Sanity Check

Series 1: The Foundation Pilot

Objective: Audit for basic C# syntax, scope, and script modularity.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
1A: Capacity	Variable types.	Using `string` for numbers.	"Use an `int` for coin count".
1B: Scope	Access modifiers.	Every variable is `public`.	"Use `[SerializeField]`".
1C: Gateway	Logic sequences.	Subtracting before checking.	"Check balance first".
1D: Efficiency	The "Heartbeat" (`Update`).	`GetComponent` in `Update()`.	"Cache in `Start()`".
1E: Blueprint	Modular architecture.	"God Scripts" (handling all).	"Create a separate class".
1F: Sanity	Logical output.	Impossible behavior (infinite jumping).	"Audit the math".

Series 2: Logic & Data Flow

Objective: Audit for execution efficiency, memory integrity, and clean architectural data flow.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
2A: Switchboard	Advanced Conditionals	"Arrow Code" (Deep nested if-else)	"Use a switch expression"
2B:	Memory Allocation	Lists growing without capacity	"Set initial capacity to 50"
2C: Iteration	Execution Safety	Modifying a list inside	"Use a reverse-for loop"
2D:	Data Integrity	"Get" methods with Side	"Make this a pure
2E: Value/Ref	Memory Storage	Boxing Value Types into	"Use a Generic List<int>"
2F: Utilities	Memory Persistence	Static variables for unique state	"Use Instance-based variables"
2G: Logic	System Integration	Cascading logic/memory failures	"Perform a Full-System Audit"

Series 3: The Variable Vault

Objective: Audit for numerical precision, logic clarity, and Inspector ergonomics.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
3A: Magic	Logic Rigidity	"Hardcoded Values" (numbers like `10.5f` buried in code)	"Use Serialized Named Constants"
3B: Precision	Math Integrity	"Rounding Deadlock" (using `int` for smooth movement)	"Use Floats for sub-pixel precision"
3C: Strings	Silent Crashes	"Blind Coupling" (relying on raw text tags/names)	"Replace raw strings with Enums/Constants"
3D: Logic	Audit Fatigue	"Double Negatives" (naming variables `isNotGrounded`)	"Use Positive Inquiry (isGrounded)"
3E: Visuals	Dashboard Ergonomics	"The Variable Dump" (long, unlabeled list in Inspector)	"Use [Header] and [Space] attributes"
3F: Guidance	Documentation Debt	"Mystery Dials" (no context for what a variable does)	"Add [Tooltip] instructions for all fields"

Series 4: The Control Tower

Objective: Audit for control flow efficiency, logic flattening, and method isolation.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
4A: Pyramid	Depth Perception	"Nested If-Statements" (logic buried 3+ levels deep)	"Use Guard Clauses to flatten logic"
4B: Switches	State Efficiency	"Else-If Ladders" (re-checking the same Enum multiple times)	"Use Switch Statements for state logic"
4C: Echoes	Math Waste	"Redundant Math" (calculating Distance twice in one frame)	"Store expensive math in local variables"
4D: Ternary	Code Density	"Clunky Assignments" (5 lines for a simple A/B choice)	"Use Ternary Operators for simple logic"
4E: Gates	Logic Ambiguity	"Ungrouped Gates" (mixing && and || without brackets)	"Use Parentheses for explicit evaluation"
4F: Isolation	Update Bloat	"The Update Dump" (low-level math clogging the main loop)	"Extract conditions into named methods"

Series 5: The Collection Agency

Objective: Audit for data structure efficiency, lookup speed, and memory sanitation.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
5A: Stability	Fixed Hardware	"Dynamic Drifting" (using a List for a fixed set of 4 engines)	"Use a fixed-size Array for static hardware"
5B: Growth	Mission Scaling	"Rigid Overflow" (using a fixed Array for an unknown number of items)	"Use a List for dynamic mission objectives"
5C: Speed	Search Friction	"The Scavenger Hunt" (looping through 1,000 items to find one ID)	"Use a Dictionary for instant ID lookups"
5D: Iteration	Cycling Drag	"Garbage Generators" (using `foreach` in high-frequency Update loops)	"Use an indexed `for` loop to avoid allocations"
5E: Sanitation	Memory Ghosting	"Permanent Passengers" (leaving data in a List after a mission ends)	"Clear collections on Reset or Disable"
5F: Visibility	Dashboard Blindness	"The Hidden Inventory" (private collections invisible to the Pilot)	"Use `[SerializeField]` to expose lists"

