```python
import sys
import os
sys.path.insert(1, os.path.join(sys.path[0], '..'))

from environment.environment_base import SimpleTrader
import numpy as np
from termcolor import colored

# Eric's implementation of the environment
class EricTrader(SimpleTrader):

    def __init__(self, ticker_list, initial_funds=2000, starting_date="2023-04-05", ending_date="2023-10-05", observation_metrics=3):
        super().__init__(ticker_list, initial_funds=2000, starting_date="2023-04-05", ending_date="2023-10-05", observation_metrics=3)

        self.purchase_price = np.zeros(self.num_stocks) #used to store the purchase price of the stock
        self.sell_price = np.zeros(self.num_stocks)     #used to store the sell price of the stock
        self.price_delta = np.zeros(self.num_stocks)    #used to store the price delta

    def step(self, action_list):

        self.curr_step += 1

        done = self.curr_step >= self.num_trading_days

        reward = self._perform_action_eric_0(action_list)

        if self.render_episodes == True:
            self.episode_funds.append(self.curr_funds - self.initial_funds)
            self.episode_portfolio.append(self.portfolio_value)

            if done:
                self.funds_history.append(self.episode_funds)
                self.portfolio_history.append(self.episode_portfolio)
                self.episode_funds, self.episode_portfolio = [], []
                self._get_total_action_count()
                self._get_total_buy_sell_percents()

        if not done:
            observation = self._get_observation_eric_0()
        else:
            observation = None

        return observation, reward, done, False, {}

    def _perform_action_eric_0(self, action_list):

        #initialise delta reward to be zero
        delta_reward = 0

        curr_date = self.trading_days[self.curr_step - 1]

        opening_price = [self.stock_data.loc[(ticker, curr_date), "Open"] for ticker in self.ticker_list]
        closing_price = [self.stock_data.loc[(ticker, curr_date), "Adj Close"] for ticker in self.ticker_list]

        self.previous_portfolio = self.portfolio_value

        for sell_stock in range(self.num_stocks):

            action = action_list[sell_stock]

            if action < 0:
                max_sell_shares = self.owned_shares[sell_stock]
                num_shares = int(abs(action) * max_sell_shares)
                if num_shares >= 1:
                    self.sell_price[sell_stock] = opening_price[sell_stock]
                    self.price_delta[sell_stock] = self.sell_price[sell_stock] - self.purchase_price[sell_stock]
                    revenue = num_shares * opening_price[sell_stock]
                    delta_reward = delta_reward + 100 *(self.sell_price[sell_stock] - self.purchase_price[sell_stock])
                    self.owned_shares[sell_stock] -= num_shares
                    self.curr_funds += revenue
                    self.num_sells += 1
                    #print(colored(f'SOLD {num_shares} {self.ticker_list[sell_stock]} for {revenue}', 'green'))
                    #print(colored(f'DELTA REWARD {delta_reward}', 'cyan'))


        for buy_stock in range(self.num_stocks):

            action = action_list[buy_stock]

            if action > 0:
                max_buy_shares = self.curr_funds / opening_price[buy_stock]
                num_shares = int(action * max_buy_shares)
                if num_shares >= 1:
                    cost = num_shares * opening_price[buy_stock]
                    if cost <= self.curr_funds:
                        self.purchase_price[buy_stock] = opening_price[buy_stock]
                        self.owned_shares[buy_stock] = self.owned_shares[buy_stock] + num_shares
                        self.curr_funds = self.curr_funds - cost
                        self.num_buys = self.num_buys + 1
                        #print(colored(f'BOUGHT {num_shares} {self.ticker_list[buy_stock]} for {cost}', 'red'))
        # print(f"purchase price: {self.purchase_price}")
        # print(f"sell price: {self.sell_price}")
        # print(f"delta: {self.price_delta}")

        self.portfolio_value = self.curr_funds + sum(self.owned_shares * closing_price)

        reward = self._get_reward_eric_0() + delta_reward
        # print(colored(f"total reward: {reward}",'cyan'))

        return reward



    def _get_reward_eric_0(self):

