**L3-Simulation Logic**

**A. Backtest Engine**

**MarketInterface Class:**

1. *Without Market Impact*

submit\_order(side, quantity, timestamp, limit)

Market.simulate\_agent\_message(message)

* submission template\_id = 99999
* append order to agent\_message\_list

cancel\_order(agent\_msg\_num)

Market.simulate\_agent\_message(message)

* cancellation template\_id = 66666
* mark order as cancelled by changing template\_id to 33333

1. *With Market Impact*

submit\_order\_impact(side, quantity, timestamp, limit)

Market.update\_with\_agent\_message()

Market.\_order\_submit\_impact(message)

* + add order to internal state, treat as historic order

Cancel\_order\_impact(side, quantity, timestamp, limit)

Market.update\_with\_agent\_message()

Market.\_order\_cancel\_impact(message)

* + remove order from internal state

**Market Class:**

1. Match Simulated Orders against Simulation State:

update\_simulation\_with\_exchange\_message()

update\_with\_exchange\_message(message\_packet)

* Updates state by processing messages according to template\_id
* Will be called in every simulation step (not during agent submissions)
* Call \_simulate\_agent\_order\_matching() to check for possible matches

\_simulate\_agent\_order\_matching()

\_build\_simulation\_state()

* Simulation\_state for simulated matching of agent orders
* Copy relevant part of internal state and add agent orders

Trade\_list = match\_new(simulation\_state)

* Match executable agent orders from agent\_message\_list with historical orders from simulation\_state, return trade\_list

\_process\_executed\_agent\_orders(trade\_list)

* Process executed agent orders by changing the quantity and templade\_id (executed -> template\_id = 11111) in the agent\_message\_list.

\_store\_agent\_trades(trade\_list)

* Store agent trades to agent\_trade\_list

1. Match Simulated Orders Against Execution Summaries

Note: Set Market argument “match\_agent\_agains\_execution\_summary” True

update\_simulation\_with\_exchange\_message()

if template\_id == 13202 – executiuon summary:

\_match\_agent\_orders\_against\_execution\_summary()

* Check if agent orders could be filled for the execution price
* Check if agent orders have higher price-time priority than historically executed orders (template\_id 13104/13105)
* Return trade\_list

\_process\_executed\_agent\_orders()

\_store\_agent\_trades()

Note: It is assumed that the historically executed orders remain executed, even if it is possible to execute agent orders (avoid large deviation from historical market activity)

**SnapshotParser Class**

* Change dictionary keys of snapshot
* Convert numbers from str to int
* Wrapper, used as decorator in Market

**MessagePacketParser Class**

* Change dictionary keys of messages
* Convert numbers from str to int
* Wrapper, used as decorator in Market

**Episode Class**

Generates data episode from historical message and snapshot data

\_\_next\_\_() – Return next message packet

**Replay Class**

generate\_episode()

step()

message\_packet = Episode.\_\_next\_\_()

Update\_with\_exchane\_message(message\_packet)

**Context Class**

Stores time series of market states in class instance

**Order Class**

Stores Agent Orders in class instance

**Trade Class**

Stores Agent Trades in class instance

**B. Reinforcement Learning**

**ObservationSpace Class**

Generate Features based on Context

**TradingEnvironment Class**

RL-Environment in Open AI Gym Convention

reset()

step()

**AgentState**

Track Agent State (inventory, time elapsed, PnL etc.)

**Reward**

Compute Reward

**Trainer**

Run training episodes