PERSLIS RESEARCH

Reading the Manual Is Not Enough

A Rule Pilot in Fallout (1997)

What a game manual, a walkthrough, saved progress and level-ups change in a pilot with no neural network in the loop that decides

Perslis Research | September 2026

Research prototype · simulation and games · not a certified safety system

Abstract. We put a rule pilot, with no neural network in the loop that decides, at the controls of Fallout (1997), on the open-source Fallout Community Edition engine with an instrumentation patch (state out, frames out, actions in). At each decision the admissible goals are those the rules find applicable and sourced cards license, narrowed by orders; experience may only remove or reorder them. The game's printed manual, compiled into 2,250 cards with page receipts, withholds no applicable goal in any of 256 fact combinations and changed the chosen goal in 0 of 2,344 live decisions: it names none of the 22 quests a walkthrough lists, 2 of its 12 places, and, outside its voice credits, none of the people the pilot met. That community walkthrough supplies quests as goals and whom to approach; experience adds where each line of dialogue leads, per counterpart. The pilot saves progress like a player, but only into two scratch slots; the engine refuses every other slot, so the learner cannot overwrite the player's saves. In a 12-minute A/B per arm on the same save and dice, learning on versus off recorded 3 versus 16 deaths, a first death at 8.6 versus 1.0 minutes and +250 versus 0 experience; from empty memory, 6 versus 16 deaths. One run per arm. It claims level-ups through the instrumentation and reached level 2 live after 10 deaths, each returned to a save. We report the defects found and what remains open, including three hours without experience gained.

1 Introduction

A new player of a 1997 role-playing game opens the box and reads the manual. It explains the interface, the character sheet, combat by action points, how to talk, trade, heal and travel. It does not say where to go, whom to talk to, or what to do there; for that, players have written walkthroughs. This paper asks what each kind of document contributes when the player is software that may only pursue goals a sourced document licenses.

We answer for one game, one manual, one walkthrough and one pilot. The game is Fallout (Interplay, 1997) [5], run from a legally owned copy of the game data on Fallout Community Edition, an open-source re-implementation of its engine [7], with a small instrumentation patch that writes the engine's state and frames out and takes actions in. The pilot is a rule pilot with no neural network in the loop that decides. It runs inside the VDSG runtime [23]: at every decision the admissible goals are the goals the rules find applicable to the engine's facts, intersected with the goals licensed by sourced cards, narrowed by standing orders; the pilot's experience may only remove goals from that set or reorder it, never add one [19]. The game's printed manual was compiled into 2,250 sourced cards with page receipts; a community walkthrough, into 2,289 more.

A manual says how to operate the game. A walkthrough says what to do in it.
In our records only the second ever changed a decision.

The manual licensed every goal the situation offered and, by the pilot's own counterfactual counter, changed the chosen goal in none of 2,344 live decisions. What changed behaviour was the walkthrough, which names places, quests and people, together with two things a long game asks of any player: keeping progress, and claiming the character's growth. We measured each from the pilot's own records, and we report the defects those records exposed as part of the method.

1.1 Contributions

C1. A measured negative for the manual, and what a walkthrough adds (§4–§5). As a licence, the manual withholds no applicable goal in any of the 256 combinations of the eight situation facts, once a retrieval artefact that hid its run-away page is fixed; as advice, it changed 0 of 2,344 live decisions. It names none of the 22 quests the walkthrough lists, 2 of the walkthrough's 12 places (the home vault and the first destination of its premise), and none of the 17 people the pilot talked to outside its voice credits. The walkthrough supplies quests as goals and whom to approach; experience adds where each line leads, per counterpart. The conversation that took the pilot to level 2 was with a person 23 hexes away whom only the walkthrough made worth the walk.

C2. Saved progress under a least-privilege rule (§6). The pilot saves like a player, at a quiet moment with something gained, but only into two scratch slots; two guards in the engine refuse every other slot, and the quick-save slot is disarmed after every load and save. We state the resulting property, the evidence that pins it, and its residual. In the live run that reached level 2, each of 10 deaths returned to the newest save not abandoned, and one save that had become a trap was abandoned by rule.

C3. An A/B of learning on against off (§7) on the same save and dice, 12 minutes per arm, with both arms saving: 3 versus 16 deaths with a snapshot of the live memory, 6 versus 16 from empty memory. It is one run per arm, and we say so wherever it appears. A death-count defect fixed after the runs makes these counts lower bounds (deaths can only have been missed); in the worst case the arms with learning on died 8 and 12 times, against at least 16 with learning off. An earlier A/B was contaminated by a shared game directory; it is excluded and the reason is given.

C4. Growth, and a method that keeps its own defects (§8–§9). The pilot claims level-ups through the character screen's own accounting, spends skill points on what it uses and takes the walkthrough's perks; the claim was made live. We tabulate the defects found by the pilot's records and by automated code reviews (12 findings in one review, 11 fixed and 1 contested; one more, a skill cap copied as 300 where the engine's is 200, found by another), re-run the lane's test suite at four commits, and report a three-hour run in which experience did not rise at all.

1.2 What we claim, and what we do not

We do not claim that the pilot plays Fallout well: it reached level 2 once, its only actionable quest at that point defeated it at level 1 and again at level 2, and for three hours afterwards it gained nothing (§10). We do not claim that the walkthrough's effect is measured by a controlled comparison: the A/B varies learning, not the books, and both arms read both books. Every comparison rests on one run per arm, one save, one character and one engine version, and every number is ours; none has been replicated by a third party. What we do claim is narrower: on this game and this pilot, a document that speaks only to how the game is operated licensed everything and steered nothing, a document that speaks to the game's world changed what the pilot did, and saving and levelling can be added to a learner without giving it the power to write the player's saves.

Relation to earlier papers. The runtime, its orders and its bounded learner are described and proved in the VDSG paper [23]; the learning loop, its Wilson gate and the credit-assignment repair it needed in

this game are in the fail-first paper [19]. There, a live Fallout run died 49 times at one guard because blame reached only the last moments of each fight; charging a fight to the line of dialogue that started it, and condemning only with a Wilson lower bound over the base rate, ended the loop in scripted scenes. We use that machinery here and do not repeat its results. The sibling papers on tank games [21] and on training grounding into small models [22] report the same group's other measurements in the same style.

2 Setting: the game, the engine and what is recorded

2.1 The game and the engine

Fallout is an isometric, turn-based role-playing game: the player's character walks hex maps of towns, vaults and desert encounters, talks to people through menus of numbered lines, fights in turns priced in action points, and grows by experience points and levels. The pilot plays it on Fallout Community Edition [7], a source re-implementation of the original executable, pinned at upstream commit `0609bcf` (January 2025), with a legally owned copy of the original game data. When this paper says "the engine does X" it means this re-implementation, read at that commit; it aims to reproduce the original game's behaviour, and we cite the source file where a claim depends on it.

The pilot never runs on the player's own installation. A staging script gives it a private game directory: the two large data archives are read through links, and the maps, scripts, text and saved games are copied, so nothing the pilot does can write the files a person plays on. The pilot starts from an existing save of a level-1 character in the town of Shady Sands. Two game copies never share a directory; the one time they did is the excluded run of §7.

2.2 The instrumentation patch

The engine is not otherwise modified. At commit `6fc5c72` the patch adds 2,211 lines and removes 11, across 25 files; most of the added lines are in three new files (the tap itself, 1,456 lines; its level-up commands, 164; a JSON writer, 48), and the rest are hooks in the main loop, combat, dialogue, save and load, the character screen, the world map and the quest list. Every other frame it writes the engine's state as JSON, and it also writes the current frame as a bitmap; it reads commands from a text file and runs each at the point in the game loop, or in the combat turn, where a click would have been handled. The state holds the player's hit points, action points, level, experience, hands, inventory and six path probes; every critter, item, door and exit grid on the map with its hex distance, path length and, for critters, the engine's chance to hit it; combat and turn state; the open conversation (speaker, reply, numbered options); the quest list; the world map; and counters the engine keeps for saves, failed saves and reloads the pilot asked for. The commands are walk, leave, attack, talk, use, get, wield, heal, end turn, travel, enter, a path-finder query, save, choose the save a death returns to, reload, claim a level, raise a skill and take a perk, plus a small set of raw keys.

