# **Pre-Coding Essentials (Component: crates/vm\_core/src/rounding.rs, Version/FormulaID: VM-ENGINE v0) — 27/89**

## **1) Goal & Success**

Goal: Provide **overflow-safe integer/rational utilities** for comparisons and rounding with the **round-half-to-even** rule where the spec permits it.

Success: No floats anywhere; comparisons don’t overflow; “half” cases resolve with banker's rounding; helpers cover gate checks and report formatting.

## **2) Scope**

In scope: Ratio helpers (normalize/simplify), overflow-safe compare, half-even rounding to **integer** and to **one decimal percent** (for reporting), % threshold comparisons.

Out of scope: seat allocation math (lives in vm\_algo), serialization (in vm\_io).

## **3) Inputs → Outputs**

Inputs: integer pairs (num, den) with den>0.

Outputs: orderings, booleans (≥ threshold), rounded integers/decimals (for report layer).

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

## **5) Variables**

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

rust

CopyEdit

/// Reduce and normalize: gcd>0, den>0

pub fn simplify(num: i128, den: i128) -> (i128, i128);

/// Overflow-safe compare of a/b vs c/d using Euclid/continued-fraction method.

pub fn cmp\_ratio(a\_num: i128, a\_den: i128, b\_num: i128, b\_den: i128) -> core::cmp::Ordering;

/// Compare a/b against integer percent p (0..=100) without floats.

pub fn ge\_percent(a\_num: i128, a\_den: i128, p: u8) -> bool;

/// Banker's rounding of a/b to nearest integer.

pub fn round\_nearest\_even\_int(num: i128, den: i128) -> i128;

/// Banker's rounding of (a/b)\*100 to \*\*one decimal place\*\*; returns tenths of a percent (0..=1000).

pub fn percent\_one\_decimal\_tenths(num: i128, den: i128) -> i32;

/// Compare with half-even at the boundary: true if a/b >= p% with "exact half" resolving to even integer.

pub fn ge\_percent\_half\_even(a\_num: i128, a\_den: i128, p: u8) -> bool;

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

**simplify**

If den==0 → error (panic or Result; choose consistent API).

Move sign to numerator: if den<0 then num=-num; den=-den.

Compute g = gcd(|num|, den) (binary GCD); return (num/g, den/g).

**cmp\_ratio** (no overflow)

Handle signs and zeros early.

Use **continued-fraction style comparison**:  
  
 bash  
CopyEdit  
// compare a/b ? c/d with a,b,c,d >= 0, b,d>0

loop {

let (qa, ra) = (a / b, a % b);

let (qc, rc) = (c / d, c % d);

if qa != qc { return qa.cmp(&qc); }

if ra == 0 || rc == 0 { return (ra == 0 && rc == 0).then\_some(Equal).unwrap\_or((ra==0).cmp(&(rc==0)).reverse()) }

// invert remainders

a = d; b = ra;

c = b\_old; d = rc;

}

Or equivalently, apply cross-cancel trick: a/g1 \* (d/g2) vs (c/g1) \* (b/g2) with g1=gcd(a,c), g2=gcd(b,d) then checked\_mul; if any checked\_mul overflows, fall back to the Euclid method.

**ge\_percent**

Compare 100 \* num >= p as i128 \* den using **cross-cancel** to avoid overflow:  
  
 rust  
CopyEdit  
let (num, den) = simplify(num, den);

let g1 = gcd(num.abs(), 100);

let g2 = gcd(den, p as i128);

// compare (100/g1)\*num vs (p/g2)\*den

All in i128; short-circuit on zeros.

**round\_nearest\_even\_int** (banker’s round)

Compute q = num / den, r = num % den on normalized (num,den).

If 2\*|r| < den → return q.

If 2\*|r| > den → return q + sign(num).

Else **exact half**: return the **even** of q and q + sign(num) (i.e., if q is odd, bump toward sign; if even, keep).

**percent\_one\_decimal\_tenths** (for reporting)

We want round\_half\_even((num\*1000)/den) as an integer **tenths of a percent** in 0..=1000.

Use cross-cancel to avoid overflow: reduce by g=gcd(num,den); split the multiply by 125 and 8 where helpful; use checked\_mul and, on overflow, do long division with remainder and apply half-even manually.

**ge\_percent\_half\_even**

Let target be p% → compare rounded-to-integer percentages with half-even:

Compute x = round\_nearest\_even\_int(num\*100/den).

Return x >= p.

Use the **same** half-even rule as round\_nearest\_even\_int to ensure a boundary at exactly .5% resolves to the nearest even percent.

## **8) State Flow**

Algorithms and gates call cmp\_ratio/ge\_percent (or ge\_percent\_half\_even where the spec mandates half-even).

Report layer uses percent\_one\_decimal\_tenths to render one-decimal percentages without re-rounding elsewhere.

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

Pure integer math; no floats; outcomes identical across OS/arch.

Half-even only where explicitly allowed; otherwise use exact rational comparison.

Denominators always positive; signs normalized in one place.

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

den == 0 → return Err(NumericError::ZeroDenominator) (prefer Result API) or debug\_assert! + panic in internal-only paths—pick one and keep consistent.

Extremely large num,den that overflow on mul → fall back to Euclid comparison path.

Negative num (shouldn’t happen with counts) still well-defined with sign normalization.

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

**Compare without overflow:**

cmp\_ratio(1,3, 333333333333333333, 999999999999999999) = Equal.

Random property tests vs num-rational (dev-only) on moderate ranges.

**Half-even integer rounding:**

round\_nearest\_even\_int(5,2) == 2 (2.5 → 2), round\_nearest\_even\_int(3,2) == 2 (1.5 → 2), round\_nearest\_even\_int(7,2) == 4 (3.5 → 4).

**Percent threshold:**

ge\_percent(55,100,55) true; ge\_percent(549,1000,55) false; edge with exact half using ge\_percent\_half\_even behaves per banker's rule.

**One-decimal percent:**

(1,3) → 33.3 tenths=333; (2,3) → 66.7 tenths=667; (1,8) → 12.5 tenths rounds to **12.5** → 125 (half-even unaffected).

**Determinism:** repeated runs produce identical outputs for all helpers.