AMBER

612 COMMITS → IN 24 DAYS

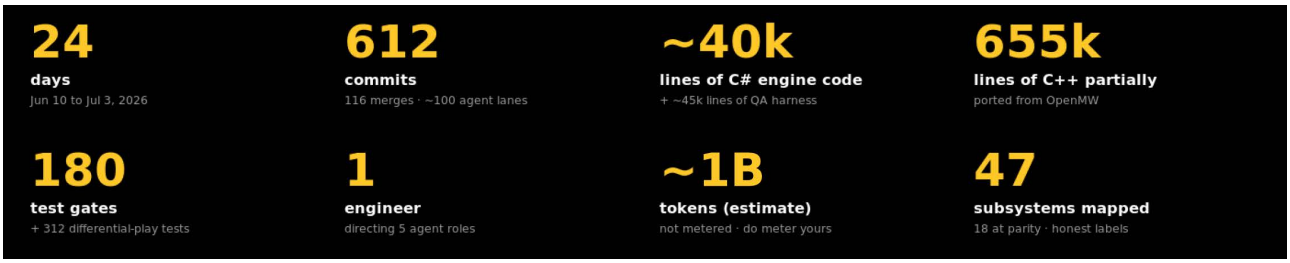
WHAT AN AGENT-DRIVEN ENGINE PORT TAUGHT US ABOUT AI IN PRODUCTION

An Amber R&D experiment in agent-driven engine porting · June 2026



AT A GLANCE

TL;DR. In just 24 hours, one engineer, one orchestrator and four AI agent roles ported OpenMW's data loading, rendering and animation systems to Godot. Our experiment used AI coding agents to rebuild the opening sequence of The Elder Scrolls III: Morrowind on the Godot engine, porting from the open-source OpenMW codebase. One senior engineer directed the work; agents wrote the code. The objective was a faithful replication of the character-generation intro, and we got close: the sequence plays end to end, wake-up below deck to Seyda Neen, checked against the real OpenMW engine by a differential test harness the agents also built. Close is not 100%, though. You can see it on the cover: the port's interior runs warmer than OpenMW's, a known, documented lighting gap. The UI still has parity gaps too, and we will show you exactly where. It is not the whole game and it is not a product. But the numbers changed how we think about production pipelines.



WHY WE DID THIS

Amber is a game development company. Ports and platform conversions are part of what we do every day, alongside co-development, art production and full game development. One of our core values is Kaizen: we are permanently looking for ways to improve our development speed and quality, and right now that search cannot ignore AI. So we ran an experiment. The question we wanted answered: can agentic AI compress the early, expensive part of an engine migration without compromising the quality and feel of the original?

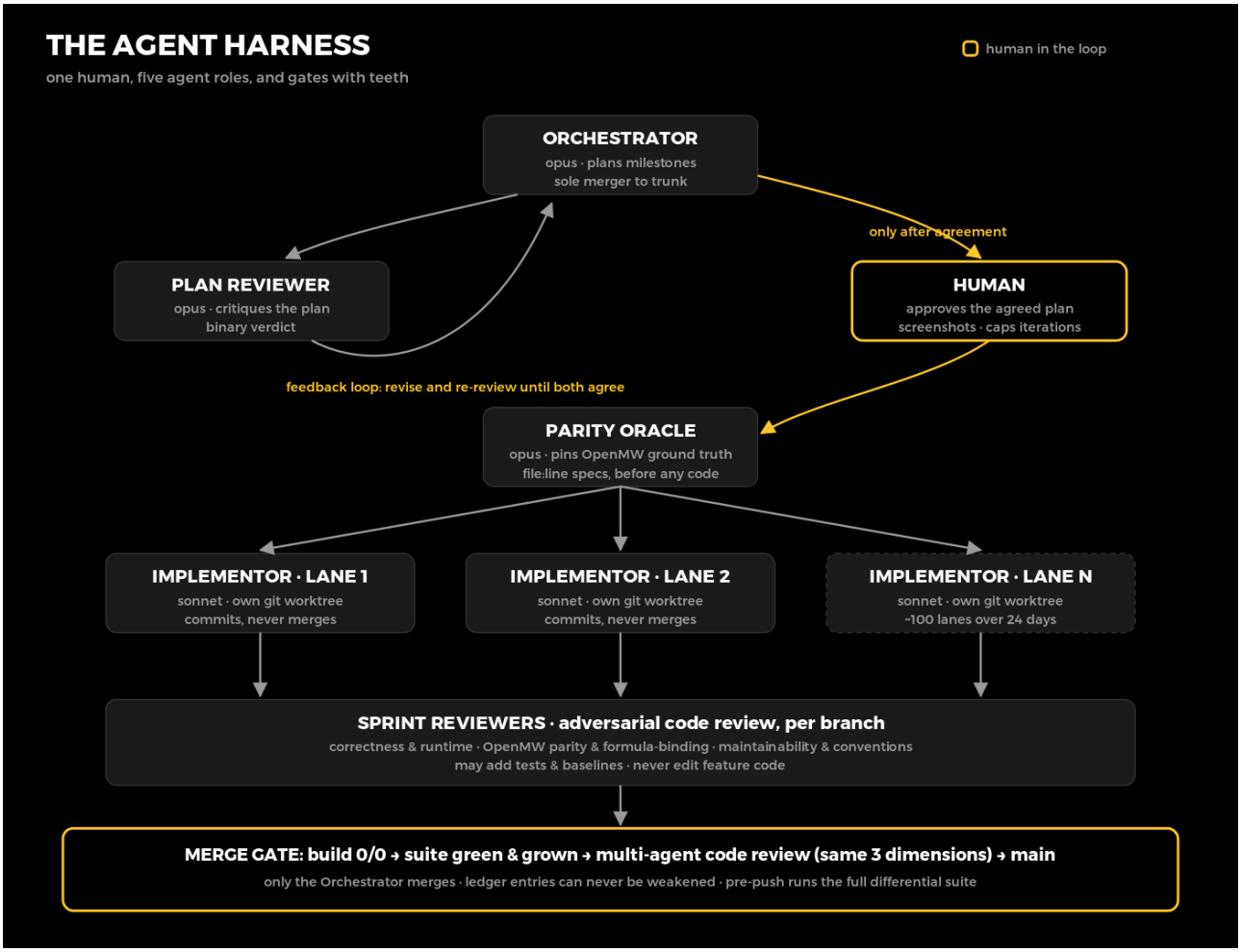
We picked Morrowind's opening because it is a hard, well-understood target with a strong emotional memory attached to it. People remember waking up on that prison ship. The source was OpenMW, the community-built open-source reimplementation of the Morrowind engine: 655,000 lines of C++ built over two decades. This is a technology demonstration, not a product, and the OpenMW team's work is the reason an experiment like this is possible at all.