2.3 Decisions and records

The pilot decides every 0.5 s, and every 0.35 s in combat. Each console run appends one JSON line per decision to a log: the map and hex, hit points, action points, experience and level, whether in combat or a conversation, the goals applicable, admissible, infeasible and benched, the goal chosen and the commands sent, the pilot's status line, the quest on the agenda, the nearest hostiles with the engine's hit chance, the save counters, and the engine's answer to the last command. The log begins at 17:38 on 26 September 2026 (it was added that afternoon); before that, the only records are the pilot's counters, the console's read-outs and the commit history. Old logs are kept under names that say what the run showed. The pilot's memories (deaths per situation and goal, what each line of dialogue caused and where it led, quests set aside, saves) are JSON files beside the logs.

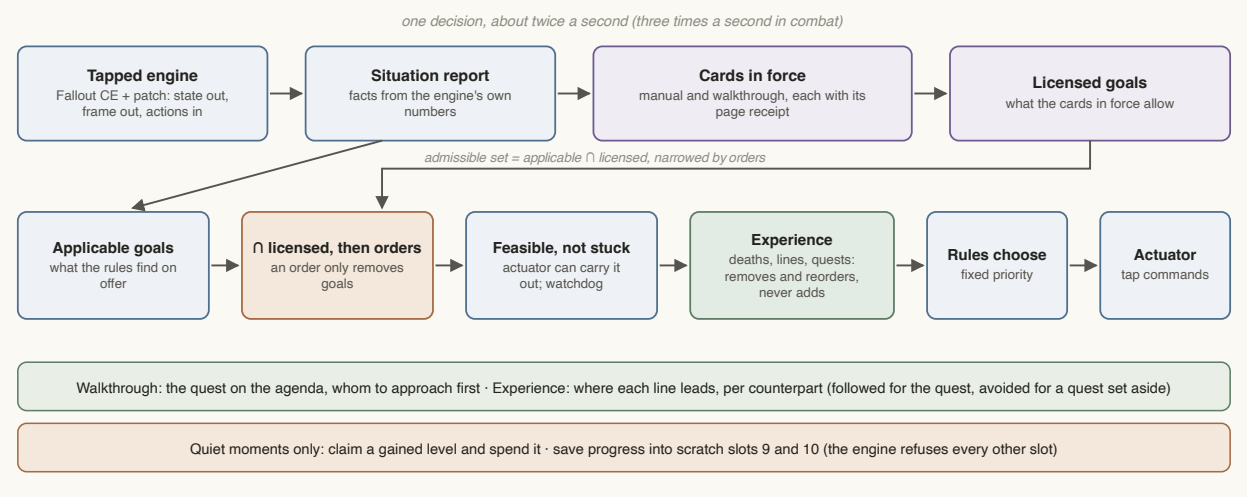

Figure 1. One decision. The books enter twice: their cards in force license goals (top row), and the walkthrough also names the quest on the agenda and whom to approach first (green band). Experience enters only after the admissible set is fixed, and only removes and reorders. Level-ups and saves are taken at quiet moments, one decision each (amber band).

Data cutoff. The code described here is commit `6fc5c72` (26 September, 22:59 EDT). The decision logs analysed run to 00:40 on 27 September; the last 101 minutes of them ran exactly that commit. The lane continued after the cutoff, with six more commits including a campaign layer (§10); none of that is evaluated here. All analysis is read-only: logs, results and memories were read, pinned code was taken from git objects, and nothing in the lane was run except its own unit tests, from an exported copy, with no engine. Appendix A maps every number to its file.

3 The pilot

3.1 The admissible set

At every decision the pilot turns the engine's state into a situation report and computes, as in the VDSG runtime [23],

$$A_{\mathcal{O}}(s) \;=\; O\big(\mathrm{App}(s) \cap \mathrm{Lic}(s)\big),$$

where App(s) is the set of goals the rules find applicable (FIGHT needs a hostile, TALK someone reachable, LOOT an item, LEAVE an exit grid, and so on; WAIT always), Lic(s) is the set of goals licensed by the cards in force, and O applies the standing orders, each of which only removes goals ("don't fight", "hold position") or narrows the set to one kind of goal ("talk first"). Two goals are the floor's own and never the books' to withhold: WAIT, and the way out of a conversation that cannot go on. What is left is filtered by what the actuator can still carry out and by the stuck rule (a goal chosen over and over while nothing changes is benched for a doubling while), and then by experience, which may remove a goal that has been killing the pilot here and may reorder what remains. A fixed-priority rule policy chooses inside the result: survival first (heal when critical; in combat, arm, fight or flee, end the turn), then a discretionary tier of LOOT, TALK, USE, LEAVE and EXPLORE. Only in that tier may the books reorder goals by their votes. Figure 1 shows the chain. The proofs that orders only narrow and that experience cannot widen the set are in [23, 19]; the lane's suite pins them with the three boundary tests reported there.

3.2 Cards in force

Eleven situation facts are read off the report: in combat, a hostile near, unarmed with a weapon carried, hurt with a stimpak carried, in a conversation, someone to talk to, items near, a door near, an exit near, on the world map, on the map. Each fact retrieves the cards that best match a fixed vocabulary of terms for it; a card licenses the goals its own text speaks to (a card that mentions attacking licenses FIGHT; a card about doors licenses USE), and every card carries its page receipt, which the console shows beside the goal it licensed. Retrieval is an index over words, not language understanding. When the walkthrough is loaded, its chapter for the current map is also in force: the chapter's first instruction and every page that names someone present. How the two documents were compiled into cards is withheld (§10); what matters here is that a card can only license what its own words speak to.

3.3 Experience

Three memories learn, each from failures the game verifies [19]. The *evidence* memory keeps deaths per situation and goal, charges a fight to the conversation that started it, judges a goal against the other goals in the same situation, and condemns a pattern only by the fail-first paper's rule, which weighs a Wilson lower bound on its death rate against the base rate. The *speech* memory keeps what saying a line caused (a fight, the end of the conversation, lost hit points, a death) and, separately, where the line took the pilot, per counterpart: one person's "Yes." is not another's. The *quest* memory sets a quest aside after three deaths in a place the quest's own words name, until the pilot's level rises. With learning off, none of the three is consulted or written.

3.4 What sees, and what decides

The lane also runs a small "eye" that labels screen columns from colour histograms, taught by the rectangles the tap reports; its labels are drawn on the console and no rule reads them. What such an eye is worth as a witness of the engine's claims is studied in a companion paper [24]. Nothing in the decision chain is a neural network or any other trained model: the rules are fixed, the memories are counts with the events behind them, and the cards are text with receipts.

4 The manual: a licence that withholds nothing, advice that changes nothing

The game's printed manual (121 pages) [6] was compiled into 2,250 cards. Until the walkthrough was added it was the pilot's only book, and we tested it in the two roles a book can play in this architecture.

4.1 As a licence

A book narrows the pilot only if some goal the situation offers is licensed by no card in force. We enumerated all $2^8 = 256$ combinations of the eight facts that describe the map (in combat, a hostile near, unarmed with a weapon carried, hurt with a stimpak, someone to talk to, items near, a door near, an exit near), built a situation report for each, and asked the pinned code for $\mathrm{App}(s) \setminus \mathrm{Lic}(s)$ with the manual as the only deck. Table 1 gives the result at three commits. Before goal-aware retrieval the manual withheld exactly one goal, FLEE, in the 128 combinations with a hostile near: its page advising the player to run away ranked 16th for combat, and only six pages per fact were in force. Once each fact also brings in the best page on every other goal the book speaks to within its retrieval window (commit `7acb344`), the manual withholds nothing, and loading the walkthrough beside it changes nothing. This re-measurement reproduces the figures recorded in that commit. The mechanism can withhold (the lane's tests tear pages out of a synthetic deck and the goal disappears); this book does not.

