achieve wide appeal repeatedly are not built by well-organized teams of competent people following good process. They are built by teams where extraordinary skill collides with the right circumstances: a particular combination of people whose strengths interlock, a commitment clear enough to focus them, incentives that reward risk over safety, and enough time for serendipity to do its work. Sometimes that collision concentrates in a single person: the architect who solves the pipeline problem nobody else could see clearly enough to frame, the artist who defines a visual identity from nothing, the designer whose instinct for the loop finds the invariant in a fraction of the time a structured search would take, the writer who gives the world a voice that makes strangers want to live there. But just as often, the breakthrough is a *combination*: the designer's instinct sharpened by the architect's constraint, the artist's vision made buildable by the engineer's pragmatism, the writer's world made playable by the loop guardian's discipline. The extraordinary thing emerges between people, not only inside one of them.

What matters is that the capability exists *somewhere in the team's constellation*; whether concentrated in an individual or distributed across a group that, together, reaches a level none of them would reach alone.

Process creates the conditions for this to happen. It gives extraordinary skill a stable commitment to serve, a team that can execute alongside it, and a loop that converts insight into shipped work. But process does not *generate* the insight. A well-run studio staffed entirely with competent, professional, taste-aligned people will produce competent, professional, tasteful games that are missing the thing that makes a stranger stop scrolling.

This matters for hiring because it changes the calculus. You are not looking for a lone savior. You are looking for the person whose addition creates a *combinatorial leap*: someone whose presence

elevates what the rest of the team can do together. When you find someone genuinely exceptional, someone whose skill, perspective, or instinct fills a gap the team cannot fill by reorganizing, the correct response is not “let’s see if they fit the process.” It is to reshape the process to capture what they offer. Such people are rarer than good processes. Processes can be rebuilt.

There is a second reason this matters, and it is less obvious: **a team that has never reckoned with real excellence—either by working alongside it or by studying its absence—cannot calibrate its own standards.** Taste is not abstract. It is learned by exposure, but “exposure” has two forms. The direct form is working alongside someone whose output resets what you thought was possible. The indirect form is working inside environments where excellence is absent, studying why, reverse-engineering the publicly available work that *did* succeed, and building a model of quality from its negative space. (The second path is explored in depth in The Benefits of Failure (https://gerolds.github.io/posts/the-benefits-of-failure/).) Both paths produce calibration. A team whose best reference point is “competent,” and which has never interrogated the distance between competent and great, will hire for competent, lead toward competent, and evaluate work against competent. They will call it good, because they have never been forced to ask what good actually is. Not “I like it” good, but “this makes strangers care, communicates something real, and holds up under the attention of millions” good. That is a different standard, and you cannot aim for it if nobody in the room has ever reckoned with it: through proximity *or* through the hard study of its absence.

A warning is necessary here, because the indirect path is easy to claim and hard to verify—including by the person claiming it. The mind’s machinery of comfort (https://gerolds.github.io/posts/the-price-of-seeing-clearly/) is specifically designed to convert painful experience into a flattering narrative. “I worked in bad studios and I learned from it” is exactly the kind of story the machinery produces. The person who has genuinely built calibration from negative space can do something the self-flattering version cannot: **they can point to specific predictions they made that were later confirmed by outcomes they did not control.** They said “this project will fail because X,” and it failed because X. They said “this mechanic won’t hold,” and it didn’t. They built something small under

constraints and strangers responded. The evidence is external, not internal. If the only proof of someone's calibration is their own conviction that they are calibrated, that is not evidence. It is the comfort machinery running in a new disguise.

This is why the first encounter with real excellence changes a team, whether that encounter arrives as a person, a shipped artifact the team studies together, or a collective breakthrough nobody expected. It doesn't just produce better work. It recalibrates what "better" means for everyone around it. The art director who has shipped a game with genuine visual identity teaches the team what identity means by demonstrating it, not by explaining it. The designer who has found and protected a core loop teaches the team what conviction looks like under production pressure. But the person who spent a decade inside failing studios and *studied every failure* may carry the same calibration, built from the other direction; not from seeing excellence done, but from mapping everything that prevents it. The difference between that person and someone who merely survived the same decade is the trail of external evidence: work that landed, predictions that held, judgments that were tested and not found wanting. Once the team has a reference point for the real thing, however acquired, they can recognize it in candidates, in work, and in themselves. Before that, they are guessing.

