DEFENSIVE PUBLICATION — TECHNICAL DISCLOSURE (TDCOMMONS DEPOSIT COPY) · 2026-06-25

Keywords: defensive-publication, prior-art, ai-agents, intrinsic-motivation, idle-detection, agent-scheduling, weighted-random

Idle-Triggered Curiosity Engine for AI Agents

A Technical Defensive Publication

Subtitle: A registry of weighted, cooldown-gated, availability-gated curiosity actions, selected by weighted-random sampling restricted to the runtime-eligible set, invoked *only* when an agent's primary work source is empty.

Field	Value
Title	Idle-Triggered Curiosity Engine for AI Agents
Document type	Technical Defensive Publication (public prior art)
Publisher / Copyright holder	Gus IT LLC (Florida, USA)
Author	Gustavo Assuncao, PhD
Publication date	2026-06-25
Version	1.0
Classification	Public
License	AGPL-3.0-or-later (copyleft; commercial license available)
Field of invention	Intrinsic-motivation behaviors in autonomous agents; computer-implemented orchestration of long-running AI agents
Deposit channel	To be assigned (IP.com / Zenodo / arXiv) — establishes a public, dated, citable prior-art record

> **Defensive-publication statement.** This disclosure is intentionally made public, with a date, to establish prior art. Its purpose is to keep the described technique freely practiceable by everyone and to bar later patenting of the technique (or trivial variants) by others. See Appendix D.

Abstract

Autonomous, reactive AI agents — chat assistants, cognitive personas, and tool-using agents driven by a perceive–reason–act loop — sit idle whenever no message, task, or event is pending. That idle time wastes provisioned compute (GPU/CPU and a held model context) and, more importantly, forfeits opportunities for the agent to do useful proactive work: revisiting stale goals, summarizing accumulated context, learning a new skill, or surfacing a forgotten follow-up. Naively filling idle time with background activity is dangerous: undisciplined background tasks compete with reactive work, repeat themselves, and fire when their preconditions are not met.

This publication discloses an **Idle-Triggered Curiosity Engine**: a small, deterministic control component that activates **only** when the agent's primary work source reports empty, and then selects one intrinsic-motivation ("curiosity") action from a **registry** of action descriptors. Each descriptor carries three independent gates — a **weight** (relative selection probability), a **cooldown** (minimum time before the same action may re-fire), and an **availability predicate** (a runtime precondition function). Selection is a **weighted-random draw restricted to the eligible set** — descriptors whose availability predicate evaluates true *and* whose cooldown has elapsed. The chosen action is enqueued into the agent's ordinary task queue, so it flows through the same execution, safety, and budget machinery as reactive work and is automatically pre-empted the moment real work arrives.

The combination — *idle-as-the-sole-trigger*, plus a *three-gate weighted registry* whose selection is *restricted to the runtime-eligible set* — yields proactive, varied, non-repetitive, precondition-respecting agent behavior. This document fully describes the architecture, mechanics, data model, an enabling clean-room reference implementation, a worked example, failure modes, framework alignment, and the complete novelty claims (one independent and fourteen dependent).

---

1. Executive Summary

1.1 Thesis

A reactive agent's idle time is a wasted, recoverable resource. It can be turned into *disciplined* proactivity — varied, non-repetitive, and precondition-respecting — by a small control component that (a) fires **only** when the primary work source is empty, and (b) chooses what to do by **weighted-random sampling over a registry, restricted to the set of actions that are currently both available and off-cooldown.** The discipline comes from making idle-detection the *sole* trigger and from making three independent gates compose into the eligibility set.

1.2 Contributions

#	Contribution	Section
C1	**Idle-as-sole-trigger.** Curiosity selection is invoked only upon detection that the primary work source is empty, guaranteeing zero interference with reactive operation.	§6, §7
C2	**Three-gate action descriptor.** Each curiosity action is described by an independent `weight`, `cooldown`, and `availability predicate`.	§7, §8
C3	**Eligibility-restricted weighted-random selection.** The weighted draw is performed *only over* descriptors that pass both the availability predicate and the cooldown gate, so probability mass is renormalized over the live eligible set.	§7
C4	**Queue-mediated execution.** The selected action is enqueued into the agent's normal task queue, inheriting its execution, safety, budget, and pre-emption semantics rather than running in a privileged side-channel.	§6, §7
C5	**Clean-room reference implementation** reducing the mechanism to practice in dependency-light Node.js, with a runnable self-check.	§9, Appendix C

1.3 Headline claim

> A method for triggering intrinsic-motivation actions in an AI agent comprising: detecting that a primary work source for the agent is empty; selecting, from a registry of action descriptors each having a weight, a cooldown, and an availability predicate, an action by weighted-random sampling **restricted** to descriptors whose availability predicate evaluates true and whose cooldown has elapsed; enqueuing the selected action into the agent's task queue; **characterized in that** the selection is invoked **only** on detection of empty primary work.

1.4 Scope

What this is. A control-component design and method for converting agent idle time into disciplined proactive activity, plus an enabling reference implementation.

What this is NOT.

- It is **not** a reinforcement-learning algorithm. It does not learn a reward model, does not perform gradient updates, and does not require an environment model. (RL "curiosity" via intrinsic reward / prediction error is *related but distinct* prior art; see §4–§5.)
- It is **not** a general scheduler or cron system; it is specifically gated on *idle detection of a primary work source* and selects from a *runtime-eligibility-restricted* weighted registry.
- It is **not** a claim over weighted-random sampling, cooldown timers, or precondition predicates *in isolation* — all of which are well-known. The disclosed subject matter is their specific *combination and triggering discipline*.

---

2. Introduction & Motivation

2.1 The concrete problem

A modern autonomous agent is typically built around a **cognitive loop**: perceive inputs, reason about them, plan, act, reflect, repeat. The loop is *reactive* — it advances when there is something to react to: an inbound chat message, a queued task, a scheduled goal, an event. When the inbound channels and the task queue are empty, the loop has nothing to do and the agent **goes idle**.

For a short-lived request/response bot, idle is fine — the process simply waits. But a growing class of agents are **long-running** and **resource-provisioned**:

- A *cognitive persona* that holds a warm model context, a working memory, and an attention state across hours or days.
- A *worker agent* that owns a slice of GPU/CPU and is expected to make continuous progress.
- An *assistant* that users expect to be proactive ("you mentioned a deadline yesterday — it is tomorrow").