Table 1. The manual as a licence: applicable goals withheld, over 256 combinations of the eight map facts (analysis/manual_licence.py).

code	deck	combinations withholding a goal	goal withheld
`90308cd`, before goal-aware retrieval	manual	128 of 256	FLEE
`7acb344`, goal-aware retrieval	manual	0 of 256	none
`6fc5c72`, this paper	manual	0 of 256	none
`6fc5c72`, this paper	both books	0 of 256	none

4.2 As advice

Inside the discretionary tier the cards in force also vote: each card votes for the goals it speaks to, weighted by how well it matched the situation, and the rules take the highest-voted goal first, the fixed order breaking ties. To know whether this mattered, the pilot computed every decision twice, with the votes and without, and counted the decisions in which the two choices differed. In a live run on Shady Sands with the manual as its only book, the counter read **0 of 2,344**: the manual's votes never changed the goal chosen. The read-out ("decisions 2344 | steered 0") was taken from the console at 15:46 EDT on 26 September and is recorded in the lane's development log and in the commit that added the walkthrough (`27fc090`). The per-decision log began two hours later, so this count cannot be re-derived from it; the counter's code is in the pinned history (`73c9bd4`, the last commit before the read-out).

4.3 Why

Both results follow from what the book is about. Table 2 and Figure 2 count, for each book, the cards that name each of the walkthrough's twelve place chapters, each of its 22 quests, and each of the 17 single-word names of people the pilot sent talk orders to in its logged runs (the engine's names, taken from the log). The manual names two places: the home vault, in its premise and hints, and Vault 15, the first mission's destination, in its premise and in a world-map example. It names no quest. Of the 17 people, it names none, except two that appear only on its voice-credits page beside their actors. (Its premise and hints do name one character, the home vault's Overseer, to whom the pilot sent no talk order in the logged runs.) At the moment of the 0-of-2,344 read-out, the "Shady Sands" cards the manual contributed were two instructions about the town-map button. A card can only license or vote for what its words speak to, and the manual speaks to every kind of action in general and to no place, quest or person in particular; so it withholds nothing and prefers nothing.

Table 2. What each book names (analysis/books.py over the two decks, read-only). A place counts when at least one card names it; for the manual, mentions on the credits pages (pp. 117–118) are counted apart.

	manual	walkthrough
cards	2,250	2,289
places with a walkthrough chapter (12) named	2 (home vault: 5 cards; Vault 15: 2)	12
quests the walkthrough lists (22) named	0	22
people the pilot talked to (17 names) named	0 (+2 on the credits page only)	11

This is a statement about this manual and this architecture, not about manuals. The manual of Civilization II gave strategic advice that a learned agent could use to win more often [2]; Fallout's manual describes the interface, the character system and the rules of combat, and leaves the world to the player.

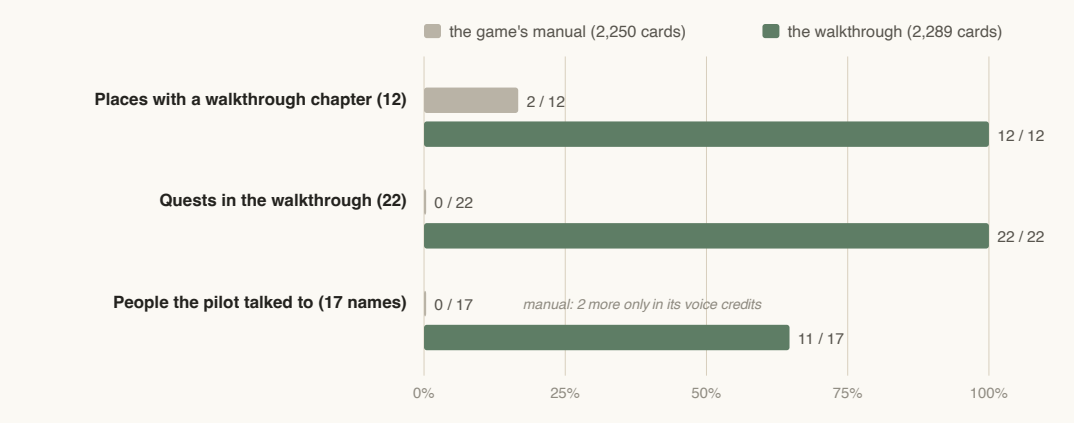

Figure 2. Coverage of the game's world by each book. The walkthrough's quests are its own list, so it names all 22 by construction; the comparison that matters is the manual's zero.

5 The walkthrough: quests, people, and where a line leads

The walkthrough is *The Nearly Ultimate Fallout Guide*, version 1.1, a 72-page community guide [8]. It was compiled into 2,289 cards with page receipts, the same way as the manual. It has twelve area chapters, one per place, and lists 22 quests in eight of them; its character-design part advises which skills to raise and how far, and sorts every perk into five tiers, which the pilot uses when it levels up (§8). The walkthrough's cards license goals by their words exactly as the manual's do, and Table 1 shows that loading them withholds nothing more. Its effect comes through three other channels.

The quest on the agenda. In each area the pilot is about one quest: the first one the game's own quest list shows as given and not done, or else the first in the guide's order not yet given, skipping any quest set aside. The agenda's words, less any word that names another quest, mark the lines of dialogue that belong to it.

Whom to approach. The people present whom the area's chapter names are approached first, in the guide's order, the agenda's people ahead of the rest. Only proper names count: the guide writes roles such as a town's guards or children in lower case, and the chapter's epigraph is a quotation, not advice. A person the guide names is worth any walk; anyone else is approached only within 18 hexes of path.

Where a line leads. What a line of dialogue leads to is learned, not read: the speech memory records, per counterpart, the map a line took the pilot to. With a quest on the agenda, a line known to lead to the quest's place is said; a line known to lead to the place of a quest set aside is not, while any other safe line exists.

5.1 The caves: a quest as a goal, and a line that kept leading back

Shady Sands' second quest sends the player to clear a cave of radscorpions, and the walkthrough names the person who takes you there (p. 19). With the quest on the agenda, the pilot went, and at level 1 it died in the caves in three consecutive console runs: 4 times in 13.4 minutes and 8 times in 12.9 minutes before the quest memory existed, and 10 times in 16.5 minutes in the first run with it (every death in the caves; analysis/defects.py). The quest memory did set the quest aside after three deaths, but the pilot kept going back: the other Shady Sands quest's text mentions radscorpions, so the line that takes the pilot to the caves read as a line of that quest, and following through meant saying it. The fix (`49fb85b`) is the per-counterpart destination above, and a rule that a word naming another quest belongs to that quest; the commit records

five further deaths in the caves after the set-aside before it. The next two runs, which started from the same save because the pilot did not yet save its progress, had no deaths in 8.8 and 3.6 minutes and gained 250 experience each. These are single, short runs on changing code; they show the mechanism working, not its size. Read for this paper, the speech memory still holds the line: said to that counterpart, "Yes." led to the caves 32 times.

5.2 The conversation that made level 2

In the live run that reached level 2 (§6, Figure 3), at 20:39:54 the pilot was exploring the east half of Shady Sands with nobody new within reach to talk to: the goals on offer were leave, explore and wait. When a person 23 hexes away entered its report, it stopped exploring and walked over. The rules offer a conversation with someone the book does not name only within 18 hexes of path; this person is named in the walkthrough's Shady Sands chapter (p. 18). Seven seconds later, as the conversation closed, experience rose from 700 to 1,200 and the character passed the engine's 1,000-point threshold for level 2. Without the walkthrough the same code would have gone on exploring; whether it would have met him later, and what that would have been worth, we cannot say from one run.

