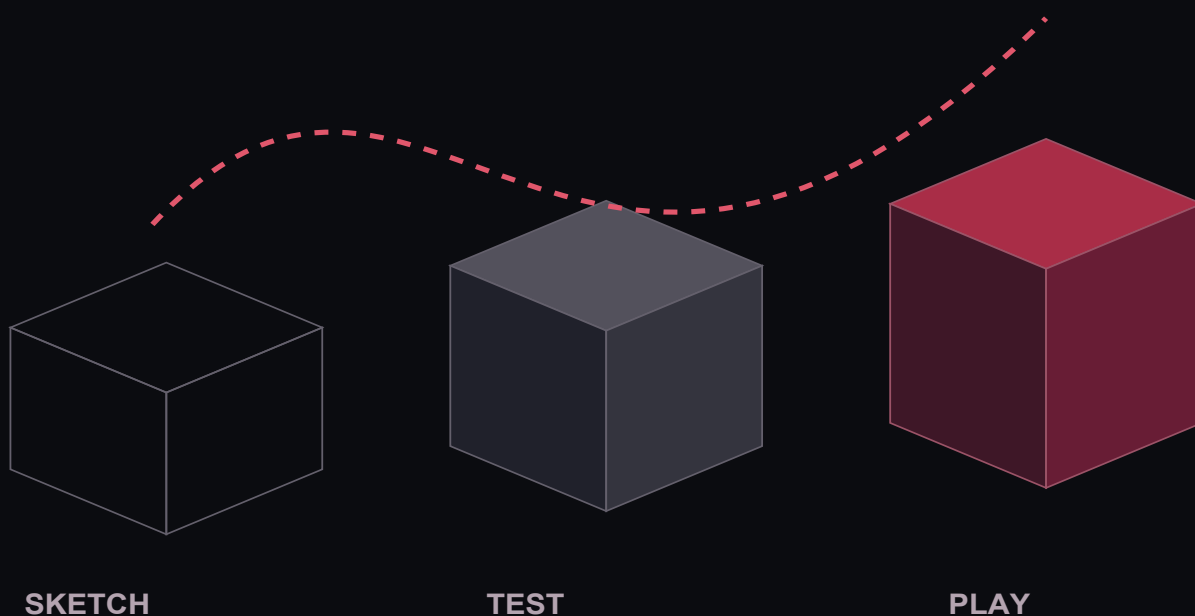


FROM IDEA TO PLAYABLE

A Practical Guide to Building Your First Game Prototype



PROTOTYPING

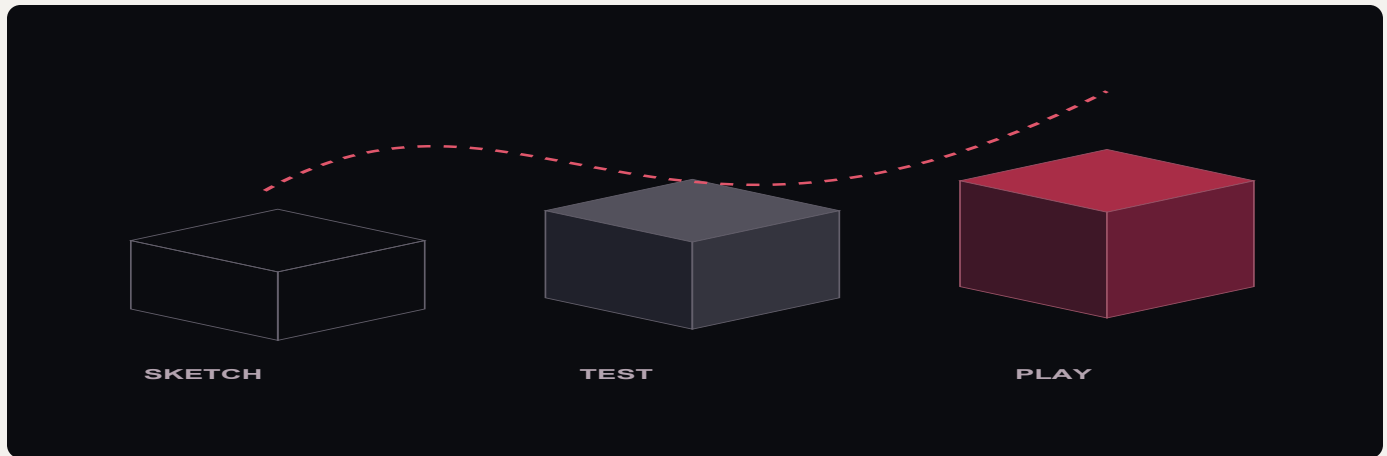
BUILD WHAT YOU IMAGINE.

