Rules of Engagement

Read this before you edit the file.

Nine house rules for every AI that touches the master map. Follow them exactly. They are short on purpose.

Why this exists

The roadmap decayed into a rat's nest once already. It was not one agent's fault. Many agents each updated a different spot. None of them were wrong on their own. Together they produced three competing sequences and a dozen contradicting status lines.

This protocol is the fix.

Rule 1 — One fact, one home

Every kind of information has exactly one legitimate home in this file. Writing it anywhere else is how the mess started.

What lives where

Build order. Lives only on View 1 and View 2. Nowhere else. Not in prose. Not in an archive full record. Not in a list-row description.

A build's current status. Lives on View 1, View 2, and that build's list-row status field. Not in the archive. The archive is history, not a status board.

Slice detail, receipts, specs. Lives in the child charter. Not here. This parent never absorbs a child's scope.

Custody, links, historical record. Lives in the archive. Not on the views. The views stay short.

Standing laws and method. Lives in the operating manual and laws sections. Not scattered into build entries.

The test, before you type a fact: does this already live somewhere in this file? If yes, go edit it there. Do not add a second copy. A second copy is a future contradiction with a delay fuse.

Rule 2 — The three-touch update

When a build's state changes, you touch exactly three places. Always the same three.

1. The list row. Update its status (pending / in progress / completed) and, if the description changed, its text.

2. View 1, the board. Update that build's status mark and, if slices closed, its slice list.

3. View 2, the detail. Flip the closed slices to done, and update the "what remains" line.

Then stop. Do not also update the archive. Do not add a note at the bottom. Three touches, done.

If those three disagree with each other after your edit, you did it wrong. Re-read all three before you close the session.

Two exceptions — and only two

A new build is five touches, not three. Rule 4 governs. It adds a child charter and an archive custody entry on top of the three.

Closing a build adds one archive receipt. Write the closure evidence into that build's archive status line. Never into its full-record paragraph. Those are frozen history.

Everything else — ignition, slice completion, blockers, halts, gate discharges — is three touches and nothing more.

Rule 3 — Use the legend, and only the legend

The eight statuses at the top of the page are the complete vocabulary. Do not invent new ones. The difference matters.

Done — Closed with receipts. Not "the code landed." Not "tests passed once." Closed, checked, receipted. If you cannot point at the receipt, it is not done.

In flight — A seat is actively building it right now, this week. A build nobody has touched in two weeks is not in flight. It is partly done or halted.

Partly done — Real work landed and real work remains. Use this liberally. It is the most under-used status, and its absence caused the worst errors in this file. Builds sat at "not started" for weeks while they were substantially built.

Halted — Was in flight, deliberately stopped, resumable. Halted is not closed and it is not pending. A halted build must say, on View 2, where it stopped, why, and what the resume ladder is.

Held — Cannot start because a named thing must happen first. An ownerless hold is a defect. Every hold states what unblocks it and who owns that.

Waiting — Not started. Nothing blocking except its turn in the queue.

Draft only — A charter exists but it is rough. It needs an ignition pass before it is buildable. Do not confuse this with waiting. Waiting means ready to go.

Standing — Never finishes. Runs on operator call.

The list-row status field can only say pending, in progress, or completed. It is coarser than the legend. The board carries the precision. The list row is a coarse index into the board.

– Done on the board maps to completed on the list.

– In flight, halted, or partly done while someone is on it maps to in progress.

– Partly done with nobody on it, held, waiting, draft only, and standing all map to pending.

Because pending is doing so much work, a pending list row whose build is actually partly done or held must say so in the first sentence. Otherwise the row reads as untouched. The board is authoritative.

Rule 4 — How to add a new build

In this order. All in one pass.

1. Create the child charter. That file owns all slice detail.

2. Add a list row here: an id, a description of purpose, slices, and gates, and status pending.

3. Add a one-line row to View 1 at its sequence position, with slice names.

4. Add a View 2 entry with the charter filename, a plain-English purpose, the slice list, and the gate.

5. Add an archive section with the code, the name, the charter link, status, and scope.

6. If it displaced anything in the sequence, fix the position numbers of everything after it, and add a reconciliation-log row saying what moved and why.