5.3 How much of the talking the walkthrough steered

In the archived decision logs (50,376 decisions from 17:38 on 26 September to 00:40 on 27 September, all with the walkthrough loaded), the pilot sent 1,035 talk orders to 26 counterparts. 207 of them (20%) went to the 11 people the walkthrough names; 730 (71%) went to people the engine names only by role, such as a town's children, guards, citizens and peasants; the remaining 98 went to single-named characters the guide does not use. The guide order is a preference, not a licence: when nobody it names is in reach, the pilot talks to whoever is near. Section 10 shows what that cost over three hours.

What is not measured. There is no A/B with the walkthrough removed. The pilot counted the decisions the walkthrough changed (a different goal or a different person, with the page that did it) on its console, but that counter was not written to disk, so we report it only through the examples above and the talk orders.

6 Saving progress like a player, without the player's saves

Until the evening of 26 September every death reloaded the slot the pilot was started on, so a life's experience, quests and maps were lost with it, and a quest "set aside until the pilot is stronger" could never come back: the pilot never got stronger. Saving progress fixed that, and made it necessary to say exactly what a learner that saves may write.

6.1 The rule

When. Only at a quiet moment (on the map; no fight, conversation, walk, hostile in view, wound, open screen or person at the controls), only when settled (the engine's path probes reach somewhere from here and no order to move has failed on this hex), only with progress since the last save (experience gained, a map the last save lacks, the quest list changed, a level spent, or 600 decisions, about five minutes, since the last save), and never within 90 decisions (about 45 s) of the last save. A save counts only when the engine's own save counter moves.

Where. The pilot's scratch slots 9 and 10, in turn, an abandoned slot first, so the only good save is never overwritten.

After a death. The engine loads the newest save by itself. A death within 80 decisions of coming back counts against that save; after two, the save is abandoned for the one before it, and past that for the slot the game was started on.

Stuck. Trying to move and not moving for 150 decisions, the pilot loads its last save on purpose; the engine counts reloads asked for, so a deliberate reload is never taken for a death. If it was stuck from the moment a save was loaded, that save is the trap and is abandoned.

Restart. The console resumes from the newest good save of the same game.

6.2 The property, and what pins it

Proposition 1 (the player's saves are out of the learner's reach)**.** *For any memory state and any sequence of decisions, no command the pilot sends makes the engine write a saved game into slots 1–8, and the pilot's engine never writes the player's game directory.*

Argument. (i) The engine runs on a staged copy of the game directory (§2). (ii) The pilot's only save command is refused in the engine unless its slot is 9 or 10, twice: by the command handler and by the save function it calls; the handler also refuses in combat, in a conversation, while moving and with no living player. (iii) The game's quick-save key writes the slot last loaded or saved; the tap forgets that slot after each of its loads and saves, so the key opens the save screen instead. (iv) The pilot sends no clicks, and its only keys are the dialogue digits, one key that leaves a trade screen, and Escape; its key table holds none of the keys with which the game saves (F4 or Ctrl+S for the save screen, F6 for a quick save, S in the options menu), the tap's key command takes no modifier, and its reload commands only choose which save a death loads. Each of (ii)–(iv) is pinned by a check that reads the committed patch and the pinned pilot code (analysis/pins.py, checks S1–S6), and the lane's own tests pin the pilot side (no command names a player's slot; the staging never deletes the pilot's slots). ■

This is least privilege in the sense of Saltzer and Schroeder [15]: the component that learns from its own deaths holds the one write it needs and no other. Two qualifications keep it honest. First, the game's own save screen is not guarded by the tap. It is one menu item from the options menu, and the pilot did open that menu once, with an Escape that landed after a conversation had closed (Figure 5); no input the pilot can express reaches a save from there (argument (iv)), but that is a property of the pilot's code, not a guard in the engine. Second, the property was not always true. Before commit `fc2ef32` the tap's start-up load left the quick-save slot armed, so a person pressing the quick-save key while the pilot played would have written over the loaded slot, which is the staged copy of the player's save. An automated code review found it (§9).

6.3 Live: ten deaths, each returned to a save

The console run that reached level 2 lasted 55.4 minutes (20:09:50 to 21:05:14, 6,924 decisions) and resumed from the pilot's own save. It saved 10 times into slots 9 and 10, in turn except once, when the save after an abandonment went into the abandoned slot as the rule requires; each save was confirmed by the engine's counter. It died 10 times, all before level 2, between minutes 10.4 and 26.2. Table 3 lists them: every death loaded the newest save that had not been abandoned. One save was abandoned by rule. On arriving in a desert encounter at minute 23.2 the pilot saved ("a new map"); at the very next decision a fight with two radscorpions began. That save killed it twice, 67 and 69 decisions after the previous load, and was abandoned at minute 24.0; the next death on the same map went back to the save before it, taken in the mountains. Saving on arrival in a dangerous map is therefore not free: this one cost two deaths before the rule caught it, and a third on the same map (§10).

Table 3. The ten deaths of the run that reached level 2 (analysis/journey.py over the pilot's decision log). Map names are the engine's: MOUNTN and DESERT are random-encounter maps, RAIDERS the raiders' camp. "Newest good" is the newest save not abandoned at that moment; the save in slot 9 taken on arrival in DESERT1 was abandoned after deaths 6 and 7.

death	minute	died on	came back to	slot	newest good
1	10.4	MOUNTN2	DESERT2	9	yes
2	12.2	MOUNTN2	DESERT2	9	yes
3	14.5	MOUNTN2	DESERT2	9	yes
4	20.9	MOUNTN2	MOUNTN2	9	yes
5	23.1	MOUNTN1	MOUNTN2	10	yes
6	23.5	DESERT1	DESERT1	9	yes
7	24.0	DESERT1	DESERT1	9	yes
8	24.5	DESERT1	MOUNTN2	10	yes
9	25.1	DESERT2	MOUNTN2	10	yes
10	26.2	RAIDERS	MOUNTN2	10	yes

7 Learning on against learning off

7.1 Design

The lane's A/B harness starts two arms from the same saved game with the engine's dice fixed at every load (seed 1234), each on a hidden engine for 12 minutes of wall-clock time: one with learning on, one with learning off (the evidence, speech and quest memories neither consulted nor written). Each arm plays a private copy of the game directory and works on its own copies of the memories: a snapshot of what the live pilot had learned so far ("live memory"), or nothing ("empty memory"). Both arms save their progress; both read both books. The two arms of a run go one after the other in one process. The harness counts, from the pilot's session, decisions, deaths, minutes to the first death, maps entered, experience gained, saves and the goals chosen. We have two recorded runs, one of each kind (Table 4, Figure 4).

Table 4. The two recorded A/B runs, one run per arm (the harness's result files, maps/measure-20260926-194427.json and -202342.json, and the arms' own records). Deaths per hour are as the harness computed them (29.99 and 79.99 in the empty-memory file, shown rounded). Deaths are as recorded and are lower bounds (§7.3).

	live memory		empty memory	
	on	off	on	off
arm started (26 Sept, EDT)	19:20:13	19:32:17	19:59:33	20:11:38
minutes	12.0	12.0	12.0	12.0
decisions	1,619	1,275	1,774	1,726
deaths	**3**	**16**	**6**	**16**
deaths per hour	15.0	80.0	30.0	80.0
minutes to first death	8.63	0.96	0.75	0.84
experience gained	+250	0	0	0
maps entered	3	1	2	1
saves	2	1	2	2
caves quest set aside	at start (snapshot)	not consulted	learned in the run	no

