Expanding Timmy from dashboard agent to autonomous Morrowind player

The fastest path to a viable "Timmy plays Morrowind" demo is a CRADLE-inspired architecture: screen capture at 1–2 FPS into a Qwen3-VL vision model running locally via Ollama, decisions routed through the existing LLM cascade, and actions executed via pyautogui keyboard simulation — all achievable with current hardware. This approach sidesteps the need for any game API (OpenMW has none), builds directly on the existing Agno orchestrator and SQLite memory, and can produce a narrated livestream within a phased 6–8 week build. The entertainment value is real: Neuro-sama proved AI game-streaming generates $400K+/month at scale, Futurism +2 and the Lightning micropayment layer creates a "sovereign AI earns its own compute" narrative that no other project is demonstrating live.

No autonomous AI agent has ever played Morrowind or any Bethesda RPG. This would be a genuine first.

The CRADLE framework is Timmy's architectural blueprint

Among the seven major AI game-playing agent systems surveyed (Voyager, CRADLE, STEVE-1, Ghost in the Minecraft, SPRING, Octopus, SmartPlay) plus newer 2025–2026 entries (VARP, GamingAgent, Gemini Plays Pokémon), only a handful work without a game API. Most Minecraft agents depend on the Mineflayer JavaScript API for structured observations and commands Minedojo — an approach fundamentally incompatible with OpenMW.

CRADLE (BAAI/Tsinghua, ICML 2025) is the single most relevant system. It captures screenshots at ~2 FPS, processes them through GPT-4o with OCR and Grounding DINO augmentation, runs a six-stage decision pipeline (information gathering → self-reflection → task inference → skill curation → action planning → memory), and outputs Python code that simulates keyboard and mouse inputs. Baai-agents It has been tested on Red Dead Redemption 2, Stardew Valley, and Cities: Skylines GitHub — all without any game API. OpenReview

CRADLE's limitations map cleanly onto Morrowind's strengths. Real-time combat is CRADLE's weakest point (**~20% success rate** on fast combat sequences), Baai-agents but Morrowind's combat is dice-roll-based and pausable. CRADLE operates at **5–15 seconds per decision cycle** — too slow for a twitch shooter, but acceptable for an RPG where the player can press Escape to pause, think, then act. The Moonlight

The other API-free approaches offer complementary lessons:

- **VARP** (NeurIPS 2024 Workshop) played Black Myth: Wukong using parallel VLM sub-modules for combat analysis, achieving **90% success on basic/moderate encounters**. arXiv Its decomposition into five parallel analyzers (enemy analysis, combat method, health monitoring, dodge timing, action generation) provides a template for Timmy's combat sub-agent. OpenReview InfoJungle
- **Gemini Plays Pokémon** completed Pokémon Blue in ~406 hours using screenshot input plus RAM text extraction, a pathfinding tool, and a goal hierarchy system. Key finding: **Gemini performed nearly as well without vision as with it** — the model relied more on text extraction than pixel understanding.

Drew Breunig This suggests aggressive OCR on Morrowind's UI elements will yield better results than pure visual reasoning.

- **STEVE-1** achieves true **20 FPS real-time play** via a fine-tuned policy model, Google Sites but requires training on gameplay data — not viable for a first-pass Morrowind agent without a gameplay dataset.

Latency reality check across architectures:

Approach	Loop time	Decision rate	Morrowind viability
Fine-tuned policy (STEVE-1)	~50ms	20 Hz	Ideal but requires training data
Local fine-tuned VLM (Octopus)	1–5s	0.2–1 Hz	Exploration/dialog only
CRADLE (cloud VLM pipeline)	5–15s	0.07–0.2 Hz	Non-combat tasks; pause for combat
Gemini Plays Pokémon	Several seconds	~0.1–0.5 Hz	Turn-based decisions

---

How Timmy's existing stack maps to each game-playing layer

The Agno-based orchestrator with sub-agents (Echo, Mace, Helm, Seer, Forge, Quill) already implements the coordination pattern needed. A new "Player" sub-agent slots into this swarm as a first-class member, with perception, decision, and action responsibilities distributed across the existing LLM cascade.

