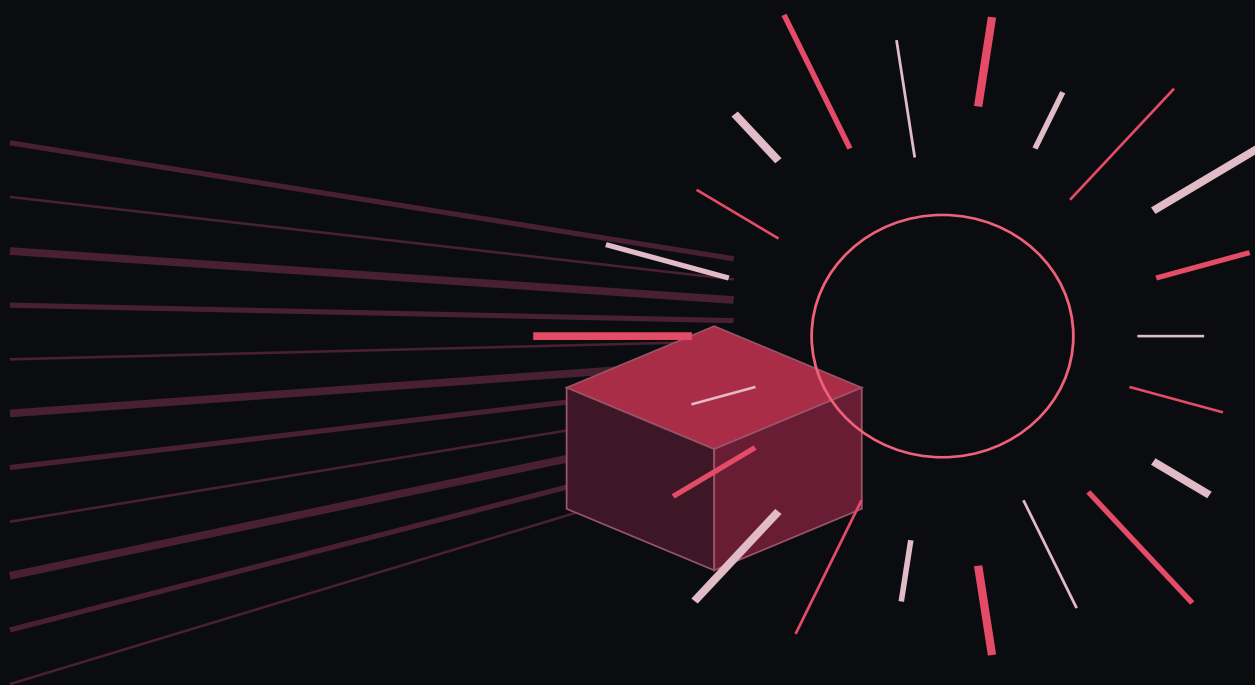


GAME FEEL

Making Simple Mechanics Feel Incredible



DESIGN & FEEDBACK

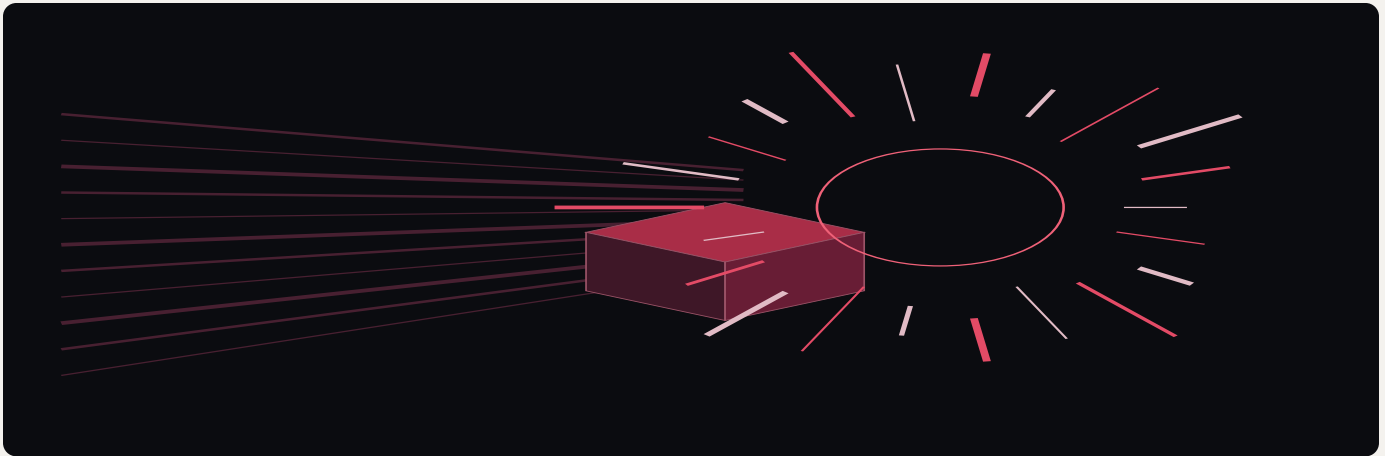
BUILD WHAT YOU IMAGINE.

