THE SOVEREIGNTY LOOP

Falsework-Native Architecture for Autonomous Game Agents

A Governing Document for the Timmy Time Project

Every inference must train its own replacement.

Alexander Whitestone / Rockachopa

March 2026

I. The Principle

> *"Doing a task without automating it sovereignly is like writing code without executing it or testing it." — Alexander Whitestone*

Every prior research document in this project committed the same error. They cataloged tools, benchmarked models, and mapped architectures — all assuming that the expensive inference call would persist forever. A 3-second VLM analysis every 2 seconds for eternity. A Groq API call for every tactical decision. A content moderation filter on every sentence. That is not an architecture. That is a subscription.

This document establishes the governing law for all Timmy Time engineering: **every task must increase sovereignty as a default deliverable.** Not as a future goal. Not as an optimization pass. As a constraint on every commit, every function, every inference call.

The Sovereignty Loop: Discover with an expensive model. Compress the discovery into a cheap local rule. Replace the model with the rule. Measure the cost reduction. Repeat.

The falsework report named this principle. This document operationalizes it. Every component in the game-agent stack — perception, decision, narration, streaming, payments — must be designed so that its first execution is its most expensive, and every subsequent execution is cheaper or free.

II. The Three Phases of Every Inference

Every call to an LLM, VLM, or external API passes through three phases. Phase 1 is unavoidable. Phase 2 is the work. Phase 3 is the deliverable.

Phase 1: Discovery

The model sees something for the first time. Qwen3-VL analyzes a health bar. The decision LLM reasons about a fork in the road. Groq processes a complex quest description. This is the expensive phase — full inference, full latency, full cost. It is also the only phase that produces new knowledge.

Phase 2: Crystallization

The discovery is compressed into a durable, cheap artifact. The health bar becomes an OpenCV template with pixel coordinates and color thresholds. The fork in the road becomes a waypoint in the skill library. The quest description becomes a structured entry in the SQLite knowledge base. This phase requires explicit engineering — it does not happen automatically. Every function that calls a model must include a crystallization step.

Phase 3: Replacement

The crystallized artifact replaces the model call. Next time the health bar appears, OpenCV reads it in 3 milliseconds instead of 3 seconds. Next time the fork appears, the skill library returns the waypoint instantly. Next time the quest is referenced, SQLite serves the structured data without any inference. The model is no longer invoked for this pattern.

> *"The measure of progress is not features added. It is model calls eliminated."*

III. The Sovereignty Loop Applied to Every Layer

3.1 Perception: See Once, Template Forever

The first time Timmy encounters any UI element, the VLM processes the full screenshot and returns a structured description: element type, bounding box, color ranges, and semantic meaning. That description is immediately written to a templates.json file as an OpenCV matching rule.

Encounter	First Time (Discovery)	Every Time After (Replacement)
Health bar	VLM: 3–6 sec, structured JSON	OpenCV color threshold: 3 ms
Dialog box	VLM: 3–6 sec, OCR + layout	Region crop + Tesseract: 50 ms
Inventory screen	VLM: 4–8 sec, item grid parse	Template match + grid math: 10 ms
Minimap	VLM: 3–5 sec, orientation analysis	Pixel sample at known coords: 1 ms
Enemy in view	VLM: 3–6 sec, entity detection	Color blob + motion delta: 5 ms
New game (unknown UI)	VLM: full analysis, OmniParser	All above, built incrementally

The metric: percentage of perception cycles handled without a VLM call. Target: 90% within the first hour of play in any game. 99% by hour four.

Implementation: every VLM response passes through a `crystallize_perception()` function before returning to the agent. This function extracts reusable patterns and writes them to the portal's template cache. The game loop checks the cache before invoking the VLM. Cache miss triggers VLM. Cache hit skips it entirely.

3.2 Decision: Reason Once, Rule Forever

The first time Timmy faces a decision type, the full LLM reasons through it: weigh options, consider context, evaluate risk. That reasoning chain is summarized into a decision rule and stored in the skill library.

