Artificial Order Generation System Documentation

Overview

The artificial order generation system creates realistic trading orders for a stock market simulation by mimicking real market behaviors through multiple interconnected mechanisms: market events, stock recovery patterns, Perlin noise for organic variation, and queue-based stock rotation.

System Architecture

Core Configuration

- **Order Frequency**: One cycle every 5 seconds (5000ms)
- **Orders Per Cycle**: 10 orders generated simultaneously
- **Base Buy Ratio**: 60% buy orders, 40% sell orders by default
- **Price Boundaries**: Orders stay within ±7% of reference price (93% to 107%)
- **Order Distribution**: 20% market orders, 80% limit orders

Stock Queue Management

The system maintains a rotating queue of all available stocks to ensure fair distribution:

1. Initializes queue with all valid stocks (those with numerical prices)
2. Processes stocks sequentially from front of queue
3. Moves processed stock to back of queue
4. Refills queue when empty to maintain continuous operation

Order Generation Mechanisms

1. Perlin Noise Integration

Purpose: Creates organic, realistic market fluctuations that avoid purely random behavior.

Implementation Details:

- Uses 2D Perlin noise with time step of 0.01 per order
- Modulates buy ratio by ±20% around the base 60% (resulting range: 40%-80%)
- Applies subtle price pressure between 99.5% and 100.5% of calculated price
- Each stock gets unique noise seed based on `(stock_id % 10)` for variation
- Time progresses continuously: `perlinTime += 0.01` after each order

Mathematical Formula:

```
buyRatioModulation = (perlinNoise - 0.5) * 0.4
finalBuyRatio = clamp(0.6 + buyRatioModulation, 0.1, 0.9)
priceInfluence = 0.995 + (perlinNoise * 0.01)
```

2. Market Events System

Event Types and Probabilities:

Tech Boom (5% chance per cycle)

- **Affected Industries**: Technology only
- **Duration**: 12 cycles (60 seconds)
- **Buy Ratio Override**: 80% buy orders
- **Price Modifier**: 103% (upward pressure)
- **Description**: Simulates technology stock bull runs

Financial Crisis (4% chance per cycle)

- **Affected Industries**: Financials only
- **Duration**: 15 cycles (75 seconds)
- **Buy Ratio Override**: 20% buy orders (80% sell)
- **Price Modifier**: 96% (downward pressure)
- **Description**: Banking sector sell-offs

Market Crash (2% chance per cycle)

- **Affected Industries**: All sectors
- **Duration**: 10 cycles (50 seconds)
- **Buy Ratio Override**: 15% buy orders (85% sell)
- **Price Modifier**: 94% (strong downward pressure)
- **Description**: Widespread panic selling

Recovery Rally (3% chance per cycle)

- **Affected Industries**: All sectors
- **Duration**: 20 cycles (100 seconds)
- **Buy Ratio Override**: 75% buy orders
- **Price Modifier**: 102% (upward pressure)
- **Description**: Market-wide recovery after downturns

Event Management Rules:

- Maximum 2 simultaneous active events
- Events cannot duplicate (same event type cannot run twice)
- Event effects completely override base buy ratios
- Price modifiers are multiplicative with other factors

3. Stock Recovery Mechanism

Trigger Conditions:

- Monitors each stock's price movement across cycles
- Tracks consecutive downward price movements

- Initiates recovery after 5 consecutive down cycles

Recovery Parameters:

- **Recovery Strength**: Random value between 60%-90% buy ratio
- **Recovery Duration**: Random duration between 8-12 cycles
- **Price Boost**: Additional 1%-3% upward price modifier
- **Priority**: Recovery mode overrides market events and base ratios

Slump Pressure Logic: When a stock has 3+ consecutive down moves AND is not in recovery mode AND the order would be a sell:

```
pressureFactor = 0.98 - (min(consecutiveDownCount, 10) * 0.005)
finalPrice = max(floorPrice, calculatedPrice * pressureFactor)
```

This creates additional downward price pressure during extended declines.

Order Type Distribution

Market Orders (20% of all orders)

- **Buy/Sell Ratio**: Follows the calculated buy ratio (influenced by Perlin noise, events, or recovery)
- **Price**: No price specified (executes at market price)
- **Logic**: `Math.random() > buyRatio ? 'Market Sell' : 'Market Buy'`

Limit Orders (80% of all orders)

- **Buy/Sell Ratio**: Fixed 50/50 distribution regardless of other factors
- **Price**: Calculated price within ±7% bounds
- **Logic**: `Math.random() < 0.5 ? 'Limit Buy' : 'Limit Sell'`

Price Calculation Process

Step-by-Step Price Determination:

1. **Base Price Range**:

   ```
   referencePrice = stock.latestPrice
   floorPrice = referencePrice * 0.93  // 7% below
   ceilPrice = referencePrice * 1.07   // 7% above
   ```

2. **Random Price Selection**:

   ```
   modifiedPrice = floorPrice + Math.random() * (ceilPrice - floorPrice)
   ```

3. **Apply Price Modifiers** (multiplicative):

    - Perlin noise influence (99.5% - 100.5%)
    - Market event modifier (94% - 103% depending on event)
    - Recovery mode modifier (101% - 103%)

4. **Apply Slump Pressure** (if conditions met):

    - Additional downward pressure for stocks in extended decline

5. **Boundary Enforcement**:

    ```
    finalPrice = max(floorPrice, min(ceilPrice, calculatedPrice))
    ```

6. **Precision Formatting**:

    ```
    finalPrice = parseFloat(finalPrice.toFixed(2))
    ```

Data Flow and Dependencies

Input Requirements:

- **Stock Data**: Must include stock_id, symbol, industry, and valid numerical reference_price
- **Authentication**: Admin JWT token for order creation API calls
- **External Service**: `getAllStocksWithLatestPricesService()` for current stock data

Order Structure:

```
{
    stockId: number,        // Database stock identifier
    quantity: number,       // Random value 1-100
    price: number|undefined, // Undefined for market orders
    orderType: string       // 'Market Buy', 'Market Sell', 'Limit Buy', 'Limit Sell'
}
```

API Integration:

- **Endpoint**: `POST http://localhost:3000/api/orders/createArtiOrder`
- **Authentication**: Bearer token in Authorization header
- **Error Handling**: Logs detailed error information and exits on failure

System State Management

Persistent State Variables:

- `activeEvents[]`: Array of currently running market events with remaining cycle counts
- `consecutiveDownMoves{}`: Object tracking price movement history per stock

- `perlinTime`: Continuously incrementing time value for noise generation
- `stockQueue[]`: Rotating queue of stocks for fair processing distribution

State Updates Per Cycle:

1. Decrement active event durations, remove expired events
2. Check for new market event triggers
3. Update consecutive down move counters for all stocks
4. Process recovery triggers and durations
5. Advance Perlin noise time parameter

Performance and Scalability Considerations

- **Memory Usage**: Tracks state for every stock in the system
- **API Load**: 10 HTTP requests every 5 seconds
- **Processing Complexity**: O(n) where n = number of stocks, executed every cycle
- **Queue Management**: Ensures even distribution across all stocks over time

Realistic Market Simulation Features

1. **Temporal Consistency**: Uses time-based Perlin noise for smooth transitions
2. **Sector-Specific Events**: Different industries affected by appropriate market conditions
3. **Mean Reversion**: Recovery mechanism prevents indefinite price declines
4. **Volume Variation**: Random quantities between 1-100 shares per order
5. **Order Type Realism**: 80/20 limit/market order distribution matches real markets
6. **Price Boundaries**: ±7% limits prevent unrealistic price jumps