FIELD GUIDE

A PRACTICAL FIELD GUIDE

Game Feel

Making Simple Mechanics Feel Incredible



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 the rules tangible	4
01 Preserve the player's input	5
02 Shape acceleration, stopping, and air control	6
03 Let the camera explain the action	7
04 Design anticipation, contact, and recovery	8
05 Build impact without losing information	9
06 Make sound and particles describe the event	10
07 Connect UI feedback to the world	11
08 Combine the layers, then remove the excess	12
Workshop checklist: one action, fully tuned	13
Conclusion: coherence beats intensity	14

About Vertex Arc / 15

Introduction: make the rules tangible

Game feel is the relationship between an intention and the response the player perceives. It includes input, simulation, collision, animation, camera behavior, sound, effects, and interface feedback. A change in any one layer can alter the whole action. That is why adding more particles rarely repairs a controller that drops inputs or a camera that hides the landing surface.

This handbook uses an original hypothetical Unity prototype called Switchyard. A small maintenance robot moves between platforms, strikes jammed switches, and redirects moving cargo. There is no finished game or performance claim behind the example. It is a compact test space for movement, impacts, and feedback. The same principles can be adapted to a platformer, action game, puzzle game, or slower exploration experience.

Begin with a repeatable scene and a named tuning profile. Change one meaningful relationship at a time and compare against the previous build. Ask testers what they expected, what they noticed, and when they felt control return. Their answers will help distinguish a timing problem from a presentation problem.

The aim is not maximum intensity. It is a coherent response that supports the game's character and the player's decisions. A gentle, precise interaction can feel excellent. A loud, shaking, brightly flashing interaction can still feel vague. Use every layer to communicate something specific.

THE FEEDBACK CHAIN



Use the result to shape the next attempt.

01 / Preserve the player's input

THE IDEA

An action begins with a request. Capture that request reliably, then decide whether the current game state permits it. Keep those two steps separate. If an action is intentionally unavailable, provide suitable feedback or a readable state. If the request vanishes between input collection and simulation, the result is an implementation failure rather than deliberate difficulty.

Continuous input and momentary input need different treatment. A movement direction describes an ongoing intention. A jump press describes an event that may need to survive until the movement system consumes it. A short input buffer can preserve a press made just before landing, but it must expire and be consumed once. Otherwise an old request can fire unexpectedly.

UNITY EXAMPLE

Switchyard records the time of a jump request separately from its grounded state. When a valid landing occurs, the controller checks whether the request is still recent enough to use. The window is exposed as a tuning value. The developer tests it alongside the actual animation and camera rather than copying a supposedly perfect duration from another project.

KEY TAKEAWAY

Reliable capture comes before generous timing. Make the system understand the request before deciding how forgiving it should be.

TRY THIS

Add a temporary display showing input requests, the current movement state, and the action that consumed each request. Test presses just before landing, during pause, after focus returns, and while a controller is reconnecting. Remove stale requests at deliberate boundaries. Ask a player to attempt rapid sequences and compare their intention with the log.

DEVELOPER NOTE

Do not let several scripts independently interpret the same button unless their responsibilities are explicit. A menu and a gameplay action responding together can make an otherwise correct input system feel unreliable.

02 / Shape acceleration, stopping, and air control

THE IDEA

Maximum speed does not describe how movement feels. Acceleration determines how quickly the character responds, deceleration determines stopping, and turning behavior determines how direction changes. Air control, gravity, and jump timing create additional relationships. Tune these separately before combining them into a final profile.

Think in observable outcomes. Can the player stop on a small platform? Can they reverse direction without an unintended slide? Can they predict the jump apex? These questions are easier to evaluate than whether a controller is generally "snappy." Heavy movement can be responsive when its inertia is consistent and visible.

