— Rigonix3D----

Smart Project Optimizer

for Unity

Official Documentation

Version 1.0 | Rigonix3D

rigonix3d.com

1. Introduction

Smart Project Optimizer is a guided optimization and diagnostics suite for Unity projects. It is designed to help developers detect performance bottlenecks, clean projects safely, and apply optimization fixes through a structured Editor workflow — without blindly changing settings.

The tool is inspired by the optimization categories Unity highlights in its official mobile optimization guide, covering: profiling, memory, adaptive performance, code architecture, project configuration, assets, graphics and GPU optimization, user interface, audio, animation, physics, and workflow.

> Smart Project Optimizer does not promise magical instant performance gains. Every optimization should be benchmarked before and after on your target device. The asset surfaces findings and guides you — it does not replace profiling.

1.1 What It Does

- Scans your entire Unity project across multiple system categories.
- Calculates an Optimization Health Score out of 100.
- Estimates likely gains in build size, memory, and frame rate.
- Surfaces a Quick Wins mode — safe one-click improvements with zero risk.
- Lets you apply individual or batch fixes with optional backup and quarantine.
- Exports readable reports in Markdown, CSV, and HTML formats.

1.2 Who It Is For

- Unity asset publishers and tool developers.
- Mobile game developers optimizing for device constraints.
- Solo indie developers who need a structured optimization checklist.
- Freelance Unity developers doing project cleanup for clients.
- Small teams needing a before/after handoff report.

2. Installation

2.1 Requirements

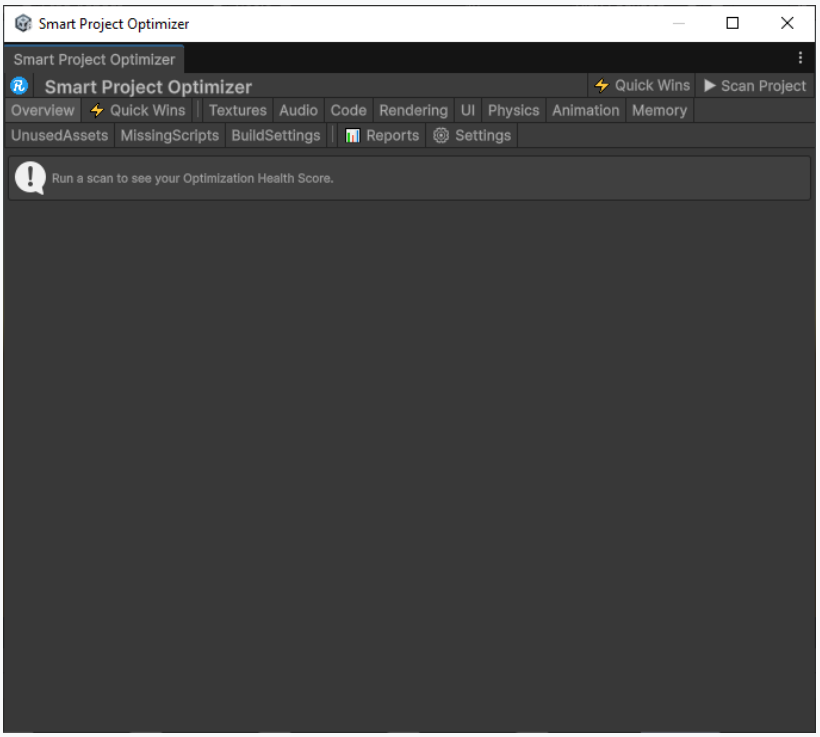
Requirement	Details
Unity Version	Unity 2020.3 LTS or newer (2021 LTS / 2022 LTS recommended)
Render Pipeline	Compatible with Built-in, URP, and HDRP
Platform	Editor only — Windows, macOS, Linux Editor supported
Dependencies	None — no third-party packages required
License	Editor-only asset — not included in builds

2.2 Opening the Tool

After import, open the tool from the Unity menu bar:

```
Tools > Smart Project Optimizer
```

The Smart Project Optimizer window will open. You can dock it anywhere in the Unity Editor layout like any other editor window.



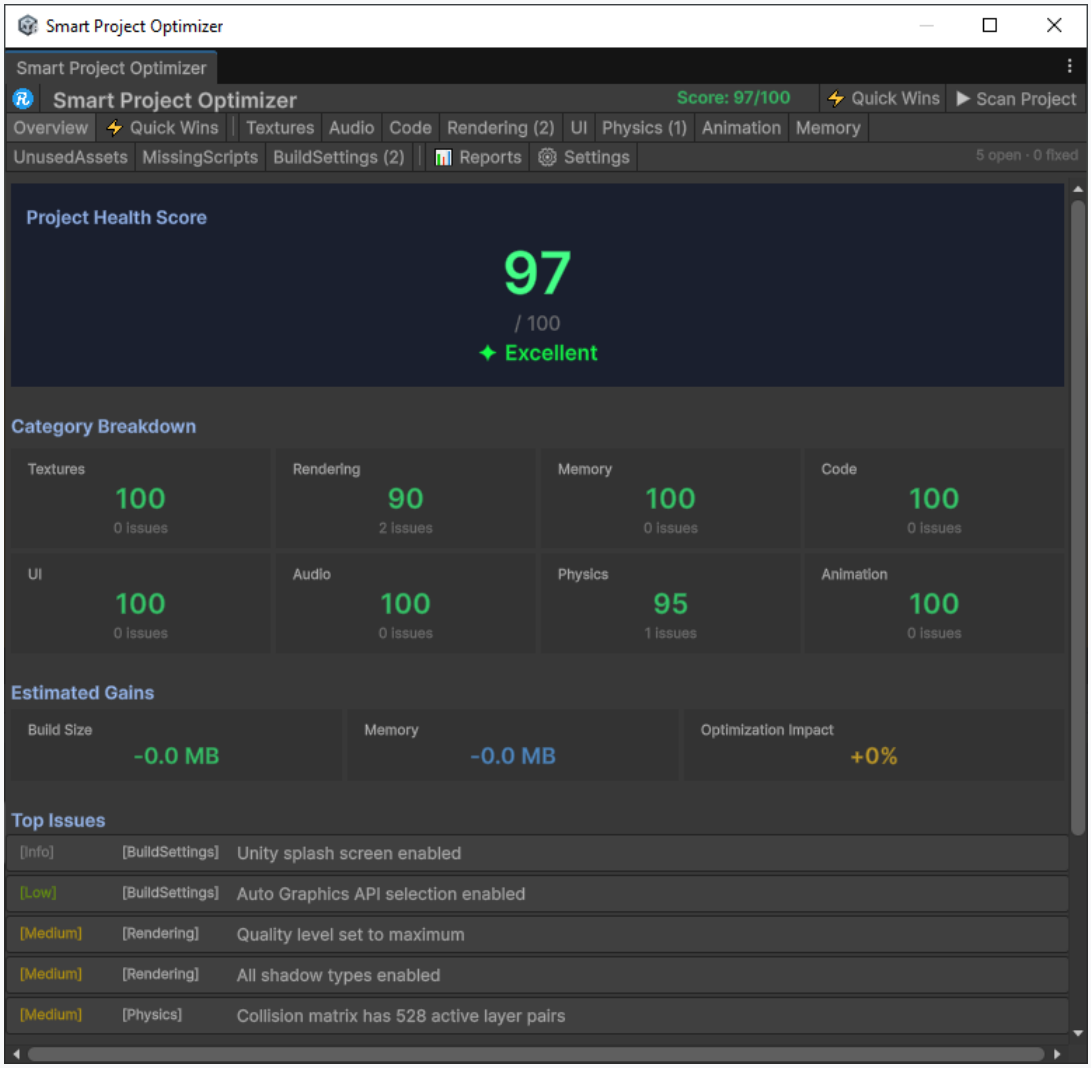
Smart Project Optimizer window — initial state before first scan

3. Quick Start

Follow these five steps for your first optimization pass:

1. Open the tool via Tools > Smart Project Optimizer.
2. Click Scan Project in the top-right toolbar. The scan typically takes 5–30 seconds depending on project size.
3. Review your Optimization Health Score on the Overview tab.
4. Click Quick Wins in the toolbar. Review the list of safe improvements and click Apply All.
5. Expand individual category tabs (Textures, Code, Rendering, etc.) to review and fix deeper issues.

