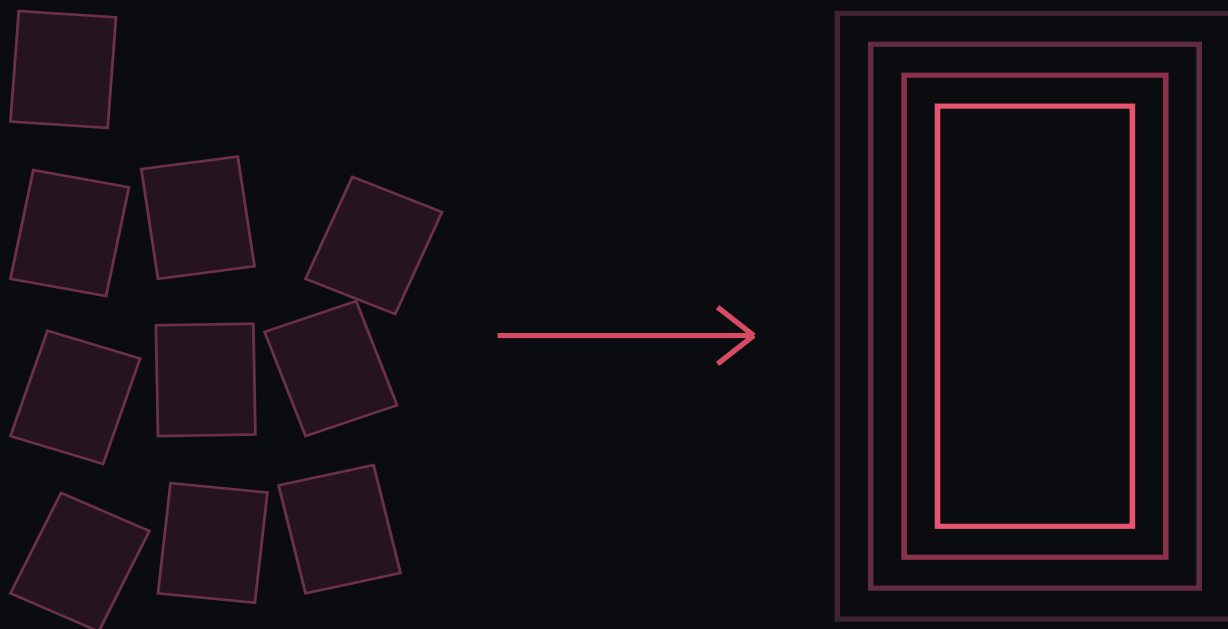


SHIP THE GAME

The Indie Developer's Guide to Actually Finishing



RELEASE / 1.0

PRODUCTION & RELEASE

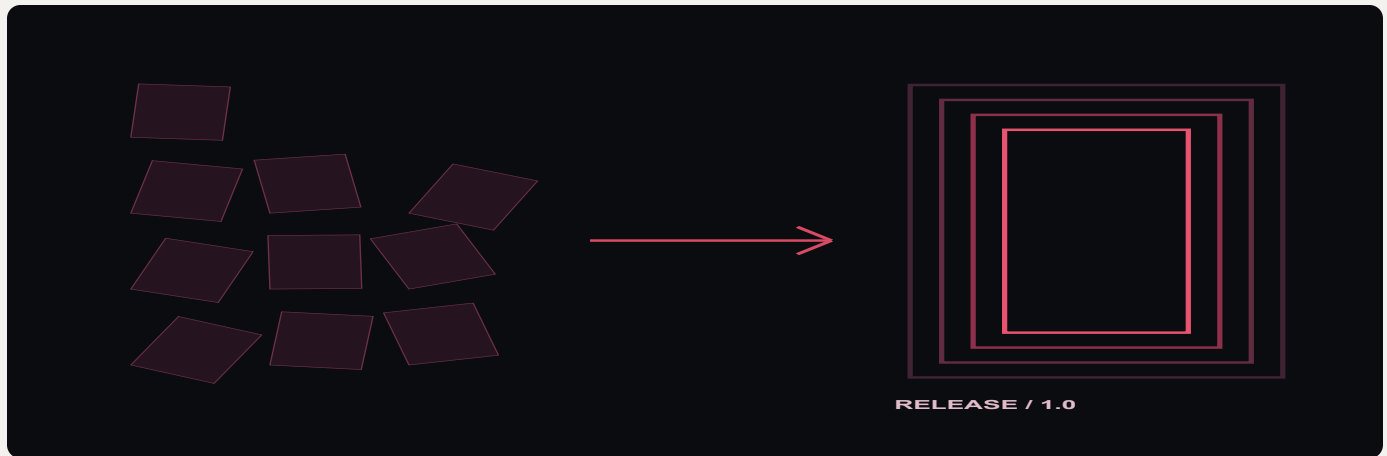
BUILD WHAT YOU IMAGINE.