A build that exists in fewer than all five places is a landmine. Eight builds once went missing that way. If this rule had existed when they were created, none of them would have gone missing. That is the entire argument for it.

Rule 5 — How to record a halt

A build stopped mid-flight is the most dangerous state in this file, because the next agent inherits it cold.

Status becomes halted. Never done. Never back to waiting.

On View 2, record four things:

– Which slice it stopped inside

– What is already on disk (saved versus not saved)

– What was in progress and left unfinished

– The resume ladder — the ordered list of what the next seat does first

Put the deep version in the child charter under a clearly marked pick-up marker. Put the pointer here.

Say why it halted. “Halted by operator priority” and “halted by a defect” send the next agent down completely different paths.

Rule 6 — How to change the order

Ordering changes are the single biggest source of drift in this file’s history. When the sequence changes:

1. Change it on View 1. Move the row. Renumber.

2. Change it on View 2. Move the entry to match.

3. Add a dated line to the reconciliation log: what moved, from where to where, on whose ruling, and the date.

4. Search the whole file for the moved build's code. Fix every stale "next," "after X," and "once Y closes" claim you find — especially inside archive full-record paragraphs, which is exactly where these rot unseen.

Step 4 is not optional. It is the step everyone skips.

Rule 7 — The dated-correction rule

When you correct something that was previously wrong — not merely out of date — do not silently overwrite it. Write the correction inline in this form:

Corrected on this date — this previously read "X," which was wrong because Z. Truth is Y.

Two reasons. A future agent who read the old version elsewhere needs to know it was retracted. And a silent overwrite hides the fact that the file was producing wrong answers, which is the information most worth keeping.

Rule 8 — What you may never do

Never add a second sequence. Not a summary list. Not a "quick reference." Not a status block at the bottom. Not a fresh section because the existing one felt hard to edit. This file had three competing sequences. If View 1 is wrong, fix View 1.

Never write ordering language inside an archive full record. No "next." No "immediately after." No "runs before." The archive records what a build is and what happened. The views record when it runs.

Never delete detail. The details are load-bearing. They are guidelines for others to pick up on, or come back to, or cross-reference. Restructure, relocate, and re-title freely. Delete nothing.

Never mark a build done without a receipt. See Rule 3.

Never leave the parent stale. If you changed a child charter's state this session, you update this file in the same session. A child that moved while the parent stayed still is how the next agent gets lost.

Never trust a status you did not verify this session. Statuses here are written by agents who may have been wrong. Before you rely on one for a real decision, open the child charter and check. Three status lines in this file were flatly wrong, and had been for weeks.

Rule 9 — Session-exit check

Sixty seconds. Do it every time. Before you end any session in which a build's state changed, ask:

1. Does the list-row status match View 1?

2. Does View 1 match View 2?

3. Does "where we are right now" still name the correct active build and the correct next three?

4. Did anything you closed unblock a hold? If so, that build's status changes too. A satisfied gate that nobody flipped is a build stalled for no reason.

5. Did you introduce any ordering language outside the two views?

If all five pass, you are clean. If any fail, you are handing the next agent the mess this section exists to prevent.

A starter you can steal

A master plan dies when the same fact is copied into three places and only one of them is updated. Give every kind of fact one home: order on the two views, status on those views plus the coarse list row, slice detail in the child plan, custody in the archive, laws in the operating manual.

A normal state change touches exactly those three status homes. Adding a build writes five homes in one pass. Closing one adds a single archive receipt. Use eight board words, map them onto pending, in progress, or completed, and never invent a ninth. The archive is history, so it does not say what runs next. Done without a receipt string is not done.