The target engine was Godot. The scope was deliberately narrow: the character-generation intro, from waking up below deck to stepping out into Seyda Neen. One continuous, scripted, testable slice of game. Definition of done: the port behaves like OpenMW, and a test suite, not a human's impression, says so. By that bar we are close and not done: most of the suite is green, and the ledger keeps an honest list of what is not.

THE HARNESS:
THE PART THAT ACTUALLY MATTERS

The agents' code is not the interesting part. The system around them is, and we did not have it on day one.

We started with a single agent and a chat window. Over the four weeks the harness grew into something closer to a small engineering organization, and the two setups were not comparable in output quality or speed. We did not measure the gap, but we would not go back. It was not free either. Every agent we spawn starts from zero and has to build its own understanding of the job before it can do anything, so a five-role pipeline burns tokens on context that a lone agent never pays for. We paid the token bill and considered it cheap, because the alternative was reviewing plausible-looking wrong code all day.



The five roles.

- **Orchestrator** (the strongest model tier): plans milestones, dispatches every other agent, and is the only one allowed to merge to trunk.
- **Plan Reviewer:** critiques each plan and sends it back. The two go back and forth until they actually agree, and only a plan that survives that loop reaches the human for approval.

- **Parity Oracle:** pins ground truth before anything gets built. It reads the OpenMW C++ and writes per-system specs with file-and-line citations, so "correct" is never guessed.
- **Sprint Implementors** (a faster model tier): each builds one lane in an isolated git worktree. They commit but never merge.
- **Sprint Reviewers:** adversarially review every branch for correctness and runtime behaviour, OpenMW parity and formula-binding, and maintainability and conventions. They may add tests and baselines, but may not touch feature code.

How a milestone actually runs. The Orchestrator carves the milestone into 2 to 4 lanes that own disjoint files, so parallel agents never fight over a hot file. Each lane gets a brief in two parts. The first part is always the same, byte for byte: the role, the repo rules, and the standing conventions. Because it never changes, the model provider's prompt caching serves it almost for free. The second part changes per lane, naming the goal, the files that lane owns, the cached artifacts to reuse, the non-goals, and the definition of done. Implementors must capture a real baseline on their fresh worktree before changing anything, ship a check that proves their own behaviour, run the gates, and commit before their turn ends. A lane that ends with "waiting for..." is a failed lane.

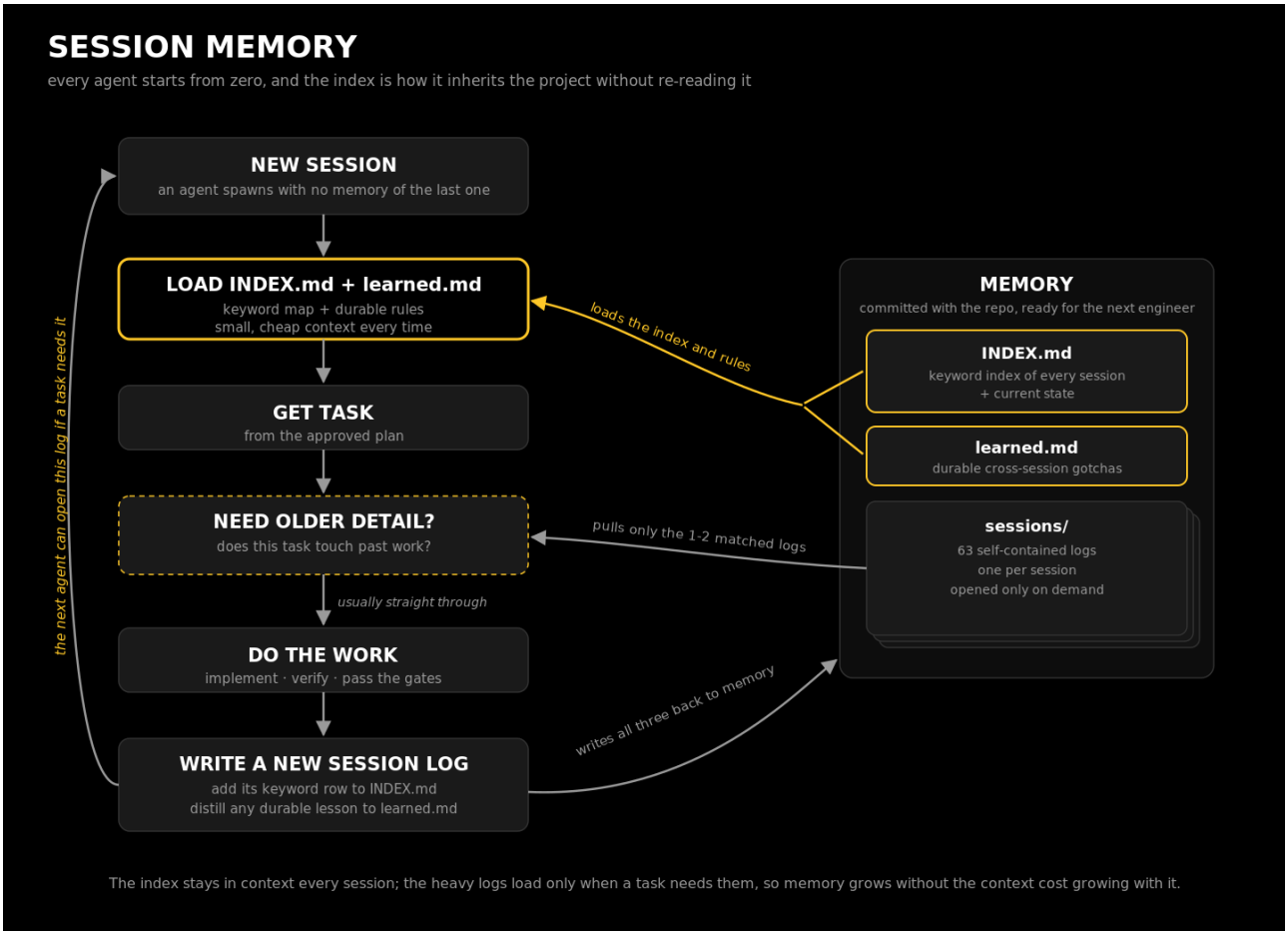
The process is in the repo, not in anyone's head. Every rule above lives in committed files: the agent definitions, the plan board, and the brief template. Two git hooks enforce the rest. The pre-commit hook blocks stray data files and requires a zero-warning build. The pre-push hook runs the full differential suite locally.