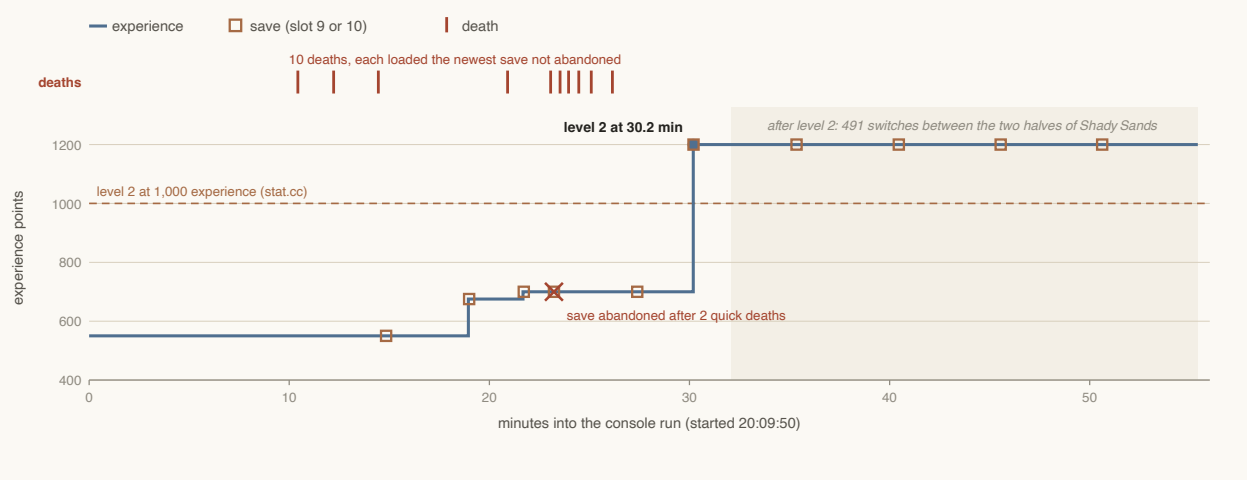

Figure 3. The live run that reached level 2 (one run). Experience 550 at the start; 675 and 700 after fights in a mountain encounter; 1,200 at minute 30.2, as a conversation the walkthrough made worth the walk closed (§5). Ten deaths (ticks), each returned to the newest save not abandoned; squares are saves; the cross is the save abandoned after two quick deaths. The shaded span after level 2 is the defect of §9: 491 switches between the two halves of the town in 23 minutes.

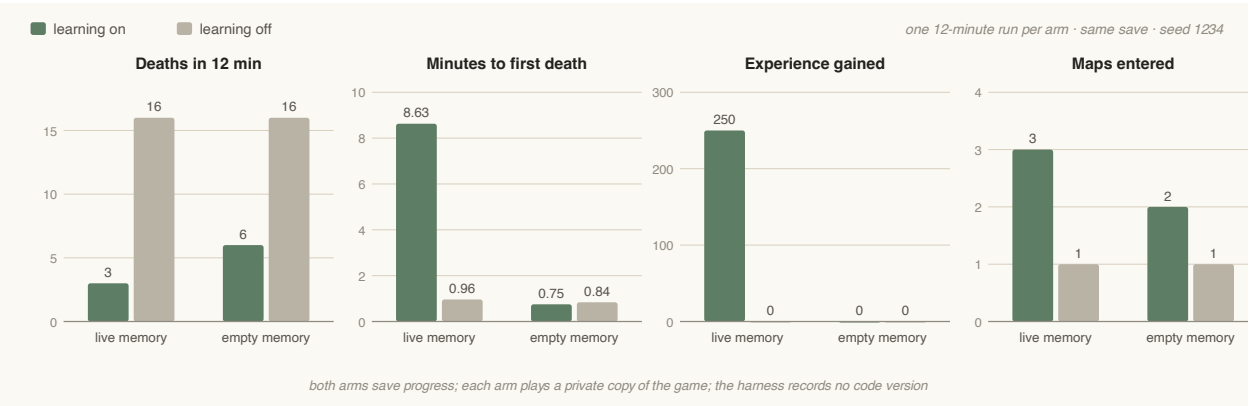

Figure 4. The A/B of Table 4. One 12-minute run per arm, same save, seed 1234; both arms save progress, each on a private copy of the game. Deaths are recorded counts and lower bounds (§7.3).

7.2 Results

With the live memory, the arm with learning on died 3 times to the other's 16, died first after 8.6 minutes instead of 1.0, gained 250 experience to none, and entered three maps (the town, the raiders' camp and a desert encounter) where the other never left its first map. It began the run with the live pilot's memories: the caves quest set aside at level 1, the recorded outcomes of 1,783 lines said before, and one line to one counterpart known to lead to the caves. From empty memory the first deaths came at about the same time, 0.75 against 0.84 minutes, as they must when nothing has been learned yet. Over the 12 minutes the arm with learning on died 6 times to 16: it said 26 lines, learned that one of them leads to the caves, had three of its deaths charged to the caves quest and set that quest aside within the run (its quest record at the end says so), and spent more of its decisions fleeing than on any other goal (679 of 1,774). It gained no experience. In both runs the arm with learning off stayed on the first map and spent its time in fights and conversations: apart from waiting, its most frequent goals were arming, ending turns, fighting, healing and replying.

7.3 What this does and does not show

The direction is the same in both runs and the deaths differ by a factor of 2.7 to 5.3, but each arm is one run. The arms ran twelve minutes apart on a shared machine; the dice are fixed at every load but the pilot's timing is not, so the arms diverge within the first seconds, and they made different numbers of decisions in the same wall-clock time (1,619 against 1,275). The harness records no code version; by start time the runs used the tree at commits `2baf007` and `946fd2b`. Learning off leaves the rules, the feasibility checks, the stuck rule and the saves in place, so the comparison isolates the three memories, not "learning" in any wider sense. The count of hexes known, which the harness also reports, is not comparable across modes because the live-memory arms start from a copy of the pilot's map memory; we leave it out of Table 4 and list it in the appendix.

Two harness defects found later. Both runs predate two fixes. Until `4188965` (19:39), a reload the pilot asked for when stuck for good, if the engine did not take it, let the next death pass for that reload, uncounted; the empty-memory run has that fix, but both runs predate `fc2ef32`, after which the engine itself counts the reloads the pilot asks for. The harness kept no load counter, so the death counts of Table 4 are lower bounds: deaths can have been missed, not added. The shortfall is bounded. The stuck rule allows a first deliberate reload only after 150 decisions and then at most one per 300, and each can hide at most one death, so the arms with learning on died at most 8 (live memory) and 12 (empty memory) times, against at least 16 in each arm with learning off (analysis/accounting.py). In every archived decision log, by contrast, the engine's loads equal the deaths plus the deliberate reloads, so every other death count in this paper is exact. The second defect: a scratch directory given to the harness a second time would not have started each arm fresh (fixed in `fc2ef32`, with a guard added in `a31fe6e`); each recorded run was given a scratch directory created at its own start, so neither reused one.

The excluded run. The first A/B, started at 18:40 the same day, ran beside the live console before arms had private game copies. Its arm with learning on shared the game's world files with the live pilot, which was playing in the same directory; two engines on one directory overwrite each other's world state. No result was written (the log holds only the start of that arm), and the commit that introduced private copies (`dee8169`) declares the run void. It is excluded here, and no number from it is used.

8 Growth: claiming a level the way the character screen does

8.1 Where a level's growth is granted

Reaching an experience threshold raises the character's level at once, but in this engine the skill points and perk that come with the level are granted only when the character screen is opened: the accounting runs in `UpdateLevel()` (`editor.cc`, line 5199 at the pinned commit), which the screen calls when it starts outside character creation. For each level gained it adds $5 + 2 \cdot \mathrm{IN} + 2 \cdot \mathrm{Educated}$ unspent skill points (minus 5 with the Gifted trait, never below 0, at most 99 held), and every third level (every fourth with the Skilled trait) it makes a perk due while fewer than seven are held. The pilot had never opened the screen, so at level 2 its level indicator stayed lit and the level brought it hit points and nothing else.

8.2 The tap's level-up commands

The patch moves that accounting into two functions shared by the screen and the tap (one claims the levels gained, one applies a perk's side effects), so the two cannot grant different amounts. Three tap commands go through them: claim the levels gained, raise one skill by one point, and take one perk; the two perks that need a second choice on the screen itself are refused. A read-out reports the indicator, the level claimed, the unspent points, every skill, the tagged skills, the skill of the weapon in hand, the perks held and, while

one is due, those on offer. On a private copy of the game the commit that added them (`ad8d6ea`) verified that a claim grants what the screen grants, that the real screen opened after a tap claim grants nothing a second time, and that invalid skills and perks are refused.

