Appendix

```
class TraderMMM02(Trader):
    """
    MMM: Minimal Market-Maker: a minimally simple trader that buys & sells to
make profit

    MMM02 long-only buy-and-hold strategy in pseudocode:

    1 wait until the market has been open for 5 minutes (to give prices a chance to
settle)
    2 then repeat forever:
    2.1 if (I am not holding a unit)
    2.1.1  and (best ask price is "cheap" -- i.e., less than average of recent transaction
prices)
    2.1.2  and (I have enough money in my bank to pay the asking price)
    2.2 then
    2.2.1   (buy the unit -- lift the ask)
    2.2.2   (remember the purchase-price I paid for it)
    2.3 else if (I am holding a unit)
    2.4 then
    2.4.1   (my asking-price is that unit’s purchase-price plus my profit margin)
    2.4.1   if (best bid price is more than my asking price)
    2.4.1   then
    2.4.1.1    (sell my unit -- hit the bid)
    2.4.1.2    (put the money in my bank)
    """

    def __init__(self, ttype, tid, balance, params, time):
        """
        Construct an MMM trader
        :param ttype: the ticker-symbol for the type of trader (its strategy)
        :param tid: the trader id
        :param balance: the trader's bank balance
        :param params: a dictionary of optional parameter-values to override the defaults
        :param time: the current time.
        """

        Trader.__init__(self, ttype, tid, balance, params, time)
        self.job = 'Buy'  # flag switches between 'Buy' & 'Sell'; shows what MMM02 is
currently trying to do
        self.last_purchase_price = None
        self.thr_pri= None
        # 初始化 KD 指标的变量
        self.k_values = []  # 新增這一行以初始化 k_values 為一個空列表
        self.last_K = 0  # 可以将其初始化为 0 或合适的起始值
        self.last_D = 0  # 可以将其初始化为 0 或合适的起始值
        self.purchase_price = None  # 初始化买入价格
        self.stop_loss_price = None  # 初始化止损价格
        #self.trail_stop_percent = 1  # 止损价保持在市价的 95%
        if params is not None:
            self.d_period = params.get('d_period', 3)
        else:
            self.d_period = 3

        # Default parameter-values
        self.n_past_trades = 14      # how many recent trades used to compute average
price (avg_p)?
        self.bid_percent = 0.9999   # what percentage of avg_p should best_ask be for
this trader to bid
        self.ask_delta = 1          # how much (absolute value) to improve on purchase
price

        # Did the caller provide different params?
        if type(params) is dict:
            if 'bid_delta' in params:
                self.bid_percent = params['bid_percent']
                if self.bid_percent > 1.0 or self.bid_percent < 0.01:
                    sys.exit('FAIL: self.bid_percent=%f not in range [0.01,1.0])' %
self.bid_percent)
            if 'ask_delta' in params:
                self.ask_delta = params['ask_delta']
            if 'n_past_trades' in params:
                self.n_past_trades2 = int(round(params['self_past_trades']))
                if self.n_past_trades2 < 1:
                    sys.exit("Fail: MM01 n_past trades must be 1 or more")

    def getorder(self, time, countdown, lob):
        """
        return this trader's order when it is polled in the main market_session loop.
        :param time: the current time.
        :param countdown: the time remaining until market closes (not currently used).
        :param lob: the public lob.
        :return: trader's new order, or None.
        """
        # this test for negative countdown is purely to stop PyCharm warning about
unused parameter value
        if countdown < 0:
            sys.exit('Negative countdown')

        if len(self.orders) < 1 or time < 5 * 60:
            order = None
        else:
            quoteprice = self.orders[0].price
            order = Order(self.tid,
                          self.orders[0].otype,
                          quoteprice,
                          self.orders[0].qty,
                          time, lob['QID'])
            self.lastquote = order
        return order

    # def update_trail_stop(self, current_price):
    #     # 仅在价格上涨时提升止损价
    #     new_stop_price = current_price * self.trail_stop_percent
    #     if new_stop_price > self.stop_loss_price:
    #         self.stop_loss_price = new_stop_price

    def respond(self, time, lob, trade, vrbs):
        """
        Respond to the current state of the public lob.
        Buys if best bid is less than simple moving average of recent transcaction prices.
        Sells as soon as it can make an acceptable profit.