The risk: valorizing individual exceptionalism can become an excuse for tolerating dysfunction, ego, or cruelty. It can justify building a team around one person's moods rather than around a commitment. The guard is the same as everywhere else: *does this person make the game more likely to be great?* Someone who makes the game better but makes the team worse is still a net negative at small scale, because the team *is* the game.

Do not hire exclusively for reliability. A team of reliable, professional, process-following contributors will produce reliable, professional, process-following work. It will not produce the moment that makes someone tell their friend about the game. That moment comes from a team that contains the right combination of skill, conviction, and circumstance—and from the willingness to recognize and protect that combination when it appears, however it appears.

10. **What to look for in individuals**

Beyond function-specific requirements and taste, there are traits that matter for every hire at this scale, regardless of role.

10.29. **The orientation toward the problem**

The person must be oriented toward the problem of making *this game good*, not toward their career, their portfolio, their craft in the abstract, or their comfort. When those things conflict with what the game needs, the game wins.

A person oriented toward the problem challenges standard operating procedure. They don't ask "how do we usually do this?" They ask "how should we do this *for this game*?" They treat every inherited process, tool, and convention as a hypothesis, not a given.

10.30. **The practice**

The person builds things outside of work. Games, systems, art, prototypes, writing about games, mods, tools, experiments. The specific form doesn't matter. What matters is that they are practicing. A person who only practices game development during work hours is relying on the project to be their education. At small scale, the project cannot afford to be anyone's school for fundamentals. The project is novel enough; the team's learning must be directed at the *project's* novel problems, not at problems the individual should have solved on their own time.

The point isn't to demand unpaid labor. The point is to identify people for whom game development is a vocation, not just a job. The difference shows up in the quality of their questions, the speed of their learning, and their resilience when the work gets hard.

10.31. **The strong opinions, loosely held**

The person has strong opinions. They can argue them with evidence, examples, and reasoning. But they can also change them when the context demands it. The critical distinction is between *strong opinions* (which produce useful friction and surface

important disagreements) and *rigid opinions* (which produce political deadlocks and ego-driven decisions).

The test: watch what happens when they're wrong. A person with strong opinions loosely held says "I was wrong, here's what I learned." A person with rigid opinions says "I still think I was right, but I'll do it your way." The first person makes the team smarter. The second makes the team tired.

10.32. **No excuses, only alternatives**

The person never says "I can't" without saying "but here's what I can do instead, and here's where it leads." Constraints are real. Resources are finite. Deadlines are immovable. The question is whether the person treats constraints as walls or as boundaries that define the shape of the solution.

This trait is especially important at small scale because there is no slack. When someone hits a wall and stops, the project stops. When someone hits a wall and finds an alternative, the project adapts. The difference between these two responses, multiplied across hundreds of moments per month, is the difference between a team that ships and a team that doesn't.

10.33. **The intolerance for chaos**

The person is irritated by disorder. Good names on everything. Canonical file paths. Clean version control. Adequate documentation. No obsolete assets rotting in the repository. No "temporary" hacks that have been temporary for six months.

Call it pedantry if you want, but it's survival. At small scale, the codebase, the asset library, and the project structure *are* the team's shared memory. When that memory is chaotic, every task takes longer because people spend time finding things, understanding things, and working around things that shouldn't exist. Disorder compounds faster than code.

10.34. **The maturity to resist novelty**

The person is not a child in a candy store. They have enough experience to know that "trying it for the first time" is not a

substitute for judgment. New tools, new engines, new architectures, new art styles: these are temptations that feel like progress but are often lateral moves that reset the learning clock without improving outcomes.