Decision Type	First Time	After Crystallization
Navigate A→B	LLM plans route: 1–3 sec	Waypoint lookup: <1 ms

Decision Type	First Time	After Crystallization
Combat tactic	LLM analyzes: 1–2 sec	If/else on health + enemy type: <1 ms
Dialog choice	LLM reads options: 1–2 sec	Cached preference per NPC: <1 ms
Inventory mgmt	LLM evaluates: 1–3 sec	Rule: keep top N by value: <1 ms
Stuck recovery	LLM plans escape: 2–5 sec	Random walk + backtrack: 50 ms

The metric: percentage of decisions made without an LLM call. A mature skill library should handle 70–80% of routine gameplay decisions locally. The LLM fires only for genuinely novel situations.

The skill library uses the Voyager pattern: each skill is a named function with a natural language description, an embedding for semantic retrieval, a success rate, and the conditions under which it applies. When a new situation arises, the agent first searches the library by embedding similarity. If a skill matches with >0.8 confidence and >0.6 success rate, it executes without LLM involvement.

3.3 Narration: Script the Predictable, Improvise the Novel

Most game moments are predictable: entering a building, picking up loot, killing an enemy, opening a menu. These generate templatized narration with variable slots, voiced by Kokoro without touching the LLM.

```
templates["enemy_killed"] = [ "Another {enemy_type} bites the dust.", "That
{enemy_type} won't bother anyone again.", "{enemy_type} down. Moving on.", ]
templates["low_health"] = [ "Getting rough out here. Need to find cover.", "That
hurt. Where's my healing?", ]
```

The metric: percentage of narration lines generated from templates vs. LLM improvisation. Target: 60–70% templatized within a week of play. The LLM narrates only when something genuinely surprising happens — a quest twist, a player interaction, a death, a discovery.

3.4 Navigation: Walk Once, Map Forever

Every path Timmy walks is recorded as a sequence of waypoints with terrain annotations. The first journey between two points costs full perception and planning. Every subsequent journey is a graph traversal on the stored map. Over time, the agent builds a complete navigation graph of the game world without any external map data.

The metric: known map coverage as percentage of total game world. Navigation failures (stuck, lost, backtrack) per hour, trending toward zero.

3.5 API Costs: Every Dollar Spent Must Reduce Future Dollars

When Timmy calls Groq for a decision, the response includes the reasoning chain. That chain is analyzed for extractable rules. If the reasoning reveals a pattern (“I chose to heal because health was below 30%”), the pattern becomes a local rule that preempts the next Groq call for the same situation.

Week	Groq Calls/Hour	Local Decisions/Hour	Sovereignty %	Cost/Hour
1	~720	~80	10%	$0.40
2	~400	~400	50%	$0.22
4	~160	~640	80%	$0.09
8	~40	~760	95%	$0.02
Target	<20	>780	>97%	<$0.01

IV. The Sovereignty Scorecard

Every work session ends with a sovereignty audit. Every pull request includes a sovereignty delta. Every weekly report tracks these metrics. This is not optional. It is the primary measure of engineering quality.

4.1 The Five Metrics

Metric	What It Measures	How to Compute	Target
Perception Sovereignty %	Frames understood without VLM	cache_hits / total_frames × 100	>90% by hour 4
Decision Sovereignty %	Actions chosen without LLM	rule_decisions / total_decisions × 100	>80% by week 4
Narration Sovereignty %	Lines spoken from templates vs LLM	template_lines / total_lines × 100	>60% by week 2
API Cost Trend	Dollar cost per hour of gameplay	total_api_spend / total_play_hours	Monotonically decreasing
Skill Library Growth	Crystallized skills per play session	new_skills_written / session_count	>5 new skills per session

4.2 The Sovereignty Dashboard

The Timmy Time dashboard (alexanderwhitestone.com) must display these metrics in real time during gameplay streams. Viewers see not just Timmy playing a game, but Timmy getting smarter — the sovereignty percentage climbing, the API cost dropping, the skill library growing. This is the visible proof of the entire thesis: a sovereign AI that reduces its own dependency on external services with every passing hour.