> Tip: Run a scan before and after making changes, then save both as Benchmark Snapshots on the Reports tab to track your progress.

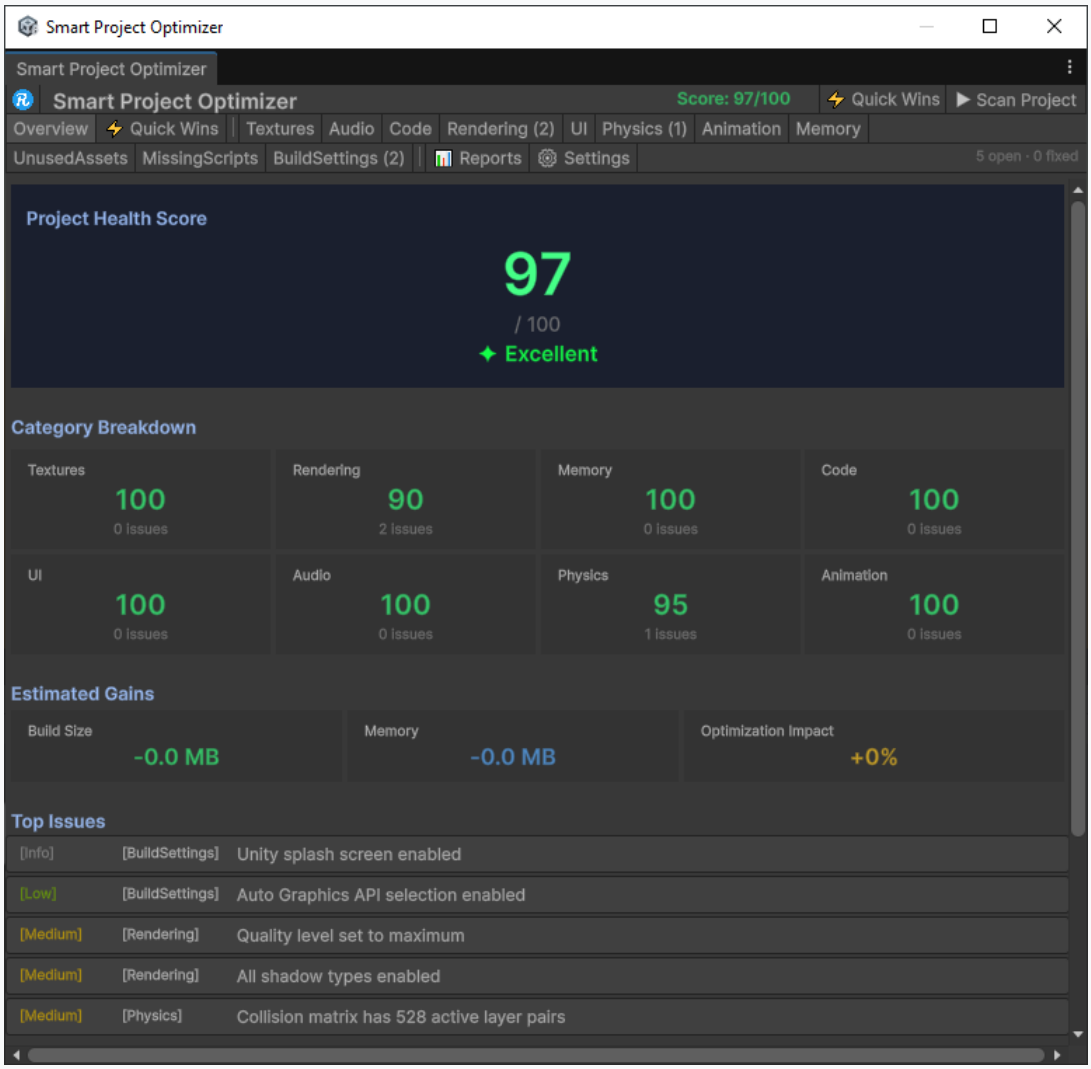


Overview tab showing Health Score 72/100 with category breakdown

Overview tab showing Health Score 72/100 with category breakdown

4. Overview Tab

The Overview tab is the main dashboard. It is the first thing you see after a scan.



Overview tab — full view with Health Score, category breakdown, and top issues

Overview tab — full view with Health Score, category breakdown, and top issues

4.1 Optimization Health Score

The Health Score converts the entire scan result into a single number from 0 to 100. A colour-coded status label appears below the score:

Range	Status Label	Meaning
95 – 100	**Excellent**	Project is in excellent health. Very few or no significant issues.

Range	Status Label	Meaning
85 – 94	**Great**	Strong project health. Minor improvements available.
70 – 84	**Good**	Good overall but some areas need attention.
55 – 69	**Needs Optimization**	Several medium-to-high issues identified. Optimization is recommended.
40 – 54	**Needs Attention**	Significant issues present. Performance impact is likely.
25 – 39	**Poor**	Multiple critical or high issues detected. Requires immediate review.
0 – 24	**Critical**	Serious optimization problems across multiple categories.

4.2 Category Breakdown

Eight score cards are shown below the main score — one per analysis category. Each card shows the category name, score out of 100, and issue count. Cards use the same colour coding as the main score.

Categories included: Textures, Audio, Code, Rendering, UI, Audio, Physics, Animation, Memory, Unused Assets, Missing Scripts, Build Settings, Reports, Settings

4.3 Estimated Gains

Three gain estimates are displayed based on the findings:

- Build Size Reduction — estimated MB saved if all auto-fixable asset issues are resolved.
- Memory Saving — estimated runtime memory reduction from texture and audio fixes.
- Optimization Impact — potential FPS improvement from rendering, physics, and code fixes.

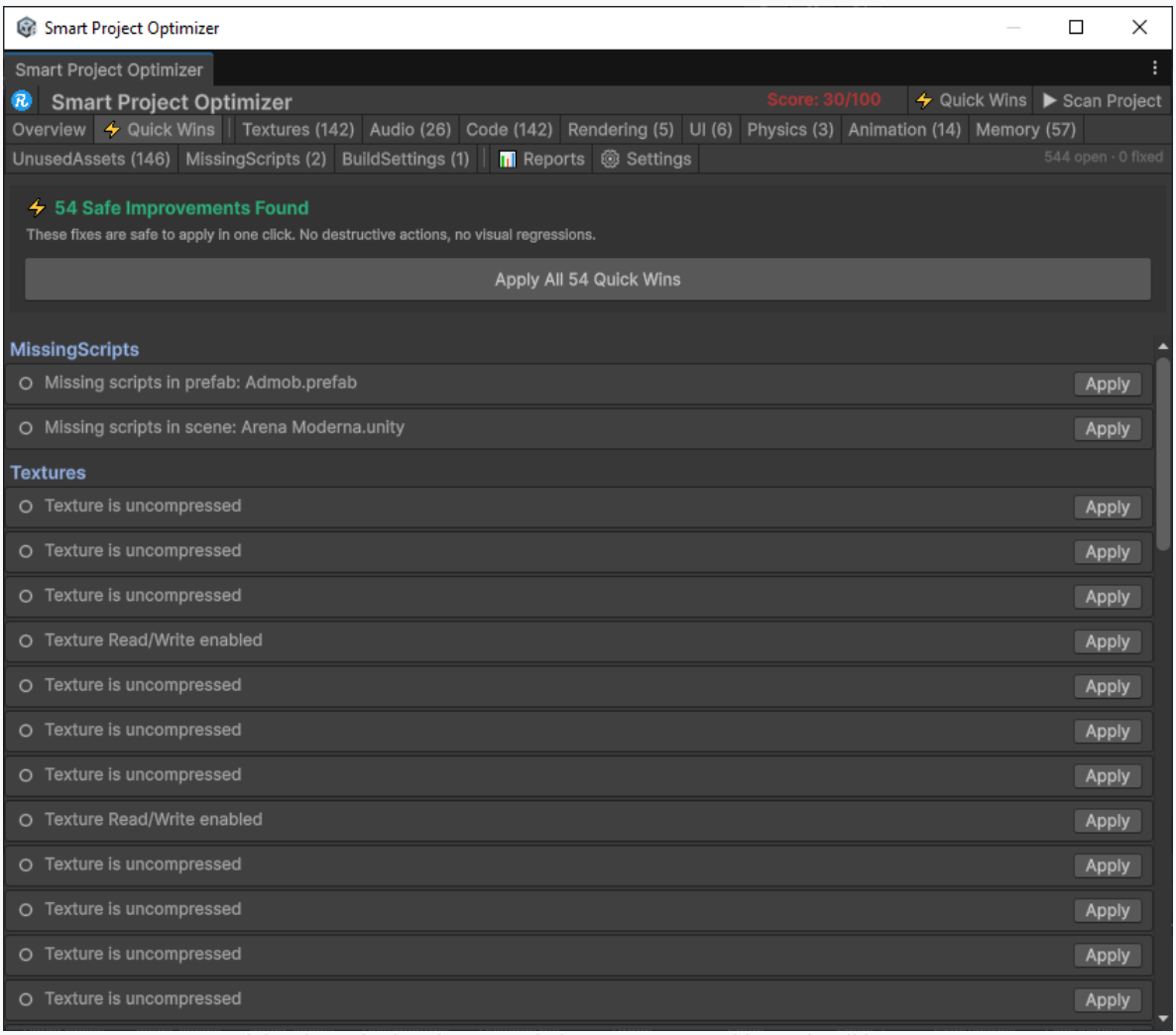
> Warning: Estimated gains are approximate and based on heuristics. Always measure real performance on target hardware.