FIELD GUIDE

A PRACTICAL FIELD GUIDE

Ship the Game

The Indie Developer's Guide to Actually Finishing



Published by Vertex Arc

First edition / October 2026

Original text and procedural illustrations created for Vertex Arc. Copyright 2026 Vertex Arc. Free to download and read. All examples describe hypothetical projects; they do not claim client work or shipped results.

Use this guide as a set of practical experiments. Test advice in your own project and consult documentation for the engine version you ship. Game-design observations are design hypotheses, not clinical claims or guarantees. No affiliation with or endorsement by Unity Technologies is implied.

vertexarc.site/books

Inside this guide

Read in sequence, or choose the lesson that answers the next question in your project. Page numbers refer to the PDF page counter.

Introduction: finishing is a design constraint	4
01 Recognize idea addiction	5
02 Define a realistic minimum complete game	6
03 Turn milestones into observable outcomes	7
04 Build vertically before expanding horizontally	8
05 Cut features with a plan	9
06 Manage bugs by impact and recovery	10
07 Use playtesting to make decisions	11
08 Polish what supports the whole game	12
09 Prepare the build and the promise	13
10 Plan the first update before launch	14
Release checklist: the final candidate	15
Conclusion: finish one honest promise	16

About Vertex Arc / 17

Introduction: finishing is a design constraint

Finishing a game means producing a complete experience that another person can obtain, understand, play, and leave without needing its developer nearby. It includes the title screen, the last encounter, the settings, the build process, the support information, and the awkward edge cases between them. Those parts are less visible than a new mechanic, but they are part of the game you are making.

This book follows an original hypothetical small project called Signal Harbor. The player repairs navigation beacons across a compact coastal map. The team has a short campaign, a few reusable interactions, and limited production time. The example is not a claim about a shipped Vertex Arc project. It is a practical frame for decisions about scope, milestones, testing, and release.

The advice is deliberately modest. You do not need a large production department to write an exit criterion, preserve a working build, or decide which feature to cut. You do need to make those decisions visible and follow them when new ideas become more exciting than unfinished work.

Keep a release definition, a milestone board, a bug list, and a decision log. Make them short enough to use. The purpose is not to create administration around the game; it is to stop hidden obligations and repeated debates from consuming the time needed to finish it.

THE RELEASE PATH



Use the result to shape the next attempt.

01 / Recognize idea addiction

THE IDEA

A new idea offers possibility without the accumulated constraints of an existing project. That makes starting feel easier than finishing. The danger is not having ideas; it is using them to avoid decisions about the game already in progress. Capture new concepts in a separate place so they remain available without automatically becoming current work.

Ask what problem a proposed feature solves in the current experience. If the answer is only that it would be exciting to build, it may belong in a future experiment. Creative exploration can still have a place, but give it a bounded purpose and a return point to the main project.

EXAMPLE

Signal Harbor's developer wants to add sailing after the beacon interactions already work. Sailing could become a separate game loop with controls, collision, camera, art, and testing costs. The idea goes into an experiment list. The current release keeps travel simple because its intended experience is repairing and reconnecting the harbor, not mastering a boat.

KEY TAKEAWAY

Keep new ideas without letting them silently replace the project you committed to finish.