2.2 The "idle tax"

For these agents, idle time carries a real cost — an *idle tax*:

Cost dimension	Description
Wasted compute	Provisioned GPU/CPU and a held model context produce nothing during idle windows.
Forfeited proactivity	Stale goals are not revisited; accumulated context is not summarized; follow-ups are not surfaced; skills are not practiced.
Perceived passivity	Users experience the agent as inert — it only ever responds, never initiates — degrading the product.
Lost learning/maintenance windows	Idle is exactly when low-priority background work (cache warming, index refresh, self-reflection) could run for free.

2.3 Why naive solutions fall short

The obvious fix — "do something in the background when idle" — fails without discipline:

1. **A fixed background task** repeats the same activity forever; it is monotonous and quickly exhausts its usefulness.
2. **A round-robin list** of activities fires actions whose preconditions are not met (e.g. "summarize recent conversation" when there has been no recent conversation), producing noise and errors.
3. **A pure random pick** has no notion of relative importance and re-fires the same heavy action back-to-back.
4. **A background thread/side-channel** that bypasses the normal task queue competes with reactive work for the model and for tools, and is not automatically pre-empted when real work arrives — defeating the whole point.
5. **A cron scheduler** fires on wall-clock time regardless of whether the agent is busy, reintroducing interference.

2.4 Why now

Three trends make this timely in 2026: (a) agents are increasingly *long-running and stateful* rather than stateless request handlers; (b) provisioned inference capacity is *expensive*, so idle waste is material; and (c) users now *expect proactivity* from agentic products. The disclosed mechanism is a small, low-risk way to capture idle value without destabilizing reactive behavior.

---

3. Problem Statement

3.1 Formal framing

Let an agent expose a **primary work source** `W` (e.g., the union of inbound message channels and the reactive task queue). At each loop tick `t`, define the predicate `idle(t) := (|W(t)| = 0)` — the primary work source is empty.

Let `R = {a_1, …, a_n}` be a **registry** of curiosity action descriptors. Each descriptor `a_i` carries:

- a **weight** `w_i ∈ ℝ_{>0}` (relative selection probability),
- a **cooldown** `c_i ∈ ℝ_{≥0}` (minimum seconds between firings of `a_i`),
- an **availability predicate** `p_i : Context → {true, false}` (a runtime precondition),

- a **last-fired timestamp** `τ_i` (mutable state, initially −∞).

At tick `t`, define the **eligible set** `E(t) := { a_i ∈ R : p_i(ctx_t) = true ∧ (t − τ_i) ≥ c_i }`.

The engine's behavior:

```
if idle(t) and E(t) ≠ ∅:
    a* ← sample a_i from E(t) with probability  w_i / Σ_{a_j ∈ E(t)} w_j
    enqueue(task_for(a*))
    τ_{a*} ← t
```

The *only* time the engine acts is when `idle(t)` holds. The draw is over `E(t)`, not over `R`, so probability mass is renormalized across exactly the actions that are currently both available and off-cooldown.

3.2 Derived requirements

Req	Requirement	Addressed in
R1	The engine must act **only** when the primary work source is empty; reactive work must never be delayed by curiosity.	§6, §7, §11
R2	Each action must declare a **relative weight** so important/cheap actions are favored without starving others.	§7, §8
R3	Each action must declare a **cooldown** so the same action cannot re-fire too soon.	§7, §8
R4	Each action must declare an **availability predicate** so it fires only when its preconditions hold.	§7, §8
R5	Selection must be a **weighted-random draw restricted to the eligible set** (renormalized over `E(t)`).	§7
R6	The selected action must be **enqueued into the normal task queue**, inheriting execution, safety, budget, and pre-emption semantics.	§6, §7, §11
R7	The engine must **degrade gracefully**: an empty eligible set, a throwing predicate, or a missing context must not crash the loop (fail-open into idle).	§7, §11
R8	The registry must be **configurable/extensible** at runtime without code changes to the engine core.	§7, §9
R9	Firings must be **observable** (logged/metered) for audit and tuning.	§8, §11, §13
R10	The mechanism must be **agent-framework agnostic** (usable as a library or sidecar).	§6, §16

---

4. Related Work & Prior Art

This work builds on, and is deliberately positioned against, a substantial body of existing art. We name the real sources we draw on.

- **Utility AI (game AI).** Decision-making by scoring candidate actions with utility ("appraisal") functions and selecting the highest — or sampling among the top — is standard in game AI (e.g., Dave Mark's *Behavioral Mathematics for Game AI*, and the IAUS / Infinite Axis Utility System). Utility AI provides **scored action selection with considerations (predicates)**, which is conceptually adjacent to our weight + availability-predicate idea.

- **GOAP (Goal-Oriented Action Planning).** Jeff Orkin's GOAP (F.E.A.R., 2006) selects actions whose **preconditions** are satisfied to achieve a goal. This is direct prior art for **precondition-gated action selection**.
- **Behavior Trees.** A widely used game/robotics structure for prioritized, fallback-driven action selection. Prior art for **structured, prioritized selection among actions**.
- **Cooldown timers.** Ubiquitous in games and rate limiters: a per-action minimum interval before re-use. Prior art for **R3** in isolation.
- **Weighted-random / roulette-wheel selection.** A textbook technique (genetic algorithms, loot tables, A/B sampling): pick an item with probability proportional to its weight. Prior art for **R5** in isolation.
- **Reinforcement-learning intrinsic motivation / "curiosity".** Schmidhuber's formal theory of curiosity (1991–2010) and prediction-error intrinsic reward (Pathak et al., ICM, 2017; Burda et al., RND, 2018) reward an agent for exploring novel/surprising states. This is the closest *conceptual* neighbor to the word "curiosity" but is a **learning** mechanism (reward shaping, gradient updates) — **distinct** from our deterministic, training-free selection-over-a-registry.
- **Generative Agents (Park et al., 2023).** LLM agents with memory streams, reflection, and planning that act proactively in a sandbox. They demonstrate **proactive LLM-agent behavior**, but their proactivity is plan/reflection-driven, not gated on *idle-detection-of-a-primary-work-source* with a *three-gate weighted registry*.
- **Cron / scheduled jobs and "sleep-time"/background-compute agents (e.g., MemGPT/Letta sleep-time agents).** These run background work on a timer or during downtime. They establish **background work during downtime** but typically fire on wall-clock schedules or framework hooks rather than on *empty-primary-work detection*, and select tasks by fixed policy rather than an eligibility-restricted weighted draw.
- **OS idle tasks / `requestIdleCallback`.** The browser/OS pattern of running low-priority work when the main thread is idle is the systems-level analogue and prior art for **R1**'s spirit (run-when-idle). We adopt its *philosophy* but specialize the trigger and the selection policy to agent registries.

