WiDS_Week1_ShortReport

This project implements a simplified market simulator that models the core mechanics of an order-driven exchange. The design focuses on correct system structure rather than trading performance, with an emphasis on centralized validation, deterministic execution, and clean separation of responsibilities. All design choices are directly reflected in the simulator's code.

The simulator is organized around a "MarketEnvironment" class, which represents the exchange. This component owns the global clock, validates incoming order, and is the only entity allowed to interact with the order book. Agents are not permitted to modify the order book directly. This design enforces fairness, prevents manipulation of timestamps or queue positions, and mirrors how real exchanges centralize rule enforcement.

The order book is implemented as a price-indexed structure, where each price level maps to a FIFO queue of individual orders. Two sorted price lists maintain price priority for bids and asks, while queues (deque) enforce time priority within each price level. Matching follows strict price-time priority, and partial fills are handled explicitly by decrementing order quantities and leaving residual quantities in the book when applicable. These mechanisms replicate the execution logic of real limit order books.

When opposing liquidity is insufficient, orders are partially filled using the minimum of available quantities. This behavior naturally arises from the matching logic and reflects realistic trading conditions. To ensure reproducibility, the simulator is deterministic given a fixed random seed. As a result, identical agent actions and arrival sequences always produce identical trades and price paths, enabling auditability and controlled experimentation.

Market participants are represented as agents that interact with the market exclusively through a well-defined API. Each agent implements an "act(env)" method that returns an order intent, which the environment then validates and processes. This separation prevents agents from bypassing market rules and allows different agent types, for example random traders or market makers, to be swapped in without modifying core market logic.

All orders and trades are recorded by a centralized logger. This logging layer enables post-simulation analysis, visualization of spreads and depth and replay of market events. Visualization components operate strictly on logged data and do not affect market state, preserving correctness and modularity.

The simulator captures essential market mechanics through explicit design choices implemented in code, i.e. centralized validation, price-time priority, partial fills and deterministic execution. While simplified, the architecture mirrors real exchange systems and provides a robust foundation for future extensions such as order cancellation, latency modelling or reinforcement learning agents.