TRY THIS

Review your last five new feature ideas. For each, identify the player problem it solves and the existing work it would displace. Move ideas without a clear answer out of the current milestone. Reserve a small future review point so postponement does not feel like forgetting.

DEVELOPER NOTE

If you repeatedly avoid the same unfinished system, investigate why. It may be unclear, too large, technically blocked, or no longer valuable. A smaller task or a deliberate cut is more useful than another unrelated feature.

02 / Define a realistic minimum complete game

THE IDEA

An MVP is not a collection of unfinished features. It is a deliberately limited product that still delivers a coherent experience. Define the player's beginning, central loop, and ending. Include the supporting work required for that journey: onboarding, settings, saving if needed, error recovery, and a way to exit.

Separate the promise of the release from the long-term vision. A short game can be complete when it fulfills its promise. A large game can remain incomplete when important paths end abruptly or systems depend on content that does not exist. Scope should describe the actual experience you can deliver.

EXAMPLE

Signal Harbor's first release has three beacon sites, one repair interaction with meaningful variations, and a final return to the harbor. It does not include an open-ended crafting system. The ending confirms the result of the player's work, so the small campaign feels intentional rather than like a demo that simply stops.

KEY TAKEAWAY

Define a small complete journey before counting features. Completeness is about the player's experience, not the size of the task list.

TRY THIS

Write a one-paragraph release promise and a start-to-finish player journey. Highlight every dependency required to complete it. Remove optional branches until the core path is feasible. Ask someone unfamiliar with the project what they would expect from the promise, then correct misleading language or missing work.

DEVELOPER NOTE

Minimum does not mean ignoring reliability or accessibility. A smaller feature set should make it easier to deliver the essential experience well. Do not use the label to conceal a known blocker in the main path.

03 / Turn milestones into observable outcomes

THE IDEA

A milestone should end with something that can be demonstrated. "Work on enemies" is an activity. "A new player can complete one encounter in a packaged build" is an outcome. The second exposes integration and testing work that the first can hide indefinitely.

Break the outcome into tasks with clear ownership and dependencies. Keep work in progress limited so the team finishes connected pieces rather than starting everything. Include a review point and a decision about what happens if the milestone exceeds its capacity: reduce scope, change the approach, or revise the date honestly.

EXAMPLE

Signal Harbor's next milestone is one complete beacon site. It includes arrival, instruction, repair, feedback, saving the result, and returning to the map. The team does not mark it complete when the repair animation works in isolation. It is complete when the whole path works in the intended build.

KEY TAKEAWAY

Measure milestones by usable outcomes. Partial activity is not the same as a finished experience.

TRY THIS

Rewrite the next milestone as a sentence beginning "A player can..." List the evidence needed to verify it. Add tasks for integration, target-device testing, and recovery from failure. Then identify the smallest cut that would preserve the outcome if time runs short.

DEVELOPER NOTE

Estimate using recent completed work rather than only optimism about future speed. Include revisions and interruptions. A schedule becomes more useful when assumptions are visible and can be updated from evidence.

04 / Build vertically before expanding horizontally

THE IDEA

Horizontal work adds more content or systems at the same unfinished level. Vertical work completes a small segment through every required discipline. Both are eventually necessary, but a vertical slice reveals the cost of finishing before that cost is multiplied across a large game.

Choose a representative segment and take it through gameplay, art, sound, UI, performance, saving, and delivery as applicable. Do not select only the easiest case. The slice should reveal the ordinary production pipeline and at least one meaningful risk without becoming the largest set piece in the project.

EXAMPLE

The first beacon site in Signal Harbor becomes the reference for later sites. Its finished interaction establishes camera framing, audio cues, accessibility behavior, and save ownership. The team learns that each site needs a clear approach landmark. That finding is applied before ten more locations have been decorated incorrectly.

KEY TAKEAWAY

Finish one representative unit before multiplying its assumptions across the game.

TRY THIS

Choose one level, encounter, or interaction that represents the release. Write completion criteria for each discipline. Track the actual time spent on first implementation and revision. Use those measurements to revise the planned content count, leaving room for integration and final testing.