We **adopt** the strengths of utility AI (scored, predicate-gated selection), GOAP (precondition gating), cooldown timers, weighted-random sampling, and the run-when-idle philosophy — and **combine** them under a specific triggering and eligibility discipline tailored to long-running LLM agents.

---

5. Prior-Art Delta

The table below isolates, per novel feature, what the closest prior source already has, what it lacks, and what this disclosure adds.

Prior source	What it HAS	What it LACKS	What THIS adds
Utility AI (game AI)	Scored action selection with per-action considerations/predicates.	No notion of *idle-detection of a host work source* as the gate; no per-action cooldown as a first-class composable gate; runs every frame, not only when idle.	Makes *empty-primary-work* the sole trigger and composes weight + cooldown + availability into a renormalized eligible-set draw.
GOAP / Behavior Trees	Precondition-gated, prioritized action selection toward goals.	Deterministic/highest-priority selection, not *weighted-random over the eligible set*; no idle trigger; no cooldown gate.	Weighted-random sampling restricted to the eligible set, fired only on idle, with cooldowns.
Cooldown timers	Per-action minimum re-fire interval.	No selection policy, no availability predicate, no idle trigger.	Cooldown as one of three composable gates inside an idle-triggered weighted registry.
Weighted-random selection	Probability ∝ weight.	No eligibility restriction (no availability/cooldown gates), no idle trigger.	Draw is *restricted to and renormalized over* the runtime-eligible set.
RL intrinsic-motivation / curiosity (ICM, RND, Schmidhuber)	Intrinsic reward driving exploration of novel states.	Requires learning (reward model, gradients, environment); not a deterministic registry selection; not idle-gated.	Training-free, deterministic registry selection; no learning loop; idle-gated and queue-mediated.
Generative Agents (Park 2023)	Proactive LLM-agent behavior via memory + reflection + planning.	Proactivity is plan/reflection-driven, not gated on empty-primary-work; no three-gate weighted registry.	Idle-as-sole-trigger + three-gate weighted registry + queue mediation.
Cron / sleep-time / background-compute agents	Background work during downtime/timer.	Fires on wall-clock/framework hook, not on empty-primary-work detection; fixed task policy, not eligibility-restricted weighted draw.	Trigger is *idle detection of the primary work source*; selection is eligibility-restricted weighted-random.
`requestIdleCallback` / OS idle tasks	Run low-priority work when the host is idle.	No agent-action registry; no weight/cooldown/availability gating; no weighted selection.	A registry-based, three-gated, weighted selection specialized for agent curiosity actions, enqueued into the agent's own task queue.

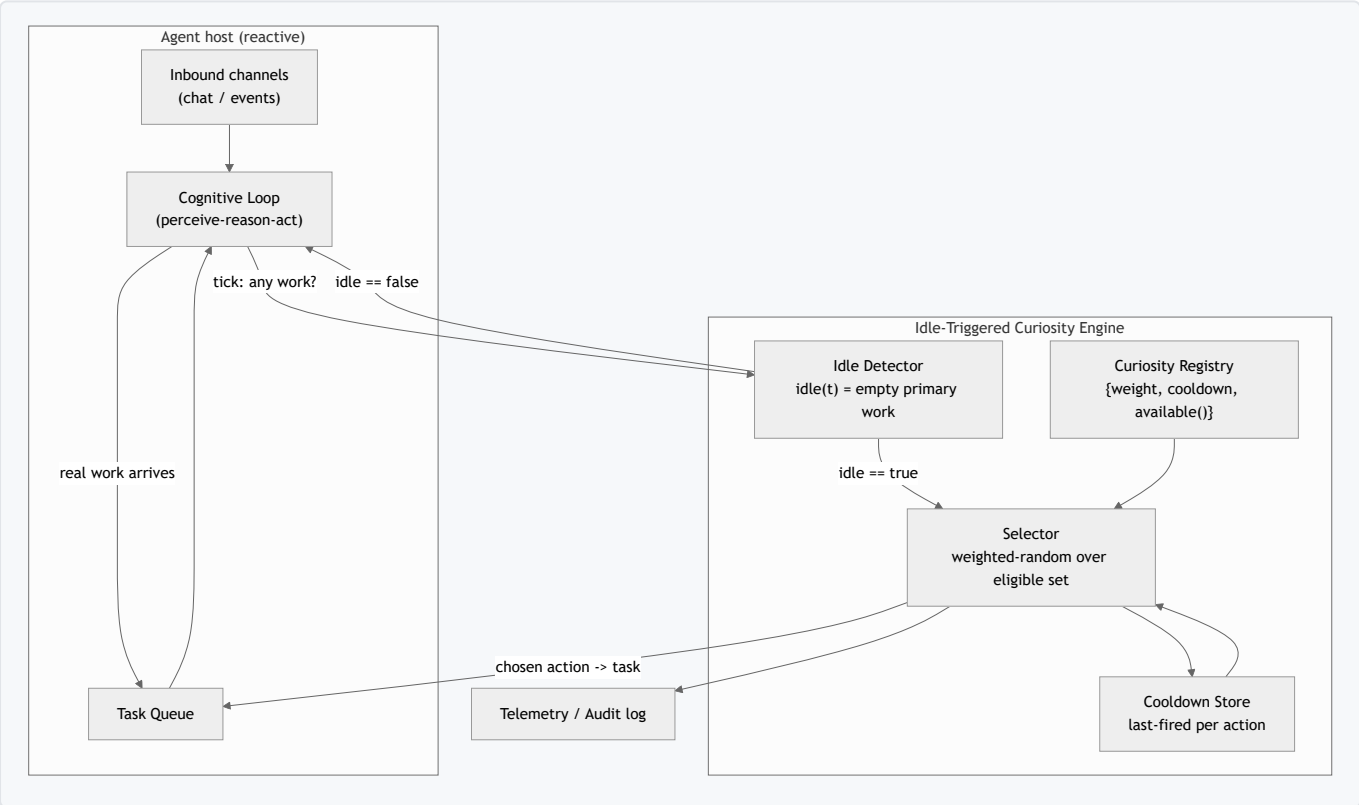
Net novel combination (the delta): *idle-detection-of-the-primary-work-source as the sole trigger + a three-gate (weight, cooldown, availability) action registry + weighted-random selection restricted to and renormalized over the eligible set + queue-mediated execution inheriting the agent's pre-emption semantics.* No single prior source combines all four.