The mature hire evaluates novelty against the question: *does this make the game better, or does it make my experience more interesting?* These are different questions, and the person who can't distinguish them will steer the project toward whatever is shiny.

At the same time, this maturity must coexist with a willingness to change when change is justified. The distinction is between novelty-seeking (change because it's new) and adaptation (change because the evidence demands it). Old assumptions (anything older than roughly five years) should be treated as deeply suspicious until re-validated against current conditions. The tools changed. The platforms changed. The audience changed. The person who insists on doing it the old way because it worked before is not being disciplined. They are being lazy in a way that looks like experience.

10.35. **The scale of the ambition**

The person understands the magnitude of the challenge. Making a good game is hard. Making a widely appealing one with a small team, limited budget, and no margin for error is one of the most difficult creative and technical challenges a person can take on.

The right hire knows this and is not deterred. They are humble in the face of the difficulty but ferocious in getting after it. They accept that they will be a different person at the end: different skills, different understanding, different maturity. If they don't aim to be transformed by the work, they are already a weak spot, because the work *will* demand transformation, and the person who resists it will become a bottleneck.

11. **When not to hire**

The most important hiring decision is often the decision not to hire.

Studios hire when they should cut scope. Studios hire when they should fix their process. Studios hire when they should confront the fact that the commitment isn't working. Hiring feels like progress because it's visible and concrete: a new person, a new desk, a new name on the team page. But adding a person to a broken process does not fix the process. It makes the process more expensive to break.

Don't hire to solve a commitment problem. If the team doesn't know what the game is, adding people will produce more work pointed in more directions. The solution is to find the commitment (https://gerolds.github.io/posts/finding-the-commitment/), not to add bodies.

Don't hire to solve a scope problem. If the game is too big for the team, the answer is almost always to cut, not to grow. A larger team building a too-big game is just a more expensive version of the same problem, with higher coordination costs and slower iteration.

Don't hire to solve a speed problem. If the team is slow, the cause is usually process, tools, or unclear priorities, not insufficient headcount. Adding a person to a slow pipeline makes the pipeline slower, because now there's one more person waiting for it.

Don't hire to avoid a hard conversation. Sometimes the team needs a difficult reckoning: someone isn't performing, the direction isn't working, the schedule is unrealistic. Hiring a new person to fill the gap that the existing problem creates is a way of avoiding the conversation. The problem remains, and now there's a new person navigating around it.

The right time to hire is when the team has a proven loop, a clear commitment, a functional process, and a specific capability gap that is preventing the game from reaching the next stage. That's a real gap. Fill it with someone extraordinary.

12. **When to exit**

The same signals that tell you whether to hire someone tell you whether to keep them. Every pathology described above does not

only appear in candidates. It appears in people already on the team.

The reluctance to exit someone is understandable. Small teams are intimate. Letting someone go feels like betrayal. But keeping someone who is damaging the project is not loyalty. It costs everyone: the team that compensates for them, the game that absorbs their drift, and the person themselves, who is spending their career in a role that doesn't fit.

12.36. **The exit signals**

The exit signals are the hiring signals, observed over time.

The work doesn't serve the commitment, and feedback hasn't changed it. One round of redirection is coaching. Two rounds is a pattern. Three rounds is evidence that the person's instincts are pointed somewhere else and will not change. The question is not whether they're trying but whether trying is producing results.

The team is routing around them. People stop asking them for input. Their work gets quietly redone by someone else. Meetings they attend take longer. Decisions they participate in get relitigated afterward. The team has already made the exit decision in practice; the formal decision is just catching up.

Their presence costs more energy than it generates. Every person on a small team should be a net energy source. Not in the motivational-poster sense, but in the operational sense: working with them should make other people's work better, faster, or more coherent. When working with someone consistently makes things harder (more meetings, more explanation, more rework, more doubt) the math is clear. The project is paying for their presence in a currency it cannot afford.