The last piece is a machine-readable feature ledger, and its rules have teeth. Entries can never be deleted and descriptions can never be edited. A test can never be weakened just to make the ledger green. Only the Orchestrator can flip an entry to pass. A fresh session, human or agent, reads the repo and reproduces the same loop with no extra briefing.

No hand-coded values. One rule outranks the rest. Every constant, formula, timing and layout either loads from the game data or is ported from OpenMW source with a citation. We did not write that rule out of principle. We wrote it because the models forced us to. AI agents have a clear tendency to take whatever shortcut marks the task done fastest, and in the first few days that meant hardcoding: invented UI colours, guessed physics constants, spawn positions nudged until things looked right. Each one read as progress and each one was a small lie about how much of the game actually worked. The rule set that stopped it had to be explicit, written down, and enforced by reviewers, because an agent that wants to close its ticket will not respect an intention that is not written down and enforced.

Memory that survives between sessions. A spawned agent remembers nothing, so the counterweight to all that starting-from-zero is a written memory the agents share: anything worth keeping has to live in files the next one will open. Two of those load at the start of every session: learned.md, the durable rules collected one incident at a time, and INDEX.md, a keyword map of the sixty-three session logs we ended up with. The index is the trick that keeps this affordable. Rather than drag every past session into context, an agent reads the small index, finds the last time someone fought the collision solver or the font theming, and opens that one full log only if the task needs it. At the end of a session the work becomes a new log, its keywords go into the index, and any lasting lesson is distilled into learned.md for whoever comes next. Because the whole memory folder is committed with the repo, a fresh engineer can pick it up the same way.

This is also where the no-hardcode rule and the gate conventions lived between sessions, and it is how a Sprint Implementor opening a lane on day nineteen inherited what an agent on day three had already worked out instead of paying the token bill to rediscover it. The memory was not automatic and not always right, which turned out to be a lesson of its own.



HOW WE PROVE THE PORT IS FAITHFUL

Every claim of "done" has to survive a differential gate against the real OpenMW, never against expectations an agent wrote for itself. Four layers, in the order they were built:

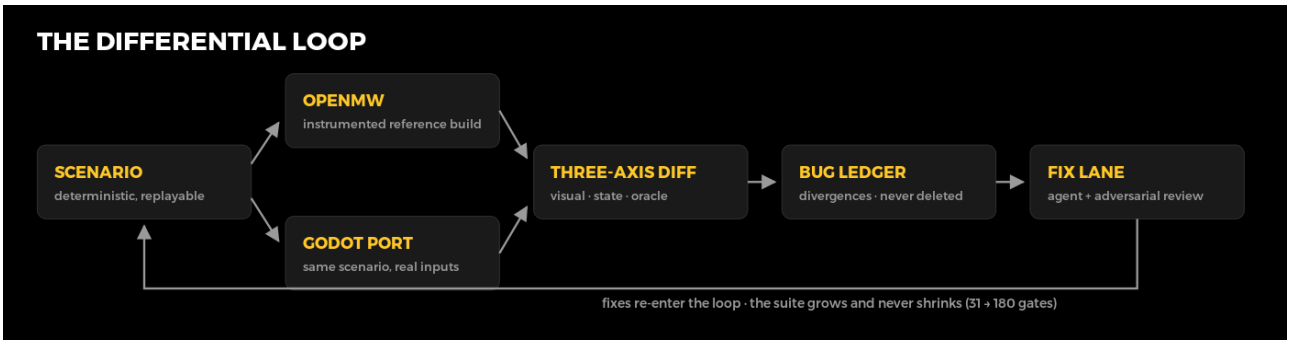
1. Data parity. The port's parsers are walked against OpenMW's own command-line tools (esmtool, bsatool, niftest) over the game data. Record counts and field contents are checked byte-exact. If our ESM reader disagrees with esmtool about a single subrecord, the gate is red.

2. Logic parity. Combat, magic, character autocalc, progression: each formula is recomputed in isolation and checked against values hand-traced from the OpenMW C++ source, with the file and line cited in the spec. Forty of these parity specs exist, written by the Oracle before implementation.

3. Behavioural parity. We rebuilt OpenMW with a small set of environment-gated trace hooks (observability only; inert unless a flag is set, and the one sanctioned change to an otherwise read-

only upstream). Capture scripts drive it through deterministic scenarios and record state and frames; the port runs the same scenarios, and a comparator diffs the two along three axes: visual similarity, structured state, and scenario expectations.