Series 6: The Communication Array

Objective: Audit for script decoupling, signal clarity, and modular dependency management.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
6A: Links	Dependency Drag	"Hard Coupling" (Drone script won't compile without the UI script)	"Decouple the drone logic from the UI using events"
6B: Signals	Broadcast Scope	"Manual Messengers" (Drone manually calling UI, Audio, and FX)	"Use UnityEvents to broadcast signals to listeners"
6C: Dispatch	Subscription Health	"Zombie Links" (C# Actions that never unsubscribe)	"Ensure every Action has a matching -= unsubscription"
6D: Statics	Global Corruption	"Shared Steering" (Multiple drones sharing a single static variable)	"Remove static keywords from instance-specific data"

6E: Managers	Singleton Bloat	"The God Object" (One GameManager managing everything)	"Refactor into specialized, modular managers"
6F: Bridges	Type Rigidity	"Class Discrimination" (Checking for specific types like 'Wall' or 'Tree')	"Use an Interface (IInteractable) to bridge types"

Series 7: The Syntax Scout

Objective: Audit for code legibility, proper encapsulation, and organizational scaffolding.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
7A: Access	Exposed Wires	"Naked Variables" (Sensitive data marked as `public`)	"Make variables private and use [SerializeField] for Inspector access"
7B: Casing	Visual Static	"Naming Soup" (Using `snake_case` or inconsistent capitalization)	"Follow C# standards: camelCase for variables, PascalCase for methods"
7C: Fog	Magic Numbers	"Mystery Math" (Hard-coded numbers like `0.75f` without a label)	"Replace magic numbers with named `const` variables"
7D: Traffic	Naming Overlap	"Hangar Collisions" (Scripts with the same name in the global namespace)	"Wrap scripts in unique Namespaces (e.g., Academy.Flight)"
7E: Noise	Redundancy	"Echo Comments" (Comments that just restate the code: `// adds 1 to x`)	"Prune obvious comments; only explain the 'Why' or the intent"
7F: Regions	Scroll of Doom	"Structural Clutter" (A 500-line script with no organization)	"Organize code into collapsible #region blocks"

Series 100: The Flight Engineer

Objective: Audit for engine-specific systems like Data, Physics, and UI.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)

100A: Data	Data decoupling.	Data "welded" into logic.	"Use a `ScriptableObject`".
100B: Physics	Interaction types.	`OnCollisionEnter` for pickups.	"Use `OnTriggerEnter`".
100C: Instrumentation	UI frameworks.	Legacy `UnityEngine.UI`.	"Use the UI Toolkit (UXML)".

Series 101: Asset Workflow

Objective: Audit for Inspector clarity, serialization safety, and fleet-wide modularity.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
101A: Clarity	Inspector UI	"Wall of Numbers" (unlabeled variables)	"Use Headers and Tooltips"
101B: Serialization	Nested Data	Custom classes missing from Inspector	"Mark this class as [Serializable]"
101C: Modular	Strategy Patterns	"The Strategy Weld" (hard-coded logic)	"Use an abstract FlightBehavior asset"
101D: Tools	Testing Workflow	"Manual Testing Drag" (hitting Play to test logic)	"Create a Custom Editor button"
101E: I/O	Data Persistence	"Black Box" path errors (hard-coded C:/ drive)	"Use persistentDataPath for JSON"
101F: Prefabs	Fleet Sync	"Instance Drift" (unique logic on prefab instance)	"Create a Prefab Variant"
101G: Events	Workflow Audit	Cascading asset/serialization failures	"Perform a Full-Workflow Audit"