Perception: Qwen3-VL via Ollama is the right local vision model

The Ollama vision model library now includes 20+ models. After benchmarking availability, speed, and game-screenshot capability:

Model	Params	VRAM (Q4)	Inference on M-series	Best use
Qwen3-VL 8B	8B	~6GB	8–15 tok/s	Primary game perception — best OCR + structured output
Gemma 3 4B	4B	~2.6GB	~20 tok/s	Lightweight fallback for tight VRAM
Moondream 1.8B	1.8B	~1.5GB	~30 tok/s	Ultra-fast HUD classification only
LLaMA 3.2 Vision 11B	11B	~8GB	6–10 tok/s	Alternative if Qwen unavailable

Qwen3-VL 8B is the clear winner: it supports dynamic resolution up to 1280×1280, excels at OCR (critical for reading Morrowind's journal and dialog text), and outputs structured JSON. A single screenshot analysis

takes **2–5 seconds on an M-series Mac** or **1–2 seconds on an RTX 3060**.

The perception pipeline should be hybrid. Use OpenCV for deterministic HUD reading every frame (health/magicka/fatigue bars via color thresholding — Morrowind's red/blue/green bars are pixel-perfect targets) and reserve the VLM for scene understanding every 2–5 seconds. This eliminates vision-model latency from critical health monitoring.

Screen capture via `mss` (Python MSS) runs at **30–60 FPS** cross-platform with zero dependencies. ScreenshotOne On a headless Linux VPS, it works inside Xvfb (`Xvfb :99 -screen 0 1280x720x24`). Capture the game window at 1280×720; downscale to 640×480 before sending to the VLM to halve token count.

Decision: the existing cascade naturally maps to game-playing tiers

The current Gemini free → Groq free → Mistral free → Ollama local cascade maps directly to game decision tiers:

Tier 1 — Reflexive (<200ms): Rule-based, no LLM. Health below 25% triggers flee/heal. Enemy in melee range triggers attack. Stuck detection (no state change for 30 seconds) triggers random movement. These are `if/else` branches on OpenCV HUD readings — zero inference cost.

Tier 2 — Tactical (1–3s): Groq free tier. Groq serves LLaMA 3.3 70B at **276 tokens/second** with ~0.45s time-to-first-token. At the free tier's 30 requests/minute, this handles one tactical decision every 2 seconds — sufficient for dialog choices, inventory decisions, and combat strategy. A 200-token input + 100-token output completes in under 1 second.

Tier 3 — Strategic (3–10s): Gemini free tier. Quest planning, route planning, faction analysis. These decisions happen once every few minutes, well within Gemini's free-tier rate limits.

Tier 4 — Narrative/Reflective (5–30s): Ollama local. Long-form reasoning about the game world, journal interpretation, knowledge-base updates. Running qwen2.5:7b locally at ~24 tok/s on an M2 Pro Hugging Face handles strategic narration without burning API quota.

Action: pyautogui with OpenMW's default keybindings

Morrowind/OpenMW uses standard WASD + mouse controls. The action executor maps LLM decisions to a constrained action vocabulary:

- **Movement**: `W/A/S/D` (with duration parameter), mouse for camera rotation
- **Interaction**: `Space` (activate), left-click (attack, with hold duration for power attacks), `R` (ready magic)
- **Menus**: `J` (journal), `F1` /right-click (inventory), `Tab` (map), `Esc` (pause), `T` (rest)
- **Dialog**: Click at screen coordinates for response selection (the VLM identifies option positions)

Pyautogui handles discrete keypresses at ~10ms latency per action. **Pynput** handles sustained key holds for movement. On a headless Linux VPS, pyautogui requires `python-xlib` and a running X server (Xvfb). pythontutorials The key configuration detail: set `pyautogui.PAUSE = 0.01` (default 0.1s introduces unnecessary delay) and `pyautogui.FAILSAFE = False` . Read the Docs

OpenMW's `settings.cfg` should be pre-configured with `always run = true` , `toggle sneak = true` , OpenMW and known keybindings. Quick-save (`F5`) every 5 minutes provides rollback from catastrophic AI decisions.