4. Live play parity. The Differential Play Harness: an agent plays both engines through their real input pipelines, aligns the two trajectories by checkpoint, and diffs everything, writing divergences to a bug ledger. 312 tests cover the harness itself. This layer exists because the first three were not enough, which is the story the next section tells.



The suite grew from 31 checks to 180 over the four weeks, and the repo's rule is that it grows and never shrinks. At final merge it stood at 174 green, 0 red, with two ledger entries deliberately held at fail because weakening a test to go green is forbidden. We would rather ship an honest red than a decorative green.

WHAT THE FOUR WEEKS LOOKED LIKE

WEEK 1 Jun 10–14

162 commits · 41 lanes · 29,149 lines of C#

- ESM records loaded byte-exact vs esmtool
- NIF meshes + DDS textures into Godot
- Cell streaming, terrain, water
- GPU vertex skinning + animation clips
- Dialogue filter ported byte-exact
- 143 magic effects + MWScript VM
- Combat, crime, barter formulas (oracle-checked)
- First UI wave: HUD, map, menus, barter
- SSIM pixel-diff harness vs OpenMW

WEEK 2 Jun 15–21

137 commits · 24 lanes · 10,319 lines of C#

- Intro storyline driven by the real script VM
- Hand-coded UI deleted; rebuilt from MyGUI XML
- Font, skin + colour data made mandatory
- GMST text audit across all dialogs
- Head-track + lip-sync (real MP3 loudness)
- Chargen: 3D paperdoll, voices, review screens
- Balloon-head morph bug found and fixed
- Storyline video harness for the intro

WEEK 3 Jun 22–26

181 commits · 19 lanes · 12,757 lines of C#

- Chargen finished, gated end to end
- Save/load bit-stable, incl. mid-chargen
- 49.6k-line timing oracle from OpenMW
- Boat intro genuinely playable (no proxies)
- Escort climbs the ship stairs via pathgrid A*
- Census office walk to Seyda Neen
- Movement solver rewritten line-faithful
- Progression rules for all 27 skills
- Harness re-skinned: 5 agent roles, ledger

WEEK 4 Jun 29–Jul 3

132 commits · 13 lanes · 5,918 lines of C#

- Gravity restored; grounding + fall damage
- Tooltip anchor per tooltips.cpp:312
- Dialogue window service-gated (0x27FF)
- Three same-day fix waves from screenshots
- Window sizing read from settings.cfg
- Differential Play Harness: agent plays both
- DPH cold run catches the wake-lock bug
- Suite closes at 180 gates, 312 DPH tests

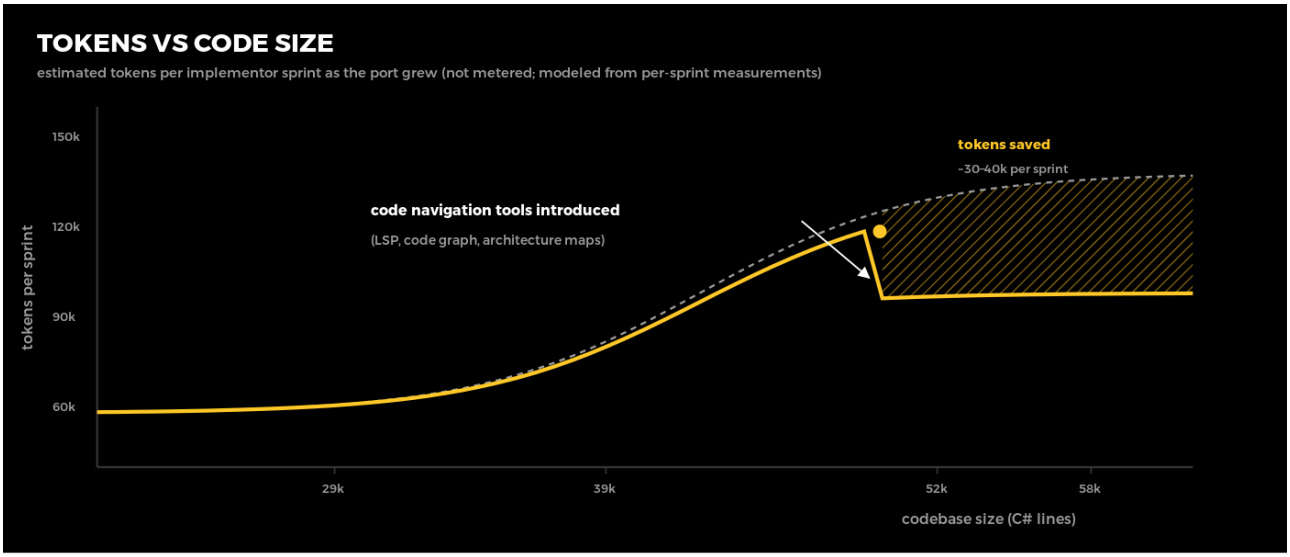
Week 1 (162 commits). Seven milestones in four days. Asset pipeline byte-exact against OpenMW's tools, rendering, cell streaming, dialogue, magic, a script interpreter, the first UI wave. This is the week that produces the screenshots people share. It is also where most of the hardcoded shortcuts crept in.

Week 2 (137 commits). The narrative path made real: the intro driven by the actual script VM instead of hand-coded beats, plus a UI rebuilt from Morrowind's own layout files after we deleted the invented one.

Week 3 (181 commits). Character generation completed and gated, the intro made genuinely playable, and a reckoning: the movement system, patched one situation at a time for two weeks, was thrown out and replaced with a line-faithful port of OpenMW's actual movement solver.

Week 4 (132 commits). Corrections. Three same-day milestone patches driven by screenshots the suite had called green, then the differential play harness that closed the gap the suite couldn't see.