Series 102: The Inspector Insight

Objective: Audit for component dependencies, reference safety, and Inspector clarity.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
102A: Requirements	Component Integrity	"Missing Engines" (Code that crashes because a Rigidbody or AudioSource is missing)	"Use [RequireComponent] to enforce the presence of needed components"
102B: Nulls	Defensive Execution	"The Red Wall" (NullReferenceExceptions caused by unassigned Inspector slots)	"Always perform a null check before accessing an Inspector-assigned reference"

102C: Identity	Silent Failures	"String Typos" (Using `other.tag == "player"` instead of the case-sensitive standard)	"Use CompareTag() for identity checks and avoid manual string comparisons"
102D: Timing	Race Conditions	"Early Ignition" (Accessing other scripts in Awake before they are ready)	"Follow the protocol: Awake for Self, Start for Others"
102E: Visibility	Dashboard Noise	"Cluttered Cockpit" (Exposing internal math or private data to the Inspector)	"Use [SerializeField] surgically; hide internal variables from the Pilot"
102F: Context	Parameter Safety	"Cryptic Boxes" (Variables with no labels, units, or safe boundaries)	"Add [Tooltip] and [Range] to guide the Pilot and prevent bad data entry"

Series 103: The Trigger Zone

Objective: Audit for physical interaction topology, layer efficiency, and memory-safe spatial queries.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
103A: Topology	Interaction Intent	"Invisible Walls" (Using solid collisions for ghost sensors/zones)	"Set the collider to IsTrigger and use OnTriggerEnter for ghost detection"
103B: Matrix	Processing Smog	"Universal Observation" (Everything on the Default layer checking collisions)	"Assign unique layers and disable unnecessary interactions in the Physics Matrix"
103C: Masking	Sensor Precision	"Self-Sensing" (Raycasts hitting the drone's own hull or ignoring layers)	"Use a LayerMask to filter raycasts at the engine level"
103D: States	Temporal Logic	"Status Hanging" (Forgetting to reset a flag in OnTriggerExit)	"Implement OnTriggerExit to clear states when an object leaves the zone"
103E: Jitter	Signal Noise	"Rapid-Fire Events" (Alarms flickering on/off at the exact edge of a zone)	"Add a 0.5s temporal buffer before triggering the state change"

103F: Queries	Memory Health	"Allocation Spam" (Creating new arrays inside Update() for physics sweeps)	"Use NonAlloc physics queries with pre-defined result arrays"

Series 104: The Physics Lab

Objective: Audit for simulation stability, layer collision efficiency, and frame-rate independent execution.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
104A: RigidBody	Movement Logic	"**The Teleporter**" (Moving a Rigidbody via `transform.position`)	"Apply `velocity` or `AddForce` to move physically"
104B: Colliders	Interaction Type	"**The Solid Coin**" (Bouncing off items intended for collection)	"Set `IsTrigger` to True and use `OnTriggerEnter`"
104C: Matrix	Collision Rules	"**The Self-Destruct**" (Bullets hitting the ship that fired them)	"Uncheck the collision pair in the Physics Matrix"
104D: Forces	Energy Application	"**The Feather Jump**" (Using weak continuous force for an explosion)	"Use `ForceMode.Impulse` for instant acceleration"
104E: Raycast	Sensor Precision	"**The Infinite Laser**" (Raycasting without a LayerMask)	"Apply a LayerMask to filter specific targets"
104F: Timing	Loop Safety	"**The Jitterbug**" (Applying physics forces inside `Update`)	"Move all physics logic to `FixedUpdate`"

Series 105: The Clone Factory

Objective: Audit for instantiation efficiency, hierarchy organization, and initialization order safety.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)