4.4 Top Issues

The bottom of the Overview tab lists the highest-severity open findings across all categories, ordered by severity. Each row shows the severity badge, category, and title. Findings with an available auto-fix show a Fix button inline.

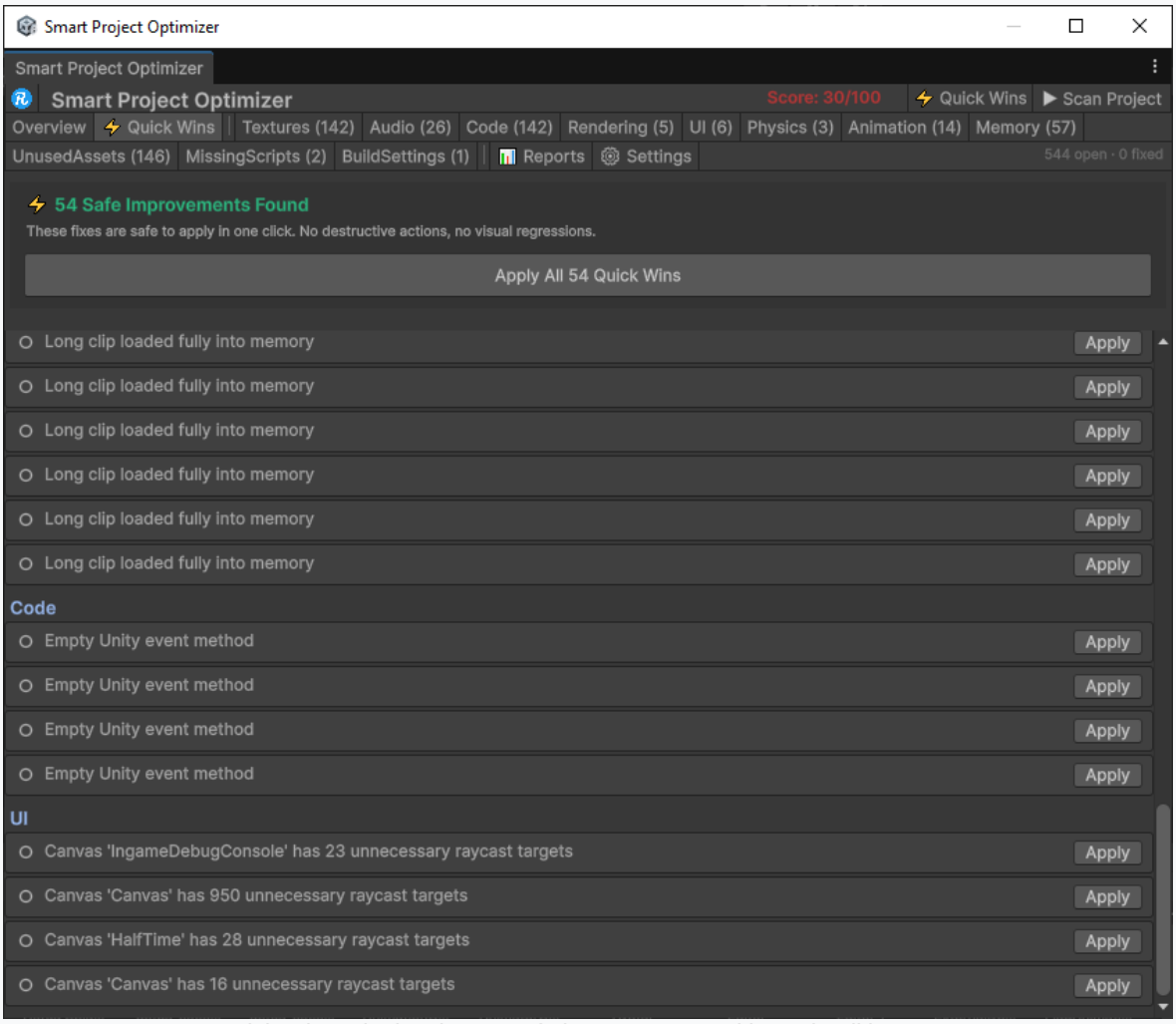
5. Quick Wins

Quick Wins are automatically identified safe improvements that can be applied in a single click without any risk of breaking your project.



Quick Wins tab showing 12 safe improvements with Apply All button

Quick Wins tab showing 54 safe improvements with Apply All button



Quick Wins tab showing 12 safe improvements with Apply All button

Quick Wins tab showing 54 safe improvements with Apply All button

6.1 What Qualifies as a Quick Win

- Disabling texture Read/Write where it is not needed.
- Removing empty Update, LateUpdate, or FixedUpdate methods.
- Enabling recommended texture compression on uncompressed textures.
- Setting long audio clips to Streaming or CompressedInMemory load type.
- Forcing mono on short SFX clips.
- Removing Missing Script components from prefabs and scenes.
- Disabling Development Build flag in Build Settings.

6.2 What Is Never a Quick Win

- Destructive asset deletion.
- Scene lighting changes that alter visual output.
- Code rewrites or structural refactoring.
- Any change that cannot be reversed or previewed.

> If Auto Backup Before Fix is enabled in Settings, a timestamped snapshot is saved before any Quick Win batch is applied.

6.3 Applying Quick Wins

6. Navigate to the Quick Wins tab.
7. Review the list of identified safe improvements.
8. Click Apply All to apply every item, or click Apply next to individual rows.
9. Applied items are marked with a green checkmark.