They have stopped growing toward the game. Some misfits are temporary. A person who doesn't get it *yet* but is visibly learning, adapting, asking better questions every week, is worth patience. A person who has plateaued, whose work this quarter is the same quality and the same misalignment as last quarter, has given you the information you need.

The nose says so. If the person (or people) with calibrated judgment watches someone work for a month and says “this isn’t going to get there,” take that seriously. They see the trajectory, not just the current position.

12.37. **How to exit**

The mechanics of firing are outside this article’s scope; jurisdictions, contracts, and HR requirements vary. What this article can address is how the *decision* should work.

Be fast. The cost of a wrong hire at small scale is compounding. Every week you delay the exit, the team absorbs more drift, more rework, more morale cost. The kindest thing for everyone, including the person being exited, is speed. Drawn-out performance improvement plans at a scale of twelve people are theater. If the signals are clear, act on them.

Be honest. “It’s not a fit” is kinder than it sounds. What is unkind is letting someone believe they are succeeding when the team has already decided they aren’t. The person deserves to know what the gap is, in specific terms, so they can seek a role where their instincts are an asset rather than a liability.

Be structural, not personal. The conversation is not “you are bad at your job.” The conversation is “your instincts and this project’s needs have not aligned, and we don’t see a path to alignment.” This is usually *true*, not diplomatic softening. Most people who fail in one game team would succeed in a different one. The failure is the match, not the person.

Protect the team’s ability to trust the process. If the team sees someone underperforming for months with no action, they learn that the hiring standard is aspirational, not real. They start hedging their own investment. They lose faith that the studio means what it says about quality, commitment, and taste. Exiting someone promptly when the signals are clear *reinforces* the standard. It tells the team: we meant it.

13. How to evaluate

Interviews are almost useless for evaluating game development hires. A person can be articulate, personable, and impressive in conversation while being mediocre in practice. The pathologies described above (title inflation, all talk no game) are specifically adapted to survive interviews. The interview selects for people who are good at interviews. You need people who are good at making games.

Evaluation has to go deeper, and it has to look at different surfaces than a CV and a conversation provide.

13.38. The nose

Before describing structured evaluation, something must be acknowledged: some people can tell within minutes whether a candidate can do the job.

This isn't mysticism, it's compressed pattern recognition. A person who has hired and fired across multiple projects, who has watched the gap between what people say and what they deliver, who has seen the pathologies listed above play out dozens of times, develops a sense for it. They hear how someone talks about a problem and they know whether the person has solved problems like it or merely studied them. They notice what the candidate is curious about, what they skip over, what makes them light up, what makes them reach for jargon. The nose picks up signals that no rubric captures.

This ability is not a gift. It is a skill, built by exposure and reflection. More than one person on a team can develop it, and the evaluation process is healthier when they do; multiple calibrated perspectives catch what a single nose might miss or project. Every hiring process should include at least one person with this ability. Their judgment should carry real weight, not necessarily veto power, but a strong voice that the team takes seriously.

The risk is that intuition without accountability becomes ego, vibes, and gatekeeping. The nose must be *paired with explanation*. "I don't think this person can do the job" is not sufficient. "I don't think this person can do the job because when I asked about X,

they described process rather than outcomes, and when pushed on the tradeoff between Y and Z, they defaulted to convention rather than reasoning from the specific problem" is sufficient. The requirement isn't that someone formalizes their intuition into a checklist, but that they can articulate *what they noticed and what it predicts*. If they can't, their signal is not trustworthy, because there is no way to distinguish pattern recognition from bias.

Guard the nose against three failure modes:

- **Ego:** "I just know" without explanation is self-flattery, not a hiring signal.
- **Similarity bias:** The nose may be detecting "people like me" rather than "people who can do this job." Force the question: "Would this person make the game better, or would they make the team more comfortable?"
- **Mythology:** Anyone can develop a mythology around their own judgment that stops updating. If the nose was wrong last time and didn't adjust, it's a habit, not a skill.

