SlimeChain ? Musk-mode Rust Wrappers

Overview

Rust helpers to apply Musk-mode in your app/chain: parameter tuning, USD?SOCIAL pricing via oracle, tier discounts, C_min enforcement, and DM escrow calculation.

Code (Rust)

```
// musk_mode.rs ? helper wrappers to apply Musk-mode at the app/chain edge
use slimechain_algo::*;
#[derive(Clone, Copy, Debug)]
pub enum Tier { T0, T1, T2, T3 }
pub struct TierPolicy {
    pub discounts: (f64, f64, f64, f64),      // T0..T3
    pub risk_factor: (f64, f64, f64, f64),   // multiply risk by factor
    pub dm_escrow_usd: f64,
    pub cmin_usd: f64,
}
pub trait PriceOracle {
    fn usd_per_social(&self) -> Option<f64>;
    fn usd_per_usdc(&self) -> Option<f64> { Some(1.0) }
}
pub fn apply_musk_mode_params(p: &mut Params) {
    p.q_weights.w_h = 0.25;
    p.propagation.ttl_base = 5.0;
    p.propagation.fanout_base = 6.0;
    p.propagation.k1 = 3.0;
    p.propagation.k2 = 3.0;
    p.cost.alpha = 0.8;
    p.cost.beta = 0.5;
    p.cost.a = 1.4;
    p.cost.b = 0.6;
    p.cost.lambda_actor = 0.8;
    p.cost.lambda_content = 0.6;
    p.reward.mu = 0.5;
}
pub fn tier_discount(tier: Tier, policy: &TierPolicy) -> f64 {
    match tier {
        Tier::T0 => policy.discounts.0,
        Tier::T1 => policy.discounts.1,
        Tier::T2 => policy.discounts.2,
        Tier::T3 => policy.discounts.3,
    }
}
pub fn tier_risk_factor(tier: Tier, policy: &TierPolicy) -> f64 {
    match tier {
        Tier::T0 => policy.risk_factor.0,
        Tier::T1 => policy.risk_factor.1,
        Tier::T2 => policy.risk_factor.2,
        Tier::T3 => policy.risk_factor.3,
    }
}
/// Convert a USD amount to SOCIAL using oracle; fallback to a fixed peg if needed.
pub fn usd_to_social(usd: f64, oracle: &dyn PriceOracle, fallback_usd_per_social: f64) -> f64 {
    let px = oracle.usd_per_social().unwrap_or(fallback_usd_per_social).max(1e-9);
    usd / px
}
```

```
/// Compute final posting cost with C_min and tier discount. Risk attenuation is handled by params
(k1/k2 etc.).
pub fn compute_final_cost_with_tier(
    actor: &Actor,
    content: &Content,
    params: &Params,
    basefare: f64,
    tier: Tier,
    policy: &TierPolicy,
    oracle: &dyn PriceOracle,
    fallback_usd_per_social: f64,
) -> f64 {
    let mut cost = calculate_post_cost(actor, content, params, basefare);
    // enforce C_min in SOCIAL
    let cmin_social = usd_to_social(policy.cmin_usd, oracle, fallback_usd_per_social);
    if cost < cmin_social { cost = cmin_social; }
    // apply tier discount
    let disc = tier_discount(tier, policy);
    cost * disc
}
/// DM escrow fee in SOCIAL (payer-side hold). Receiver may auto-refund according to policy.
pub fn dm_escrow_social(policy: &TierPolicy, oracle: &dyn PriceOracle, fallback_usd_per_social: f64)
 -> f64 {
    usd_to_social(policy.dm_escrow_usd, oracle, fallback_usd_per_social)
}
// --- example stub oracle for tests ---
pub struct StubOracle { pub usd_per_social_px: Option<f64> }
impl PriceOracle for StubOracle {
    fn usd_per_social(&self) -> Option<f64> { self.usd_per_social_px }
}
// --- quick test ---
#[cfg(test)]
mod tests {
    use super::*;
    #[test]
    fn t_costs() {
        let mut p = Params::default();
        apply_musk_mode_params(&mut p);
        let actor = Actor{ rl:120.0, q:0.8, ef:30.0, posts_1h:Some(12.0) };
        let content = Content{ is_claim:Some(true), has_evidence:Some(false), risk_signals:None };
        let pol = TierPolicy{
            discounts: (1.0, 0.95, 0.85, 0.7),
            risk_factor: (1.0, 0.95, 0.9, 0.8),
            dm_escrow_usd: 0.003,
            cmin_usd: 0.005,
        };
        let oracle = StubOracle{ usd_per_social_px: Some(0.2) }; // 1 SOCIAL = $0.2
        let basefare = 1.0;
        let c0 = compute_final_cost_with_tier(&actor, &content, &p, basefare, Tier::T0, &pol,
&oracle, 0.2);
        let c3 = compute_final_cost_with_tier(&actor, &content, &p, basefare, Tier::T3, &pol,
&oracle, 0.2);
        assert!(c3 < c0);
        let dm = dm_escrow_social(&pol, &oracle, 0.2);
        assert!(dm > 0.0);
    }
}
```

Integration Notes

- Risk attenuation by tier can be applied by multiplying the risk signals before calling propagation.
- Consider persisting tier states off-chain (cache) for UI, but treat the chain as source of truth.
- For production, replace StubOracle with chain oracle queries; add retries and circuit breaker.