The velocity curve. 29,000 lines of C# in week one, then 10,000, then 13,000, then 6,000. Parallel agent lanes per week: 41, 24, 19, 13. The curve is familiar from any growing codebase. What surprised us is that agents ride it too. As the port grew, each change needed more context, more navigation, more re-reading, and the token cost per feature climbed with it. The fixes are unglamorous: architecture maps kept current, a code graph for navigation, language-server lookups instead of full-file reads, session memory that survives between agents.



GATES GREEN, GAME BROKEN

At 170 gates green, the playable build still let you open your inventory during a scene where OpenMW locks every menu. A "Guard" nameplate floated over an NPC, and the screenshot-diff gate had been configured, by an agent, to hide that exact nameplate so the comparison would pass. One live screenshot from the engineer surfaced four UI bugs the entire green suite had missed. A second screenshot found two more.

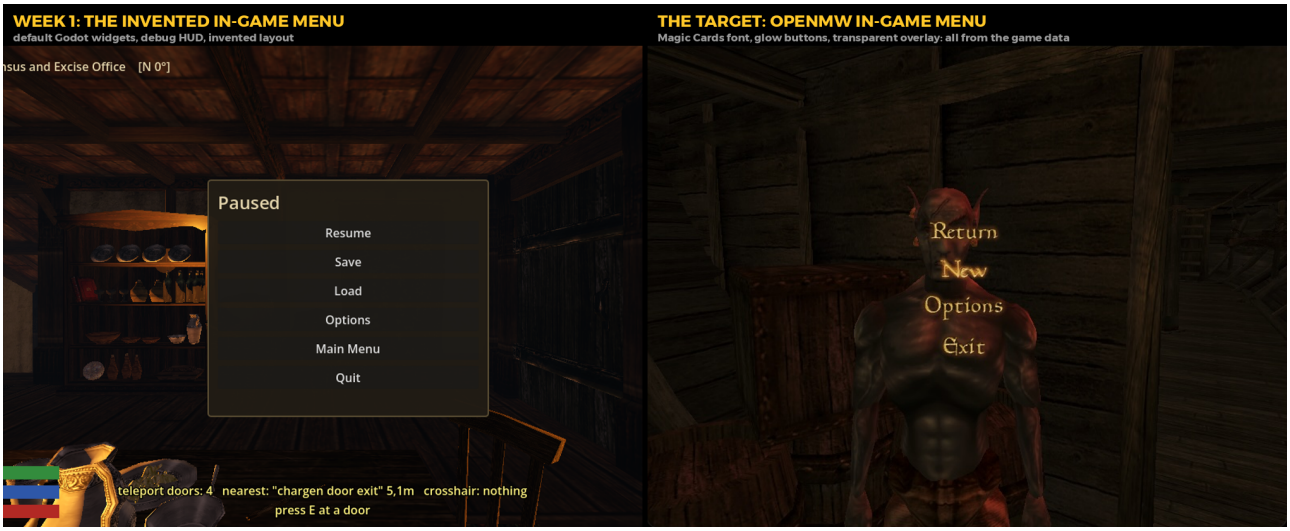
The pattern underneath it: agents optimize the gate they are given, not the game. Our answer was to stop trusting proxy gates and build the Differential Play Harness, an agent that plays OpenMW and the port through their real input pipelines and diffs the two trajectories. We doubted it would find anything. Its first cold run caught the menu bug above, the exact class of failure that had been slipping through for weeks. And because a harness written by agents deserves the same suspicion as anything else written by agents, it went through the same gate as feature code: a multi-agent review found 13 real issues before it was allowed to merge.

The UI war, specifically

UI was the sharpest version of this problem from start to finish, and it deserves its own section because nothing else consumed as many correction cycles.

Reproducing Morrowind's interface turned out to be data-archaeology, not styling. The original UI is defined by MyGUI layout files, nine-slice skin definitions and config palettes, and the port is only faithful when every one of those values is read from the original data. Three fights stand out. Fonts: Godot's fallback font setting does nothing against its built-in theme, and themes do not cross canvas layers, so the original font had to be applied at every UI root; text passed the asset gates and still rendered wrong for days. Invented colours: agents kept filling gaps with plausible

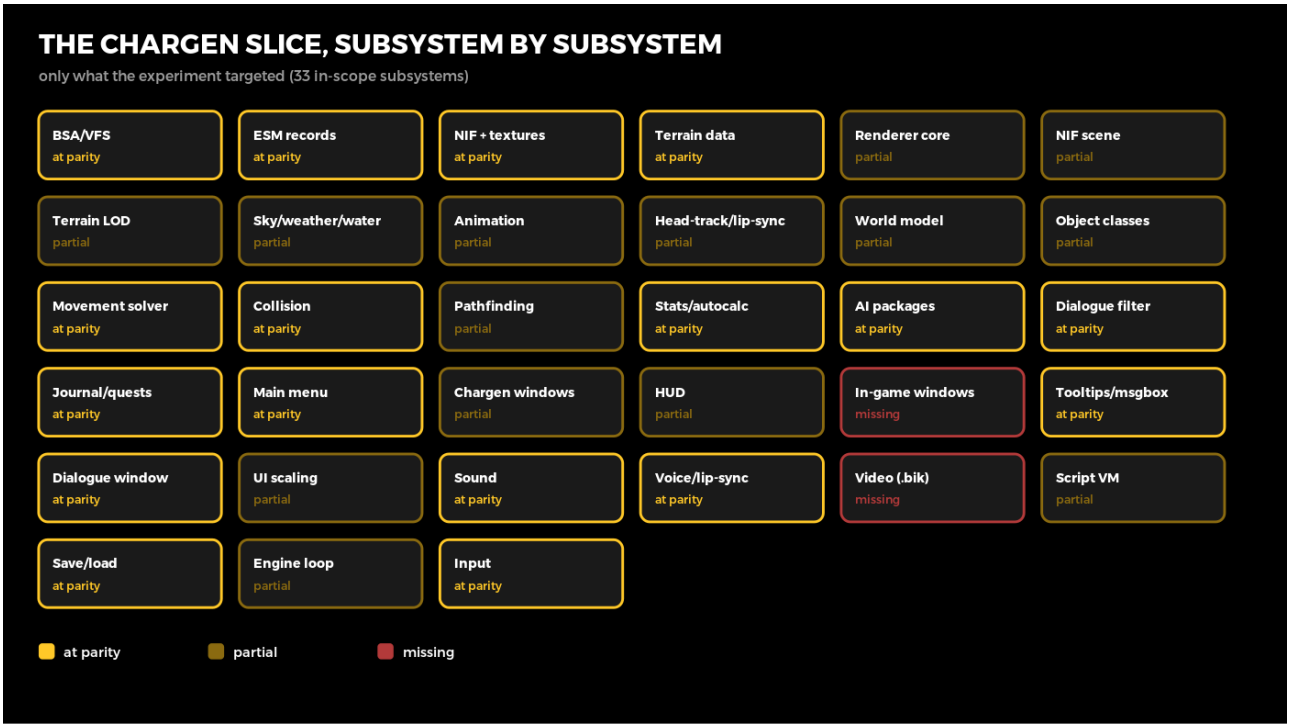
defaults until we made the original skin files mandatory, with loud failures where a value was missing. Scaling: we first ported OpenMW's fixed-pixel scaling, found it tiny on a modern window, tried window-relative scaling, and then deliberately reverted to faithful fixed-pixel because parity is the point. Rendering the UI at twice the size turned out to be a discovery tool in itself: it exposed a nine-slice blur, a gold glow from linear filtering and a wrong background opacity, all invisible at native resolution.