6. System Architecture

6.1 Components

Component	Responsibility
Cognitive Loop (host)	The agent's reactive perceive–reason–act loop. At each tick it asks: is there primary work? It owns the **task queue**.
Idle Detector	A thin predicate over the primary work source: returns true iff inbound channels *and* the reactive task queue are empty for this agent.
Curiosity Registry	The set of action descriptors `{ id, weight, cooldown, available(ctx), build(ctx) }`. Configurable/extensible at runtime.
Cooldown Store	Per-agent, per-action `last-fired` timestamps; answers "has the cooldown elapsed?"
Selector	Computes the eligible set, performs weighted-random draw restricted to it, returns the chosen descriptor (or none).
Task Queue (host)	The agent's normal queue. The selected curiosity action is enqueued here as an ordinary task.
Telemetry	Logs/meters firings and skips for audit and tuning.

6.2 Architecture diagram (Figure 1)



6.3 Cross-cutting properties

- **Non-interference (R1).** The selector is invoked *only* on `idle == true`. The selected action is enqueued, not executed inline; the host loop processes the queue with its normal priorities, so any reactive task enqueued afterward can be ordered ahead of it.

- **Statelessness of the core.** The selector is a pure function of `(registry, cooldown store, context, RNG)`. The only mutable state is the cooldown store.
- **Fail-open (R7).** Any error in the engine resolves to "do nothing this tick," leaving the agent idle — never crashing the host loop.
- **Framework-agnostic (R10).** The engine depends only on three injected capabilities: `isIdle()`, `enqueue(task)`, and a `context` provider. It can ship as a library or sidecar.

---

7. Detailed Mechanics

7.1 The action descriptor

Each registry entry is a descriptor:

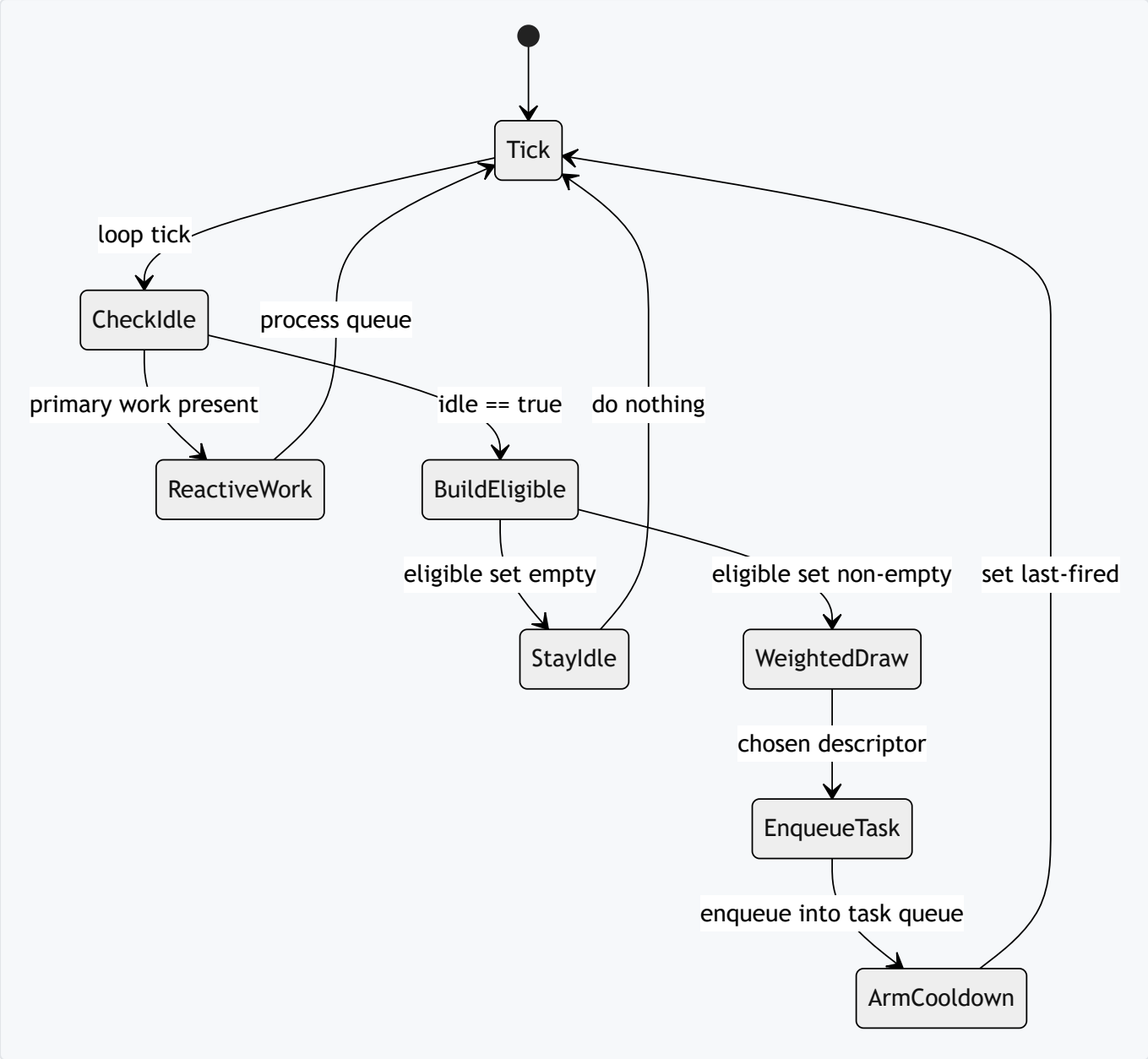
```
ActionDescriptor {
  id:         string         // unique, stable
  weight:     number > 0     // relative selection probability (R2)
  cooldownMs: number >= 0    // minimum ms between firings (R3)
  available:  (ctx) => bool  // runtime precondition predicate (R4)
  build:      (ctx) => Task  // constructs the queue task for this action
}
```

`weight`, `cooldownMs`, and `available` are **independent** gates. `weight` shapes *probability among eligible actions*; `cooldownMs` and `available` shape *membership in the eligible set*.