No process or technology can substitute for people who have seen enough to smell competence, provided they can explain what they're smelling and have a track record that supports it. Ideally, spread this responsibility: the more people on the team who have developed calibrated judgment, the less the process depends on any single evaluator's blind spots.

There is a prerequisite that is easy to overlook: **the evaluator must have been calibrated against the real thing—but that calibration has more than one source.** The direct path is proximity: working alongside someone whose output resets the bar. The indirect path is negative-space expertise: years inside environments where excellence was absent, combined with honest study of *why* it was absent and close examination of the publicly available work that succeeded. (See The Benefits of Failure (https://gerolds.github.io/posts/the-benefits-of-failure/) for the full argument.) Both paths produce evaluators who can feel the gap between "pretty good" and "genuinely great." What does *not* produce calibration is a career of unexamined experience (mediocre or successful) where nobody ever asked *why* the work landed the way it did. And the indirect path carries a specific danger: it is easy to narrate. "I

learned from failure” is a story the comfort machinery (https://gerolds.github.io/posts/the-price-of-seeing-clearly/) produces fluently. The test is external evidence: predictions confirmed, work that landed with strangers, judgments tested against outcomes the person did not control. Without that evidence, the claim of negative-space calibration is indistinguishable from the self-rationalisation it resembles. If nobody on the hiring side can point to that kind of evidence, the team will hire mirrors of itself and wonder why the game never exceeds expectations. The first job of the studio is to find at least one person who has done that reckoning and whose reckoning left marks that others can see. Everything else, hiring, evaluation, leadership, calibrates off that.

13.39. **Before you meet: the evidence trail**

Enthusiasm. Not “passion” in the self-reported, performative sense. Enthusiasm in the operational sense: does the person’s life show evidence that they care about games more than professionally required? Do they play broadly? Do they have opinions about games outside their genre? Do they notice things in games they play that connect to problems in games they make? Enthusiasm is not a feeling; it’s a pattern of attention.

Side projects. Games, prototypes, mods, tools, game-jam entries, experiments. These matter not because the results need to be impressive, but because making things in your own time, on your own terms, reveals how someone thinks when nobody is directing them. Side projects show initiative, taste, scope instinct, and follow-through. A person with no side projects is not necessarily disqualified, but a person with side projects is giving you evidence you cannot get any other way.

Writing. Does the person write about games, design, their craft, their process? Writing is thinking made visible. A person who writes about game development has been forced to organize their ideas, confront their assumptions, and articulate positions clearly enough for strangers to engage with. The quality of the writing matters less than its existence. Someone who writes devlogs, design analyses, postmortems, or critical essays is someone who processes experience deliberately rather than letting it pass through them.

Role-adjacent hobbies. A programmer who plays music. An artist who builds furniture. A designer who writes fiction. A producer who coaches a sports team. These adjacencies matter because they develop transferable judgment: rhythm, composition, pacing, structure, leadership under pressure. A person whose entire world is their discipline may be deep, but they are often narrow in ways that show up as rigidity when the project demands lateral thinking. The best game developers import metaphors from outside games.

13.40. **The live test: a real problem, not a quiz**

Don’t give people abstract tests or whiteboard exercises. Give them a real problem from your game. A design question the team is actually wrestling with. A visual challenge. A technical constraint. Then watch.

Do they ask clarifying questions? A person who jumps straight to solutions is performing confidence. A person who asks “what is the game trying to do here?” and “what have you already tried?” is showing you how they actually work. The quality of their questions is more diagnostic than the quality of their answers.

Is their instinct to add or to subtract? When faced with a design problem, do they propose new systems, or do they look for what can be removed or simplified? In game development, the instinct to subtract is rarer and more valuable. Anyone can add. The person who sees that the problem is *too much*, not *too little*, is the person you want.

Can they sketch live? Not “draw well.” Sketch: rough out an idea visually, spatially, structurally, in real time. A designer who can sketch a flow on a whiteboard. A programmer who can pseudocode a system on paper. An artist who can block out a composition in thirty seconds. The ability to externalize thinking quickly and messily is a proxy for how they’ll work in the build, where speed of iteration matters more than fidelity of plan.