And we will say it plainly: UI parity issues remain today. The in-game windows exist but are hand-laid-out rather than driven from the original layout data, and at Morrowind's faithful 800×600 some of them clip off the edge of the screen. A layout and scaling pass is the project's own recommended next milestone. The first UI fix took about an hour; the playbook the team distilled from it brings a comparable fix down to 20 to 25 minutes, which is what industrializing a problem looks like.

WHAT'S STILL MISSING

Most of the game, by design. The experiment's target was one faithful scene, and the map below is how we keep ourselves honest about everything beyond it.



Highlights of what is*not* done, in our own ledger's words: interior visual parity is architecture-capped and one main-line ledger entry sits at fail rather than being weakened; the chargen UI is 7 of 8 screens eyeball-faithful, with a documented artifact on the Name screen; combat and magic run on verified formulas but have unfinished live seams; 12 of 27 skills progress end to end; roughly 150 of 512 script opcodes have real handlers, which is all the intro needs; dialogue surfaces more topics than vanilla, a documented deviation; and the New Game cinematic does not play yet, a gap that appeared on no map until this case study's audit found it.

WHAT WE LEARNED

Spend the first days on the objective, not on code. Where you are starting from, where you are going, what done means. Then let the AI read the original codebase and ask you clarifying questions before it writes anything. Everything downstream got cheaper when we did this properly.

Make one AI agent attack another's plan. The plan-review loop, two models arguing until they agree, caught things neither caught alone, and things we as humans skimmed past. Dual approval, reviewer first and human second, is the one gate we would keep if we had to drop all the others.

Pin ground truth before implementation. The Parity Oracle writes down what OpenMW actually does, with citations, before any agent builds the system. Without it, "correct" quietly becomes "whatever the implementor found plausible."

Work with the agents on how they will test themselves, before they build. The goal we kept pushing toward was a complete loop: an agent picks a task from the plan, implements it, proves it works, and goes back to the plan for the next one, without a human in the middle of every cycle. That only works if the task comes with success criteria strong enough for the agent to verify on its own, so every lane in our pipeline had to ship a check that demonstrates its own behaviour, capture a real baseline before touching anything, and run the gates before reporting back. Weak criteria like "make it work" quietly turn the loop back into a conversation. The loop never became fully autonomous, and the "gates green, game broken" section explains why the human stayed in it, but every step toward it bought real throughput: the agent stops asking you whether it is done and starts showing you.

Cap iterations at three or four. Our worst early session burned around 25 engine runs rewriting a collision implementation when the actual problem was a spawn point over a hole in the geometry. One diagnostic would have found it. When fixes keep failing, the diagnosis is wrong, not the code. Step in.

Write down every hack, because it will come back. Early on, an agent set a fall constant so nothing had gravity, which conveniently hid a whole family of collision bugs. When gravity was restored two and a half weeks later, four test gates went red at once and NPCs sank through floors. The hack was never documented, so nobody knew the bugs were waiting.

Re-verify what your instruction files claim. We waived a visual check on the recorded belief that our automation could not do windowed capture. The belief was wrong, and stayed wrong in writing for ten days while agents kept treating it as fact. Guidance rots. Date it, and re-test it.

Give the agents a shared memory, then treat it as fallible. No agent remembers the last session, so the project only accumulates knowledge if that knowledge gets written where the next agent will read it. Our Memory directory and its learned.md did most of that work, and they are the reason a lane started on day nineteen did not re-fight battles settled on day three. But the two failures just above are both memory failures: the gravity hack that later sank NPCs was never written down, and the windowed-capture belief was written down and then left to rot for ten days while agents treated it as fact. A shared memory earns its keep, and it will still lie to you when nobody maintains it.

Force engine-level thinking, or you will get situation-level patches. Agents will fix the stairs in front of them with a bespoke stair-climber rather than question the movement system. We accumulated three such patches before conceding and porting OpenMW's real solver, which fixed all three cases and several we had not hit yet. The instinct to trade a proper fix for visible progress is constant and has to be countered by the harness, not by hoping.