UNITY EXAMPLE

Switchyard has two profiles: precise workshop movement and a heavier cargo-carrying mode. Both use the same controller, but the loaded mode accelerates more slowly and has a longer stopping distance. The level contains enough space to use that behavior intentionally. The robot does not suddenly become hard to steer in a corridor designed for the faster profile.

KEY TAKEAWAY

Tune motion against the tasks the player must perform. A parameter is only useful in relation to the geometry and decisions around it.

TRY THIS

Build a test lane with a start marker, a stopping target, a low ceiling, a slope, and three jump gaps. Compare two named profiles over the same route. Change only one parameter family at a time. Record overshoots, missed landings, and corrections, then ask which behavior the player could predict more easily.

DEVELOPER NOTE

Keep velocity and displacement distinct. Unity's `CharacterController.Move` takes displacement and does not apply gravity for you. If you use a `Rigidbody`, align movement with the physics approach you selected rather than mixing unrelated transform changes into the same motion. Verify behavior at different frame rates.

03 / Let the camera explain the action

THE IDEA

The camera shapes perceived speed, distance, and responsiveness. Excessive smoothing can make a responsive controller seem delayed. Abrupt collision adjustments can make a predictable route feel unstable. Treat the camera as part of the control system and tune it with the actual player movement, not after movement is considered finished.

Decide what the camera must prioritize: the player, a target, the next landing, or the broader space. These priorities may change by state, but transitions should be understandable. A camera that anticipates movement can help navigation; one that repeatedly overcorrects can force the player to fight its interpretation.

UNITY EXAMPLE

In Switchyard, the camera includes a modest look-ahead toward the robot's travel direction. That offset changes gradually and returns toward neutral when the robot stops. During a precise interaction, it stays stable enough to keep the switch visible. Camera collision is tested near the same obstacles used in ordinary movement, including narrow corners and moving cargo.

KEY TAKEAWAY

Camera behavior should help the player judge the next action. Smoothness is useful only when it preserves timely information.

TRY THIS

Run the same movement route with camera smoothing disabled, moderate, and high. Ask a tester to stop on a target and land on a narrow platform. Compare their corrections. Then test a reduced-motion setting that removes decorative motion while preserving essential tracking. Confirm that the game remains readable and enjoyable.

DEVELOPER NOTE

Separate aiming logic from cosmetic shake where appropriate. A visual impulse should not silently change the direction used to resolve a precise action unless that is an intentional, communicated rule. Provide sensitivity and motion controls suited to the game.

04 / Design anticipation, contact, and recovery

THE IDEA

An action has a rhythm. Anticipation communicates what is about to happen, contact communicates the decisive moment, and recovery communicates the return to control. Their relationship matters more than the length of an animation by itself. A heavy action can use visible commitment; a quick interaction may need almost no delay.

Align the event that changes game state with the perceived contact. If damage occurs before the weapon appears to connect, feedback becomes contradictory. If the animation ends while the character remains locked out for an invisible extra period, players may interpret a deliberate recovery as dropped input.

UNITY EXAMPLE

Switchyard's hammer strike has a readable wind-up, a contact event, and a short return motion. The switch changes state at contact. During tuning, the developer shows the active and recovery phases with a debug color. This makes it possible to compare what the player sees with what the controller allows.

KEY TAKEAWAY

Make action timing legible. Players should be able to recognize commitment, outcome, and the moment they can act again.

TRY THIS

Record a single action and review it frame by frame. Mark the input request, animation start, gameplay contact, impact feedback, and control return. Look for gaps or contradictions. Then test at normal speed: a technically aligned sequence can still feel slow if its anticipation is too long for the task.

DEVELOPER NOTE

Animation events can be useful, but decide how interruption works. If an animation is cancelled before a required cleanup event, the controller must not remain stuck. State ownership and interruption handling are part of feel, even when they are not visible in a screenshot.

05 / Build impact without losing information

THE IDEA

Impact feedback confirms an outcome. It can include a target reaction, brief hit-stop, a sound, camera impulse, particles, and interface information. Give each layer a job. The target reaction identifies what was hit; the sound helps establish material and weight; the UI may explain a resource change. If every layer only says "big," useful information can disappear.

Hit-stop changes timing and should have an explicit scope. Pausing selected animation is different from pausing an entire simulation. Consider input buffering, timers, audio, physics, and multiplayer before implementing a broad time-scale change. A presentation effect should not accidentally change game rules.