This is a teaching checker over a toy file. It is not a copy of any production system. Paste it into an empty folder and run it. It flags the five lies a messy plan tells first. Halt notes, dated corrections, and the session-exit questions stay human writing.

```
"""Toy checker for a toy markdown roadmap. Stdlib only.\n \nRun: python check_roadmap.py [optional.md]\nNo
argument grades the SAMPLE (it is dirty on purpose).\n \nChecks and the house rule each one teaches:\n
legend_only    — rule 3: board status is one of eight words\n  coarse_list    — rule 3: list field is
pending / in progress / completed\n  required_homes — rules 1 + 4: id in list, View 1, View 2, archive,
charter\n  archive_order  — rule 8: no next / immediately after / runs before in archive\n  done_receipt
— rules 3 + 8: done needs a receipt string\n \nRules 5, 7, 9 (halt record, dated correction, session-exit)
stay human.\n"""\nfrom __future__ import annotations\n \nimport re\nimport sys\nfrom pathlib import Path\n
\nLEGEND = {\n    "done", "in flight", "partly done", "halted",\n    "held", "waiting", "draft only",
"standing",\n}\nLIST_OK = {"pending", "in progress", "completed"}\nORDER_RE =
re.compile(r"\b(next|immediately after|runs before)\b", re.I)\n \nSAMPLE = """# Toy roadmap\n## List\n- id:
B-1\n  status: completed\n- id: B-2\n  status: pending\n- id: B-3\n  status: shipped\n## View 1\nB-1 | done
| First\nB-2 | waiting | Second\nB-3 | shipped | Third\nB-4 | held | Ghost\n## View 2\n### B-1\nstatus:
done\n### B-2\nstatus: waiting\n### B-3\nstatus: shipped\n## Archive\n### B-1\ncharter: b1.md\n### B-
2\ncharter: b2.md\nthis one runs before B-1\n### B-3\ncharter: b3.md\n"""\n \n \ndef section(text: str,
title: str) -> str:\n    for chunk in re.split(r"(?=^## )", text, flags=re.M):\n        if
chunk.startswith("## " + title):\n            return chunk\n    return ""\n \n \ndef headed(block: str) ->
dict[str, str]:\n    parts, current = {}, None\n    for line in block.splitlines():\n        hit =
re.match(r"### (B-\d+)", line)\n        if hit:\n            current = hit.group(1)\n
parts[current] = line + "\n"\n        elif current:\n            parts[current] += line + "\n"\n    return
parts\n \n \ndef check(text: str) -> list[str]:\n    flags = []\n    listing = {\n        m.group(1):
m.group(2).strip().lower()\n        for m in re.finditer(r"- id: (B-\d+)\s+status: (.+)", section(text,
"List"))\n    }\n    board = {}\n    for line in section(text, "View 1").splitlines():\n        m =
re.match(r"(B-\d+)\s*\|\s*([^|]+)", line)\n        if m:\n            board[m.group(1)] =
m.group(2).strip().lower()\n    detail, archive = {}, headed(section(text, "Archive"))\n    for bid, body
in headed(section(text, "View 2")).items():\n        m = re.search(r"status:\s*(.+)", body, re.I)\n
if m:\n            detail[bid] = m.group(1).strip().lower()\n    homes = {\n        "list": set(listing),\n
"view 1": set(board),\n        "view 2": set(detail),\n        "archive": set(archive),\n        "charter":
{i for i, b in archive.items() if "charter:" in b.lower()},\n    }\n    for bid, status in
listing.items():\n        if status not in LIST_OK:\n            flags.append(f"coarse_list {bid}:
{status!r} not pending/in progress/completed")\n    for label, table in (("view 1", board), ("view 2",
detail)):\n        for bid, status in table.items():\n            if status not in LEGEND:\n
```

```
flags.append(f"legend_only {label} {bid}: {status!r} outside the eight words")\n    for bid in
sorted(set().union(*homes.values())):\n        miss = [n for n, g in homes.items() if bid not in g]\n
if miss:\n            flags.append(f"required_homes {bid}: missing {', '.join(miss)}")\n    for bid, body
in archive.items():\n        hit = ORDER_RE.search(body)\n        if hit:\n
flags.append(f"archive_order {bid}: {hit.group(0)!r} is view language")\n    blob = section(text, "View
2")\n    for bid, status in {**board, **detail}.items():\n        if status == "done" and "receipt:" not in
(blob + archive.get(bid, "")).lower():\n            flags.append(f"done_receipt {bid}: done with no receipt
string")\n    return flags\n \ndef main() -> int:\n    path = Path(sys.argv[1]) if len(sys.argv) > 1
else None\n    text = path.read_text(encoding="utf-8") if path and path.exists() else SAMPLE\n    flags =
check(text)\n    print("clean" if not flags else "\n".join(flags))\n    return 0 if not flags else 1\n \n
\nif __name__ == "__main__":\n    raise SystemExit(main())
```

Aether guides. Read it on the page before you keep a copy.