105A: Blueprint	Asset Management	"**The Scene Duplicate**" (Manually copying objects in Hierarchy)	"Create a Prefab to serve as the master blueprint"
105B: Spawner	Creation Logic	"**The Teleporting Bullet**" (Moving the original asset instead of cloning)	"Use `Instantiate` to create a new runtime copy"
105C: Parenting	Scene Organization	"**The Hierarchy Flood**" (Spawning objects at the root level)	"Parent new instances to a Container transform"
105D: Recycler	Memory Hygiene	"**The Ghost Script**" (Destroying `this` instead of `gameObject`)	"Call `Destroy(gameObject)` with a time delay"
105E: Ignition	Execution Order	"**The Race Condition**" (Two scripts talking in `Start`)	"Initialize variables in `Awake`; talk in `Start`"
105F: Linking	Dependency Injection	"**The Lost Child**" (Clones searching for targets via `Find`)	"Capture the instance reference and set data immediately"

Series 106: The Data Vault

Objective: Audit for data persistence integrity, file path safety, and serialization structure.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
106A: Memory	Persistence Scope	"**The Static Trap**" (Assuming `static` variables survive quitting)	"Write data to Disk using `PlayerPrefs` or File I/O"
106B: Keys	String Safety	"**The Typo Hazard**" (Using raw strings like "Gold" vs "gold")	"Use `const string` variables for all Save Keys"
106C: Models	Data Structure	"**The Prefs Spam**" (Saving 50 separate keys for an inventory)	"Create a `[Serializable]` class to hold complex data"
106D: JSON	Serialization	"**The Binary Blob**" (Using obsolete `BinaryFormatter`)	"Use `JsonUtility` to serialize data to text"
106E: Paths	Platform Safety	"**The Desktop Hardcode**" (Saving to "C:/GameData")	"Use `Application.persistentDataPath`"

106F: Checksum	Integrity	"**The Open Book**" (Saving plain text values like "Gold: 999")	"Append a Hash/Checksum to verify file integrity"

Series 200: Flight Automation

Objective: Audit for state integrity, autonomous navigation, and modular event-driven communication.

Rule	The Audit	Red Flag (Danger)	Pilot Command (Safe)
200A: States	State Integrity.	"Boolean Soup" (overlapping	"Use an **Enum** and **Switch**
200B: Navigation	Autopilot Efficiency.	Manual `MoveTowards` ignoring obstacles.	"Delegate to the **NavMeshAgent**."
200C: Radio	Modular	"Welded Cockpit" (Direct	"Decouple via **C# Actions**
200D:	Global	Duplicate managers across	"Enforce the **Singleton**
200E: Yoke	Fly-By-Wire Input.	Hard-coded `KeyCode.Space` polling.	"Use **Input Actions** (New Input System)."
200F:	Visual	Jarring, "Binary" UI snaps.	"Smooth via **Coroutine**

Series 201: Flight Automation

Objective: Audit for state-based logic, non-blocking temporal flow, and predictive spatial awareness.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
201A: Autopilot	State Integrity	"Boolean Jungle" (overlapping true/ false flags)	"Use an Enum-based State Machine"
201B: Temporal	Execution Timing	"Update Bloat" (manual timers in the Update loop)	"Use a Coroutine (IEnumerator)"
201C: Async	Thread Safety	"Data Stall" (blocking the game during loading)	"Use Async/Await for data tasks"
201D: Sensors	Predictive Vision	"Blind Bumping" (reactive OnCollision code)	"Use Physics.Raycast or Overlap"
201E: NavPath	Pathfinding	"The Magnet Effect" (walking directly into walls)	"Use a NavMeshAgent"
201F: Feedback	Visual Sync	"The String Trigger" (Play("Anim") commands)	"Use Animator Parameters"

201G: Capstone	Automation Audit	Cascading logic/sensor/timing failures	"Perform a Full-Automation Audit"

Series 202: Performance Tuning

Objective: Audit for memory allocation, CPU/GPU bottlenecks, and resource management to ensure 60+ FPS stability.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
202A: Garbage	Memory Allocation	"**The String Stitcher**" (String concatenation in Update)	"Use Cached Strings or StringBuilder"
202B: Pooling	Instantiation Cost	"**The Factory Spammer**" (Instantiate/Destroy in loop)	"Implement an Object Pool"
202C: Caching	Reference Lookups	"**The Constant Search**" (GetComponent inside Update)	"Cache References in Awake"
202D: CPU Math	Loop Efficiency	"**The Elegant Stall**" (Using LINQ queries every frame)	"Use Standard For-Loops"
202E: Profiling	Telemetry	"**The Blind Optimization**" (Guessing where the lag is)	"Use Profiler.BeginSample"
202F: Batching	Draw Calls	"**The Draw Call Jam**" (Accessing .material directly)	"Use MaterialPropertyBlock"
202G: Capstone	Tuning Audit	Cascading lag, memory spikes, and stutter	"Perform a Full-Performance Audit"