DEVELOPER NOTE

Keep the slice representative, not uniquely overproduced. If it uses custom work that cannot be repeated for the rest of the game, it may demonstrate a trailer scene rather than a sustainable production process.

05 / Cut features with a plan

THE IDEA

Scope creep expands the overall project; feature creep adds capabilities inside it without a corresponding decision about cost. Both can happen through reasonable requests. Make additions explicit: what player value do they provide, what dependencies do they create, and what will be removed or delayed to make room?

Cutting is not simply deleting code. Remove the feature's promises from tutorials, store copy, UI, saves, and level design. Preserve a recoverable version and document the reason. A complete cut reduces complexity; a half-cut leaves confusing traces and hidden maintenance work.

EXAMPLE

Signal Harbor removes a weather system because its main effect is to complicate beacon visibility and testing. The team keeps a fixed atmospheric condition that supports the story. It also updates screenshots and descriptions so the release no longer implies dynamic weather. The result is a coherent smaller game rather than an absent feature players still expect.

KEY TAKEAWAY

Cut dependencies and promises together. A feature is gone only when the experience no longer relies on it.

TRY THIS

Rank unfinished features by their contribution to the release promise and their remaining full cost. Select one weak candidate and trace every place it appears. Create a reversible cut, play the entire main path, and update the plan. Check whether the game becomes clearer as well as smaller.

DEVELOPER NOTE

Do not use sunk effort as the deciding argument. Work already spent cannot be recovered by spending more. Keep useful assets or findings for later, but judge the current feature by what the release still needs.

06 / Manage bugs by impact and recovery

THE IDEA

A bug list should help decisions, not merely accumulate complaints. Record the build, reproduction steps, expected result, actual result, and relevant evidence. Separate severity from priority: a visually obvious issue may be less urgent than a rare save failure with no recovery.

Group issues by player impact and the release promise. Blocking completion, losing progress, crashing, or creating misleading critical information deserves close attention. Cosmetic defects can still affect quality, but they should not crowd out reliability work simply because they are easier to reproduce.

EXAMPLE

Signal Harbor has a misaligned prop and an intermittent save error after leaving a beacon. The prop is visible in every session, but the save error threatens the main journey. The team prioritizes the save issue, adds a repeatable transition test, and verifies the fix on the build containing it rather than assuming the code change was sufficient.

KEY TAKEAWAY

Prioritize bugs by what they do to the player and how recoverable the result is.

TRY THIS

Triage the current list into release blockers, important fixes, and accepted limitations. Give every blocker an owner and a reproduction plan. For intermittent issues, record conditions and frequency rather than inventing certainty. After each fix, retest nearby behavior that could have changed.

DEVELOPER NOTE

Keep a short smoke test for every candidate build: start, load, core action, save, resume, and exit. Automation is useful for stable rules and data checks, while exploratory testing remains necessary for unexpected combinations and clarity.

07 / Use playtesting to make decisions

THE IDEA

Playtesting is most useful when it answers a question that can change the build. Watch behavior before collecting broad opinions. A player missing an objective may reveal a navigation problem, unclear language, or competing visual emphasis. Do not automatically implement the first suggested solution.

Choose participants who can reveal different assumptions. Familiar testers are good at regression checks, but they may compensate for missing onboarding. New players expose those gaps. Explain the test, avoid coaching the task, and ask about specific moments afterward.

EXAMPLE

Signal Harbor's testers repeatedly return to a repaired beacon because its visual state still resembles an unfinished one. The team initially considers adding a larger quest log. Observation suggests a simpler fix: make the completed beacon's world state unmistakable. The next test checks whether players now move toward the remaining objective.

KEY TAKEAWAY

Convert feedback into a hypothesis, then test the smallest change that addresses it.

TRY THIS

Choose one uncertain section and ask a new player to complete it. Record three observed problems without proposed fixes. For each, write at least two plausible causes. Change the most likely cause and repeat the task. Keep the previous build so improvements can be compared directly.

DEVELOPER NOTE

Do not treat every preference as a requirement. Evaluate feedback against the intended audience and release promise. Several contradictory suggestions may point to one shared confusion rather than a need to satisfy every suggestion literally.