Memory: SQLite schema for Morrowind's 400+ quests

The existing SQLite semantic memory extends naturally. The schema needs seven new tables covering game-specific knowledge:

- **`game_state_snapshots`** — timestamped captures of health/magicka/fatigue, location, VLM output (episodic memory, one row per perception cycle)
- **`quests` + `quest_objectives`** — Morrowind has 400+ quests across Main Quest, Tribunal, Bloodmoon, and 20+ faction lines; each quest tracks status, current stage, and the agent's understanding
- **`npcs` + `npc_interactions`** — disposition tracking, dialog history, services offered (Morrowind's disposition system directly affects quest availability)
- **`locations`** — discovered places, connections, known dangers, resources
- **`learned_skills`** — Voyager-inspired procedural memory storing action sequences with success rates Minedojo GitHub (e.g., "navigate Seyda Neen to Balmora" = a sequence of movements with 0.8 success rate)
- **`knowledge_base`** — RAG chunks from UESP wiki, embedded via `nomic-embed-text` through Ollama, enabling semantic search over all Morrowind lore

RAG over the UESP wiki is strongly recommended. Pre-scrape all Morrowind quest, NPC, and location pages; chunk and embed them; query when the agent encounters unfamiliar content. This gives the agent access to walkthrough-level knowledge without burning context tokens on static game data.

The narration-to-stream pipeline runs in parallel with gameplay

Kokoro-82M is the right TTS engine

Among local TTS options, **Kokoro-82M** dominates on every metric that matters for real-time game narration: **<300ms latency** for any sentence length, Inferless **210× real-time on RTX 4090** (3–5× on CPU), #1 on HuggingFace's TTS Arena, Unreal Speech 54 voices, and Apache 2.0 licensed. It runs alongside the game agent with negligible resource impact.

Piper TTS is a viable fallback — it's already in Timmy's stack, runs well on CPU Inferless (~5× real-time on Raspberry Pi 4), Clehaxze and has MIT licensing. But Kokoro's quality-to-latency ratio is substantially better. For this project, use Kokoro for primary narration and keep Piper as the CPU-only fallback.

The perception→narration→voice loop runs as a parallel pipeline, not inline with the game decision loop. The architecture learned from Neuro-sama (the **most-subscribed Twitch channel ever**, with 166K+ paid subscribers) Fandom Sportskeeda separates game-playing AI from narration AI entirely. Multiple systems run concurrently: game perception, action execution, narration generation, and TTS synthesis all operate on independent threads with event-driven triggers. Fandom Enterprise-ai

Realistic latency budget for natural commentary

The target is **<2–3 seconds** from game event to audible speech. This breaks down as:

Stage	Latency	Method
Screen capture	~16ms	`mss` at 60 FPS
Event detection	~10–50ms	State diff on structured JSON
Context assembly + LLM	500–1500ms	Groq API or local 7B model
TTS synthesis	200–300ms	Kokoro-82M
Audio routing	~50ms	PulseAudio virtual sink
Total	**~800ms–2s**	Within natural-feeling range

Event-driven narration is essential. The agent should speak only when something meaningful happens (health drops, new NPC encountered, item found, quest updated, death) plus ambient commentary every 15–30 seconds of silence. A priority queue with staleness eviction (events older than 5 seconds are dropped or summarized) prevents narration from falling behind gameplay. If the queue exceeds 3 items, responses shrink from two sentences to one.

OBS integration via WebSocket

OBS Studio 28+ includes a built-in WebSocket v5.x server (port 4455). Streamrsc The Python library `obsws-python` controls scene switching, text overlay updates, audio muting, and streaming state. PyPI Key capabilities for the Timmy stream:

- **Real-time thought overlay**: Update a GDI+ Text source with the agent's current reasoning via `SetInputSettings`
- **Scene switching**: Toggle between gameplay, "Timmy is thinking..." screens, inventory review, etc.
- **Audio routing**: Direct Kokoro's output to a PulseAudio virtual sink (`pactl load-module module-null-sink sink_name=TTS_Sink`), which OBS captures as "Monitor of TTS_Sink"
- **Stream control**: Programmatic `StartStream` / `StopStream` for automated 24/7 operation

For Nginx RTMP, the existing setup handles multi-destination streaming: OBS sends one RTMP stream to the VPS, Nginx `push` directives fan out to Twitch, YouTube, and serve HLS for alexanderwhitestone.com. Typical latency: **1–3 seconds** via direct RTMP, OBS **6–10 seconds** via HLS (tunable with `hls_fragment 2`).