Series 203: The Avionics Interface

Objective: Auditing for modern UI architecture (Toolkit), MVC decoupling, and resolution independence.

Lesson	The "Passenger" Prompt (Weak)	The "Pilot" Command (Strong)	Why it Works
203A: Toolkit	"Create a Canvas with a Text object."	"Create a `UIDocument` and query the `rootVisualElement` to bind the interface."	Transitions from heavy "GameObject UI" (Canvas) to the professional, lightweight "Visual Tree" (UI Toolkit).

203B: Query	"Find the score label."	"Cache the element using `root.Q<Label>('Score')` in `OnEnable`."	Avoids `GameObject.Find`, which scans the entire scene universe instead of just the cockpit dashboard.
203C: MVC	"Update the health text inside the Player script."	"Fire an `OnHealthChanged` event from the Player (Model) and let a separate UI Controller update the View."	Prevents "Hard Coupling." The engine (Player) shouldn't break just because the fuel gauge (UI) is missing.
203D: Events	"Use the OnClick box in the Inspector."	"Register a callback in code using `button.RegisterCallback<ClickEvent>(OnFire)`."	Keeps logic encapsulated and trace-able, avoiding the "Invisible Wire" problem of Inspector events.
203E: Flexbox	"Set the X/Y position to 500, 500."	"Use Flexbox properties (`JustifyContent`, `AlignItems`) to position elements relative to the screen."	Ensures the cockpit instruments adjust automatically to different screen sizes (Responsive Design).
203F: Custom	"Instantiate 50 images for the grid."	"Create a custom `VisualElement` and draw the lines using the `MeshGenerationContext`."	Reduces 50 "Draw Calls" to 1, ensuring the HUD renders instantly without lag.

Series 204: The Animation Servo

Objective: Audit for state machine efficiency, kinematic accuracy, and robust event handling to ensure fluid, physics-aware motion.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
204A: Hashes	String Optimization	"**The Magic String**" (Using `SetBool("Run")` in Update)	"Cache `Animator.StringToHash` IDs"
204B: Blending	Motion Fluidity	"**The Hard Swap**" (Switching clips via `if/else`)	"Use Blend Trees for parametric mixing"

204C: Layers	Asset Complexity	"**The Combo Clip**" (Baking "Run+Shoot" into one file)	"Use Avatar Masks and Override Layers"
204D: IK Link	Physical Alignment	"**The Ghost Hand**" (Hands floating near interactables)	"Implement `OnAnimatorIK` logic"
204E: Behaviors	Logic Durability	"**The Hidden Hook**" (Events buried inside Animation Clips)	"Use `StateMachineBehaviour` scripts"
204F: Root Drive	Physics Coupling	"**The Ice Skater**" (Transform movement conflicting with feet)	"Couple Physics with `ApplyRootMotion`"
204G: Capstone	Servo Audit	Glitching limbs, fragile logic, and sliding feet	"Perform a Full Kinematic Audit"

Series 205: The Control Link

Objective: Audit for hardware abstraction, input phase handling, and device-agnostic design to ensure responsive, rebindable controls.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
205A: Actions	Hardware Abstraction	"**The KeyCode Shackle**" (Hardcoding `Input.GetKey(KeyCode.W)`)	"Use Input Actions (Abstracted Controls)"
205B: Phases	Event Lifecycle	"**The Boolean Hold**" (Manually tracking state variables)	"Subscribe to `.performed` and `.canceled`"
205C: Axes	Vector Math	"**The Pythagorean Cheat**" (Moving faster diagonally)	"Use Vector2 Composites (Normalized)"
205D: Processors	Signal Hygiene	"**The Manual Deadzone**" (Writing drift logic in gameplay code)	"Apply Processors in the Asset Pipeline"
205E: Schemes	Context Awareness	"**The Static Tutorial**" (Showing "Press Space" to a gamepad user)	"Detect `playerInput.currentControlScheme`"
205F: Rebinding	User Freedom	"**The Legacy Launcher**" (Relying on external config dialogs)	"Use the Interactive Rebinding API"