FIELD GUIDE

A PRACTICAL FIELD GUIDE

From Idea to Playable

A Practical Guide to Building Your First Game Prototype



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: make something you can question	4
01 Your idea is not your game	5
02 Find the core gameplay loop	6
03 Reduce the scope until the question is visible	7
04 Prototype before polishing	8
05 Placeholder art is a communication tool	9
06 Build systems that can change	10
07 Playtest before you can explain everything	11
08 Learn what to remove	12
09 Move from prototype to vertical slice	13
10 Know when the experiment is finished	14
Field checklist: your first playable	15
Conclusion: evidence is progress	16

About Vertex Arc / 17

Introduction: make something you can question

A first prototype is not a smaller version of every feature in your imagined game. It is a playable experiment that helps you decide what deserves more work. You might want to create a delivery adventure across a floating city. The first useful build may contain a courier, three platforms, one parcel, and a timer. Its job is to answer whether carrying that parcel changes movement in an interesting way.

This book follows that original hypothetical project, Parcel Run. The courier can move quickly while empty-handed, but a fragile parcel changes acceleration and landing behavior. The example stays deliberately small so we can discuss decisions without hiding behind a large world or an elaborate story. You can substitute your own mechanic at every step.

Use the chapters in order if this is your first project. If you already have a prototype, begin with the chapter that addresses its biggest uncertainty. Keep a playable build, a one-page question sheet, and a short decision log. Those three artifacts will tell you more about progress than a long list of features marked partly complete.

The goal is to finish with evidence: what players understood, what they chose to repeat, what broke, and what remains risky. A prototype that leads you to change direction has done useful work. The only wasted experiment is one that never asks a clear question or refuses to use the answer.

THE PROTOTYPE LOOP



Use the result to shape the next attempt.

01 / Your idea is not your game

THE IDEA

An idea describes a possibility. A game contains rules that produce actions and consequences over time. Move from the first to the second by writing a player verb and the decision attached to it. "Explore a beautiful floating city" is a setting promise. "Choose a safe route or a faster jump while carrying a fragile parcel" is a testable interaction.

Write down the uncertainty that could most change your plan. It may be whether carrying an object creates a useful tradeoff, whether players can judge jumps from the camera, or whether the control scheme is understandable. Choose one as the first target. A prototype trying to answer all three at once may fail without revealing why.

EXAMPLE

Parcel Run begins with a route-selection question. Both paths reach the same destination. One contains broad platforms; the other has shorter but narrower landings. The parcel can survive one rough landing. There are no shops, missions, or upgrades because none are required to see whether the player weighs speed against risk.

KEY TAKEAWAY

Describe the choice the player makes, not only the world in which they make it.

TRY THIS

Write your concept in one sentence using a verb, a constraint, and a consequence. Then list the objects needed for one attempt. If the list includes several unrelated systems, remove everything that does not help answer the first question. Ask another person to explain what they expect to decide during play.

DEVELOPER NOTE

Keep technical feasibility questions separate from design questions. A local prototype can test route choice without proving online synchronization or mobile performance. Name the limits of the evidence you collect.

02 / Find the core gameplay loop

THE IDEA

A loop connects an action to feedback and a new opportunity to act. In Parcel Run, the player picks up a parcel, chooses a route, attempts a delivery, sees the result, and starts another attempt. Every part matters. Without a readable result, the player cannot learn; without a quick restart, testing becomes tedious.

Draw the loop as a sequence of states. Include failure and recovery, not only success. Decide what information carries between attempts and what resets. A timer that continues after a restart or a damaged parcel that remains damaged can make a simple loop difficult to interpret.

EXAMPLE

The first delivery ends with three facts: delivered or broken, completion time, and the landing that caused damage. The player can restart immediately. This feedback supports another route choice without introducing a score economy. A high score might be useful later, but it is not necessary to learn whether the core interaction works.

KEY TAKEAWAY

A complete attempt teaches more than several disconnected mechanics. Build the return to play as carefully as the first action.

TRY THIS

Play five attempts in a row and time how much of the session is spent acting versus resetting or waiting. Remove avoidable delay. Ask a tester to describe why the previous attempt succeeded or failed. If they cannot, improve feedback before expanding the mechanic.

DEVELOPER NOTE

Use a clear run state such as ready, active, complete, or failed. Centralizing these transitions is often simpler than letting several scripts independently decide whether the game is over. Keep the implementation appropriate to the size of the experiment.