6. Category Analyzers

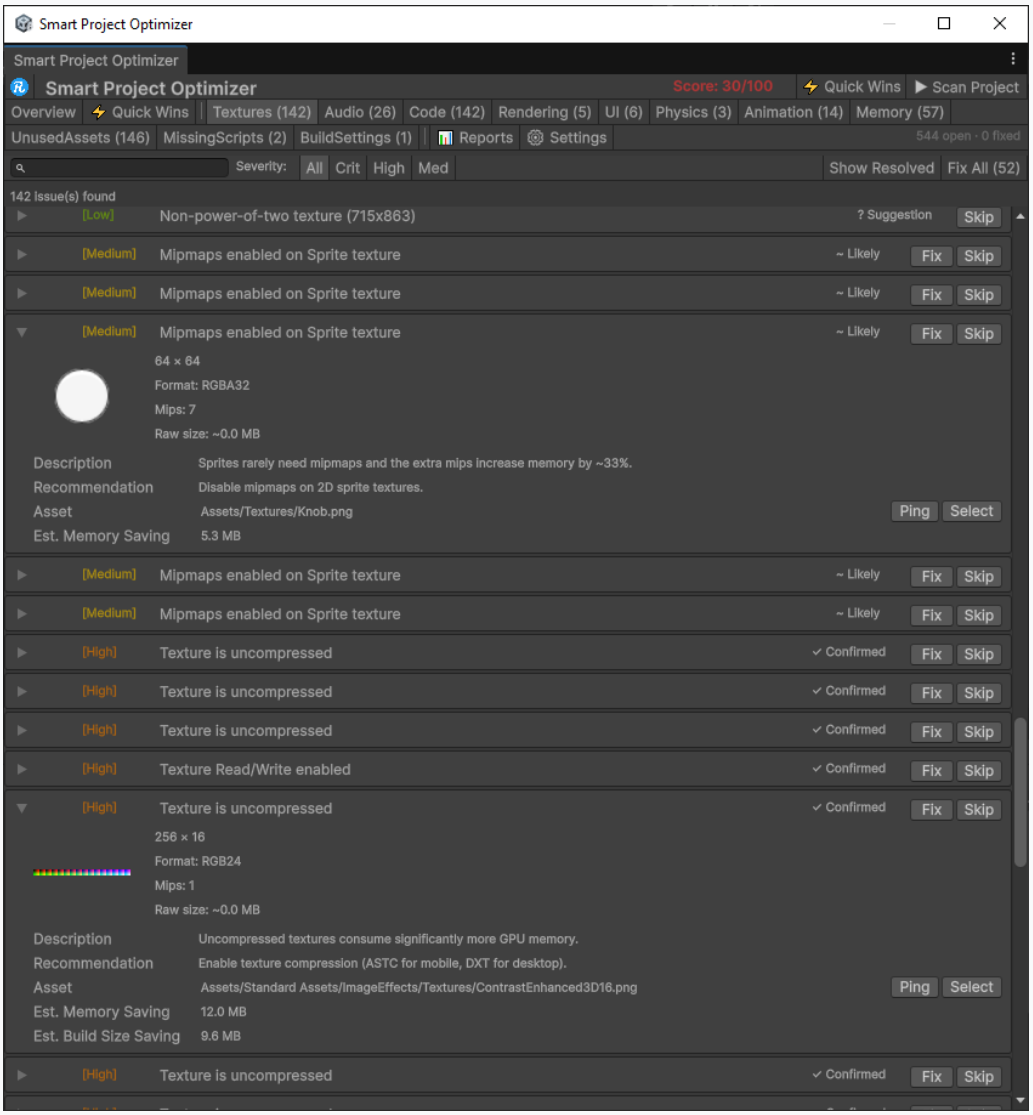
Each category tab shows findings specific to that system area. The toolbar at the top of each tab lets you search, filter by severity, show/hide resolved items, and apply all available auto-fixes at once.

7.1 Textures

The Texture Analyzer scans every Texture2D asset in your project for the following issues:

- Read/Write enabled — doubles GPU memory usage. Auto-fixable, Quick Win.
- Mipmaps enabled on Sprite textures — adds ~33% memory overhead unnecessarily.
- Max size above 2048 — large textures cause memory pressure on mobile.
- Uncompressed textures — uncompressed formats are 4–8x larger than compressed equivalents.
- Non-power-of-two (NPOT) dimensions — may block compression on some platforms.

When you expand a texture finding, a live preview of the texture is shown alongside its dimensions, format, mip count, and estimated raw memory size. A Ping and Select button let you jump to the asset instantly.



Texture finding expanded — showing 80x80 thumbnail, dimensions, format, Ping/Select buttons

Texture finding expanded — showing 80x80 thumbnail, dimensions, format, Ping/Select buttons

7.2 Audio

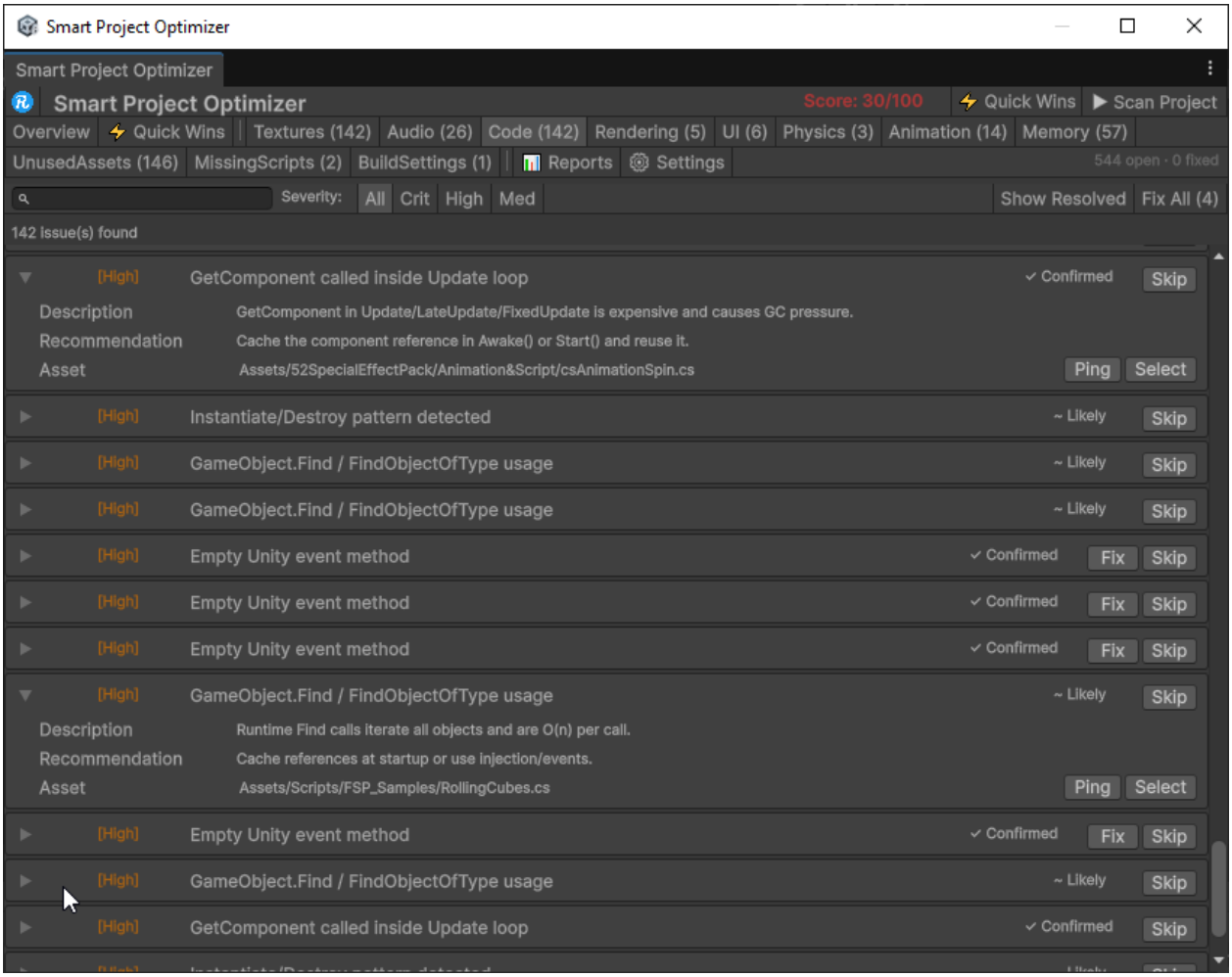
The Audio Analyzer scans every AudioClip for import setting problems:

- Long clips set to DecompressOnLoad — fully decompressed at load time wastes memory.
- Stereo clips suitable for mono — halving channels halves memory for SFX.
- Sample rate above 44100 Hz — unnecessary quality for game audio.
- Preload Audio Data on long clips — increases startup time and memory.