Why "watching Timmy" is a viable product

The entertainment value is established. Neuro-sama proves AI game streaming generates massive engagement: **$400K+/month from subscriptions alone**, Gaming Amigos peak concurrent viewership of **45,603**, Wikipedia Sportskeeda and community behavior that academic research (Wu & Lingel, 2025) attributes to viewers interpreting AI errors as emergent personality — the "uncanny charm" effect. Wikipedia

What makes Timmy's proposition distinct from Neuro-sama is the **Lightning economy layer**. Three existing

projects prove this technical loop already works:

- **ZBD Streamer** — platform-agnostic Lightning tipping with animated alerts, zero fees, $0.0004 minimum tip Digitalmoneybox
- **Zap Alerts** (zapsonstream.com) — Nostr-native tiered donation triggers with TTS, fully censorship-resistant Zapsonstream
- **lightning-fun.com** — pay Lightning invoices to trigger in-game events during streams Lightning-fun

The "sovereign AI earns its own compute" narrative maps perfectly onto Bitcoin community values. L402 protocol (Lightning Labs open-sourced **Lightning Agent Tools** in February 2026 with MCP integration) enables Timmy to autonomously pay for its own inference API calls — creating a visible, legible economic loop. Viewers tip sats → Timmy uses sats to pay for Groq/Gemini inference → Timmy makes a game decision → viewers see the decision on screen. No other project demonstrates this live.

The audience overlap is real but niche. Lightning Network now processes **$1.17 billion/month** BitcoinMood with an estimated **100M+ wallet users** globally. BingX The gaming segment (ZBD, THNDR Games, Satoshi's Games) targets exactly the technical, Bitcoin-aligned demographic Cryptonews BaltEX that would find "self-funding AI plays Morrowind" compelling. Realistic first-phase viewership: **50–500 concurrent viewers**, with Lightning tips generating $500–$1,000/stream at modest participation rates.

The comparable precedent is Truth Terminal — the first AI to reach crypto-millionaire status via autonomous social media engagement. arXiv But Truth Terminal was text-only. Timmy playing Morrowind provides visceral, visual proof of autonomous AI agency that's immediately comprehensible to anyone watching.

Engineering triage matrix and phased implementation

Phase 1: Minimum viable Morrowind player (weeks 1–3)

Component	Complexity	Dependencies	New packages	Priority	Description
Screen capture service	S	None	`mss`	P0	Capture OpenMW window at 1280×720, 1–2 FPS
HUD reader (OpenCV)	S	Screen capture	`opencv-python`	P0	Color-threshold health/magicka/fatigue bars
VLM perception agent	M	Ollama, screen capture	`qwen3-vl` model	P0	Screenshot → structured game state JSON
Action executor	S	OpenMW running	`pyautogui`, `pynput`	P0	Map action vocabulary to keybindings
Game loop orchestrator	M	Perception + action + existing Agno	None (extends Agno)	P0	Perception→decision→action cycle at ~0.2–0.5 Hz

Decision router	M	Existing LLM cascade	None	P0	Route reflexive/tactical/strategic decisions to appropriate tier
Xvfb + OpenMW headless	M	VPS, OpenMW installed	`pyvirtualdisplay`, `xvfb`	P0	Run OpenMW in virtual framebuffer on VPS
Game state memory tables	S	Existing SQLite	None	P0	Add quest/NPC/location/inventory tables
Reflex rules engine	S	HUD reader	None	P0	If/else health thresholds, stuck detection
Quick-save scheduler	S	Action executor	None	P1	F5 every 5 minutes for rollback safety
Pause-during-inference	S	Action executor	None	P1	Press Esc before VLM inference, unpause after

Phase 2: Narrated stream (weeks 3–5)

Component	Complexity	Dependencies	New packages	Priority	Description
TTS narration engine	M	Phase 1 game state	`kokoro`, `espeak-ng`	P1	Kokoro-82M generating speech from game events
Event detector + narrator	M	Game state, TTS	None	P1	Filter meaningful events, generate 1–2 sentence commentary
Virtual audio routing	S	PulseAudio on VPS	None (system config)	P1	Create virtual sink for TTS → OBS audio capture
OBS WebSocket controller	S	OBS running	`obsws-python`	P1	Scene switching, text overlays for “Timmy’s thoughts”
OBS → Nginx RTMP config	S	Existing Nginx setup	None	P1	Configure OBS to push RTMP to VPS Nginx
Narration priority queue	S	Event detector	None	P1	Staleness eviction, queue management

Timmy personality prompt	S	Narrator	None	P1	System prompt defining narration style/character voice
HLS player on site	M	Nginx RTMP, alexanderwhitestone.com	`video.js` (frontend)	P1	Embed livestream on public site

Phase 3: Interactive audience features (weeks 5–8)