08 / Polish what supports the whole game

THE IDEA

Polish should improve clarity, consistency, comfort, or emotional impact. It should not become a way to avoid unfinished essential work. A beautiful first room does not compensate for a broken ending. Protect a complete playable path before spending large amounts of time refining details within one favorite section.

Prioritize repeated interactions and prominent inconsistencies. Players may use a menu or movement action hundreds of times, so small improvements there can matter more than a rare decorative effect. Review the whole game for mismatched feedback, abrupt transitions, and unclear states.

EXAMPLE

Signal Harbor's repair sound is pleasant, but the completion response is delayed and the final route has no clear exit. The team fixes those issues before adding new particles. Later, a restrained audio and lighting pass gives every completed beacon a consistent identity. The polish supports the player's understanding of progress.

KEY TAKEAWAY

Polish after the important behavior is stable, and spend it where players repeatedly experience the game.

TRY THIS

Make two lists: unfinished essentials and polish opportunities. Complete or deliberately cut the essentials first. Then rank polish by frequency and impact. Test a longer session to ensure effects remain comfortable, and verify reduced-motion and audio settings throughout the pass.

DEVELOPER NOTE

Do not polish a feature whose design is still likely to change substantially. Use readable temporary presentation until the interaction is validated. Otherwise finish can make the team reluctant to make a necessary cut.

09 / Prepare the build and the promise

THE IDEA

A release build must be reproducible from a known project state. Record the source revision, engine version, required dependencies, and build configuration. Test installation and first launch outside the development environment. Missing files and machine-specific assumptions often appear only when someone else runs the package.

The store page or download page is part of the release promise. Use accurate screenshots, clear controls and requirements, and an honest description of what is included. Platform rules and technical requirements can change, so check the current official submission guidance for the actual storefront before publishing.

EXAMPLE

Signal Harbor's release candidate is copied to a clean test environment. The team checks a new game, saving, resuming, settings, the ending, and uninstall behavior where applicable. Its page describes a compact beacon-repair adventure rather than promising an open-world simulation. The screenshots come from the candidate build, not a discarded prototype.

KEY TAKEAWAY

Ship a reproducible package and an accurate promise. Both determine whether the player receives the experience you intended.

TRY THIS

Ask someone to obtain and run the candidate using only the public-facing instructions. Observe every point where they need help. Fix the package or instructions, then repeat. Verify licenses and credits for included assets and confirm that support contact information reaches the intended destination.

DEVELOPER NOTE

Keep release credentials out of source control and public build artifacts. Separate development configuration from shipped configuration. Do not include debug shortcuts, test accounts, or private project notes in the package by accident.

10 / Plan the first update before launch

THE IDEA

Release changes the work from building in private to maintaining a version people use. Decide how issues will be received, reproduced, prioritized, and communicated. Keep the shipped version recoverable and define how an update will be validated. A rushed patch can create a larger problem than the original issue.

Separate urgent reliability fixes from new features. Preserve save compatibility where the project promises it, and test updates from the previous version. Keep patch notes specific about actual changes. Avoid making commitments you cannot support simply because early feedback is enthusiastic or critical.

EXAMPLE

Signal Harbor receives a report that one beacon interaction fails after loading a particular save. The team requests the relevant reproduction details, reproduces the issue, fixes it, and tests both a new game and an update from the released version. A suggested new island goes into a future planning list rather than the emergency patch.

KEY TAKEAWAY

Maintain the released experience deliberately. Reliable updates build on the same discipline that made the first release possible.

TRY THIS

Write a one-page post-launch plan with the support channel, triage owner, patch validation steps, and rollback or recovery approach. Then write a retrospective template: what worked, what cost more than expected, what was cut, and what you would do differently. Use evidence from the project rather than judging learning only by sales or attention.

DEVELOPER NOTE

A project can teach valuable production lessons without becoming commercially successful. Preserve reusable tools, measured estimates, and decisions. Avoid immediately rebuilding the same scope problem under a new title without examining the previous result.