7.2 The selection algorithm (pseudocode)

```
function maybeRunCuriosity(agentId, ctx, now, rng):
    if not isIdle(agentId):                # R1: only when primary work is empty
        return SKIP_NOT_IDLE

    eligible = []
    for a in registry:
        try:
            if (now - lastFired(agentId, a.id)) < a.cooldownMs:   # R3 cooldown gate
                continue
            if not a.available(ctx):                              # R4 availability gate
                continue
        except Error as e:
            logSkip(a.id, "predicate-error", e)                   # R7 fail-open per-action
            continue
        eligible.push(a)

    if eligible is empty:                  # R7: nothing eligible -> stay idle
        return SKIP_NO_ELIGIBLE

    total = sum(a.weight for a in eligible)  # R5: renormalize over eligible set ONLY
    r = rng() * total
    acc = 0
    chosen = eligible[last]                 # safe fallback for float drift
    for a in eligible:
        acc += a.weight
        if r < acc:
            chosen = a
            break

    task = chosen.build(ctx)                # R6: build an ordinary task
    enqueue(agentId, task)                  # ... and enqueue it
    setLastFired(agentId, chosen.id, now)    # arm cooldown
    logFire(agentId, chosen.id)             # R9 observability
    return chosen.id
```

7.3 Control flow (Figure 2 — state machine)



7.4 Data structures

- **Registry**: an array (or map) of `ActionDescriptor`. Order is irrelevant to correctness; weights govern probability.
- **Cooldown store**: `Map<agentId, Map<actionId, lastFiredMs>>`. In a single-process agent this is in-memory; in a clustered deployment it is backed by a shared key-value/row store so cooldowns are honored across replicas (see §8). Missing entry ⇒ treated as $-\infty$ (eligible).
- **Context (`ctx`)**: a read-only snapshot the predicates and builders consult (e.g., counts of stale goals, recent message count, available skills). The engine does not mutate it.

7.5 Error handling (R7)

- A **throwing availability predicate** is caught per-action; that action is skipped, not the whole tick.
- A **throwing builder** is caught; the firing is abandoned and the cooldown is *not* armed, so the action remains eligible next idle tick.

- An **empty eligible set** returns a benign skip code; the agent simply stays idle.
- A **missing/failed idle detector** is treated as "not idle" (fail-closed *for triggering*), guaranteeing the engine never runs when it cannot prove the agent is idle — protecting R1.

The two fail postures are deliberate: *triggering* fails **closed** (never act unless proven idle — protects reactive work), while *selection* fails **open** (a broken action just drops out — protects liveness). See §11.

7.6 Configuration points

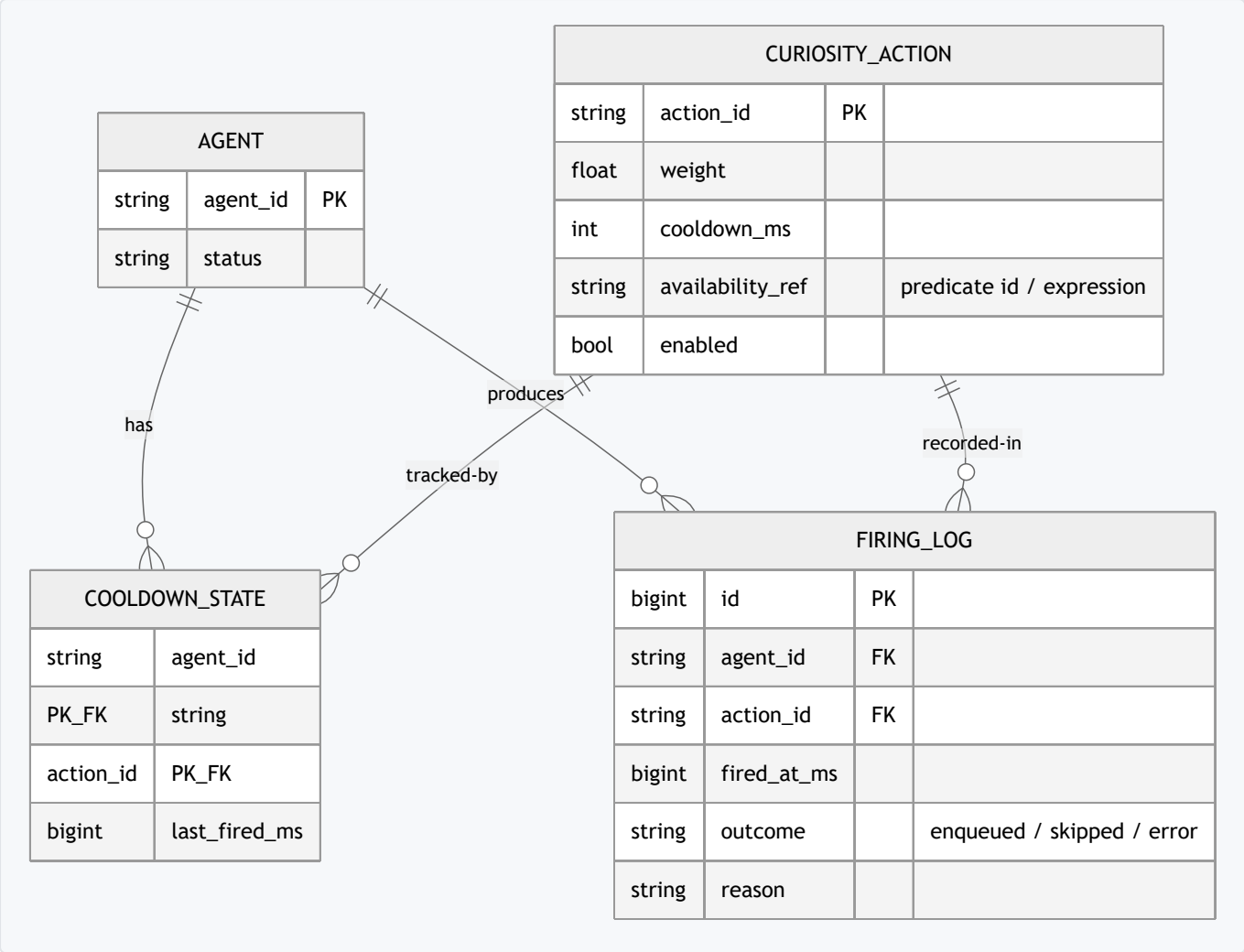
Config	Effect
`weight` per action	Bias selection toward important/cheap actions (R2).
`cooldownMs` per action	Throttle re-firing of heavy/spammy actions (R3).
`available` per action	Encode preconditions (R4).
Registry membership	Add/remove curiosity behaviors without touching the engine (R8).
RNG seed (optional)	Deterministic replay for testing.
Global enable flag	Feature-flag the whole engine off.