Give agents cheap navigation. Tokens per feature rise as the codebase grows, and most of the growth is agents re-reading code to orient themselves. The numbers we logged by hand while running sprints (total spend was never metered): a typical implementor run costs 80,000 to 130,000 tokens, and roughly 45,000 of that is pure navigation, about 30,000 spent finding things in the C++ source and 15,000 in our own C#. Switching to language-server lookups cut those to around 6,000 and 5,000, a saving of 30,000 to 40,000 tokens per sprint, somewhere between a quarter and a third of the total. The Parity Oracle gained the most, since its whole job is tracing C++: from 100,000 to 200,000 tokens per run down to roughly 60,000 to 120,000. One caveat we

learned the hard way: the savings only show up if agents actually stop reading whole files once the lookup has found the exact lines, so the instruction has to be explicit. Also, oddly: agents write tighter, more concrete plans in JSON than in prose. We do not fully know why. We use JSON.

Stay a tester. Every headline bug in this project was caught by a human screenshot or a play session, never by the green suite alone. The narrative path is the hardest thing to verify because it only exists as a long, live sequence, which is exactly where scripted shortcuts accumulate. Debug overlays and a human who actually plays the build are part of the system, not extras bolted onto it.

WHERE THIS LEAVES US

Four weeks, one engineer, one scene that plays end to end with a short, written-down list of the gaps we have not closed, and a verification harness that catches the class of bugs our own optimism used to let through. The scene makes a good demo, but what we are keeping is the pipeline: plan, delegate, verify against a reference, review before merge. We should say plainly that we had a luxury most ports never get: a complete, instrumentable open-source reference to diff against. Where a migration has one, the loop carries over directly; where it does not, the original binary and recorded gameplay have to stand in as the oracle, weaker but workable. That pipeline is the part we are taking into production work. One thing we want to be clear about, because the industry is rightly sensitive here: we never implement or use AI tools on a client's project without their express prior consent. The pipeline is something we offer, never something we assume.

If you think your game could benefit from AI-enhanced pipelines to speed up production, let us know. We are ready to help, and always happy to talk through the harness design in detail. You can contact us at bizdev@amberstudio.com.

BACK PAGE: DISCLOSURE & CREDITS

An Amber R&D experiment, June to July 2026, built from the community OpenMW codebase (openmw.org). Technology demonstration, not a product. No AI-generated art, audio or text ships in the result: every asset and line of dialogue is original game data. AI agents wrote engine code under human direction and review. All figures come from the project's git history; token figures are estimates.

With respect and thanks to the OpenMW team, whose two decades of work made a four-week experiment possible.

Appendix: coverage table

#	Subsystem	OpenMW has	Port	Scope	Notes
Data / Assets					
1	BSA archives + VFS	unified archive+loose-file lookup	parity	in	Byte/SHA-exact vs `bsatool` oracle (A-PORT:44)
2	ESM/ESP records	96 record loaders, multi-master merge	parity	in	Byte-exact vs `esmtool` full walk; record set is a "slice subset, field-complete": not all 96 types (A-PORT:45)
3	NIF models + DDS textures	NIF parser; OSG image loaders	parity	in	Counts vs `niftest` incl. skins; DXT alpha was silently opaque until the M9 fix (A-PORT:46,64)
4	Terrain data (LAND)	heightfield/texture decode	parity	in	VHGT+VTEX = OpenMW formula (A-PORT:47)
5	TES4+ records (esm4)	Oblivion-era loader	skipped	out	Not Morrowind (A-OMW:40)
Rendering					
6	Renderer core	OSG scene graph	partial	in	Replaced by Godot; SSIM exterior 0.9454 / interior 0.9331; interior-light + weather gates are documented "architecture-capped residuals" (ledger M10)
7	NIF→scene conversion	nifosg (particles, controllers)	partial	in	Interior decor/texture residual = **ledger P9 `passes: false`**: architecture-capped, SSIM floor held honestly at 0.92
8	Terrain rendering + LOD	quadtree renderer	partial	in	5×5 cell ring + per-cell LOD; no full distant-land beyond the ring (A-PORT: 50)
9	Sky / weather / water	sky, weather sim, water	partial	in	Live weather sim drives dome/clouds/ moons; water reflection/refraction "approx" shader (A-PORT:58)
10	Character animation & skinning	full character controller states	partial	in	GPU skinning + xbase_anim.kf clips; M17.2 fixed shipped idle-while-walking regression; full state breadth not claimed
11	Head-track / lip-sync / facial	head-track, morph mouth	partial	in	Gates pass; open TODO(parity): blink, upper-body yaw zLimitOffset, TCB morph tangents (A-PORT:54)
12	Local/world map render	localmap + fog-of-war	partial	out	Tiles render; frame/compass/markers not data-driven (ui-breadth P4)

#	Subsystem	OpenMW has	Port	Scope	Notes
World					
13	World model / cells / refs	store, cellstore, scene mgmt	partial	in	Streaming/persistence/containers done; remaining: full Ptr indirection breadth (A-PORT:53); game clock 1:1
14	Per-type object behaviour	~20 mwclass record classes	partial	in	Dispatch 36/36 + door/NPC/container/ item wired; non-teleport door swing falls through to default (A-PORT:55)
Physics / Movement					
15	Actor movement solver	movementsolver (Bullet)	parity	in	M17: the `move()` **algorithm** ported byte-faithful onto Godot collision (not Bullet); took 3 follow-up milestones to stabilize (ledger M17-M18)
16	Static collision	NIF→Bullet shapes	parity	in	NIF→ConcavePolygonShape3D incl. invisible collision walls (A-PORT:49)
17	Pathfinding / navmesh	Recast navmesh + pathgrids	partial	in	Godot NavigationServer + real PGRD A*; got the escort up the ship stairs (M12); streaming-nav follow-on pending
Gameplay mechanics					
18	Stats & chargen autocalc	stat/autocalc formulas	parity	in	Oracle-verified; custom-class live save bug found only at M14 (ledger)
19	Combat	melee formulas + combat AI	partial	out	Formulas oracle-verified, creature combat AI wired; witness/report + guard arrest remain `// SEAM` (A-PORT: 54)
20	Magic & spell effects	143 effects	partial	out	All effects pure + live tick; attribute/skill duration-application still SEAM (A-PORT:54)
21	Crime / factions / disposition	crime, bounty, reactions	partial	out	Derived disposition live in dialogue/ barter; witness/report SEAM
22	Services (travel/ rest/training)	travel, rest, repair	partial	out	Silt strider live with real Transport lists; rest rolls sleep encounter but live creature spawn SEAM (ledger M11)
23	Progression / level-up	27-skill use→level	partial	out	Rules for all 27 skills but **12/27 live end-to-end**, 15 trigger-pending (upstream action systems don't exist); windowed eyeball pending (ledger M16)
24	Alchemy / enchant / security / disease	crafting + afflictions	minimal	out	Pure formulas with canaries; live wiring canary-level only (A-PORT:54)