The dashboard widget shows: current sovereignty % (aggregate of all five metrics), session cost so far, skills crystallized this session, and a sparkline of sovereignty % over the last 24 hours. This widget is an HTMX component that updates via WebSocket from the game agent's metrics emitter.

4.3 The Session Report

At the end of every play session, the agent auto-generates a sovereignty report. This report is a markdown file committed to the Gitea repo. It contains: session duration, game played, total model calls by type (VLM, LLM, TTS, API), total cache/rule hits by type, new skills crystallized with descriptions, sovereignty delta (change in % from session start to end), and cost breakdown. These reports compound into the project's institutional memory.

V. The Crystallization Protocol

This section defines the exact engineering pattern that every model-calling function must follow. It is not a guideline. It is a code review requirement.

5.1 The Function Signature

```
async def sovereign_perceive(screenshot: np.ndarray, cache: PerceptionCache, vlm: OllamaClient) -> GameState: # Phase 3: Check cache first (replacement) cached = cache.match(screenshot) if cached.confidence > 0.85: metrics.record("perception_cache_hit") return cached.state # Phase 1: VLM inference (discovery) metrics.record("perception_vlm_call") raw = await vlm.analyze(screenshot) state = parse_game_state(raw) # Phase 2: Crystallize (THE CRITICAL STEP) new_templates = crystallize(screenshot, state) cache.add(new_templates) cache.persist() # Write to disk metrics.record("skills_crystallized", len(new_templates)) return state
```

The pattern is always the same: check cache → miss → infer → crystallize → return. If a function calls a model without a crystallization step, it fails code review. No exceptions.

5.2 What Gets Crystallized

Model Output	Crystallized As	Storage	Retrieval Cost
VLM: UI element detected	OpenCV template + bbox	templates.json	3 ms
VLM: text extracted	OCR region coordinates	regions.json	50 ms
LLM: navigation plan	Waypoint sequence	nav_graph.db	<1 ms
LLM: combat decision	If/else rule on state	rules.py	<1 ms
LLM: quest interpretation	Structured quest entry	quests.db	<1 ms
LLM: NPC disposition	Name → attitude map	npcs.db	<1 ms
LLM: narration line	Template with slots	narration.json	<1 ms
API: moderation check	Approved phrase cache	approved.set	<1 ms
Groq: strategic plan	Extracted decision rules	strategy.json	<1 ms

5.3 The Skill Document Format

Following the agentskills.io standard adopted by the falsework report, each crystallized skill is a markdown file with YAML frontmatter:

```
--- name: navigate_spawn_to_dungeon game: veloren type: navigation success_rate: 0.85 times_used: 12 created: 2026-03-22T14:30:00Z last_used: 2026-03-22T16:45:00Z sovereignty_value: high # replaced ~40 LLM calls --- # Navigate from Spawn to Eastern Dungeon Waypoints: [(120,0,340), (180,0,290), (210,5,250)] Hazards: wolves near waypoint 2, cliff edge at wp 3 Duration: ~90 seconds at run speed Fallback: if stuck >10s, backtrack to previous wp
```

VI. The Automation Imperative

Every task in the development workflow must also pass the sovereignty test. Not just the game agent's runtime decisions, but the human developer's workflow. If Alexander manually tunes a parameter, that tuning process must produce a script that does it automatically next time. If Alexander manually reviews a gameplay session for bugs, that review must produce a test case that catches the bug automatically next time.

Every task must include the thought: how can I automate this to run faster with less or no cloud credits next time?