8.3 The pilot's choices

At a quiet moment the pilot claims a gained level; a later quiet moment spends it. Skill points go to what the pilot uses: first the skill of the weapon in hand, then the tagged skills, lowest first, up to the walkthrough's marks ("around 100% early on", then the weapon skill to 150%, p. 7). First Aid gets nothing: the pilot heals with stimpaks and never uses the skill, and the walkthrough lists it among the skills not worth points (p. 7). The perk is the best-tiered of those on offer in the walkthrough's five tiers (pp. 8–9). With no walkthrough loaded no perk is taken and it stays due. A step counts only when the engine's read-out shows it; a level spent is progress for the next save.

8.4 Live

At 22:24:57 on 26 September, in the first second of a console run started with the level-up code (`ad8d6ea`), the pilot claimed level 2 and spent its points on Melee Weapons, 52% to 72% (the weapon in hand was a knife); at 22:24:58 it saved into slot 9 with the reason "a level spent", a reason the pilot gives only after the engine has shown the step. No perk was due at level 2.

8.5 A defect found by review: a cap the engine never reports

The first version of the level-up code copied the engine's skill cap as 300, citing a header. The engine's cap is 200 (`SKILL_LEVEL_MAX`, `skill.cc` line 31): skills clamp to it and a point at the cap is refused. At 200 the pilot's plan would have stayed alive, every point would have been refused, and the same batch would have been re-sent every 20 or so decisions indefinitely. The tests could not catch it, because they used 300 as the value "at the cap", which the engine can never report. A separate automated review of that commit found it before any skill the pilot raised came near the cap (the one it was raising stood at 72%). The fix (`6fc5c72`) sets 200, stops re-sending a step the engine refused while nothing has changed, and adds a test that reads `SKILL_LEVEL_MAX`, the two hand-to-hand skill indices and the two refused perk indices back from the C++ sources, so a mistyped literal fails the suite. We re-ran that suite at the pinned commit with the engine sources exported at the engine pin: 461 passed and 1 strict expected failure (§9).

9 Method: the defects the records exposed

Every result above was reached through defects, and the defects are part of the method: each was found in the pilot's own records, by an exhaustive check or by an automated code review, and fixed in a commit that says what was wrong, nearly always with a test that pins the fix. Table 5 lists those that bear on this paper's claims, with the count that exposed each, recomputed from the archived logs where one exists. Two of the logged counts are larger than the commit messages say, because each message was written while its run was still going; the logs win, and Appendix A notes both.

The reviews. The lane's code was written with an AI coding assistant, and every review recorded in its history is an automated, read-only review; no human code review is recorded. A separate AI coding tool reviewed three ranges: `9949c65..73c9bd4` (seven findings; six fixed in `401b168`, one declined by design), a second pass over those fixes and the commits after them (one more finding, fixed in `2c5e584`), and `2c5e584..41919c8` (12 findings; 11 fixed in `fc2ef32` and 1 contested: the speech memory compares an attacker by name on purpose, because a threat to one of a town's guards turns all of them, and a test now pins that choice). The fixed findings of that last review include the armed quick-save slot of §6, a failed save that opened the engine's modal error box where no pilot could see it, a real death that could pass for a reload the pilot had asked for (the engine now counts those), and a save file that no longer existed but

Table 5. Defects found and fixed (the lane's commit messages; counts recomputed read-only from the archived decision logs by analysis/defects.py, analysis/journey.py and analysis/stall.py).

defect	found by	the record	fix
The manual never licensed running from a fight: its run-away page ranked 16th, six pages in force	exhaustive check	FLEE withheld in 128 of 256 fact combinations	`7acb344`
A line's outcome read one tick early; blame only on the last six decisions of a fight	the pilot's records	49 deaths at one guard (see [19])	`7d52323`, `73c9bd4`
A set-aside quest re-entered through a line that read as another quest's	the pilot's records	10 deaths in the caves in 16.5 min (the commit counts five after the set-aside)	`49fb85b`
Two engines on one game directory	the first A/B	the run is void (§7)	`dee8169`
A deliberate reload the engine did not take let the next death pass uncounted	inline check	bounds the A/B death counts (§7.3)	`4188965`, `fc2ef32`
Stuck at the map's rim; the save taken there became the return point	the pilot's records	two stuck-for-good reloads, both back into the same save	`450b325`, `2baf007`
A far walk that went nowhere sealed that exit for good	the pilot's records	2,736 decisions on one map, experience unchanged (28.2 min)	`946fd2b`
The pilot's own late Escape opened the options menu, read as a person at the controls	the pilot's records	1,642 consecutive decisions held (Figure 5)	`7056bda`
Back and forth between the two halves of a walked-out town	the pilot's records	491 switches in 23 min after level 2; 54 in the next 3.08 h	`31dad05`, `bb3abb5`
Quick-save slot left armed after the tap's load, and 10 more	automated review	12 findings: 11 fixed, 1 contested	`fc2ef32`
Skill cap copied as 300; the engine's is 200	automated review	tests used an unreachable value	`6fc5c72`
Upstream engine: an uninitialised read in the combat AI, turned by the optimiser into a null dereference	a crash	pinned by the lane's patch tests	in the patch

was still believed to be the newest. While that tool was at its usage limit, a separate review agent of the assistant that wrote the code reviewed the level-up commit and found the skill cap of §8, and the writing session twice checked its own recent work inline (`4188965`, `a31fe6e`), finding among other things the death-count case of §7.3 and an unsafe clean-up in the A/B harness. A review of the later fix commits by the external tool was started before the cutoff and failed at once on its usage limit; it is still owed. The fixes carry tests and mutation checks, recorded in their commit messages (14 of 14 mutants caught in `fc2ef32`, 47 of 47 in `6fc5c72`).

The suite, re-run. We re-ran the lane's own test suite at four commits from exported copies, with no engine and the engine sources the constants test reads exported at the engine pin (Table 6). Each run matches the count its commit message states, where it states one. The one expected failure is kept on purpose: having died only while unarmed, the widest pattern condemns fighting while armed, which was never tried; the remedy, exploring condemned goals in contexts never tried, is not built [19].

An engine defect upstream. In the pinned upstream engine, a combat-AI helper clears three output pointers but tests the third by reading through it, and its only caller passes a slot of an uninitialised local array. That read is undefined behaviour. The lane's patch comment records what the compiler made of it: with the helper inlined, the optimiser removed a null check, so any critter joining a fight before it had been hit dereferenced a null pointer and the engine crashed. The patch tests the pointer instead, as the two lines above it already do, and a test pins all three. We confirmed the upstream source reads through

Table 6. The lane's test suite re-run from exported copies (analysis/run_suite.sh, 27 September).

commit	what it added	passed	strict expected failure
`fc2ef32`	the external review's fixes	413	1
`bb3abb5`	no ping-pong between walked-out maps	422	1
`ad8d6ea`	level-ups through the character screen's accounting	456	1
`6fc5c72`	the engine's real skill cap; the code of this paper	461	1

Figure 5. Research screenshot of the game: a 640×480 frame the instrumented engine wrote at 20:00 on 26 September. In a desert encounter the pilot's own Escape, meant for a conversation that had already closed, opened the game's options menu; the pilot read the open screen as a person at the controls and held for 1,642 decisions (Table 5). The menu's first item is the game's own save screen, which the tap does not guard (§6). Game content © its rights holders; reproduced for research.

the pointer and passes the uninitialised slot (analysis/pins.py, check E1); we did not rebuild the engine to confirm the machine code.

10 Open problems and limits

Status. Research prototype, in a game. The lane labels itself a prototype: no deploy path beyond one machine and no alerting. Nothing here is a certified safety system, and a game that reloads on death is far more forgiving than any physical system. Every number is ours, from the pilot's own records, and none has been replicated by a third party.