03 / Reduce the scope until the question is visible

THE IDEA

Scope is the amount of work required to produce and maintain the experience, not merely the number of mechanics. A new enemy can require art, animation, sound, tuning, navigation, UI, and testing. A new level can introduce loading and save requirements. Count these dependencies before adding something that appears small.

Divide the plan into essential, useful later, and unrelated to this experiment. Essential work allows the question to be tested. Useful work might improve interpretation but can wait. Unrelated work belongs outside the prototype. This is a practical sorting exercise, not a judgment about whether an idea is good.

EXAMPLE

Parcel Run does not need a city map to test two routes. It needs one destination visible from the start. Weather, dialogue, and customization are removed. A simple damage indicator stays because the player needs to understand the parcel's condition before choosing the risky path.

KEY TAKEAWAY

Cut work that does not change the answer to the prototype's question. Preserve the feedback needed to trust that answer.

TRY THIS

Write a task list for the next playable build. For every task, finish the sentence: "Without this, we cannot learn whether..." Move tasks with weak answers to a later list. Choose a stopping date and define the build that must exist by then. If it is still too large, simplify the question.

DEVELOPER NOTE

Keep cuts visible. Quietly postponing a feature while still designing everything around it does not reduce scope. Update the level, interface, and success criteria to match the smaller experiment.

04 / Prototype before polishing

THE IDEA

Polish improves a direction you have reason to keep. Before that direction is established, polish can make change emotionally and technically expensive. Distinguish essential clarity from finish. A clear landing response is essential to testing movement; a detailed landing effect with several custom assets may not be.

Use the minimum presentation needed for a trustworthy test. If players cannot see hazards or recognize success, the prototype is too rough. If most development time goes into details that do not alter their decisions, it may be too polished. The right level depends on the question, not a universal rule that all prototypes must look ugly.

EXAMPLE

Parcel Run uses a short sound and a change in parcel color after a damaging landing. That is enough to test whether players understand the consequence. A bespoke break animation can wait until the team knows how damage should work and whether the parcel mechanic survives testing.

KEY TAKEAWAY

Build enough clarity to learn, then postpone finish until the interaction earns it.

TRY THIS

Review the last hour of work and label each change as clarity, feasibility, or finish. If finish dominates, select one unanswered design question and build the smallest change that addresses it. Test before returning to cosmetic work. Keep screenshots or builds so you can compare behavior rather than remember impressions.

DEVELOPER NOTE

Audio and camera behavior sometimes belong in the prototype because they affect perception of timing and space. Do not remove an entire discipline automatically; include the pieces necessary to answer the specific question.

05 / Placeholder art is a communication tool

THE IDEA

Placeholder art should make roles distinct. A destination, a hazard, a movable object, and background scenery need different visual treatment even when all are simple shapes. Consistent scale and collision are especially important. Players cannot judge a jump reliably if the visible platform does not match the surface they can land on.

Choose a small visual vocabulary and write it down. For example: solid gray surfaces are walkable, crimson panels are dangerous, and a white vertical marker identifies the destination. Do not reuse a signal for a different meaning just because it looks attractive in another room.

EXAMPLE

Parcel Run uses broad, neutral platforms and a bright parcel with a clear upright orientation. The risky route has narrow geometry, not merely a red label saying "hard." The player can assess the actual landing area. The destination marker remains visible from both routes so getting lost does not contaminate the movement test.

KEY TAKEAWAY

Temporary assets should be cheap to replace and reliable to read. Rough presentation does not excuse ambiguous rules.

TRY THIS

Show a still frame of your prototype to someone unfamiliar with it. Ask what looks usable, dangerous, movable, and important. Then let them play and compare those expectations with the actual rules. Fix mismatches using shape, placement, or contrast before adding explanatory text.

DEVELOPER NOTE

Keep asset provenance clear even during prototyping. Original primitives and properly licensed assets are easier to replace or retain later than a collection of images copied without a recorded source. Mark anything that must not ship.

06 / Build systems that can change

THE IDEA

A prototype needs flexibility, not a large architecture. Separate input, movement, run state, and tuning values enough that a design change does not require editing unrelated code. Avoid building a generic framework for games you are not making. The useful structure is the smallest one that keeps this experiment understandable.

Expose variables you expect to tune, such as speed, acceleration, damage thresholds, and restart delay. Give each value one clear owner. If the same behavior is controlled by a script value, an animation event, and a hidden scene override, comparisons become unreliable.