UNITY EXAMPLE

The robot strikes a jammed metal switch. The switch compresses slightly, emits a short metallic impact, and changes its indicator. A brief local animation pause emphasizes contact. Decorative particles travel away from the contact point, leaving the new indicator readable. Missing the switch produces a lighter response, so the player can distinguish a successful hit without studying a damage number.

KEY TAKEAWAY

Impact is strongest when its layers agree about what happened. Intensity cannot replace a clear outcome.

TRY THIS

Create three feedback presets: minimal, moderate, and exaggerated. Test repeated hits, misses, and multiple targets. Ask the player to identify the result after each action. Disable sound and shake independently to check whether essential information still reaches them. Keep the strongest preset that remains comfortable and readable over repeated play, not merely the most impressive first hit.

DEVELOPER NOTE

Enemy reactions need rules too. Define when an enemy staggers, whether attacks can interrupt it, and how that is communicated. A reaction that plays without affecting behavior can imply safety the player does not actually have.

06 / Make sound and particles describe the event

THE IDEA

Sound and particles can establish material, direction, scale, and consequence. They should respond to the actual event rather than play the same full-strength effect for every interaction. A glancing contact, a solid strike, and a failed activation may need different responses. Variation is useful when it preserves recognizable meaning.

Particles should respect the player's visual task. Consider coverage, duration, contrast, and where they move. A burst that obscures an enemy's next tell can make a polished action less playable. Sound needs a similar hierarchy: secondary effects should not bury a cue required for the next decision.

UNITY EXAMPLE

Switchyard uses a small contact effect for an ordinary switch and a broader burst when a major mechanism unlocks. The latter is rare and changes the route, so its emphasis is justified. The audio system limits overlapping copies of repetitive mechanical sounds. A sustained machine hum becomes quieter during the important activation cue rather than every new sound simply becoming louder.

KEY TAKEAWAY

Use effects to describe the event's role and consequence. Reserve strong emphasis for moments that deserve attention.

TRY THIS

Test your busiest encounter with every feedback layer active. List the cues players miss. Reduce a competing layer before increasing the missed cue. Then test at a lower listening volume and on ordinary speakers. For particles, capture the frame at maximum overlap and check whether targets, hazards, and routes remain visible.

DEVELOPER NOTE

Preload or prepare frequently used effects when appropriate, and profile their actual cost. Pooling may help repeated effects, but pooled systems need complete reset behavior. A particle trail left over from a previous use can communicate an event that never happened.

07 / Connect UI feedback to the world

THE IDEA

The interface should answer questions the world cannot communicate clearly enough on its own. It can show resource state, cooldowns, objective changes, or the reason an action is unavailable. Avoid repeating the same information at full intensity in several places unless redundancy serves accessibility or an important decision.

Feedback timing matters. A delayed resource update can make an action seem free, while an early success message can contradict a later rejection. Distinguish pending, successful, and failed states for operations that are not immediate. The player should understand whether to wait, retry, or change their action.

UNITY EXAMPLE

The robot's charge meter responds immediately when a powered strike begins, while a separate world indicator confirms that the target mechanism actually activated. A blocked strike returns part of the charge according to a visible rule. The UI does not display "complete" until the mechanism's state is confirmed.

KEY TAKEAWAY

Interface feedback should clarify the relationship between an action, its cost, and its outcome.

TRY THIS

Perform each major action in success, failure, and unavailable states. Write down what the world, sound, and UI communicate at each point. Remove contradictions. Test with increased text size, muted audio, and reduced motion. Important information should remain understandable without depending on one decorative channel.

DEVELOPER NOTE

Keep controller focus and touch feedback in the test. A button's visual response should indicate that it received input, but it must not imply that a longer operation has already finished. Use clear labels for the actual state.

08 / Combine the layers, then remove the excess

THE IDEA

The final action is a composition. Input, motion, animation, camera, sound, particles, enemy response, and UI need to agree about timing and importance. Evaluate them together, but keep the ability to disable each layer. That separation makes it easier to identify which part adds clarity and which part hides a weakness elsewhere.

Write a few feel principles for the project. For Switchyard they might be: precise stopping, readable commitment, short feedback, and stable camera control. These principles prevent every new feature from becoming louder or more dramatic than the last. They also help multiple contributors make compatible choices.

