Implementation Plan: Multi-Agent Dealer Market Simulator

Based on the paper: "Multi-Agent Simulation for Pricing and Hedging in a Dealer Market" (Ganesh et al., 2019).

1. Project Architecture

The system should follow an OpenAI Gym-like structure for the environment, with modular classes for different agent types.

Core Modules:

- `MarketEnvironment` : Manages global state (Time, Mid-price, Reference Spread).
- `Exchange` : Implements the LOB analytical approximation for $S_{ref}(v)$.
- `InvestorAgent` : Implements trade generation logic and MM selection.
- `MarketMakerAgent` : Implements Tiering, Pricing, and Almgren-Chriss Hedging.
- `SimulationRunner` : Orchestrates the discrete-time steps and reward logging.

2. Phase 1: Exogenous Market Environment

Implement the external variables that drive the simulation.

2.1 Mid-price Dynamics (P_t)

- Model P_t as Geometric Brownian Motion (GBM).
- **Update Rule:** $P_t = P_{t-1} \cdot \exp((\mu - 0.5\sigma^2)\Delta t + \sigma\sqrt{\Delta t}Z)$ where $Z \sim N(0,1)$.
- **Time Step (Δt):** 15 minutes.

2.2 Reference Spread ($s_{0,t}$)

- Sample at each step from $N(\mu = 0.00015, \sigma = 0.00005)$.
- Clip values to the range $[0.00002, 0.0005]$.

2.3 Reference Price Curve ($S_{ref,t}(v)$)

- Implement Equation 13 from the Appendix.
- **Formula:** - If $\lambda = 2$: $S_{ref}(v) = -\frac{s_0}{2}\frac{1}{\tilde{v}}\ln(1-\tilde{v})$
  - If $\lambda \neq 2$: $S_{ref}(v) = \frac{s_0}{2}\frac{\omega}{\tilde{v}}(1-(1-\tilde{v})^{1/\omega})$
- **Parameters:** $\lambda = 1.6$, $\omega = (\lambda - 1)/(\lambda - 2)$, $\tilde{v} = v/v_{max}$.
- v_{max} should be set to a value larger than any expected trade (e.g., 10,000).

3. Phase 2: Investor Agents

Investors are price-takers who provide the trade flow.

3.1 Trade Generation

- **Arrival:** At each step, trade occurs with probability p_j^{trade}.
- **Size (v):** Sample from Log-Normal distribution (μ_j, σ_j).
- **Direction:** 50% Buy / 50% Sell.
- **Sophisticated Investors:** Implement an "Oracle" flag that, with probability q_j, sets the trade direction to align with the price move over the next t_m steps.

3.2 Market Maker Selection

- Query all Market Makers for their spread $s_{i,t}(v, u_{i,j})$.
- Execute with the MM providing the minimum spread.

4. Phase 3: Market Maker Agent (The Core)

MMs act as intermediaries and risk managers.

4.1 Tiering Policy

- **State:** Track `yield_history` for every participant j.
- **Metric:** `Average Yield` = Exponentially Weighted Moving Average of (Revenue / Volume).
- **Action:** At each step, sort participants by yield and assign to K tiers (e.g., $K = 5$).

4.2 Pricing Policy

- **Formula:** $s(v, u) = S_{ref}(0) \cdot (\frac{S_{ref}(v)}{S_{ref}(0)})^{\alpha} + \delta_{tier} \cdot u$
- $S_{ref}(0)$ is $s_0/2$.
- α is the volume penalty (experiment with 1.1 to 1.7).
- δ_{tier} is the fixed tier penalty.

4.3 Hedging Policy (Almgren-Chriss)

Implement the Value-at-Risk (VaR_p) optimization.

- **Goal:** Minimize $VaR_p = E[C] + \gamma\sqrt{Var(C)}$.
- **Inputs:** Net position $z_{i,t}$, volatility σ, risk aversion γ, and max hedge time N_{max}.
- **Immediate Hedge Action:** $v_{hedge} = z_{i,t} \cdot x_0$, where x_0 is the first step of the optimized liquidation schedule.

5. Phase 4: Reward Engine

Calculate the four components of the MM reward at each step t:

1. **Spread Revenue:** $\sum(s_{i,t} \cdot v_{investor})$
2. **Position Revenue:** $\sum(P_{t+t_m} - P_t) \cdot \tilde{v}_{trade}$ (Realized after delay t_m)
3. **Hedging Cost:** $-\sum(s_{j,t} \cdot v_{hedge})$ (Paid to other MMs)

4. **Risk Cost:** $\min(z_{i,t} \cdot (P_t - P_{t-1}), 0)$

6. Phase 5: Experiments & Validation

To confirm the implementation is correct, run the following tests:

1. **Internalization Test:** Verify that an MM with 100% market share has a lower `Average Net Position / Total Volume` than an MM with 50% share.
2. **Price Sensitivity Test:** Plot the market share of an investor as the MM manually moves them from Tier 0 to Tier 4.
3. **Volatility Test:** Compare total rewards for different γ (Risk Aversion) values in high vs. low volatility regimes.

Implementation Tips for Coding LLMs

- Use **NumPy** for all stochastic sampling and vector calculations.
- Use **Pandas** to log agent states and trade histories for analysis.
- Use **Scipy.optimize** if you choose to solve the Almgren-Chriss schedule numerically, though the paper suggests a heuristic linear solution is possible.
- Ensure the `MarketEnvironment` keeps a circular buffer of P_t to resolve `Position Revenue` after t_m steps.