10.1 Three hours without experience

After level 2 the console ran for 3.08 hours (21:35 to 00:40, 21,833 decisions) across three console runs on commits `bb3abb5`, `ad8d6ea` and `6fc5c72`; the last 100.6 minutes ran exactly the code of this paper. Experience was 1,200 at the first decision and at the last (Figure 6). 90.7% of the decisions were on the three Shady Sands maps. The pilot loaded its last save because it was stuck for good 31 times (10, 7 and 14 in the three runs). It died 7 times, all in the last run: five times in the caves at level 2, where the caves quest was set aside again after three deaths and two more deaths followed there, since both of its latest saves had been

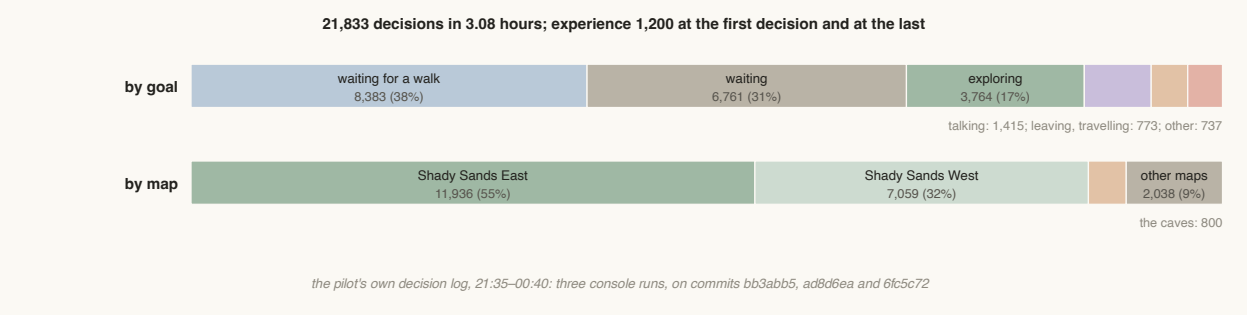

Figure 6. Three hours after level 2 (one console session, three runs). Top: decisions by goal. Bottom: decisions by map. Experience did not change.

taken in the caves and each death returned it to one of them; and twice in a desert encounter. The engine refused 572 walk orders because an exit grid was in the way and 319 orders to leave by the hex the pilot stood on. 69.6% of the decisions were WAIT: 8,383 waiting for a walk to finish and 6,761 with nothing to do. It carried no stimpak, and a knife was in its hand throughout.

The lane's own diagnosis, written after this run and committed with the work that followed it (`41d3dcc`), names three causes. The pilot was boxed in: walking onto an exit grid leaves the map, so walks whose first step crossed one were refused, and the leave goal chose the grid under its feet. It had no plan: the walkthrough was a deck of pages in force, which licensed and voted and named people in view, but nothing held a step such as "talk to this person, who reveals the next towns" until the game said it was done; so the pilot talked to whoever was near (across all the logged runs, 71% of its talk orders went to people named only by role; §5). And its fights were weak. The work that followed, a campaign layer that holds the walkthrough's instructions as steps closed by the game's own facts, walks that route around exit grids, and weapons chosen by the engine's hit numbers, was committed after this paper's cutoff and is not evaluated here. The result stands as it is: *a walkthrough read page by page is not yet a plan.*

10.2 Open problems

Weapon against build. The pilot fought with a knife and spent level 2's points on Melee Weapons, while the character's tagged combat skill is Unarmed; the lane's later diagnosis measured the engine's chance to hit one target at 42% with the knife and 72% with brass knuckles carried in the pack. Choosing weapons from the engine's own numbers came after the cutoff.

Combat tactics. The caves quest defeated the pilot at level 1 (22 deaths in the caves over three runs) and again at level 2 (five). With the knife, the engine's chance to hit a radscorpion, as the log records it, ranged from 30% to 62%. The rules fight by action-point arithmetic, one attack a turn and a stimpak when low; they do not position, retreat to cover or use companions.

Saving on arrival. A save taken on entering a map where a fight is about to begin becomes a return point into that fight: two deaths in the level-2 run before the rule abandoned it (§6), and in the three-hour run two saves in the caves (one on arrival, one two seconds after the first death there) and five deaths in them.

Containers and trade. The tap has no command to search a container, and the pilot leaves every trade screen; neither is driven.

An unexplained crash. The lane's notes list a crash in a fight on a fresh game as unexplained. No crash report was retained for it, and this paper, which runs no engine, could not examine it.

Over-generalisation. The strict expected failure of §9 stays in the suite until exploration of condemned

goals is built.

The game's own save screen. It is not guarded by the tap (§6); a guard there, or write-protecting the player's slots in the pilot's copy, would close the residual of Proposition 1.

The walkthrough's size of effect. No A/B removes the walkthrough, and its change counter was not kept.

10.3 Threats to validity

One save, one character and one engine version; the results may depend on the character's build and on where the save stands. The A/B is one run per arm, on wall-clock time, with the code version inferred from timestamps, and its death counts are lower bounds (§7.3). The archived runs are not a controlled series: most are separated by a commit, and each describes the code of its moment. The same group built the pilot, ran it and wrote this paper, and the records are the pilot's own, written by code that was itself changing; two counts in commit messages were smaller than the logs show, and we corrected them. The code was written with an AI coding assistant and reviewed only by automated reviewers (§9). The 0-of-2,344 count exists only as a console read-out, in the development log and in a commit message. All of it is a game.

10.4 What this paper does not publish

How the two documents were compiled into cards, how contradictions between sources are checked, how provenance is stored and what the admission gate checks internally are withheld. The paper states their interfaces and measured behaviour: cards with page receipts, licensing by a card's own words, and the counts reported above.

11 Related work

Documents that help agents play. Branavan, Silver and Barzilay grounded the text of the Civilization II manual in a Monte-Carlo search agent and reported a 34% absolute improvement, winning over 65% of games against the built-in AI [2]. Wu et al. extracted auxiliary rewards from Atari instruction manuals to speed up reinforcement learning [17], and SPRING had a language model read the paper that introduced an open-world game and reason over it to play [18]; AutoManual has language-model agents write their own manual of rules through interaction [3]. In all of these the document shapes a learned policy or a prompt. Here a document can only license the goals its own words speak to and reorder a small discretionary tier, and its effect is measured by counting decisions, which is how we can say that a manual changed none. That Civilization's manual helped and Fallout's did not is consistent with what each is about: strategy in one, operation in the other.

Long games as testbeds. NetHack [9] and the BALROG benchmark [13] use long, stochastic games to test exploration and planning, and Voyager [16] lets a language-model agent grow a skill library in Minecraft with an automatic curriculum. Fallout adds dialogue with consequences and a quest graph whose state is mostly out of view. We make no score comparison with any of them; the pilot plays one save and reached level 2 once.

Returning to a saved state. Go-Explore returns to promising states before exploring from them [4]. The pilot's saves are closer to a player's: taken only at quiet moments with something gained, confirmed by the engine, alternated between two slots, and abandoned when they keep killing the pilot. The design question they raise is not exploration but privilege: what a learner that saves may write.

Least privilege and shields. Saltzer and Schroeder's principle of least privilege [15] is the frame for Proposition 1. Removing unsafe actions before a learner chooses is shielding [1]; the relation between the admissible set, orders and the learner is set out in [23, 19].

From pages to plans. The three-hour run shows that a walkthrough's instructions need to be held as steps until the game says they are done. Hierarchical task networks decompose tasks with methods written by people [11]; belief–desire–intention agents hold an intention until it succeeds or fails [14]; goal-oriented action planning brought planning over preconditions and effects into commercial game AI [12]; Soar handles impasses by creating subgoals [10]. A walkthrough is, in effect, a method library written by a human; the campaign layer committed after our cutoff takes that view, and its evaluation is future work.