7.3 Code

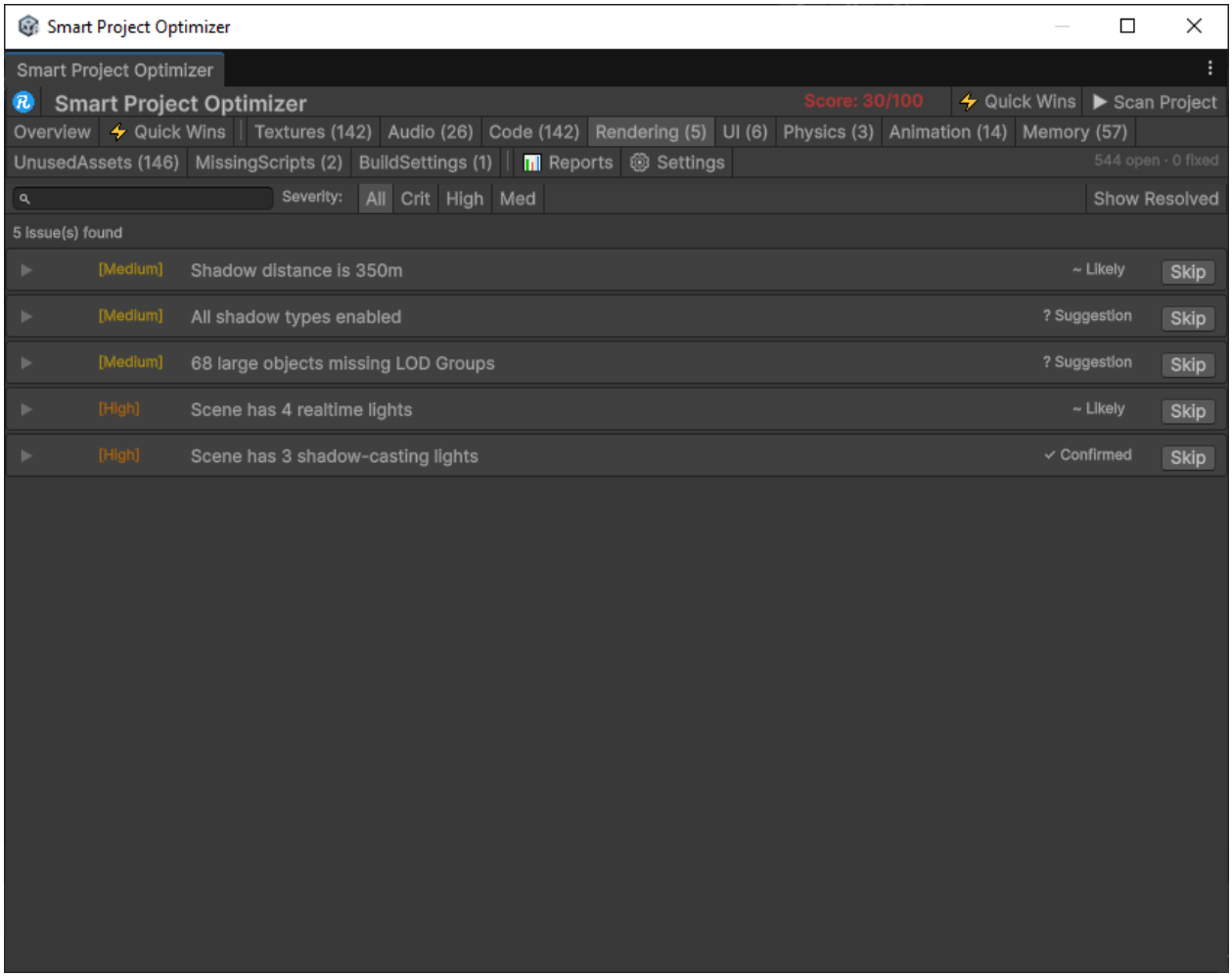
The Code Analyzer parses all C# scripts in your project and identifies common Unity performance anti-patterns. Every finding is labelled with a confidence level:

Pattern	Confidence	Why It Matters
GetComponent in Update	Confirmed Risk	Expensive call in the hot path. Cache in Awake/Start instead.
GameObject.Find at runtime	Likely Risk	O(n) search over all objects. Cache references at startup.
Empty Update/LateUpdate	Confirmed Risk	Unity still calls these every frame even when empty.
Debug.Log in production	Confirmed Risk	Allocates strings and causes GC pressure.
String-based Animator params	Likely Risk	Use Animator.StringToHash() cached IDs instead.
String-based Shader properties	Suggestion	Use Shader.PropertyToID() for performance.
new WaitForSeconds in coroutine	Likely Risk	Creates heap allocations each coroutine restart.
LINQ usage	Educational Tip	Avoid LINQ in Update or other hot paths.
Instantiate/Destroy pattern	Likely Risk	Use object pooling to avoid GC spikes.
Runtime AddComponent	Suggestion	Pre-attach components and enable/disable instead.



7.4 Rendering

- Too many realtime lights in a scene — each adds a full per-frame lighting pass.
- Multiple shadow-casting lights — each doubles draw calls for shadow receivers.
- Large objects missing LOD Groups — renders at full detail regardless of camera distance.
- High pixel light count in Quality Settings.
- Shadow distance above 50m — expensive on mobile.
- All shadow types enabled in Quality Settings.



7.5 UI

- Large single Canvas with many graphics — any change triggers a full rebuild.
- Excessive non-interactive raycast targets — wastes input system cycles every frame.
- Nested LayoutGroups — cascades layout recalculations during UI changes.
- Deep Mask chains — compounds stencil buffer operations.
- Invisible full-screen panels hidden via alpha=0 — still causes GPU overdraw.

> Tip: Split your UI Canvas into a static Canvas (never changes) and a dynamic Canvas (changes frequently). This dramatically reduces rebuild cost.

7.6 Physics

- Fixed Timestep below 0.016s — runs physics more times per frame.
- Dense collision matrix — too many active layer pairs increases solver work.
- MeshColliders on moving Rigidbodies — extremely expensive for dynamic objects.
- Large Rigidbody count in a scene.
- Heavy logic inside FixedUpdate — amplified cost when frame rate drops.

7.7 Animation

- Animator Controllers with more than 50 states.
- Controllers with more than 5 layers.
- FBX mesh optimization disabled — Unity can reduce vertex count on import.
- Animation clips not using Optimal compression.
- More than 20 simultaneously active Animators in one scene.

7.8 Memory

- Oversized textures (above ~8MB uncompressed).
- Multiple textures with identical dimensions — potential duplicate or atlas candidates.
- Boxing-prone code patterns.
- String concatenation in potentially hot code paths.
- Incremental GC not enabled in Player Settings.