8. Data Model

8.1 Logical entities

The engine needs three persisted/queryable concepts in a clustered deployment: the registry (often code-defined but optionally table-driven), the per-agent cooldown state, and a firing audit log.

8.2 ER diagram (Figure 3)



8.3 Field notes

Entity.Field	Notes
`CURIOSITY_ACTION.weight`	Strictly positive. Relative, not absolute — only ratios within the eligible set matter.
`CURIOSITY_ACTION.cooldown_ms`	`0` allowed (no cooldown).
`CURIOSITY_ACTION.availability_ref`	A reference to a predicate (code symbol) or, in a table-driven variant, a safe expression evaluated against `ctx`.
`COOLDOWN_STATE.(agent_id, action_id)`	Composite PK. Absence ⇒ never fired ⇒ eligible. In a single-process agent this collapses to an in-memory map.
`FIRING_LOG.outcome`	Drives observability/tuning (R9): which actions fire, which are starved, which error.

> **Note for clustered agents.** When the same logical agent runs as multiple replicas, `COOLDOWN_STATE` must be a *shared* store and the read-check-write of `last_fired_ms` should be atomic (e.g., a conditional update) so two replicas do not both fire the same action in the same window. In a single-process agent the in-memory map suffices.

9. Reference Implementation & Enablement

9.1 What `src/` demonstrates

The `src/` directory contains a **clean-room, dependency-light Node.js** implementation that reduces the invention to practice:

- `src/curiosity-engine.js` — the engine core: registry, eligibility computation (cooldown + availability gates), eligibility-restricted weighted-random selection, queue enqueue, cooldown arming, fail-open/fail-closed handling, and telemetry hooks.
- `src/example-agent.js` — a tiny mock agent host: a fake inbound channel + task queue, an `isIdle()` derived from them, a registry of three illustrative curiosity actions, and a driver loop that injects bursts of reactive work and shows curiosity firing *only* during idle gaps. Runs a self-check that asserts (a) no curiosity fires while reactive work is pending, (b) cooldowns are honored, (c) availability gates exclude actions, and (d) selection frequency tracks weights over many idle ticks.

It is **illustrative and clean-room** — it shares no proprietary source with any internal product and contains no secrets, infra identifiers, or customer data. It is not production code; it omits clustering, persistence, and framework glue.

9.2 How it reduces the invention to practice

The reference code exercises every claim element: idle detection as the sole trigger (Claim 1 preamble + characterizing clause), the three-gate descriptor (Claims 2–4), the eligibility-restricted weighted draw (Claims 1, 5), and queue-mediated execution (Claim 1, Claim 6). The self-check provides empirical evidence that the mechanism behaves as specified.

9.3 Configuration points exercised

Per-action `weight`, `cooldownMs`, and `available()`; registry membership; an injectable RNG for deterministic test runs; and an injectable clock. See `src/README.md`.

---

10. Worked Example / Scenario

Setup. A long-running "assistant" agent has a registry of three curiosity actions:

Action	weight	cooldown	available(ctx)
`revisit-stale-goal`	5	30 min	at least one goal not touched in 24h
`summarize-recent-context`	3	15 min	≥ 10 new messages since last summary
`practice-skill`	1	2 h	at least one skill marked "needs practice"

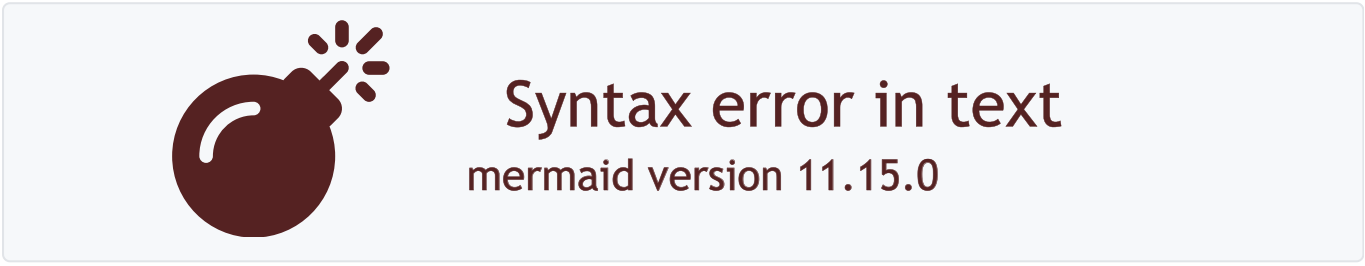
Timeline.

1. `t0`: user sends 3 messages; the agent answers them (reactive). On each tick `isIdle()` is false, so the curiosity engine does nothing.
2. `t1` (idle gap): inbound empty, task queue empty ⇒ `isIdle()` true. Eligible set: `revisit-stale-goal` (a goal is 26h stale) and `practice-skill` (one skill needs practice). `summarize-recent-context` is

excluded by its availability predicate (only 4 new messages). Weighted draw over `{5, 1}` (total 6) selects `revisit-stale-goal` (5/6). A "revisit goal X" task is enqueued; its cooldown is armed for 30 min.

3. `t2` (still idle, 1 min later): `revisit-stale-goal` is now on cooldown ⇒ excluded. Eligible set: `practice-skill`. It is selected and enqueued; cooldown 2 h armed.
4. `t3`: a new user message arrives. `isIdle()` false ⇒ engine silent; the reactive answer is processed ahead of any further curiosity work.

10.1 Sequence diagram (Figure 4)



---

11. Security, Safety & Failure Modes

11.1 Posture

- **Triggering fails CLOSED.** If idle cannot be *proven* (detector error/unknown), the engine treats the agent as busy and does nothing — guaranteeing it never steals cycles from reactive work (protects R1).
- **Selection fails OPEN.** If a single action's predicate or builder throws, that action is dropped and the rest of the eligible set proceeds — protecting liveness (R7).

