Pre-Coding Essentials (Component: crates/vm\_pipeline/src/resolve\_ties.rs, Version/FormulaID: VM-ENGINE v0) — 56/89

1. Goal & Success

Goal: Resolve only ties that **block a decision** (WTA winner, last seat, IRV elimination) using the configured policy and, if needed, a **seeded deterministic RNG**, and emit a **TieLog** consumed by Result/Report.

Success: With the same inputs (and same **tie\_seed** when policy = random), tie outcomes and TieLog are **byte-identical** across OS/arch.

1. Scope

In scope: Policy order and contexts; deterministic selection; TieLog entries. If gates failed earlier, enter here **only** to log a blocking tie (if any).

Out of scope: Tabulation/quotients, gates math, frontier mapping, labeling (handled in other stages).

1. Inputs → Outputs (with schemas/IDs)

Inputs

* Pending **TieContext** items from prior stages (e.g., “WTA winner in U:…”, “last seat in …”, “IRV elimination …”).
* **ParameterSet** snapshot (notably **VM-VAR-032 tie\_policy**, **VM-VAR-033 tie\_seed**).

Outputs

* **TieLog** entries embedded in **Result**; **tie\_seed** echoed in **RunRecord** when used.
* A mapping of resolved winners for the caller to finalize the blocked step.

1. Entities/Tables (minimal)

(N/A — structures are local to the tie stage and serialized into Result/RunRecord per Doc 5.)

1. Variables (used here)

* **VM-VAR-032 tie\_policy** ∈ {status\_quo, deterministic, random} (default: status\_quo)
* **VM-VAR-033 tie\_seed** ∈ integer (≥ 0) (default: 0) — used **only** when tie\_policy = random

1. Functions (signatures only)

pub enum TiePolicy {

StatusQuo,

Deterministic,

Random { seed: u64 }, // constructed from VM-VAR-033

}

pub enum TieKind { WtaWinner, LastSeat, IrvElimination }

pub struct TieContext {

pub kind: TieKind,

pub unit: UnitId,

pub candidates: Vec<OptionId>,

}

pub struct TieLogEntry {

pub context: String, // human-readable

pub candidates: Vec<OptionId>, // sorted, stable

pub policy: &'static str, // "status\_quo" | "deterministic" | "random"

pub detail: &'static str, // "order\_index" | "seed"

pub seed: Option<u64>, // present iff random

pub winner: OptionId,

}

pub fn resolve\_ties(

contexts: &[TieContext],

order\_index: &BTreeMap<OptionId, u32>, // deterministic key

policy: TiePolicy

) -> (Vec<TieLogEntry>, BTreeMap<(TieKind, UnitId), OptionId>);

// helpers

fn pick\_status\_quo(cands: &[OptionId], is\_sq: &BTreeSet<OptionId>) -> Option<OptionId>;

fn pick\_by\_order(cands: &[OptionId], order\_idx: &BTreeMap<OptionId,u32>) -> OptionId;

fn pick\_by\_rng(cands: &[OptionId], seed: u64) -> OptionId; // ChaCha20; reproducible

1. Algorithm Outline

**Iteration order**  
Process contexts in a **stable order** (as provided: already sorted by (kind, unit, candidates) upstream).

**Policy application**  
For each context:

a) status\_quo → if any candidate has is\_status\_quo = true, choose it; if none, **fall through** to deterministic.

b) deterministic → choose the smallest (order\_index, OptionId) among candidates (uses **Option.order\_index**).

c) random → initialize **ChaCha20** with **tie\_seed (VM-VAR-033)**; draw **uniformly** among candidates; **log** the seed in the entry.

**Logging**  
Emit TieLogEntry { context, candidates, policy, detail, seed?, winner }.

* detail = "order\_index" for deterministic; detail = "seed" for random (with seed set).

Return both: the **TieLog** vector and a map of resolved winners keyed by (TieKind, UnitId).

1. State Flow

Pipeline: … → **MAP\_FRONTIER** → **RESOLVE\_TIES** *(only if blocking)* → **LABEL\_DECISIVENESS** → **BUILD\_RESULT** → **BUILD\_RUN\_RECORD**.  
If gates **failed**, caller may still enter here **only** to log a blocking tie.

1. Determinism & Numeric Rules

* Stable iteration orders; **integer-only** logic; no floats.
* RNG: **ChaCha20** seeded **only** by **VM-VAR-033 tie\_seed** (no OS RNG/time; no parallel RNG).
* Same inputs + same seed ⇒ **identical winners & logs** across OS/arch.

1. Edge Cases & Failure Policy

* If status\_quo policy but no SQ in candidates → fall through to deterministic.
* Missing seed when policy=random → configuration error; **never** fallback to OS randomness.
* Condorcet cycle ≠ “tie” here (resolved upstream by completion rule).
* Threshold equality (e.g., exactly 55%) is **not** a tie.

1. Test Checklist (must pass)

* **Deterministic order:** candidates {A,B}, policy=deterministic ⇒ winner has lower order\_index (A before B).
* **Seeded RNG:** policy=random, seed=1337 ⇒ two runs produce identical winners and TieLog rows.
* **Context coverage:** WTA winner tie, last-seat tie, and IRV elimination tie each yield a valid entry and unblock the pipeline.
* **Result/RunRecord wiring:** TieLog appears in Result; **tie\_seed** appears in RunRecord when used.