EXAMPLE

Parcel Run stores empty-handed and loaded movement settings in two explicit profiles. Picking up a parcel changes the active profile. The designer can compare acceleration without rewriting the controller. Delivery success is reported to the run controller, which owns the timer and restart, instead of every trigger maintaining a separate copy of the game state.

KEY TAKEAWAY

Structure the code around the changes you need to make now. Clarity is more valuable than speculative extensibility.

TRY THIS

Choose one likely change, such as doubling parcel weight or adding a second destination. Identify every file and scene setting you would need to touch. If the list is unexpectedly large, simplify ownership before adding the feature. Then make the change on a branch and verify the original behavior still works.

DEVELOPER NOTE

Use source control from the first playable build. Label temporary shortcuts and preserve working checkpoints. A prototype can contain deliberate compromises, but those compromises should not become invisible obligations when production begins.

07 / Playtest before you can explain everything

THE IDEA

Watching a new player reveals assumptions the developer no longer notices. Give a clear goal, then observe without coaching every action. Record what happens before asking for an opinion. A player who repeatedly misses a landing gives you a specific behavior to investigate; "movement feels odd" is only a starting point.

Separate observation from interpretation. "The player pressed jump before reaching the edge" is evidence. "The jump needs more height" is a proposed solution. Camera framing, input buffering, or platform spacing might address the same problem more effectively.

EXAMPLE

A Parcel Run tester chooses the narrow route because the broad route looks unfinished, not because they prefer risk. The team initially interprets this as a successful tradeoff. A follow-up question reveals that the visual vocabulary misled the player. Fixing the route presentation is more useful than rebalancing parcel damage.

KEY TAKEAWAY

Ask why the player acted, but first record what they actually did. Feedback becomes useful when it is tied to a moment in the build.

TRY THIS

Run a short test with one objective and no hints after the initial instruction. Note three moments of hesitation. Ask the player what they expected at those moments and what information they used. Choose one mismatch to address, then test with a new player if possible.

DEVELOPER NOTE

Use a consistent build identifier and brief test script. Tell participants what you are observing and get permission before recording them. Keep notes focused on game behavior rather than unnecessary personal information.

08 / Learn what to remove

THE IDEA

Removing a mechanic can strengthen the remaining decisions. A feature may be individually enjoyable yet distract from the core loop, duplicate another system, or create costs the project cannot support. Evaluate its role in the whole experience rather than defending the time already spent on it.

Use a reversible comparison. Disable the feature in one build, preserve the original, and test the same sequence. Ask what became clearer, what was lost, and whether another mechanic now carries more weight. This is more informative than debating the feature entirely in a planning document.

EXAMPLE

Parcel Run includes a stamina meter that limits sprinting. Tests show that players watch the meter instead of planning landings, while parcel weight already creates a speed tradeoff. Removing stamina makes the route choice easier to understand without eliminating the central risk. The feature is archived with notes instead of silently deleted from history.

KEY TAKEAWAY

Keep features because they support the experience, not because they took effort to build.

TRY THIS

Select the feature with the weakest connection to your core question. Disable it for one test. Write down the decisions players still make and the ones that disappear. If the useful decisions remain, consider cutting the feature. Update tutorials, UI, and dependencies so the cut is complete.

DEVELOPER NOTE

Distinguish removal from simplification. Sometimes a confusing system needs one clear use rather than several overlapping uses. Test the smallest coherent version before deciding whether to abandon the entire idea.

09 / Move from prototype to vertical slice

THE IDEA

A prototype tests whether an interaction is worth pursuing. A vertical slice tests whether the team can produce a representative finished segment through the intended pipeline. It should include the disciplines and delivery work the full game depends on: art, sound, UI, saving where relevant, performance, packaging, and testing.

Do not multiply prototype content before learning the cost of finishing one unit. Choose a small but representative segment. An unusually easy level can hide production risks, while the most ambitious set piece can make the experiment too large. Select something that reflects ordinary work in the planned game.

EXAMPLE

Parcel Run's slice contains one finished delivery route, one parcel type, a start screen, readable feedback, a result screen, and a packaged target-device build. The team measures how long it takes to revise the route after art is added. That revision cost informs the number of routes the full project can realistically support.

KEY TAKEAWAY

A vertical slice proves a production process. It is not simply a prototype with prettier textures.

TRY THIS

List every discipline required for a shippable segment of your game. Choose one segment and define completion for each discipline. Include a clean-machine build check and a performance scenario. Record actual work and revisions, then use that evidence to revise the full-game plan.

DEVELOPER NOTE

Review prototype code before carrying it forward. Reuse parts that are dependable and understandable; replace shortcuts that conflict with production requirements. Neither a total rewrite nor unconditional reuse is automatically the right choice.