205G: Capstone	Control Audit	Drift, hardware lock-in, and confusing prompts	"Perform a Full Input Architecture Audit"

Series 300: The Navigator

Objective: Audit for spatial intelligence, asynchronous logic, and visibility optimization.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
300A: Radar	Raycast	Unfiltered/Infinite rays.	"Use `LayerMask` + Distance".
300B/C: Timing	Task management.	`Update()` timers / Sync I/O.	"Use `IEnumerator` / `async`".
300D: Sight	FOV detection.	3D trigger cones for	"Use `Vector3.Dot` math".
300E: Stability	Rotation math.	`eulerAngles += ...`.	"Use `Quaternion.RotateToward`
300F: Awareness	Situational culling.	"Always On" off-screen logic.	"Use Visibility Culling".

Series 206: The Sonic Engine

Objective: Audit for signal routing, spatial realism, and memory-efficient audio handling to ensure a professional soundscape.

Rule	The Audit (Focus)	Red Flag (Danger)	Pilot Command (Safe)
206A: Mixers	Signal Routing	"**The Script Slider**" (Setting `source.volume` directly)	"Route signals through Audio Mixer Groups"
206B: 3D Ear	Spatialization	"**The Flat Boom**" (2D sounds ignoring distance)	"Enable Spatial Blend and Rolloff Curves"
206C: Ducking	Dynamic Mixing	"**The Coroutine Fade**" (Code fighting to lower volume)	"Use Sidechain Compression in the Mixer"
206D: Variance	Organic Feel	"**The Robot Repeater**" (Identical pitch/sample looping)	"Modulate Pitch and Clip selection"

206E: Pooling	Allocation	"**The Instant Spawner**" (`PlayClipAtPoint` garbage)	"Use an Audio Source Pool"
206F: Snapshots	Transitions	"**The Effect Toggle**" (Enabling filters causing pops)	"Transition via Mixer Snapshots"
206G: Capstone	Sonic Audit	Phasing artifacts, garbage spikes, and flat audio	"Perform a Full Sonic Architecture Audit"

Series 301: The Navigator Protocols

Objective: Auditing for global persistence, non-blocking scene transitions, and memory hygiene.

Lesson	The "Passenger" Prompt (Weak)	The "Pilot" Command (Strong)	Why it Works
301A: Compass	"Make a persistent manager."	"Create a **Singleton** for GameManager. In Awake, **destroy any duplicates** to prevent instance overlap."	Prevents "Telemetry Amnesia" while ensuring only one "Commander" exists in memory.
301B: Airspace	"Load the next level."	"Use **LoadSceneAsync** to load the next sector in the background and update a loading bar."	Eliminates the "Synchronous Jump" that freezes the cockpit during transit.
301C: Mission	"Save the quest status."	"Store quest states in a **ScriptableObject** and check them via a global MissionManager."	Decouples mission logic from individual levels to prevent "Mission Amnesia".
301D: Comms	"Tell the UI the quest is done."	"In MissionManager, broadcast a **static Action** OnQuestDone. Have the UI subscribe to it."	Replaces "Spaghetti Circuits" with a clean signal bus between global systems.
301E: Buffers	"Load the interior scene."	"Use **LoadSceneMode.Additive** to stream the building interior without unloading the world."	Maintains "Regional Context"—critical for Digital Twins and seamless world-streaming.
301F: Garbage	"Close the mission window."	"When the window closes, **unsubscribe** from all global events to prevent ghost references."	Cleans the "Hangar" of dead telemetry, preventing slow memory-based crashes.

