IMC Prosperity 2024

April 7th 2024

Tomás Zúquete da Fonseca (Zukt)

This challenge invites participants to develop and submit a Python trading algorithm class named Trader. This algorithm will compete against bots on a simulated island exchange to earn as many SeaShells (the island's currency) as possible.

The document outlines several Python classes designed for an algorithmic trading challenge on a simulated exchange. These classes form the backbone of the simulation environment, providing the structure for participants to interact with the market, place orders, and manage trading strategies.

1. Trader Class

This class is where participants will write their trading algorithm. It requires a single method, run, which encapsulates the trading logic. When the simulation environment executes, it calls this run method, providing a TradingState object as input. The run method processes this state and decides on the orders to place. The method's output includes orders, conversions, and a traderData string for maintaining state across iterations.

Key Elements:

- **run method**: Takes TradingState as input, and outputs trading orders based on algorithmic analysis and strategy.
- **result dictionary**: Contains all the orders that the algorithm decides to send, keyed by product.
- **traderData:** A string that allows participants to maintain state information across different calls to the run method.

2. TradingState Class

This class provides a comprehensive snapshot of the current market situation to the Trader class each time its run method is called. It includes details about both the participant's trades and overall market activity.

Attributes:

- **traderData**: String value holding state data required for the next execution.
- **timestamp**: The current time in the simulation.
- **listings**: A dictionary of available products for trading.
- **order_depths**: A per product overview of all outstanding buy and sell orders.
- **own_trades & market_trades**: Details of trades executed by the algorithm and other market participants, respectively.
- **position**: The algorithm's current position in every tradable product.
- **observations**: Market observations that may influence trading decisions.

3. Order and Execution Mechanics

Understanding the dynamics of order placement and execution is vital. Orders are detailed via the Order class and are immediately matched against existing orders if compatible, ensuring dynamic market interaction.

OrderDepth Class

Shows all market orders for a product, influencing buying or selling decisions.
buy_orders and sell_orders: Dictate market depth and liquidity, providing price levels and volumes.

Trade Class

Documents each trade transaction within TradingState, offering insights into recent market activities.

4. OrderDepth Class

Encapsulates the state of the market for a particular product, showing all outstanding buy and sell orders. It's provided to the Trader class via the TradingState object.

Attributes:

- **buy_orders & sell_orders**: Dictionaries where keys are prices and values are the total volume of orders at those prices. Sell order quantities are negative.

5. Trade Class

Represents an individual trade transaction. Instances of this class are used within the TradingState object to inform the Trader about recent trades.

Attributes:

- **symbol**: The product being traded.
- **price**: The price at which the trade occurred.
- **quantity**: The amount of the product traded.
- **buyer & seller**: Identities of the trade participants. For the algorithm's trades, these will be populated with specific identifier

6. Order Class

Defines a trading order. The Trader class uses instances of Order to specify the details of orders it wants to place.

Attributes:

- **symbol**: The product for which the order is placed.
- **price**: In the case of a buy order, the maximum price the algorithm is willing to pay; for a sell order, the minimum price at which it's willing to sell.
- **quantity**: The amount of the product to buy or sell. A positive value indicates a buy order, and a negative value indicates a sell order.

7. Listing Class

The Listing class represents the metadata for tradable products on the exchange. It is essential for understanding the characteristics of each product available for trading.

Key Attributes:

- **symbol**: A unique identifier for the tradable product.
- **product**: The name or type of the product, which could be anything from financial instruments to virtual goods.
- **denomination**: The currency or unit of account in which the product's trades are denominated.

This class helps the algorithm differentiate between various products and plan strategies, accordingly, taking into account the specific dynamics and denomination of each product.

8. Observation Class

The Observation class encapsulates market insights and environmental conditions that could impact trading decisions. It provides a broader view of the market beyond just price and volume data.

Key Attributes:

- **plainValueObservations**: A dictionary mapping products to their observed plain values, offering direct insights into market valuations.
- **conversionObservations**: A dictionary mapping products to their respective ConversionObservation instances, furnishing detailed data for making informed conversion decisions.

9. ConversionObservation Class

This class provides data relevant to the conversion of products, which is a crucial aspect for strategies that involve trading across different product types or converting one form of asset into another.

Key Attributes:

- **bidPrice & askPrice**: The current buy and sell prices for conversions, enabling the algorithm to calculate the cost and potential profit of conversion trades.
- **transportFees**: Costs associated with transporting or converting the product, adding a layer of complexity to conversion strategies.
- **exportTariff** & **importTariff**: Fees imposed on exporting and importing goods, respectively, which can affect the net outcome of conversion trades.
- **sunlight & humidity**: Environmental factors that might influence the conversion process or the valuation of certain products. These attributes simulate dynamic external conditions affecting market dynamics.

10. Placing Orders with the Order Class

When the algorithm determines a strategy for trading, it communicates its decisions through orders using the Order class, which specifies:

- **Product Symbol**: Identifies the product to be traded.

- **Price**: Sets the maximum/minimum price for buying/selling.
- **Quantity**: Indicates how much of the product the algorithm intends to trade, with the sign indicating buy (positive) or sell (negative) orders.

Orders are matched immediately if compatible counterparty orders exist, leading to instant execution. If unmatched, they remain visible to bots, which might trade against them in future iterations. This mechanism ensures dynamic interaction with the market, allowing for both strategic positioning and opportunistic trading.

Understanding Position Limits

Position limits are crucial safeguards that prevent excessive exposure to a single product. They are defined per product and limit the absolute size of an open position (long or short) that an algorithm can hold. Exceeding these limits results in automatic rejection of orders that would breach the limit. This constraint necessitates careful order volume calculation and strategic position management, ensuring algorithms do not overextend and maintain balanced exposure across products.

Enhancing Trading Strategies

Incorporating insights from the **Listing**, **ConversionObservation**, and **Observation** classes enables algorithms to:

- **Tailor Strategies per Product**: Understand unique product characteristics and adjust trading strategies accordingly.
- **Exploit Conversion Opportunities**: Analyze conversion costs and environmental factors to identify profitable arbitrage opportunities.
- **Adapt to Market Conditions**: Use market observations to modify strategies in response to environmental changes or new information**.**
- **Data-Driven Decisions**: Utilizing Observation data, especially conversion opportunities, allows algorithms to make nuanced trading and conversion decisions based on a comprehensive market view.
- **Order Strategy**: Effective use of the Order class, considering the immediate execution mechanism and visibility to bots, is essential for tactical trading and capitalizing on market movements.
- **Position Management**: Adhering to position limits while strategically entering and exiting trades is key to managing risk and exploiting market opportunities within the allowed constraints.