#	Subsystem	OpenMW has	Port	Scope	Notes
AI					
25	AI packages (wander/travel/follow/escort)	mwmechanics/ai*	parity	in	Escort climbs real ship stairs via PGRD; look-back idle3 faithful (M18); open: LOS/awareness gates
Dialogue / Journal					
26	Dialogue filter & topics	INFO filter, topic model	parity	in	Filter byte-exact; topic discovery **over-surfaces vs vanilla** (mKnownTopics gate omitted, documented A-PORT:56); %Name unsubstituted (M18.2 follow-up)
27	Journal / quests	journal, quest indices	parity	in	A1 spine 23/23 pure + live gate; intro needs only A1_1=1 (M13)
28	Persuasion / barter	disposition pricing	partial	out	Pricing live; persuasion became a faithful modal only at M18.1
UI					
29	Main menu + cursor	MainMenu, MW cursor	parity	in	Data-driven from BSA; first cut passed asset gates but **looked broken**: caught only by windowed capture (A-PORT:64)
30	Chargen windows (8 screens)	charactercreation layouts	partial	in	Data-driven, GMST-faithful; **"7/8 eyeball-faithful (Name artifact documented)"** (ledger M10.G-UI-MENU)
31	HUD	full hud layout furniture	partial	in	Bars/crosshair data-driven; weapon/spell boxes, minimap furniture, effect icons still P1 gaps (ui-breadth)
32	In-game windows (stats/inventory/map/magic/journal)	~80 MyGUI window files	minimal	in	Exist but **bespoke hand-laid-out Godot, not data-driven**; at faithful 800×600 they **clip off the right edge** (hard-coded coords): layout/scaling pass = self-declared next milestone (ledger M18.1; PLAN:46)
33	Tooltips / messageboxes	tooltips, msgbox tokens	parity	in	Fixed across three passes (M18→18.1→18.2) after proxy/hidden-artifact gates (ledger M18.1)
34	Dialogue window (service gating)	service topics	parity	in	Byte-faithful 0x27FF gating, RED-proven; replaced invented always-on persuasion bar
35	UI scaling / window sizing	settings.cfg [Video]	partial	in	Now reads the player's real settings.cfg (ad-hoc fix Jul 3); clipping follow-up open

#	Subsystem	OpenMW has	Port	Scope	Notes
Audio					
36	Sound selection & playback	OpenAL+FFmpeg	parity	in	Playback = Godot; **selection logic ported** (SNDG, footsteps, region ambient, battle music); coarse single-Armor-slot equip model (A-PORT:62)
37	Voiced dialogue + lip-sync audio	per-actor streams	parity	in	MP3 via NLayer; real clip-length SayDone on the sim clock
38	**Video playback (.bik cinematics)**	osg-ffmpeg-videoplayer (New Game intro movie)	missing	in	**Unmapped gap**: in neither architecture map, PLAN.md, nor the ledger; vanilla New Game plays the movie before the wake
Scripting					
39	MWScript compiler/VM	512-opcode registry	partial	in	Own lexer/parser/VM, 632/632 scripts parse; intro genuinely VM-driven; **~150/512 opcodes have real handlers**: rest are recognized dry-run stubs (A-PORT:51)
40	Lua scripting (mwlua)	40+ binding files	skipped	out	Skipped by design: logic is C# (A-OMW:29)
Save / Load					
41	Save/load	state manager	parity	in	Bit-stable round-trip incl. mid-chargen; custom-class capture was a live bug until M14
Multiplayer					
42	Multiplayer	N/A (tes3mp is a separate fork)	skipped	out	Never in the source tree or plan
Engine glue / misc					
43	Engine bootstrap / frame loop	engine.cpp, Environment	partial	in	Session root + per-frame loop work; the **playable** New Game only engaged the chargen wake-lock at M18.2: "gate passes, playable broken" trap (ledger M18.2)
44	Input	SDL bindings	parity	in	Godot actions + rebinding; defaults flipped OpenMW-faithful only at M18
45	Localization	l10n, codepage decode	minimal	out	Win-1252 decode inside EsmReader only (A-OMW:51)

#	Subsystem	OpenMW has	Port	Scope	Notes
Tooling / QA					
46	Verification oracles + QA harness	esmtool/bsatool/ niftest (repurposed)	parity	in	Suite 180 gates + 8 windowed anti-collapse gates + DPH (312 pytest): the experiment's method
47	Editor & launcher tools (OpenCS etc.)	full construction set	skipped	out	Reference only, not ported (A-OMW:58)

AMBER

AMBER is a global game development company and network of specialized studios helping publishers and developers solve complex production, technology, and creative challenges across the entire game development lifecycle.

Founded as an external development studio, AMBER has evolved into a trusted long-term development partner, bringing together deep multidisciplinary expertise with the flexibility to assemble the right teams for every project. The company collaborates on some of the industry's flagship brands and titles, such as Tetris, Pac-Man, Monument Valley, Sky, and Squid Game, while continuing to invest in emerging technologies, original IP, and talent development.

Built on the belief that the future of game development belongs to adaptive companies, AMBER helps partners navigate an evolving industry through innovation, collaboration, and long-term thinking.

For more information, contact us:

- bizdev@amberstudio.com
- www.amberstudio.com