# **Pre-Coding Essentials (Component: crates/vm\_algo/src/allocation/sainte\_lague.rs, Version/FormulaID: VM-ENGINE v0) — 44/89**

## **1) Goal & Success**

Goal: Implement **Sainte-Laguë (highest averages, favor small)**: sequential awards using **odd divisors 1,3,5,…**, after applying the **PR entry threshold**. Deterministic and integer-only.

Success: Seat vector per Unit sums to m; below-threshold options excluded; last-seat ties resolved per policy. Baselines match tests (e.g., 1–2–3–4 with m=10; and 3–2–2 in the convergence case).

## **2) Scope**

In scope: Per-Unit Sainte-Laguë allocation, threshold filter, quotient loop with odd divisors, deterministic/reproducible tie handling.

Out of scope: Tabulation, aggregation, gates/frontier, any I/O.

## **3) Inputs → Outputs**

Inputs:

seats: u32 (Unit.magnitude ≥1)

scores: &BTreeMap<OptionId,u64> (natural tallies)

options: &[OptionItem] (gives (order\_index, id) and status-quo flag)

threshold\_pct: u8 (**VM-VAR-012**)

tie\_policy: TiePolicy, optional rng: &mut TieRng when random

Output: BTreeMap<OptionId,u32> where the sum equals seats.

## **4) Entities/Tables (minimal)**

## **5) Variables (used here)**

## **6) Functions (signatures only)**

rust

CopyEdit

use std::collections::BTreeMap;

use vm\_core::{

ids::OptionId, entities::OptionItem,

rng::TieRng, variables::TiePolicy,

};

/// Sainte-Laguë allocation (odd divisors 1,3,5,…).

pub fn allocate\_sainte\_lague(

seats: u32,

scores: &BTreeMap<OptionId, u64>,

options: &[OptionItem],

threshold\_pct: u8,

tie\_policy: TiePolicy,

rng: Option<&mut TieRng>,

) -> Result<BTreeMap<OptionId, u32>, AllocError>;

// helpers

fn filter\_by\_threshold(scores: &BTreeMap<OptionId,u64>, threshold\_pct: u8) -> BTreeMap<OptionId,u64>;

fn next\_award(

seats\_so\_far: &BTreeMap<OptionId,u32>,

eligible\_scores: &BTreeMap<OptionId,u64>,

options: &[OptionItem],

tie\_policy: TiePolicy,

rng: Option<&mut TieRng>,

) -> OptionId; // argmax of v / (2\*k + 1) via integer cross-multiplication

## **7) Algorithm Outline (implementation plan)**

**Threshold filter**: drop options **strictly below** threshold\_pct share (share computed from ballot’s **natural** totals).

**Init**: alloc[opt]=0 for all **eligible** options; keep options ordered by (order\_index, id) for deterministic scans.

**Seat loop** (seats times): for each eligible opt, compute quotient q = scores[opt] / (2\*alloc[opt] + 1); pick the max using **integer cross-multiplication** (no floats). Ties: higher raw score first; if still tied, deterministic order; if tie\_policy=random, draw with seeded RNG.

**Finish**: return alloc (sum==seats). Provide optional award trail for tests.

## **8) State Flow**

Called by **AllocateUnit** after Tabulate; before aggregation; respects threshold and tie rules from Doc 4B/4C.

## **9) Determinism & Numeric Rules**

Integer comparisons only; stable option ordering; RNG used **only** if tie\_policy=random (seeded, reproducible). (General tie behavior per allocation spec.)

## **10) Edge Cases & Failure Policy**

seats == 0 ⇒ empty allocation.

No eligible options after threshold ⇒ AllocError::NoEligibleOptions.

All scores == 0 with seats > 0 ⇒ allocate entirely by tie policy (deterministic order unless random).

Use u128 when cross-multiplying to avoid overflow on extreme inputs.

## **11) Test Checklist (must pass)**

**VM-TST-001**: A/B/C/D = 10/20/30/40, m=10 ⇒ seats **1/2/3/4**.

**VM-TST-003 (convergence)**: A/B/C shares 34/33/33, m=7 ⇒ seats **3/2/2**.

Determinism: permuting input map/iteration yields identical results due to canonical ordering.

Threshold behavior: raising threshold\_pct excludes sub-threshold options from any seat award.