        :param time: the current time
        :param lob: the current public lob
        :param trade:
        :param vrbs: if True then print running commentary, else stay silent
        :return: <nothing>
        """

        vrbs = False
        vstr = 't=%f MM01 respond: ' % time

        # what is average price of most recent n trades?
        # work backwards from end of tape (most recent trade)
        tape_position = -1
        n_prices = 0
        sum_prices = 0
        avg_price_ok = False
        avg_price = -1
        # 初始化变量
        recent_prices = []
        # 逆向遍历交易记录

        while n_prices < self.n_past_trades and abs(tape_position) <= len(lob['tape']):
            trade = lob['tape'][tape_position]
            if trade['type'] == 'Trade':
                price = trade['price']
                recent_prices.append(price)  # 收集价格信息
                sum_prices += price  # 累加价格
                n_prices += 1
            tape_position -= 1

        # 如果收集到足够的价格数据来计算 RSV 和平均价格
        if n_prices == self.n_past_trades:
            avg_price = int(round(sum_prices / n_prices))  # 计算平均价
            avg_price_ok = True
            # 从收集到的价格中计算最高价和最低价
```

```
            low_min = min(recent_prices)
            high_max = max(recent_prices)
            # 计算 RSV
            RSV = ((recent_prices[-1] - low_min) / (high_max - low_min) * 100) if 
high_max > low_min else 0
            # 计算%K
            self.last_K = self.last_K * (2 / 3) + RSV * (1 / 3)
            # 更新%K 历史记录用于计算%D
            self.k_values.append(self.last_K)
            if len(self.k_values) > self.d_period:
                # 移除最老的%K 值，以保持列表长度
                self.k_values.pop(0)
            # 计算%D
            if len(self.k_values) == self.d_period:
                self.last_D = sum(self.k_values) / self.d_period
            else:
                # 如果不够计算%D，使用最后一个%K 值
                self.last_D = self.last_K



        # buying?
        if self.job == 'Buy' and avg_price_ok and self.last_K < 25 and self.last_K < 
self.last_D:
            vstr += 'Buying - '
            # see what's on the LOB
            if lob['asks']['n'] > 0:
                # there is at least one ask on the LOB
                best_ask = lob['asks']['best']
                if best_ask / avg_price < self.bid_percent:
                    # bestask is good value: send a spread-crossing bid to lift the ask
                    bidprice = best_ask + 1
                    if bidprice < self.balance:
                        # can afford to buy
                        # create the bid by issuing order to self, which will be processed in 
getorder()
                        order = Order(self.tid, 'Bid', bidprice, 1, time, lob['QID'])
                        self.orders = [order]
                        vstr += 'Best ask=%d, bidprice=%d, order=%s ' % (best_ask, bidprice, 
order)
                        self.thr_pri = bidprice #theashold price
                else:
                    vstr += 'bestask=%d >= avg_price=%d' % (best_ask, avg_price)
            else:
                vstr += 'No asks on LOB'

        elif self.job == 'Sell'and self.thr_pri is not None:

            vstr += 'Selling - '
            if lob['bids']['n']:

                best_bid = lob['bids']['best']

                if best_bid >= self.thr_pri:

                    askprice = self.thr_pri + self.ask_delta

                    if askprice <= best_bid:

                        order = Order(self.tid, 'Ask', askprice, 1, time, lob['QID'])

                        self.orders = [order]

                        vstr += 'Best bid=%d, selling at askprice=%d, order=%s' % (best_bid,
askprice, order)

                    else:

                        vstr += 'Best bid=%d, but holding for higher price than askprice=%d'
% (best_bid, askprice)

                else:

                    # 在这里加入止损逻辑，如果最佳买入价低于购买价格，不卖出

                    vstr += 'Best bid=%d is below purchase price %d, holding...' % (best_bid,
self.thr_pri)

            else:

                vstr += 'No bids on LOB'

        self.profitpertime = self.profitpertime_update(time, self.birthtime, self.balance)

        if vrbs:
            print(vstr)

    def bookkeep(self, time, trade, order, vrbs):
        """
        Update trader's records of its bank balance, current orders, and current job
        :param trade: the current time
        :param order: this trader's successful order
        :param vrbs: if True then print a running commentary, otherwise stay silent.
        :param time: the current time.
        :return: <nothing>
        """
        vrbs = False
        # output string outstr is printed if vrbs==True
        mins = int(time//60)
        secs = time - 60 * mins
        hrs = int(mins//60)
        mins = mins - 60 * hrs
        outstr = 't=%f (%dh%02dm%02ds) %s (%s) bookkeep: orders=' % (time, hrs,
mins, secs, self.tid, self.ttype)
        for order in self.orders:
            outstr = outstr + str(order)

        self.blotter.append(trade)  # add trade record to trader's blotter

        # NB What follows is **LAZY** -- assumes all orders are quantity=1
        transactionprice = trade['price']
        if self.orders[0].otype == 'Bid':
            # Bid order succeeded, remember the price and adjust the balance
            self.balance -= transactionprice
            self.last_purchase_price = transactionprice
            self.job = 'Sell'  # now try to sell it for a profit
        elif self.orders[0].otype == 'Ask':
            # Sold! put the money in the bank
            self.balance += transactionprice
            self.last_purchase_price = 0
            self.job = 'Buy'  # now go back and buy another one
        else:
            sys.exit('FATAL: MMM02 doesn\'t know .otype %s\n' % self.orders[0].otype)

        if vrbs:
            net_worth = self.balance + self.last_purchase_price
            print('%s Balance=%d NetWorth=%d' % (outstr, self.balance, net_worth))

        self.del_order(order)  # delete the order

    # end of MMM02 definition
```