10 / Know when the experiment is finished

THE IDEA

A prototype needs an exit decision. Without one, it can absorb endless improvements while the central uncertainty remains unresolved. Choose between proceeding, changing the question, or stopping. Each can be a successful result when supported by evidence.

Write the exit criteria before the final test. They should describe observable behavior or feasibility, not an impossible promise that everyone will love the game. For example: players can explain the route tradeoff, complete an attempt without coaching, and choose to try a different approach after failure.

EXAMPLE

Parcel Run tests show that the parcel creates an interesting decision, but the current camera makes narrow landings hard to judge. The team does not begin a city-sized production. It proceeds to a focused camera experiment, preserving the movement findings and identifying the remaining risk. The next step is smaller and clearer because the prototype ended with an honest decision.

KEY TAKEAWAY

Stop when the prototype has produced the evidence it was built to gather. More features do not automatically produce a better answer.

TRY THIS

Write a one-page closure note: question, build link, test conditions, observations, decision, and remaining risks. Ask a collaborator to read it without your verbal explanation. If they cannot understand why the next step follows, clarify the evidence or reduce the certainty of the conclusion.

DEVELOPER NOTE

Archive the playable build as well as the source. Tools and dependencies change. A recorded video can preserve context, but the interactive build is the stronger reference when timing and control are central to the finding.

Field checklist: your first playable

- A player verb, constraint, and consequence define the experiment.
- One core question determines the essential feature list.
- Success, failure, and restart form a complete loop.
- Placeholder visuals communicate consistent roles and collision.
- Tuning variables have clear names and ownership.
- The project is in source control with a recoverable working build.
- A new player can attempt the task without continuous coaching.
- Observations are separated from proposed solutions.
- At least one unnecessary feature has been challenged or removed.
- The final decision states what the prototype proves and what it does not.

Conclusion: evidence is progress

Your first prototype should leave you with more than a folder of scripts. It should leave you with a clearer understanding of the game: which decision matters, how players read it, what can be simplified, and which risks deserve the next experiment. Those findings remain valuable even if the original concept changes substantially.

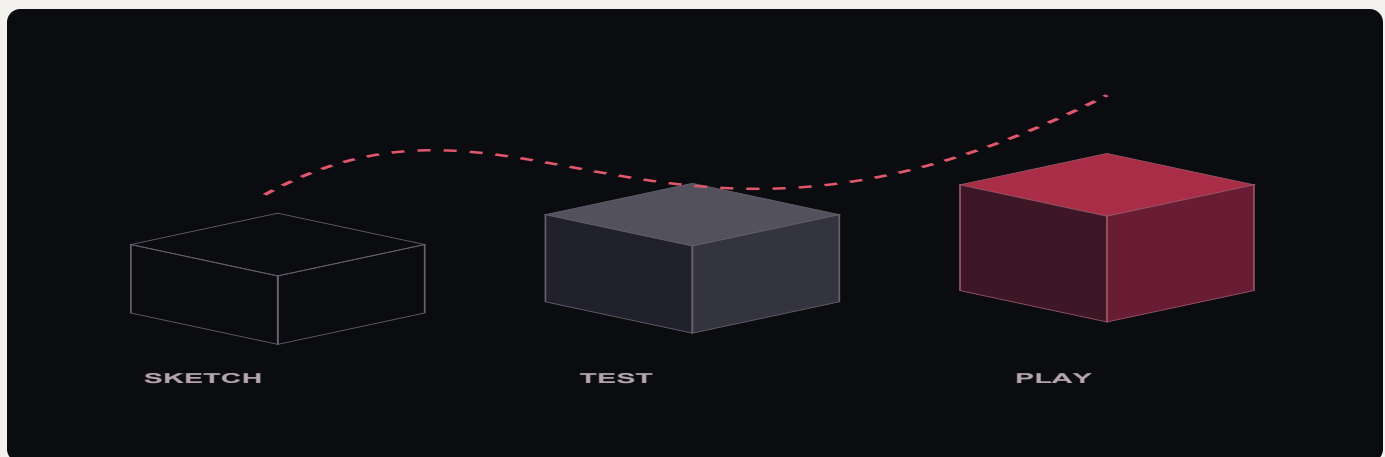
Build the smallest complete loop that can answer a meaningful question. Make it readable, let someone else play it, and keep the result honest. Then choose the next step deliberately. The path from idea to playable is not a straight line from imagination to execution; it is a sequence of useful questions answered through play.

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](#)