Release checklist: the final candidate

- The release promise matches the playable content and current screenshots.
- A new player can begin, complete, and exit the main journey.
- Saving, loading, and update compatibility are tested where applicable.
- Settings persist and accessibility options work in ordinary gameplay.
- Supported input methods, displays, and target devices have been checked.
- Known blockers are fixed; accepted limitations are recorded deliberately.
- The build is reproducible from a tagged or otherwise identified source revision.
- A clean environment can obtain, install, launch, and run the package.
- Asset rights, credits, required notices, and storefront requirements are reviewed.
- Debug data, private credentials, and test-only content are excluded.
- Support information, version identification, and patch ownership are clear.
- The released build and a recovery path are preserved before the next update.

Conclusion: finish one honest promise

Finishing is not the moment when no more improvements are imaginable. It is the point where the game fulfills a deliberate promise at a quality level you can support. That requires choosing a scope, completing connected work, listening to evidence, and allowing good ideas to wait when they would prevent the current project from ending.

Choose the next milestone as a player outcome. Remove one unnecessary dependency. Produce a build another person can use. Those actions are modest, but they move the game toward release more reliably than expanding the vision whenever the work becomes difficult.

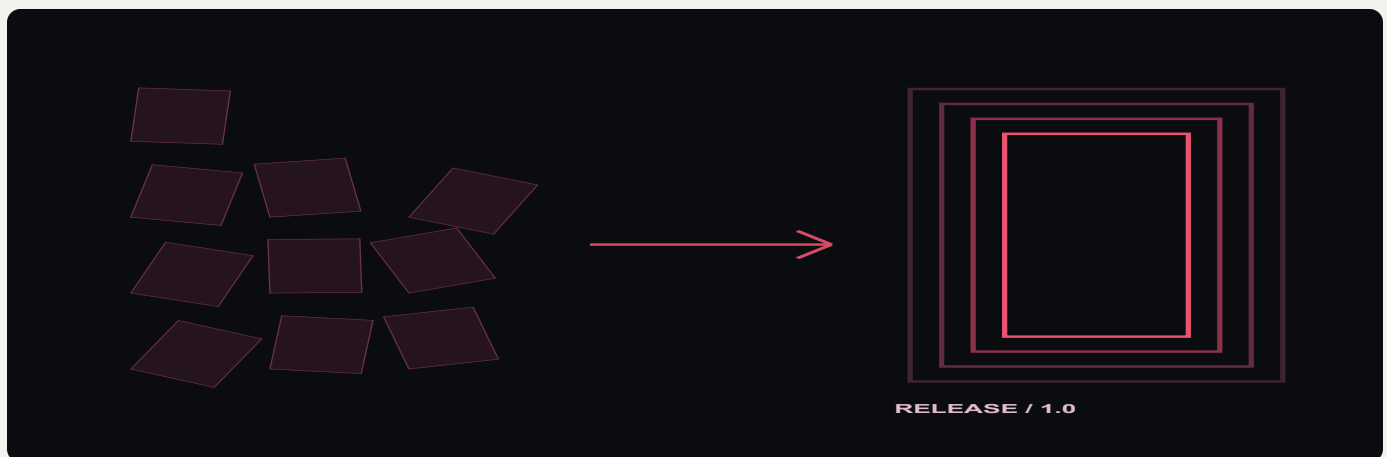
When the project ships, examine what you learned with the same care you gave its design. Keep the useful tools and estimates, name the mistakes without dramatizing them, and carry a smaller set of clearer principles into the next project. A finished game becomes both an experience for players and evidence for the developer you are becoming.

About Vertex Arc

BUILD WHAT YOU IMAGINE.

Vertex Arc is an independent game-development studio and creative engineering partner. Its work spans playable prototypes, game systems, and interactive experiences across PC, mobile, and web. This publishing series shares practical ways to ask better questions, make focused experiments, and improve the experience players actually receive.

The guides are designed to be used beside a project. Keep the exercises small, record what you observe, and adapt the next step to the evidence. Finishing one useful experiment is a better result than collecting advice that never reaches a playable build.



[Browse the complete free library](#)

[Explore the studio and its work](#)

[Start a project with Vertex Arc](#)