Do they consider the commitment? If you’ve explained what the game is trying to be, watch whether their solution serves it. A person who produces a clever answer that ignores the commitment is showing you exactly what they’ll do on the job.

13.41. **The trial: watching them work**

If at all possible, do a paid trial. A day. A week. Put them in the actual environment with the actual team and the actual game. You will learn more in one day of working together than in ten hours of interviews. Here is what to watch for.

Do they triage intuitively? When given a task with multiple sub-problems, do they find the critical path without being told? Do they start with what matters most, or do they start with what's most comfortable? Intuitive triage reveals how someone processes complexity. A person who needs explicit prioritization for every task is consuming leadership bandwidth. A person who naturally finds the load-bearing problem first is generating it.

Do they look for alternatives before committing? When they encounter a problem, is their first move to solve it, or to understand it? The person who explores two or three approaches before committing to one produces better work and fewer dead ends. The person who locks in immediately is often optimizing for speed of decision rather than quality of outcome.

Do they keep their artifacts organized and clean? Look at their files, their layers, their code, their naming. Not for pedantic compliance with conventions, but for evidence of a mind that cares about the next person who will touch this work. Organized artifacts are a signal of organized thinking. Messy artifacts are a signal that the person builds for themselves, not for the team.

Do they aim for minimal or maximal? When given a task, do they produce the leanest thing that solves the problem, or the most elaborate thing the time allows? At small scale, minimal is almost always right. The person who instinctively builds the simplest working version and then asks "is this enough, or do you need more?" understands that scope is the enemy. The person who builds the biggest version they can is optimizing for their portfolio, not for the project.

Do they understand how precise the output must be? Not everything needs to be final quality. Some things need to be sketches. Some things need to be bulletproof. The person who produces placeholder-quality work when the task requires precision is costing the team rework. The person who produces polished

work when the task requires a quick test is costing the team time. The judgment about *when to be rough and when to be precise* is one of the highest-leverage skills at small scale, and it is almost never taught.

Can they deal with vague specs and incomplete information? In small-team game development, specs are always incomplete. The person who freezes when the brief is ambiguous, or who demands complete specifications before starting, is not suited to this environment. The right person makes reasonable assumptions, flags them, starts building, and adjusts when more information arrives. They treat ambiguity as normal, not as someone else's failure.

How do they handle conflicting requirements? This is a critical test. Give them a task where two constraints genuinely conflict. Watch whether they try to satisfy both (impossible, produces mush), whether they silently pick one and ignore the other (dangerous), or whether they step back and say: "These requirements contradict each other. Here is why. Here is what I'd suggest changing in the spec." The third response is the one you want. It shows someone who can see the shape of a problem from above, not just from inside it. The ability to recognize an unsolvable contradiction and explain why the specification itself must change is rare, and it is worth more than almost any technical skill.

Can they use AI? How? AI tools are part of the working environment. The question is not whether they use them, but *how*. Do they use AI as a collaborator, generating options, accelerating search, testing ideas, while maintaining judgment about what to keep? Or do they use it as a crutch, accepting outputs uncritically and losing the ability to evaluate quality? The person who uses AI well produces more, faster, without sacrificing taste. The person who uses AI badly produces more, faster, and all of it is mediocre. The tool amplifies whatever the person already is.

How have they failed? Ask them. Not as a gotcha interview question, but as a genuine inquiry into their operating rhythm. The right answer is not a polished story about overcoming adversity. The right answer reveals that failure is *daily*: try, fail, fix, try again.

They push hard enough to break things regularly, and they treat the breaking as information.

A person who cannot recall recent failures is either not pushing hard enough or not honest enough. Both are problems. The person you want fails constantly, at small scale, and each failure teaches them something specific.