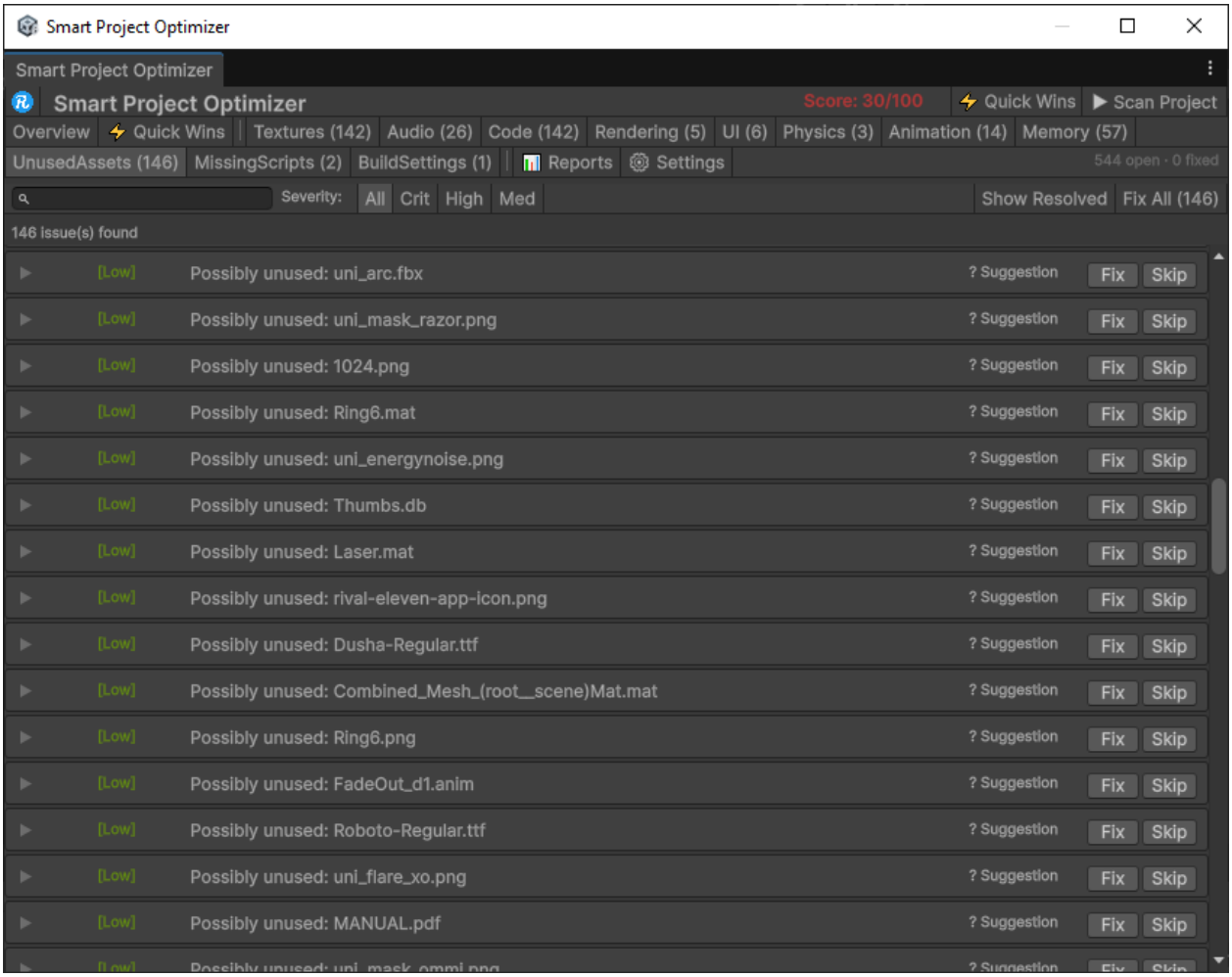
7.9 Unused Assets

The Unused Asset Analyzer builds a dependency graph from all build-settings scenes, every scene in the project, all prefabs, ScriptableObjects, the active render pipeline asset, Resources folders, and sprite atlases. Assets not reachable through any tracked path are flagged as possibly unused.

> Warning: Unused asset detection can never be 100% certain. Assets loaded dynamically via string paths (Resources.Load, Addressables, AssetBundles) may not appear in the static dependency graph. Always review before deleting — use Quarantine instead of Delete.

The following are never flagged, regardless of dependency status:

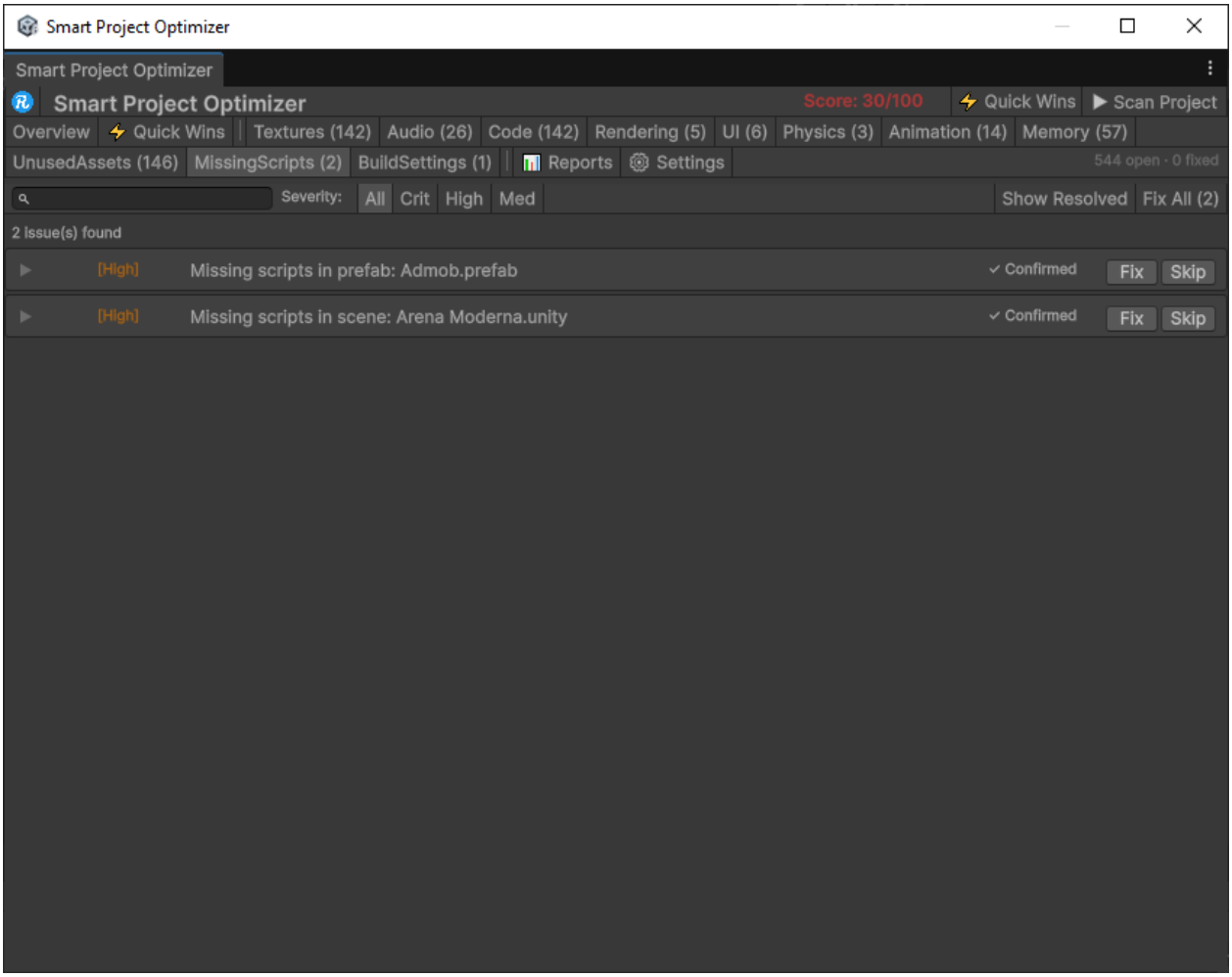
- .asset files (ScriptableObjects, render pipeline assets, volume profiles).
- URP/HDRP assets, renderer assets, volume profiles.
- Resources folder contents.
- Addressables, StreamingAssets, Plugins, Packages.
- InputActions, Samples, Gizmos, Editor Default Resources.
- All code and shader files.



7.10 Missing Scripts

The Missing Script Detector scans all prefabs and build-settings scenes for null MonoBehaviour references (broken script components). These appear as a pink/empty script slot in the Inspector.

For each affected asset it shows the number of broken objects and their names. The Fix button calls GameObjectUtility.RemoveMonoBehavioursWithMissingScript, which is Unity's official API for this operation. Undo support is provided.



7.11 Build Settings

- Scripting backend set to Mono instead of IL2CPP on mobile — IL2CPP is faster and produces smaller builds.
- Managed stripping disabled — includes unused code in the build.
- Development Build enabled — adds profiler overhead to a release build.
- Auto Graphics API selection — may include unnecessary legacy APIs.
- Unity splash screen enabled (informational).