11.2 Failure-mode table

Failure mode	Cause	Effect if unhandled	Mitigation
Curiosity steals reactive cycles	Engine runs while work pending	Latency on user-facing work	Idle is the *sole* trigger; trigger fails closed (§7.5).
Action spam / thrash	No throttle	Same heavy action re-fires repeatedly	Per-action cooldown gate (R3).
Firing on unmet precondition	No precondition check	Errors, noise	Availability predicate gate (R4).
Starvation of low-weight actions	Weights too skewed	Some actions never run	Weights are relative within the *eligible* set; cooldowns on heavy actions naturally cede turns; tunable via telemetry (R9).
Predicate/builder exception	Buggy action	Loop crash	Per-action try/catch, fail-open (§7.5).
Duplicate firing in a cluster	Two replicas, shared agent	Same action twice	Shared cooldown store + atomic conditional update (§8.3).
Runaway resource use	Curiosity tasks heavy	Cost spike	Tasks flow through the agent's normal budget/safety machinery (R6); curiosity inherits, not bypasses, those limits.
Information leakage	Curiosity action reads/writes data	Privacy issue	Curiosity tasks run under the agent's existing authorization context — no elevated side-channel (R6).

11.3 STRIDE-style notes

Threat	Relevance	Note
Spoofing	Low	No new identity surface; runs as the agent.
Tampering	Med	A poisoned registry could schedule unwanted actions ⇒ registry should be config-controlled and validated.
Repudiation	Low	Firing log (R9) gives an audit trail.
Info disclosure	Med	Curiosity tasks inherit the agent's auth scope; they must not be granted broader access (R6).
DoS	Med	Cooldowns + idle-gating + queue budgets bound resource use.
Elevation	Low	No privilege escalation; queue-mediated, same scope.

---

12. Standards & Framework Mapping

We claim **semantic alignment** with the following frameworks — not certified compliance.

Framework	Relevant element	How this aligns
NIST AI RMF (Govern/Manage)	Bounded, observable autonomous behavior	Curiosity is gated, throttled, precondition-checked, and logged — a managed form of autonomy, not free-running.
ISO/IEC 42001 (AI management systems)	Controlled operation of AI components	Feature-flag, per-action config, and firing audit log provide operational control points.
IEEE 7000-series (ethically aligned design)	Transparency of autonomous initiative	Every proactive action is recorded with reason, supporting explainability of *why the agent acted on its own*.
OWASP for LLM/agent apps	Excessive agency / unbounded action	The three gates + idle trigger + queue budgets directly bound agent "excessive agency."
W3C `requestIdleCallback` philosophy	Run low-priority work only when idle	Adopts the run-when-idle principle, specialized to agent action registries.

No certification is asserted; these are conceptual alignments useful to auditors.

13. Evaluation Methodology

Numbers below are **illustrative** placeholders to define *how* one would measure the mechanism, not benchmark results.

Dimension	Metric	How to measure	Interpretation
Non-interference (R1)	Reactive-latency delta	p95 reactive response time with engine on vs. off	≈ 0 delta ⇒ no interference (illustrative target: < 1% delta).
Idle utilization	Idle-tick action rate	fraction of idle ticks that fired a curiosity action	Higher ⇒ more idle value captured (subject to cooldowns).
Variety	Distinct-action entropy	Shannon entropy of fired action distribution	Higher ⇒ less monotony.
Weight fidelity (R2)	Selection-vs-weight error	observed selection frequency vs. expected weight share within eligible sets	Near-zero ⇒ selection respects weights.
Precondition correctness (R4)	False-fire rate	fraction of firings whose precondition was actually false at fire time	Should be 0.
Cooldown correctness (R3)	Cooldown-violation rate	firings closer than `cooldownMs` for the same action	Should be 0.
Robustness (R7)	Crash rate under fault injection	inject throwing predicates/builders; count host-loop crashes	Should be 0 (fail-open).
Observability (R9)	Log completeness	fraction of decisions with a logged outcome+reason	Should be 1.0.

The reference self-check (§9) operationalizes the non-interference, cooldown, availability, and weight-fidelity rows on a small scale.

14. Novelty & Inventive Claims

The following claims are presented in prose for prior-art purposes (defensive publication — not a patent application). One independent claim and fourteen dependent claims are provided, each dependent claim narrowing on a distinct feature.

Claim 1 (independent)

> A computer-implemented method for triggering intrinsic-motivation actions in an AI agent, the method comprising: **detecting** that a primary work source for the agent is empty; **maintaining** a registry of action descriptors, each descriptor having a weight, a cooldown, and an availability predicate; **forming** an eligible set consisting of those descriptors whose availability predicate evaluates true and whose cooldown has elapsed since the descriptor's most recent firing; **selecting** one descriptor from the eligible set by weighted-random sampling in which each eligible descriptor's selection probability is proportional to its weight normalized over the eligible set; **enqueuing** an action constructed from the selected descriptor into the agent's task queue; and **arming** the selected descriptor's cooldown; **characterized in that** the said selecting and enqueuing are invoked only upon the said detecting that the primary work source is empty.

Dependent claims

Claim 2. The method of claim 1, wherein the said weight, cooldown, and availability predicate are three **independent** gates such that the weight governs probability only among descriptors already admitted to the eligible set, while the cooldown and availability predicate govern membership in the eligible set.

Claim 3. The method of claim 1, wherein the weighted-random sampling **renormalizes** selection probability over the eligible set such that descriptors excluded by cooldown or availability contribute no probability mass.

Claim 4. The method of claim 1, wherein the availability predicate is a runtime function evaluated against a read-only context snapshot describing the agent's current state, and a descriptor whose predicate throws an exception is **excluded** from the eligible set without aborting the selection.

Claim 5. The method of claim 1, wherein the cooldown is enforced by comparing a current timestamp against a stored most-recent-firing timestamp for the descriptor, an absent stored timestamp being treated as eligible.

Claim 6. The method of claim 1, wherein the enqueued action is processed by the same task queue, execution, authorization, and budget machinery as reactive tasks, such that a reactive task subsequently enqueued may be ordered ahead of the curiosity action.

Claim 7. The method of claim 1, wherein the detecting of an empty primary work source comprises confirming that both an inbound message channel set and a reactive task queue are empty for the agent.

Claim 8. The method of claim 1, wherein the said detecting **fails closed** such that, if emptiness of the primary work source cannot be confirmed, the selecting and enqueuing are not invoked.

Claim 9. The method of claim 1, wherein the registry is configurable at runtime to add or remove descriptors without modifying the selecting logic.

Claim 10. The method of claim 1, further comprising recording, for each invocation, a firing-log entry capturing the selected descriptor identifier, a timestamp, and an outcome, to support audit and weight tuning.

Claim 11. The method of claim 1, wherein, when the eligible set is empty, the method performs no enqueuing and leaves the agent idle.