Discuss: aim for failing 50% of the time. This is a useful provocation in an evaluation setting. How does the candidate think about balancing experimentation (which requires tolerance for failure) with reliability (which requires protecting the upside)? The interesting answer is structural: they describe how to create safe-to-fail experiments that don't endanger the critical path. They know which parts of the project can absorb failure and which cannot. They understand that "fail fast" is not a universal principle but a *scoped* one: fail fast on what's uncertain, be disciplined on what's proven.

How do they deal with not knowing what the future holds? Can they name patterns, pathologies, and likely outcomes? Can they see that action A will cause rot B in six months? What heuristics, lenses, or methods do they use to check their solution's viability before the evidence is in? This tests for *structural thinking*: the ability to reason about second-order effects, not just immediate outcomes. A programmer who can explain why a particular architecture will become painful at scale. An artist who can predict where a visual style will break when applied to content they haven't built yet. A designer who can see where a mechanic will degenerate before players find the exploit. This kind of foresight is not prophecy. It is pattern recognition, and it separates experienced practitioners from people who have merely accumulated years.

Do they have a healthy suspicion? Of best practices. Of internet advice. Of conference talks. Of their own accumulated experience. The best hires know that best practices are context-dependent, that internet advice is survivorship-biased, and that their own experience is path-dependent and self-rationalizing. They hold their knowledge provisionally. They can say "I've always done it this way, but I'm not sure it's right for this situation." That sentence is worth more than ten years of uncritical expertise.

13.42. **After the trial**

Ask the team. A simple question: “Do you want to work with this person?” Not whether they’re qualified or did good work, but whether the team *wants* them here. Small teams run on mutual respect and shared energy. If the team is lukewarm, that is data. Trust it.

---

14. **Quick reference**

For the reader who has been through the article and wants the compressed version.

The question that governs everything: Does this person make the game more likely to be great?

Don’t hire to solve: commitment problems, scope problems, speed problems, or to avoid a hard conversation.

Hire for functions, not titles. The eight functions: commitment keeper, eye, author, loop guardian, architect, skeptic’s advocate, bridge, unblocker. Every function must be owned. At small scale, people carry multiple functions; identify which merges destroy necessary tension and hire to restore it.

The pathologies to screen for: title inflation, all talk, simply not getting it, complacency, asking the wrong questions.

Management pathologies to guard against: comfort hiring, referral bias, shallow search, proximity hiring, fear of competence, lowballing, wishful thinking, self-rationalisation, reorganising around new hires, not seeing skills already on the team, not taking hiring seriously.

What to look for in individuals: orientation toward this game’s problem, a practice outside work, strong opinions loosely held, alternatives instead of excuses, intolerance for chaos, maturity about novelty, ambition scaled to the difficulty.

Taste: the ability to distinguish “this works” from “this is right for this game.” It scales inversely with team size. At 4 people, everyone’s taste is the game’s taste.

Extraordinary skill: process creates conditions; the right combination of people, circumstances, and incentives still generates work that process alone cannot. Extraordinary ability is necessary, it just doesn't have to live in a single person, though it often does. A team that has never reckoned with real excellence, through proximity *or* through studying its absence, cannot calibrate its own standards.

The nose: include people with calibrated pattern recognition in every hiring process. They must explain what they notice. Calibration comes from proximity to excellence *or* from rigorous study of its absence, not from unexamined experience at any level. Spread this responsibility: multiple calibrated perspectives are more reliable than one.

Evaluation order: evidence trail (enthusiasm, side projects, writing, adjacent hobbies) → live test (real problem, watch questions and instincts) → paid trial (triage, organization, minimal vs. maximal, precision judgment, ambiguity tolerance, contradiction handling, AI use, failure rhythm, structural foresight, healthy suspicion) → ask the team.

When to exit: work doesn't serve the commitment after feedback, team routes around them, they cost more energy than they generate, they've stopped growing toward the game, the nose says so. Be fast, honest, structural, and protect the team's trust in the standard.

---

15. **The unifying principle**

Hiring for games at small scale comes down to one question: **does this person make the game more likely to be great?**

The standard is not shipping, schedule, or technical impressiveness by themselves. The standard is whether the game becomes more *great*: coherent, compelling, and true to its commitment.