        #penalty for holding funds scale: (dollar value)
        funds_penalty = - (self.curr_funds * 0.05)

        #reward for making profit from initial funds scale: (dollar value)
        dollar_profit_reward = self.portfolio_value - self.initial_funds

        #reward for making profit from previous porfolio value scale: dollar value
        dollar_portfolio_reward = self.portfolio_value - self.previous_portfolio

        #reward/penalty for making more than the initial investment value
        #setting a minimum % it ahs to return (4%) scale: x% value

        percent_portfolio_return = ((self.portfolio_value - self.initial_funds)/self.initial_funds) * 100
        if  percent_portfolio_return > 4:                       #small reward for performing good
            percent_portfolio_reward= 10 * ( percent_portfolio_return - 3)
        elif  percent_portfolio_return < 4:                     #larger penalisation for bad performance
            percent_portfolio_reward = -(15 * (4 -  percent_portfolio_return))
        elif  percent_portfolio_return < 0:                        #negative performance = large penalisation
            percent_portfolio_reward = 50 * percent_portfolio_return

        #has to make a 3% improve from the previous portfolio
        #calculate percentage return scale: x% value
        percent_prev_portfolio_return = ((self.portfolio_value - self.previous_portfolio)/self.previous_portfolio) * 100

        if percent_prev_portfolio_return > 3:                       #small reward for performing good
            percent_prev_portfolio_reward = 10 * (percent_prev_portfolio_return - 3)
        elif percent_prev_portfolio_return < 3:                     #larger penalisation for bad performance
            percent_prev_portfolio_reward = -(15 * (3 - percent_prev_portfolio_return))
        elif percent_prev_portfolio_return < 0:                        #negative performance = large penalisation
            percent_prev_portfolio_reward = 50 * percent_prev_portfolio_return


        reward = funds_penalty + dollar_profit_reward + dollar_portfolio_reward + percent_portfolio_reward + percent_prev_portfolio_reward

        # print(colored(f'funds penalty {funds_penalty}', 'magenta'))
        # print(colored(f'dollar profit reward  {dollar_profit_reward}', 'magenta'))
        # print(colored(f'dollar portfolio reward {dollar_portfolio_reward}', 'magenta'))
        # print(colored(f'percent portfolio reward {percent_portfolio_reward}', 'magenta'))
        # print(colored(f'percent prev portfolio reward {percent_prev_portfolio_reward}', 'magenta'))

        return reward

    def _get_observation_eric_0(self):

        curr_date = self.trading_days[self.curr_step - 1]

        if self.curr_step != 1:
            yesterday = self.trading_days[self.curr_step - 2]
        else:
            yesterday = self.trading_days[self.curr_step - 1]

        opening_price = np.array([self.stock_data.loc[(ticker, curr_date), "Open"] for ticker in self.ticker_list])
        yesterday_price = np.array([self.stock_data.loc[(ticker, yesterday), "Open"] for ticker in self.ticker_list])
        #volume = np.array([self.stock_data.loc[(ticker, curr_date), "Volume"] for ticker in self.ticker_list])

        observation = [round(self.curr_funds, 3)]
        for ii in range(self.num_stocks):
            observation.append(int(self.owned_shares[ii]))
            observation.append(round(opening_price[ii], 3))
            observation.append(round(yesterday_price[ii], 3))
            #observation.append(round(volume[ii], 3))

        return observation

    def reset(self, render=False, seed=None):

        if seed != None:
            np.random.seed(seed)

        if render == True:
            self._render_on_completion()

        if not hasattr(self, "funds_history"):
            self.funds_history, self.portfolio_history = [], []
            self.episode_funds, self.episode_portfolio = [], []
            self.render_episodes = False

        self.curr_step = 0
        self.curr_funds = self.initial_funds
        self.portfolio_value = self.initial_funds

        self.num_buys, self.num_sells = 0, 0
        self.buy_percents, self.sell_percents = 0.0, 0.0
        self.owned_shares = np.zeros(self.num_stocks)

        observation = self._get_observation_eric_0()

        return observation, {}
```