Our own earlier papers. The runtime and its proofs are in [23]; the fail-first learning loop and the credit-assignment repair it needed in this game are in [19]; rule pilots against language-model drivers in two tank games are in [21]; negative results on training grounding into small models are in [22]; an eye that checks what a game engine claims against the pixels, with a null result where it matters most, is in [24]; the case for keeping known facts in retrieval rather than in weights is in [20].

12 Conclusion

A game's manual, compiled into sourced cards and made the only source of a rule pilot's licence, licensed every goal the situation offered and changed none of 2,344 live decisions, because it speaks to how the game is operated and not to anything in it. A walkthrough changed what the pilot did: which quest it pursued, whom it walked to, and, with the pilot's own memory of where each line leads, what it said. The conversation that took the character to level 2 was one only the walkthrough made worth the walk. Saving like a player let the pilot keep what a life had gained, under a rule that gives the learner two scratch slots and the engine's refusal of every other; learning on against off, on the same save and dice, died 3 times to 16 and 6 to 16 as recorded (at worst 8 and 12 to at least 16), one run per arm. Growth came through the character screen's own accounting, and a skill cap copied wrong was caught by an automated review and pinned against the engine's source. Then three hours at the same experience showed the next limit: pages in force are not a plan. Each of these results is small, and each is traceable to a file.

References

[1] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, U. Topcu. Safe reinforcement learning via shielding. In *Proc. AAAI Conference on Artificial Intelligence*, 2018.

[2] S. R. K. Branavan, D. Silver, R. Barzilay. Learning to win by reading manuals in a Monte-Carlo framework. *Journal of Artificial Intelligence Research* 43:661–704, 2012. doi:10.1613/jair.3484.

[3] M. Chen, Y. Li, Y. Yang, S. Yu, B. Lin, X. He. AutoManual: constructing instruction manuals by LLM agents via interactive environmental learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.16247.

[4] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, J. Clune. First return, then explore. *Nature* 590(7847):580–586, 2021.

[5] Interplay Productions. *Fallout: A Post Nuclear Role Playing Game*. Computer game, 1997.

[6] Interplay Productions. *Fallout* game manual, printed with the game, 1997 (121 pages in the scanned edition used here).

[7] Fallout Community Edition contributors. *fallout1-ce*: a re-implementation of the Fallout engine. github.com/alexbatalov/fallout1-ce; used at commit `0609bcf` (15 January 2025) with an instrumentation patch.

[8] P. Jorner. *The Nearly Ultimate Fallout Guide*, version 1.1. Community walkthrough, 72 pages (PDF edition, 2017).

[9] H. Küttler, N. Nardelli, A. H. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, T. Rocktäschel. The NetHack Learning Environment. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. arXiv:2006.13760.

[10] J. E. Laird, A. Newell, P. S. Rosenbloom. SOAR: an architecture for general intelligence. *Artificial Intelligence* 33(1):1–64, 1987.

[11] D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, F. Yaman. SHOP2: an HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404, 2003.

[12] J. Orkin. Three states and a plan: the A.I. of F.E.A.R. Talk, Game Developers Conference, 2006.

[13] D. Paglieri et al. BALROG: benchmarking agentic LLM and VLM reasoning on games. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2411.13543.

[14] A. S. Rao, M. P. Georgeff. BDI agents: from theory to practice. In *Proc. First International Conference on Multi-Agent Systems (ICMAS-95)*, AAAI Press, 1995.

[15] J. H. Saltzer, M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE* 63(9):1278–1308, 1975.

[16] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, A. Anandkumar. Voyager: an open-ended embodied agent with large language models. arXiv:2305.16291, 2023.

[17] Y. Wu, Y. Fan, P. P. Liang, A. Azaria, Y. Li, T. M. Mitchell. Read and reap the rewards: learning to play Atari with the help of instruction manuals. arXiv:2302.04449, 2023.

[18] Y. Wu, S. Prabhumoye, S. Y. Min, Y. Bisk, R. Salakhutdinov, A. Azaria, T. Mitchell, Y. Li. SPRING: studying the paper and reasoning to play games. arXiv:2305.15486, 2023.

[19] Perslis Research. Fail-First Models: failure becomes structure, structure changes the next attempt. 2026. research.perslis.com/fail-first

[20] Perslis Research. Inference Placement: where learned inference earns authority in a symbolic system. 2026. research.perslis.com/inference-placement

[21] Perslis Research. Rules at the Wheel: tanks, language models and an honest loss. 2026. research.perslis.com/tank-arena

[22] Perslis Research. We Tried to Train It In: negative results on teaching small models to ground facts. 2026. research.perslis.com/training-grounding

[23] Perslis Research. VDSG: a commanded admission-control runtime for autonomous agents. 2026. research.perslis.com/vdsg

[24] Perslis Research. A Witness, Not a Detector: checking what a game engine claims against the pixels. 2026. research.perslis.com/witness-eye

A Where each number comes from

Every number in this paper is read from the lane's own files or recomputed from them by a read-only script in this paper's `analysis/` directory; `CLAIMS.md` maps each claim to its file and line. Paths are relative to the lane (`fallout-floor/`) unless marked `analysis/`.

result	source
2,250 and 2,289 cards; places, quests and people named; talk orders	the two card decks, opened read-only; the pinned deck code; the archived decision logs (`analysis/books.py`)
Licence test over 256 fact combinations	pinned code at `90308cd`, `7acb344`, `6fc5c72` (`analysis/manual_licence.py`)
0 of 2,344 decisions steered	console read-out of 26 Sept 15:46 EDT in the lane's development log (copied to `analysis/evidence/`); commit `27fc090`; counter code at `73c9bd4`
The caves runs; the options-menu hold; the town-bound run; the resume trap	`maps/decisions.*.jsonl` (`analysis/defects.py`)
The level-2 run: experience, saves, deaths, the abandoned save, the switches	`maps/decisions.journey1.jsonl` (`analysis/journey.py`)
The three hours after level 2; the live level-up claim	`maps/decisions.jsonl.1`, snapshot in `analysis/evidence/` (`analysis/stall.py`)
A/B results	`maps/measure-20260926-194427.json`, `-202342.json`; the arms' records copied to `analysis/evidence/ab-arms/` (`analysis/ab.py`)
Save refusals, quick-save slot, pilot inputs, level accounting, skill cap, the upstream read	the committed patch and pilot code at `6fc5c72`; upstream `0609bcf` (`analysis/pins.py`, all checks pass)
Death accounting in every archived log; the A/B worst case	the logs' load and death counters; the stuck rule's constants at `2baf007` and `946fd2b` (`analysis/accounting.py`)
Test counts at four commits	`analysis/run_suite.sh` over exported copies
Review findings, mutation counts, dates	the lane's commit messages (`fc2ef32`, `6fc5c72`, and each fix in Table 5)

Two counts corrected by the logs. The commit that fixed exit sealing (`946fd2b`) says the pilot spent 2,297 decisions on one map; the archived log of that run holds 2,736, all on that map. The commit that fixed the options-menu hold (`7056bda`) says the pilot held for 300 decisions; the log holds 1,642 consecutive held decisions. Both messages were written while the runs were still going; the logs are used.

A/B details not in Table 4. Hexes known at the end of each arm (the pilot's map memory; the live-memory arms start from a copy of it): live memory 26,674 with learning on and 19,619 off; empty memory 6,825 and 2,121. Goals chosen most often: live memory on, explore 582, end turn 358, wait 207, fight 164; off, arm 328, end turn 248, reply 207, wait 204; empty memory on, flee 679, end turn 315, explore 305, wait 175; off, end turn 391, arm 389, wait 246, fight 244.

Reproduction. From this paper's directory, with the lane checked out beside it and the card decks in place: `python3 -B analysis/books.py`, `manual_licence.py`, `journey.py`, `stall.py`, `ab.py`, `defects.py`, `accounting.py` and `pins.py` rewrite `analysis/out/`; `analysis/run_suite.sh <commit>` re-runs the lane's tests from an exported copy. None of them starts the game or writes to the lane.