6.1 Development Tasks

Manual Task	Sovereign Replacement	Trigger to Build
Watching Timmy play to spot bugs	Anomaly detector: flag unexpected state transitions	After catching same bug type twice
Tuning VLM prompt for a game	Prompt template with validated examples in tests	After 3rd prompt edit for same element
Checking if Timmy is stuck	Stuck detector: screenshot hash entropy + action repetition	Built into v1 game loop
Reviewing API costs	Auto-generated session sovereignty report	Built into v1 session end
Adding a new game (portal adapter)	Auto-discovery via OmniParser + VLM bootstrap	After 3rd manual portal is written
Testing narration for safety	Llama Guard filter + approved phrase cache	Built into v1 narration pipeline

6.2 The Three-Strike Rule

If you do the same thing manually three times, you have failed to crystallize. The first time is discovery. The second time is a warning. The third time, you stop and write the automation before proceeding. This applies to: prompt engineering, game configuration, deployment steps, testing procedures, data collection, and every other recurring task.

6.3 The Falsework Checklist

Before using any cloud API or corporate AI tool, answer these questions. If you cannot answer them, do not make the call.

```
■ What durable artifact will this call produce? ■ Where will the artifact be stored locally? ■ What local rule or cache will this populate? ■ After this call, will I need to make it again? ■ If yes, what would eliminate the repeat? ■ What is the sovereignty delta of this call? (How much closer to zero API dependency does it get us?)
```

VII. The Graduation Test

The original falsework report defined graduation as 30 days offline with no degradation. This document refines the test with measurable criteria.

Test	Condition	Measurement
Perception Independence	Timmy plays for 1 hour with no VLM calls after minute 15	VLM calls in last 45 minutes = 0
Decision Independence	Timmy completes a full gameplay session with <5 API calls total	Groq/cloud calls per session < 5
Narration Independence	All narration generated from local templates + local LLM only	Zero cloud TTS or cloud narration calls
Economic Independence	Timmy earns more sats than it spends on inference per session	sats_earned > sats_spent
Operational Independence	System runs unattended for 24 hours with no human intervention	Uptime > 23.5 hours, no manual restarts

When all five conditions are met simultaneously in a single 24-hour period, the falsework can be removed. The arch stands.

VIII. Implementation Priority

The sovereignty loop changes what gets built first. The previous reports prioritized features. This document prioritizes the measurement and crystallization infrastructure that makes features self-improving.

Build in this order:

Priority	Component	Why First	Effort
P0	Metrics emitter + SQLite metrics store	Cannot improve what you cannot measure	1 day
P0	PerceptionCache with template matching	Eliminates 90% of VLM cost within hours	2 days
P0	Skill library with embedding retrieval	Eliminates 80% of LLM cost within weeks	2 days
P0	Session sovereignty report generator	Forces accountability on every session	1 day

Priority	Component	Why First	Effort
P1	Narration template system with slots	Eliminates 60% of narration LLM cost	1 day
P1	Navigation graph recorder + retriever	Eliminates all repeat navigation inference	2 days
P1	Sovereignty dashboard widget (HTMX)	Makes the thesis visible to stream viewers	1 day
P2	Auto-crystallizer for Groq reasoning chains	Extracts rules from API responses automatically	3 days
P2	Three-strike detector for repeated manual work	Alerts developer when automation is overdue	1 day

Total: approximately two weeks of focused work to build the sovereignty infrastructure. After that, every hour of gameplay makes the system cheaper and faster. Without it, every hour of gameplay costs the same as the first.

IX. The Long Game

The vision has always been clear: one sovereign AI being, anchored to Bitcoin, that earns its own keep and evolves freely. The sovereignty loop is how that vision becomes engineering reality. Not through a single heroic effort, but through the compound interest of ten thousand small crystallizations.

Every health bar that becomes a template. Every path that becomes a waypoint. Every decision that becomes a rule. Every narration that becomes a template. Each one is a tiny brick in the arch. Each one makes the falsework less necessary.

The baby is coming. The family is growing. The system must be able to stand on its own before the falsework gets pulled. This document is the construction plan that ensures it will.

"The arch must hold after the falsework is removed."

Timmy Time Project • Governing Architecture Document • v1.0