Claim 12. The method of claim 1, wherein the agent comprises a plurality of replicas sharing a common logical identity, and the cooldown is enforced via a **shared** most-recent- firing store updated atomically so that no two replicas enqueue the same descriptor within its cooldown window.

Claim 13. The method of claim 1, wherein the weighted-random sampling uses an injectable pseudo-random source and an injectable clock, enabling deterministic replay for testing.

Claim 14. The method of claim 1, wherein at least one descriptor's action comprises one of: revisiting a stale goal, summarizing accumulated conversational context, surfacing a forgotten follow-up, or practicing a declared skill.

Claim 15. The method of claim 1, wherein the entire mechanism is gated behind a single feature flag whose disabling suppresses all curiosity selection while leaving the agent's reactive operation unchanged.

---

15. Limitations & Threats to Validity

- **Constituent ideas are well-trodden.** Weighted-random selection, cooldowns, precondition gating, and run-when-idle all exist independently (§4–§5). The contribution is their specific *combination and triggering discipline*, not any single gate.
- **"Curiosity" is a chosen metaphor.** This is *not* RL intrinsic-motivation curiosity; the word denotes proactive self-directed activity, not learned reward. Readers expecting an RL mechanism should note the distinction (§2.4, §5).
- **Tuning is empirical.** Good weights/cooldowns/predicates are domain-specific and require observation; this disclosure provides the mechanism, not tuned constants.
- **Clustered correctness depends on a shared store.** Single-process correctness is simple; multi-replica correctness requires the shared atomic cooldown store of Claim 12.
- **Idle detection is only as good as the work-source model.** If a channel is omitted from the primary-work definition, the engine can mis-judge idleness.
- **Withheld.** Specific tuned weight/cooldown values and domain-specific predicate libraries from production deployments are intentionally not disclosed; only the mechanism is.

---

16. Future Work & Open-Source Reference App

A minimal **open-source reference application** is planned: a framework-agnostic library (plus optional sidecar) implementing the engine, with adapters for common agent loops and a table-driven registry. See

docs/OPEN-SOURCE-APP.md for the design and a generic Kubernetes/AKS deployment sketch (no internal cluster identifiers).

Roadmap sketch: (1) extract the clean-room core into a published package; (2) add a shared cooldown store adapter (SQL/KV) for clustered agents; (3) add a telemetry/tuning dashboard for weight/cooldown adjustment; (4) provide host adapters; (5) publish illustrative tuning case-studies.

---

17. Conclusion

Idle time in long-running reactive agents is a recoverable resource. By making *empty-primary-work* the **sole** trigger, by describing each proactive action with three independent gates (**weight, cooldown, availability**), by performing a **weighted-random draw restricted to and renormalized over the eligible set**, and by **enqueuing** the result into the agent's normal task queue, an agent gains varied, non-repetitive, precondition-respecting proactivity *without* interfering with its reactive duties. This document discloses that mechanism completely and publicly, with an enabling clean-room implementation, to establish prior art and keep the technique freely practiceable by all.

Appendix A — Prior-Art Landscape

A.1 Well-trodden (treat as known)

- Weighted-random / roulette-wheel selection (GAs, loot tables).
- Per-action cooldown timers (games, rate limiters).
- Precondition-gated action selection (GOAP, behavior trees, utility AI considerations).
- Run-when-idle scheduling (`requestIdleCallback`, OS idle tasks, cron/sleep-time agents).
- RL intrinsic motivation / curiosity (Schmidhuber; ICM, Pathak 2017; RND, Burda 2018).
- Proactive LLM agents (Generative Agents, Park 2023).

A.2 Candidate-novel (the disclosed delta)

- **Idle-detection-of-the-primary-work-source as the *sole* trigger** for proactive selection.
- A **three-gate (weight + cooldown + availability) action registry** in which the gates compose into an eligible set.
- **Weighted-random selection *restricted to and renormalized over* the eligible set.**
- **Queue-mediated execution** so curiosity inherits the agent's pre-emption, safety, and budget semantics rather than running in a privileged side-channel.

A.3 Honesty attestation

The prior-art searches behind this document are **directional, not exhaustive.** They reflect a good-faith review of game-AI, RL, and agent-orchestration literature and products as of the publication date. No claim is made that every relevant reference has been found. This attestation is itself part of the defensive-publication record: the value of the publication is the **dated public disclosure of the mechanism**, independent of search completeness.

Appendix B — Glossary

Term	Definition
Reactive agent	An agent that advances only in response to inputs (messages, tasks, events).
Cognitive loop	The agent's perceive–reason–act control loop.
Primary work source	The union of inputs the agent reacts to (inbound channels + reactive task queue).
Idle	State in which the primary work source is empty.
Curiosity action	A self-directed, proactive action an agent may take when idle.
Action descriptor	A registry entry: `{ id, weight, cooldown, available, build }`.
Weight	Relative selection probability of an action within the eligible set.
Cooldown	Minimum time before the same action may re-fire.
Availability predicate	A runtime precondition function controlling eligibility.
Eligible set	Descriptors passing both the availability predicate and the cooldown gate at a tick.
Weighted-random selection	Choosing an item with probability proportional to its weight.
Queue-mediated	Executed by enqueuing into the agent's normal task queue.
Fail-open / fail-closed	Resolve errors toward continued operation / toward inaction, respectively.

Appendix C — Reference-Implementation Index

File	Purpose
`src/curiosity-engine.js`	Engine core: registry, eligibility (cooldown + availability), eligibility-restricted weighted-random selection, enqueue, cooldown arming, fail-open/closed, telemetry hook.
`src/example-agent.js`	Mock agent host + three illustrative curiosity actions + driver loop + runnable self-check (asserts R1, R3, R4, weight-fidelity).
`src/README.md`	What the sample shows, how to run it, and the clean-room/illustrative disclaimer.

Appendix D — Defensive-Publication Deposit & Timestamp

- **Publication date:** 2026-06-25
- **Publisher / copyright holder:** Gus IT LLC (Florida, USA)
- **Author:** Gustavo Assuncao, PhD
- **Version:** 1.0
- **Deposit channel:** to be assigned (IP.com / Zenodo / arXiv) — establishes a public, dated, citable prior-art record. No DOI is asserted here.
- **Intent:** This disclosure is intentionally public to bar later patenting of the described mechanism (or trivial variants) by others, and to keep the technique freely practiceable. The dated public availability of this document is the operative act establishing prior art.