Series 302: System Architecture Protocols

Objective: Transitioning from "Monolithic Scripts" to "Decoupled Patterns" (Command, Strategy, Factory).

Lesson	The "Passenger" Prompt (Weak)	The "Pilot" Command (Strong)	Why it Works
302A: Command	"Shoot when I press Space."	"Bind the input to an `ICommand` interface so we can rebind keys or undo actions later."	Decouples "Input" from "Action," preventing the cockpit controls from being welded to the engine.
302B: Strategy	"If walking, walk. If flying, fly."	"Implement an `IMovementStrategy` interface to swap movement behaviors at runtime without `if/else` chains."	Eliminates "Monolith Methods" and allows the aircraft to change engines mid-flight.
302C: Factory	"Spawn a red drone here."	"Request a 'Red' unit from the `DroneFactory` to ensure ammo and team settings are initialized correctly."	Centralizes "Construction Logic," preventing "Initialization Drift" where copies differ from the original.
302D: Services	"Call `Audio.Instance.Play()`."	"Retrieve the `IAudioService` from the Service Locator rather than hard-linking to a singleton."	Decouples systems, allowing you to swap out the "Real Radio" for a "Test Radio" without crashing the plane.
302E: Components	"Make a Flying Tank class."	"Use Composition: Attach a `FlightComponent` and `TrackComponent` to the entity chassis."	Avoids "Inheritance Hell" (The Diamond Problem) by bolting capabilities onto objects instead of extending classes.
302F: Events	"Tell the UI to update the score."	"Publish a `ScoreChanged` event to the Bus, and let the UI subscribe to it independently."	Prevents "Spaghetti Wiring" where the Engine script accidentally knows about the UI script.

Series 303: Procedural Engine Protocols

Objective: Transitioning from "Hand-Crafted Scenes" to "Algorithmic Generation" while maintaining 60 FPS.

Lesson	The "Passenger" Prompt (Weak)	The "Pilot" Command (Strong)	Why it Works
303A: Noise	"Generate a random terrain."	"Use `Mathf.PerlinNoise` scaled by frequency to generate a coherent heightmap."	Prevents "TV Static" topology (jagged spikes) by ensuring height values are related to their neighbors.
303B: Meshes	"Create a plane with code."	"Generate a procedural mesh using a shared-vertex grid to minimize the memory footprint."	Avoids "Geometry Bloat" by ensuring the vertex count is optimized for the GPU.
303C: Seeds	"Randomly place trees."	"Initialize `Random.InitState(seed)` before generation to ensure the world is deterministic."	Prevents "Ephemeral Data" so that bugs can be reproduced and multiplayer maps match.
303D: Chunks	"Make an infinite runner."	"Implement a Chunk Manager that pools and recycles terrain segments behind the player."	Prevents "Resource Exhaustion" (Crash) by keeping the active memory footprint constant.
303E: Jobs	"Calculate 10,000 points."	"Schedule an `IJobParallelFor` struct with the Burst Compiler to handle the math off the main thread."	Eliminates "The Main Thread Freeze" by utilizing all CPU cores instead of just one.
303F: Poisson	"Scatter coins on the floor."	"Use Poisson Disc Sampling to generate positions with a strict minimum radius."	Prevents "Collision Chaos" (overlapping objects) while maintaining a natural, organic distribution.

Series 304: Neural Link Protocols

Objective: Transitioning from "Reactive Logic" to "Cognitive Architecture" using Behavior Trees and Swarm Optimization.

