Page contents

- Put each effect in the context that owns it
- Use the object API before a source patch
- Validate that a patch actually applied
- Version and compatibility boundaries
- Sources and limits

Offline copy of Scoring contexts, hooks and Lovely patches .

Balatro: scoring contexts, hooks and Lovely patches

Documentation status: In progress. **Sources checked:** 2026-10-12. **In-game validation:** not recorded. Read the Joker quick start and object-model chapter first. Steamodded's calculate reference and Lovely's patch documentation describe two different ways to influence game behavior.

Put each effect in the context that owns it

`calculate(self, card, context)` receives calls at several phases. The Steamodded reference describes `context.joker_main` for normal Joker scoring, `context.pre_joker` for an edition's pre-Joker effect, and `context.other_joker` when one object responds to another Joker. The `card` instance can store changing run values in `card.ability.extra`; the `self` definition is shared. For a scaling Joker, initialize numeric values under `config.extra`, update only on the intended trigger, then return the score contribution in the correct scoring phase. Updating a shared prototype would leak state across instances.

Build a context table before writing the callback: trigger event, required `context` flag, available cards/areas, state to change, returned effect, and whether the game may call it more than once. The +4 Mult quick start uses `context.joker_main` deliberately. A Joker that also reacts to selling or discarding needs another explicit branch and a test

that the score effect does not fire there. Inspect the full calculate reference for the current flags rather than guessing from a field name in an old mod.

Keep state mutation separate from score reporting when possible. Test a single Joker, two copies, Blueprint-compatible copies, retriggers, debuffs, and save/reload. The Mod Object reference lists opt-in features such as `retrigger_joker`, `post_trigger`, `quantum_enhancements` and extra card areas; enabling one changes which contexts can appear. Record the Steamodded release and enabled optional features with each result.

Use the object API before a source patch

Steamodded exposes registration and lifecycle callbacks on its object pages. A Joker calculation, consumable use, blind calculation or voucher redemption should use that object's API when it covers the desired behavior. A Lovely patch is for a specific source location that the API does not expose. Lovely's README says its patch format is still unstable and documents pattern, regex, copy and module patches. A mod with `lovely.toml` or a `lovely/*.toml` file must remain in its own folder or ZIP inside the mod directory.

For a narrow source patch, identify the game file and exact target pattern from the installed Balatro build. Use a unique nearby sequence and constrain the expected match count; the Lovely example uses `times = 1` for a single match. Prefer a pattern patch for a small insertion, and use regex only when pattern matching cannot express the target. Place new Lua code under your mod folder and reference it with paths relative to that folder where the patch format requires. Do not distribute a modified game executable or copied full game Lua file to implement a short hook.

Validate that a patch actually applied

Lovely writes logs and patched-source dumps under `MOD_DIR/lovely/log` and `MOD_DIR/lovely/dump` according to its patch debugging section. Inspect the log for match failures, then inspect the dumped target around the injection point. Confirm the inserted code runs once at the intended stage and has the variables it expects. Test with only Lovely, Steamodded and this mod, then with mods known to patch the same target. A successful TOML parse does not prove a pattern still matches after a Balatro update.

Lovely's `manifest.priority` controls ordering among loaded patch files, while Steamodded metadata dependencies control whether a Lua mod loads; these are distinct controls. Changing priority does not fix an absent dependency or a stale pattern. Write down the game build, Lovely version, Steamodded version and patch target in the release notes. Retest every source patch after a game or loader update, and inspect dumped output before calling a release compatible.

Version and compatibility boundaries

Steamodded's release notes list new APIs by release, including 26.829.0 and 26.1002.0 on the documentation index at this check. A method marked “added in” a release is unavailable on older versions. Keep a version floor in metadata only after confirming all required methods, and use a clean profile for that floor. Optional integrations should check the other mod's load state and keep their objects behind object-level `dependencies` where appropriate. Preserve a test matrix for new run, mid-run reload, another mod changing the same object, and another Lovely patch at the same target.

Sources and limits

- Steamodded calculate functions, game-object API, Mod Object and release notes — SMODS Team references; checked 2026-10-12.
- Lovely injector README — maintainer documentation for patch types, loading and debugging; checked 2026-10-12.
- This is original ModDocs prose. No Balatro source or third-party patch payload is mirrored. A full compiled patch example and cross-version test remain outstanding.