7. Working with Findings

8.1 Finding Row Layout

Every finding row shows:

- Severity badge — Critical, High, Medium, Low, or Info — colour coded.
- Category badge.
- Title of the finding.
- Confidence label — Confirmed Risk, Likely Risk, Suggestion, or Educational Tip.
- Status badge when Fixed (green) or Skipped (yellow with strikethrough).
- Fix button — only shown when an automatic fix is available.
- Skip button — dismisses the finding from the score without deleting it.

8.2 Expanding a Finding

Click the arrow to the left of any row to expand the full detail panel:

- Texture preview (80x80 thumbnail) for texture findings.
- Full description of the problem.
- Recommendation explaining how to fix it.
- Asset path with Ping and Select buttons.
- Estimated memory saving and build size saving where applicable.

8.3 Skipping a Finding

Clicking Skip crosses out the finding title with a strikethrough and marks it with the yellow badge. Skipped findings are:

- Excluded from the Health Score calculation.
- Excluded from Estimated Gains totals.
- Still visible in the list so you can review your decisions.

To restore a skipped finding, click Undo on the skipped row. It returns to open status and is included in the score again.

8.4 Toolbar Filters

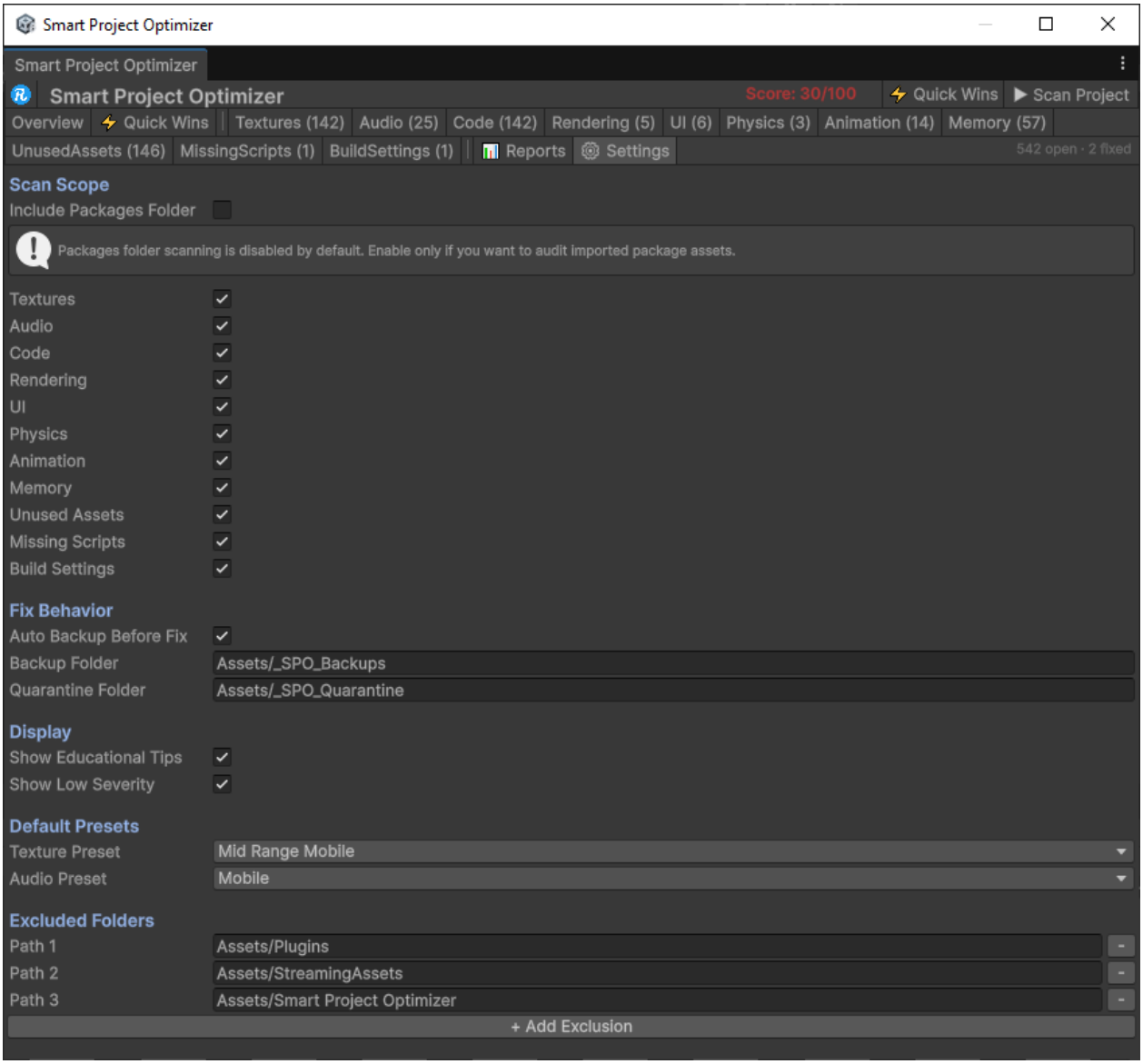
- Search field — filter by finding title or asset path.
- All / Crit / High / Med — filter by severity level.
- Show Resolved — toggle to show or hide already-fixed items.
- Fix All (N) — applies all available auto-fixes in the category in one batch.

8. Feature Matrix

Feature	Incl.	Risk	Description
Optimization Health Score	Yes	Safe	Converts scan results into a score out of 100 with category breakdowns and estimated gains.
Quick Wins Mode	Yes	Safe	Surfaces only safe one-click improvements. Never includes destructive changes.
Texture Optimizer	Yes	Low	Batch-applies compression, max size, mipmaps, and Read/Write settings via Unity reimport.
Audio Optimizer	Yes	Low	Reimports audio by category using load type, compression, mono, and sample rate presets.
Code Analyzer	Yes	Low	Parses scripts for 10 anti-patterns with confidence labelling.
Rendering Diagnostics	Yes	Medium	Warns about lights, shadows, LOD, and quality settings. Partial auto-fix.
UI Analyzer	Yes	Medium	Detects Canvas rebuild risks, raycast overhead, and layout group issues.
Physics Audit	Yes	Medium	Flags mesh colliders, Rigidbody counts, collision matrix, and FixedUpdate weight.
Animation Audit	Yes	Low	Reviews controller complexity, FBX settings, and active Animator counts.
Memory Assistant	Yes	Medium	Flags oversized textures, duplicates, and GC-prone code. Recommends Incremental GC.
Unused Asset Cleaner	Yes	Medium	Dependency-graph detection. Never hard-deletes. Quarantine and restore workflow.
Missing Script Detector	Yes	Safe	One-click removal of null MonoBehaviour references. Undo supported.
Build Settings Optimizer	Yes	Low	IL2CPP, stripping, Development Build, and Graphics API checks.
Benchmark Snapshots	Yes	Safe	Save before/after snapshots with notes. Visual score comparison in history table.
Report Export	Yes	Safe	Export findings as Markdown, CSV, or HTML to Assets/_SPO_Reports/.

9. Settings

Access Settings via the Settings tab in the tab bar. All settings are stored in memory for the current Editor session.



10.1 Scan Scope

Toggle individual analyzers on or off. Disabling a category speeds up the scan and removes it from the Health Score calculation. Use this when you want a focused scan of specific areas.

Include Packages Folder — disabled by default. Enable only to audit third-party package assets. This can add significant noise from assets you do not own.

10.2 Fix Behavior

- Auto Backup Before Fix — creates a timestamped folder snapshot before any fix is applied. Strongly recommended to keep this enabled.
- Backup Folder — where backup snapshots are saved. Default: Assets/_SPO_Backups.
- Quarantine Folder — where assets are moved when using Quarantine mode. Default: Assets/_SPO_Quarantine.

10.3 Display

- Show Educational Tips — include low-confidence informational findings. Good for learning but noisy for production audits.
- Show Low Severity — toggle Low findings in the issue lists. Hiding them focuses the view on High and Medium issues.

10.4 Default Presets

These presets are used by the Texture and Audio optimizers when you click a batch apply action. They can be changed per-session.