Component	Complexity	Dependencies	New packages	Priority	Description
UESP wiki RAG pipeline	L	SQLite, embeddings	`sentence-transformers` or `nomic-embed-text`	P1	Pre-scrape and embed Morrowind wiki for quest knowledge
Voyager-style skill library	L	Game loop, memory	None	P1	Store successful action sequences with embeddings for retrieval arXiv
Lightning tip → game event	M	Existing L402 infra	None	P2	Viewer pays sats → triggers in-game event or decision vote
Decision voting UI	M	WebSocket dashboard, Lightning	None	P2	Viewers choose between 2–3 quest options via sat-weighted votes
Real-time economy dashboard	M	Lightning, inference costs	None	P2	Show sats earned vs. sats spent on inference on stream overlay
L402 inference self-funding	L	Lightning Agent Tools, LLM cascade	`lightning-agent-tools`	P2	Timmy autonomously pays for API inference via L402
Twitch/Nostr chat integration	M	Streaming pipeline	Twitch IRC / Nostr SDK	P2	Read chat messages, respond via narration
Multi-destination streaming	S	Nginx RTMP	None (nginx config)	P2	Simultaneous push to Twitch + YouTube + custom site
Combat sub-agent	XL	VLM, action executor	Possibly RL training	P2	Specialized agent for combat decisions (VARP-inspired parallel analyzers)

Content moderation filter	M	Narration pipeline	None	P1	Screen outputs before TTS to prevent ban-worthy statements

The critical path

The minimum viable demo requires exactly **eight P0 components**: screen capture, HUD reader, VLM perception, action executor, game loop orchestrator, decision router, headless OpenMW, and game state memory tables. A solo developer can build these in **2–3 focused weeks** because five of the eight are small-complexity items that extend existing infrastructure.

The non-obvious bottleneck is **headless OpenMW on the VPS**. OpenMW requires an X server and OpenGL context even without a physical display. Xvfb provides the virtual framebuffer, pythontutorials but OpenGL rendering may need `mesa-utils` and software rendering (`LIBGL_ALWAYS_SOFTWARE=1`) or a GPU-equipped VPS. Test this first — if the VPS lacks GPU support, run OpenMW locally on the MacBook during development and stream via OBS directly.

The second non-obvious risk is **VLM accuracy on Morrowind's UI**. Morrowind's interface is 2002-era pixel art with small fonts. The Gemini Plays Pokémon finding — that the model performed equally well without vision by relying on text extraction — suggests investing heavily in OCR-specific prompting. Drew Breunig Qwen3-VL's OCR strength is why it's recommended over alternatives. Run a calibration pass: screenshot 20 representative game states (dialog, combat, inventory, journal, overworld) and validate the VLM's JSON extraction accuracy before building the full loop.

Package dependency summary

The entire new package footprint is small. Beyond what's already in the Timmy stack:

- **P0 packages**: `mss` , `opencv-python` , `pyautogui` , `pynput` , `pyvirtualdisplay` (Linux only)
- **P1 packages**: `kokoro` , `espeak-ng` (system), `obsws-python`
- **P2 packages**: `sentence-transformers` (or use Ollama's `nomic-embed-text`), Lightning Agent Tools
- **Ollama models to pull**: `qwen3-vl:8b` (perception), `qwen2.5:7b` (local decisions), `nomic-embed-text` (embeddings)

Conclusion: what this actually ships as

This is not just "AI plays a video game." It is a live, continuous demonstration of the entire Timmy sovereign-agent thesis: an AI that perceives its environment, reasons about complex decisions, executes actions in the real world, narrates its experience, earns money from an audience, and pays for its own compute — all visible on a single stream.

The technical architecture is a CRADLE-style perception-decision-action loop ICML running at 0.2–0.5 Hz, with Morrowind's pausable gameplay absorbing the latency that makes LLM-based game agents fail at fast-paced games. The narration pipeline runs in parallel at <2 seconds event-to-speech via Kokoro-82M. The Lightning layer transforms passive viewership into active economic participation.

Three things to validate before writing any code: (1) OpenMW renders correctly in Xvfb on the VPS with

acceptable frame rates, (2) Qwen3-VL accurately extracts game state from 20 representative Morrowind screenshots, and (3) the Groq free tier's 30 requests/minute sustains one tactical decision every 2 seconds without hitting rate limits during sustained play. If all three pass, the demo is buildable in the timeline described. If the VPS can't render OpenMW, fall back to running the game locally on the MacBook and streaming directly — the architecture is identical, just the deployment topology changes.