UNITY EXAMPLE

An early version of the powered strike uses shake, zoom, hit-stop, a large burst, a UI flash, and a loud impact. It looks impressive once but becomes tiring during a sequence of switches. The revised version keeps target deformation, a distinct sound, and a small contact burst. The stronger camera impulse is reserved for a major mechanism opening.

KEY TAKEAWAY

Polish is the quality of the relationship between layers, not the number of layers active at once.

TRY THIS

Run a ten-minute session with repeated use of the same action. Record fatigue, missed information, and moments where the effect delays the next decision. Remove one redundant layer and compare. Ask whether the action became less understandable or merely less noisy. Keep accessibility settings active during part of the test rather than checking them only at the end.

DEVELOPER NOTE

Preserve named presets and a previous build. Human memory is poor at comparing subtle timing over long gaps. A direct A/B comparison in the same test scene is more useful than debating which version felt better yesterday.

Workshop checklist: one action, fully tuned

- Input is captured reliably and unavailable states are understandable.
- Movement is tested against real stopping and landing tasks.
- Camera tracking preserves the information needed for the next action.
- Anticipation, contact, and recovery match the gameplay state.
- Hit-stop has a defined scope and does not corrupt timers or input.
- Success, failure, and interruption have distinct responses.
- Particles preserve target and hazard visibility at maximum overlap.
- Sound priorities remain clear during the busiest encounter.
- UI accurately represents cost, progress, and confirmed outcome.
- Reduced motion, muted audio, and alternative inputs remain playable.
- Repeated use stays comfortable after the first impressive moment.
- The final settings are named, documented, and easy to compare.

Conclusion: coherence beats intensity

Start with a reliable action and make its rules visible. Then add only the feedback that helps the player understand and enjoy the result. If a layer creates confusion, fatigue, or an unintended delay, its visual polish does not justify keeping it. The game needs a dependable conversation between intention and response.

Choose one ordinary action in your project and tune it through the whole pipeline before polishing every feature separately. That action becomes a reference for timing, readability, comfort, and intensity. A shared reference makes the rest of the game's feel easier to build consistently.

TECHNICAL REFERENCE

Unity `CharacterController.Move` documentation explains displacement and gravity behavior. Consult the documentation matching your installed version:

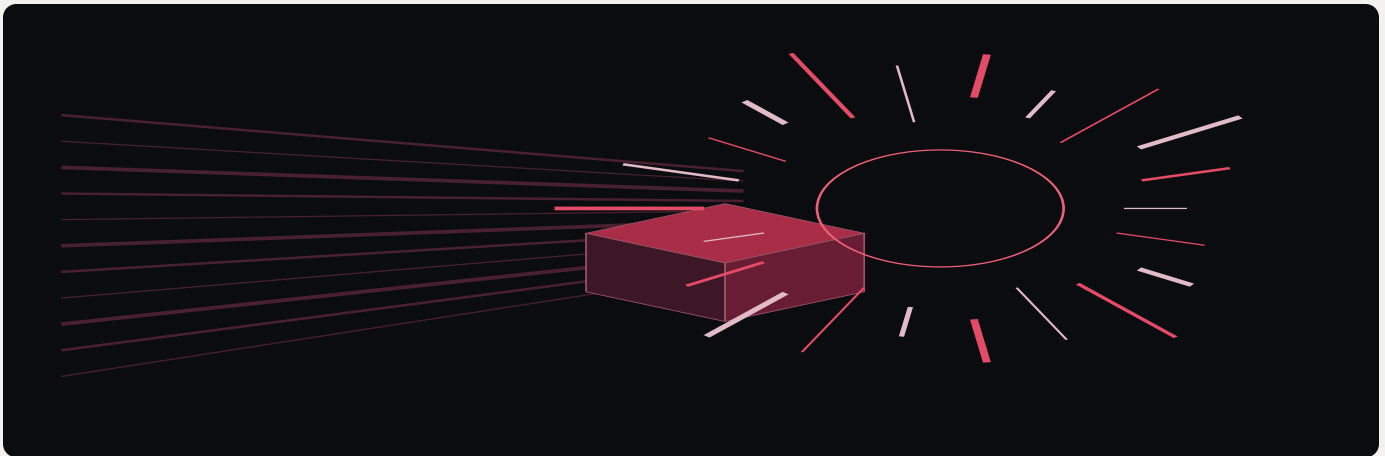
<https://docs.unity3d.com/ScriptReference/CharacterController.Move.html>

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](#)