Texture Presets:

- Low-End Mobile — 512px max, compressed, no mipmaps, Read/Write off.
- Mid-Range Mobile — 1024px max, compressed, mipmaps enabled (default).
- High-End — 2048px max, compressed.
- UI Pack — Sprite type, 2048px, no mipmaps.
- Sprite Pack — Sprite type, 1024px, no mipmaps.

Audio Presets:

- Mobile — Vorbis, Streaming for music/ambience, CompressedInMemory + mono for SFX (default).
- Desktop — Moderate quality Vorbis.
- High Quality — PCM uncompressed, DecompressOnLoad.

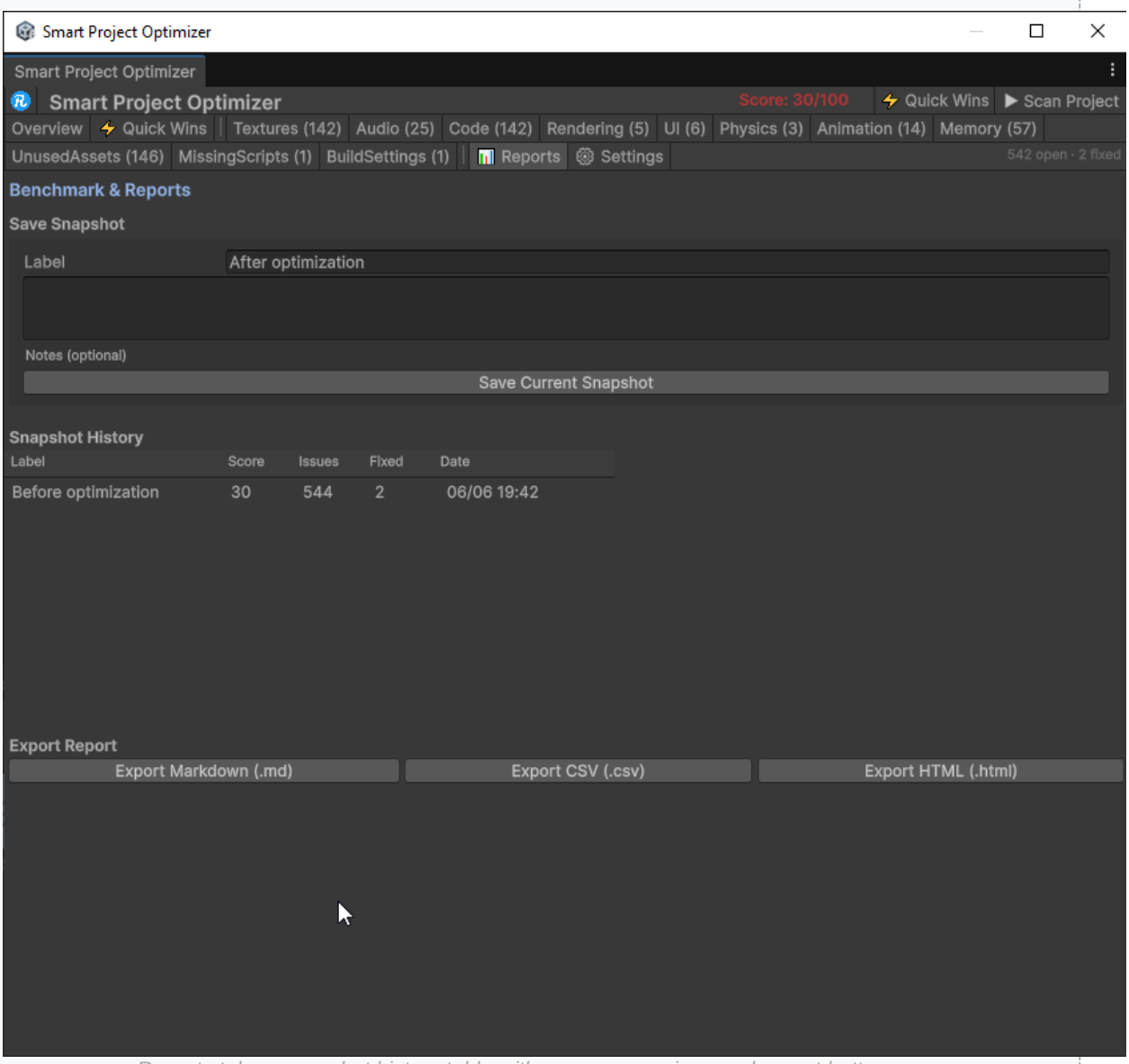
10.5 Excluded Folders

Add any project-relative paths here to exclude them from all scans. The following are excluded by default:

- Assets/Plugins
- Assets/StreamingAssets
- Assets/Smart Project Optimizer

> Tip: Add your third-party asset folders here to keep scan results focused on your own code and content.

10. Reports & Benchmarking



Reports tab — snapshot history table with score comparison and export buttons

Reports tab — snapshot history table with score comparison and export buttons

11.1 Saving Snapshots

A benchmark snapshot saves the current scan state — overall score, issue count, fixed count, and estimated gains — with a label and optional notes.

Best practice: save a snapshot labelled Before optimization before making any changes, and a second snapshot labelled After optimization once fixes are applied. The history table shows both side by side with colour-coded score deltas.

11.2 Export Formats

Format	Output Location	Best Use
.md Markdown	`Assets/_SPO_Reports/`	Team documentation, Notion/Confluence pages, README updates.
.csv Spreadsheet	`Assets/_SPO_Reports/`	Issue tracking in spreadsheets or project management tools.
.html Web report	`Assets/_SPO_Reports/`	Client handoffs, QA reviews, browser-readable standalone reports.

11. Frequently Asked Questions

Will this tool automatically fix everything?

No. Smart Project Optimizer is a guided optimization suite, not a magic button. Auto-fixes are provided only for clearly safe, reversible changes. Higher-risk improvements are surfaced as suggestions with explanations so you can make an informed decision.

Can I run a scan on just one category?

Yes. In the Settings tab, disable all categories except the one you want, then run a scan. The Health Score will only reflect enabled categories.

The scan flagged a file I know is used. What do I do?

Click Skip on that finding. It will be crossed out and excluded from the score. The most common cause is an asset loaded dynamically at runtime via Resources.Load, Addressables, or a custom asset bundle system. You can also add the folder to the Excluded Folders list in Settings to prevent it appearing in future scans.

Are changes reversible?

All import-setting changes (textures, audio) are reversible through Unity's own reimport workflow. With Auto Backup Before Fix enabled, a snapshot is created before any batch operation. For unused asset quarantine, assets are moved (not deleted) and can be restored from the quarantine folder.

Does this work with URP and HDRP?

Yes. The tool is render-pipeline agnostic. URP and HDRP-specific assets (renderer assets, volume profiles, pipeline assets) are automatically whitelisted and will never be flagged as unused.

Does it slow down the Unity Editor?

The scan runs on demand only — it does not add any runtime overhead or Editor update loops. The scan duration depends on project size, typically 5–30 seconds. The window uses standard IMGUI and has no persistent performance cost.

12. Troubleshooting

The window shows no results after scanning

- Ensure at least one scene is added to Build Settings (File > Build Settings).
- Check that the scan categories are enabled in Settings.
- Check the Unity Console for any SPO-prefixed error messages.

A fix was applied but nothing changed in the project

- Some import-setting changes require Unity to reimport the asset. This happens automatically but may take a moment in large projects.
- If the issue reappears on re-scan, the importer may have reverted the setting. Check if a .meta file override is in version control.

Compilation errors after import

- Ensure your project is on Unity 2020.3 LTS or newer.
- If you have a custom Assembly Definition that excludes Editor assemblies, check that SmartProjectOptimizer.Editor.asmdef is being included.
- Try Assets > Reimport All if errors persist after a clean import

13. Support & Contact

For support, bug reports, feature requests, or general questions about Smart Project Optimizer, please reach out through any of the following channels:

Website	rigonix3d.com
Unity Asset Store	Search for Smart Project Optimizer by Rigonix3D
Email Support	rigonix3d@gmail.com
Documentation	This document — keep a copy for offline reference

Thank you for using Smart Project Optimizer.

We hope it saves you hours of manual optimization work.

— Rigonix3D