Lesson	The "Passenger" Prompt (Weak)	The "Pilot" Command (Strong)	Why it Works

304A: Trees	"Make the enemy patrol, chase, and flee."	"Implement a Behavior Tree with a Selector for high-level states (Flee/Chase) and Sequences for actions."	Prevents "Spaghetti State Machines" where transition logic is duplicated across every single state.
304B: Memory	"Pass the target to the gun script."	"Write the target to a shared Blackboard variable that the Gun Node can read independently."	Decouples the sensors from the weapons, allowing systems to communicate without "Referential Glue."
304C: Utility	"If health is low, heal."	"Calculate a Utility Score based on Health % and Threat Distance, and execute the highest-scoring action."	Replaces robotic "Threshold Blindness" with fuzzy logic, allowing the AI to make nuanced decisions.
304D: Grids	"Make drones avoid each other."	"Register units into a Spatial Hash Grid and only check collisions against local neighbors."	Eliminates the "N-Squared Catastrophe" where checking every unit against every other unit melts the CPU.
304E: Flocking	"Follow the leader."	"Apply Separation, Alignment, and Cohesion forces to create emergent flocking behavior."	Replaces "Mechanical Parenting" with organic group dynamics that can flow around obstacles.
304F: Pulse	"Look for the player."	"Use a Coroutine with a random start delay to time-slice the sensor checks across different frames."	Prevents "Frame Stutter" caused by hundreds of AI agents firing Raycasts at the exact same millisecond.

Series 305: Connected Fleet Protocols

Objective: Transitioning from "Single-Player Solipsism" to "Server-Authoritative Networking" using Netcode for GameObjects (NGO).

Lesson	The "Passenger" Prompt (Weak)	The "Pilot" Command (Strong)	Why it Works
305A: Host	"Move the player when I press W."	"Wrap input logic in an `IsOwner` check to ensure the local client only controls their own avatar."	Prevents "Input Bleed" where one keyboard press inadvertently moves every player on the server.

305B: NetVars	"Create a health variable."	"Use a `NetworkVariable<int>` and only modify it on the Server to ensure all clients see the same value."	Prevents "State Desync" where a player dies on one screen but remains alive on another.
305C: RPCs	"Shoot a bullet."	"Call a `ServerRpc` to request the shot, then let the server handle the logic."	Prevents "Phantom Actions" where bullets exist only on the shooter's screen and do no damage.
305D: Authority	"Open the door."	"Validate the request (Distance/Locked Status) inside the `ServerRpc` before applying changes."	Prevents "Client Trust" vulnerabilities where hackers can trigger actions from across the map.
305E: Ghosts	"Sync the position."	"Add a `NetworkTransform` component with interpolation enabled to smooth out latency."	Eliminates "Lag Jitter" (teleporting objects) caused by raw position updates over the internet.
305F: Spawning	"Instantiate a fireball."	"Instantiate the object on the Server and call `.Spawn()` on its `NetworkObject` component."	Prevents "Ghost Objects" that exist locally but are invisible to the rest of the fleet.

Series 306: Render Pipeline Protocols

Objective: Transitioning from "CPU Logic" to "GPU Math" using Shaders, HLSL, and Compute Buffers.

Lesson	The "Passenger" Prompt (Weak)	The "Pilot" Command (Strong)	Why it Works
306A: Graph	"Scroll the water texture."	"Create a Shader Graph that multiplies the Time Node by a Speed Property to offset UVs."	Prevents "Bus Traffic" (Data Transfer) by keeping the math entirely on the video card.
306B: HLSL	"Make the pixel transparent if it is black."	"Use the HLSL `clip()` or `step()` function to mask pixels without using if/else logic."	Prevents "Thread Divergence" caused by branching logic, which stalls GPU cores.

306C: Vertex	"Make the flag wave."	"Apply a Sine Wave offset to `positionOS` (Object Space) inside the Vertex Shader stage."	Eliminates "Geometry Bottlenecks" by avoiding looping through thousands of vertices on the CPU.
306D: Batch	"Give every asteroid a random color."	"Use a `MaterialPropertyBlock` to set the color property without breaking the Instancing Batch."	Prevents "State Thrashing" (Draw Call spikes) caused by creating unique Material instances.
306E: Compute	"Update 50,000 particles."	"Write a Compute Shader Kernel to update positions in parallel and Dispatch it from C#."	Moves physics from "Serial Processing" (CPU) to "Massive Parallel Processing" (GPU).
306F: PostFX	"Make the screen red when hit."	"Get the Vignette override from the active `PostProcessVolume` and animate its intensity."	Avoids legacy "GUI Overdraw" and integrates effects seamlessly into the render pipeline.