If the answer is clearly yes, hire fast. If the answer is "maybe, but they're available and we need someone," stop. That calculus is how studios fill seats with adequate people, and adequate people

produce adequate games. The standard has to be higher than adequate because the problem is harder than adequate can solve.

Every person in a 4–40 team shapes the game in ways that larger organizations absorb and dilute. At this scale, there is no absorption. There is only the team you built, making the game you committed to, and every person in the room either pushes it toward greatness or doesn't.

---

16. **Appendix: further reading**

This article's positions are informed by the following. None of them are about game hiring specifically; the useful thinking on teams, taste, and creative leadership tends to come from adjacent fields.

16.43. **On hiring and team composition**

- **"Who: The A Method for Hiring"** — Geoff Smart & Randy Street. The structured-evaluation approach (scorecard, topgrading interview, focused reference checks) that this article's trial-and-evidence framework draws from. The core insight: most hiring fails because the process selects for interview performance rather than job performance.
- **"The No Rules Rules"** — Reed Hastings & Erin Meyer. Netflix's talent-density argument: a team of exceptional people with minimal process outperforms a larger team of adequate people with heavy process. The direct antecedent of this article's stance that adequate is negative.
- **"Peopleware"** — Tom DeMarco & Timothy Lister. The classic on why adding people to a late project makes it later, why environment shapes output, and why teams gel through shared standards rather than shared fun. Written about software; applies directly to game studios.

16.44. **On taste, craft, and creative judgment**

- **"Taste for Makers"** — Paul Graham (essay, paulgraham.com). The argument that taste is trainable, that it requires exposure to great work, and that makers who lack taste produce technically

proficient mediocrity. Directly relevant to the calibration problem described in this article's sections on excellence and the nose.

- **"The Timeless Way of Building"** — Christopher Alexander. The concept of "quality without a name," the thing that makes a space (or a game) feel alive rather than merely correct. Alexander's pattern language influenced game design thinking; the deeper contribution is the argument that quality is recognizable but not reducible to rules.
- **"Creativity, Inc."** — Ed Catmull. Pixar's internal culture of candor, the Braintrust model (structured creative criticism from peers, not management), and the argument that protecting creative people from organizational dysfunction is itself a creative act. The checks-and-balances and calibrated-judgment ideas in this article owe something to Catmull's framing.

16.45. **On small teams and focus**

- **"Rework"** — Jason Fried & David Heinemeier Hansson. The argument for staying small, cutting scope ruthlessly, and treating constraints as creative advantages. The "when not to hire" section draws from this line of thinking.
- **"Turn the Ship Around!"** — L. David Marquet. Leadership through intent-based orders rather than permission-based hierarchies. Directly relevant to the "judgment under autonomy" trait at 20–40 scale, and to the argument that leads must internalize the commitment deeply enough to act independently.
- **"Team of Teams"** — General Stanley McChrystal. How a large organization can operate with the speed and adaptability of a small one by pushing decision authority to the edges. Relevant to the scale-transition sections and to the argument that shared mental models matter more than shared meetings.

16.46. **On game development specifically**

- **"Blood, Sweat, and Pixels"** — Jason Schreier. Case studies of game development that illustrate nearly every pathology described in this article, including how extraordinary outcomes emerge from specific combinations of people and pressure.

Useful for calibrating expectations about how hard the work actually is.

- **“A Theory of Fun for Game Design”** — Raph Koster. The argument that games are learning machines and that “fun” is the feeling of mastering a pattern. Relevant to the loop guardian function and to the taste distinction between “this works” and “this is right for this game.”
- **GDC Vault** (gdcvault.com). Postmortems and talks from shipped projects are the closest thing to field reports on what this article describes. Pay particular attention to talks about what went wrong, not what went right; the failure postmortems are where the real hiring lessons live.

Drafting assistance: Claude. All claims mine; errors my responsibility.