```python
# Author: Jordan Richards
# Date: 19/10/2023
# Purpose: Implemented the action, observation and reward functions for the reinforcement learning
#          environment.
# NOTE: This implementation uses the Portfolio and Share files which has not been included in this submission. They are however, available in the main
#       repository for this code.
# Todo: None

# Imports
from environment.environment_base import SimpleTrader
from environment.Portfolio import Portiolio
import numpy as np
import math
import logging
import matplotlib.pyplot as plt

# Jordan's implementation of the environment
class Trader(SimpleTrader):
    def __init__(self, ticker_list, initial_funds, observation_metrics, starting_date, ending_date):
        super().__init__(ticker_list, initial_funds=initial_funds, observation_metrics = observation_metrics, starting_date=starting_date, ending_date=ending_date)

        # Add a portfolio
        # self.portfolio = Portiolio(initial_funds, starting_date, ending_date, ticker_list)
        self.initial_funds = initial_funds
        self.starting_date = starting_date
        self.ending_date=ending_date
        self.ticker_list = ticker_list

    def step(self, action_list):

        self.action_history.append(action_list)

        # Starts at step 1 and increments there after
        self.curr_step += 1

        # Check that the trading days and steps are in sync
        done = self.curr_step >= self.num_trading_days

        # All the information from the current state to the model
        reward = self._perform_action_jordan_0(action_list)

        if self.render_episodes == True:
            self.episode_portfolio.append(self.portfolio.total_funds)

            if done:
                self.funds_history.append(self.episode_funds)
                self.portfolio_history.append(self.episode_portfolio)
                self.episode_funds, self.episode_portfolio = [], []
                self.portfolio._get_num_buys()
                self.portfolio._get_num_sells()
                self.total_buy_actions = self.portfolio.num_buys
                self.total_sell_actions = self.portfolio.num_sells
        # State_current_step
        if not done:
            observation = self._get_observation_jordan_0()
        else:
            observation = None

        """
        print("Step summary: ", self.curr_step)
        print(f"Action: {action_list}")
        print(f"Reward :{reward}")
        print(f"Observation: {observation}")
        print("========================")
        self.portfolio.print_portfolio()
        print("========================\n")
        """

        self.portfolio.update()
        return observation, reward, done, False, {}

    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)

        self.portfolio = Portiolio(self.initial_funds, self.starting_date, self.ending_date, self.ticker_list, self.historic_stock_data)
        observation = self._get_observation_jordan_0()

        return observation, {}

    def _perform_action_jordan_0(self, action_list):

        # Establish current date
        self.curr_date = self.trading_days[self.curr_step - 1]

        # Get the opening prices of each of the stock for the current date
        opening_prices = [self.stock_data.loc[(ticker, self.curr_date), "Open"] for ticker in self.ticker_list]
        closing_prices = [self.stock_data.loc[(ticker, self.curr_date), "Adj Close"] for ticker in self.ticker_list]

        self.portfolio.set_opening_prices(opening_prices)
```

```python
        self.portfolio.set_closing_prices(closing_prices)
        self.portfolio.update()

        # Apply the sell action
        for index, action in enumerate(action_list):

            # If there is a sell action
            if action < 0:  # sell signal

                # Determine the amount of shares that will be sold
                num_shares = int(abs(action) * self.portfolio.shares[index].num_shares)

                # Perform sell action
                self.portfolio.sell(self.ticker_list[index],  self.curr_date, num_shares, opening_prices[index])

        # Apply the buy action
        for index, action in enumerate(action_list):
            if action > 0:  # buy signal

                # Number of shares to be bought
                num_shares = int((action * self.portfolio.current_funds) / opening_prices[index])

                # Perform buy action
                self.portfolio.buy(self.ticker_list[index], self.curr_date, num_shares, opening_prices[index])


        self.portfolio.update_position()
        # Retrieve reward for previous action
        reward = self._get_reward_jordan_0(action_list)

        return reward

    ####### ATTEMPT 1

    # Naive and simple approach
    def _get_reward_jordan_0(self, action_list):
            reward = -1
            action_sum = 0

            for action in action_list:
                action_sum += abs(action)

            # Stop the agent from performing all the same action with 100% intensity
            if abs(action_sum) != len(action_list):
                for index, action in enumerate(action_list):

                    # Make sure the model is not trying to do invalid actions
                    if action < 0 and self.portfolio.shares[index].num_shares == 0:
                        reward += -1 * self.portfolio.total_funds
                    elif action > 0 and self.portfolio.current_funds == 0:
                        reward += -1 * self.portfolio.total_funds

                    # Otherwise reward the agent if it has made money
                    else:
                        if self.curr_step > 1:
                            if self.portfolio.historic_portfolio_values[-2] < self.portfolio.historic_portfolio_values[-1]:
                                reward += self.portfolio.total_funds
                        else:
                            reward += 0
            else:
                reward = -1 * self.portfolio.total_funds
            return reward

    # Show the agent all the relevant metrics/information
    def _get_observation_jordan_0(self):

        # Establish the current date/trading day
        curr_date = self.trading_days[self.curr_step - 1]

        # Get the opening price and the adjusted closing price for today and previous day
        opening_price = np.array([self.stock_data.loc[(ticker, curr_date), "Open"] for ticker in self.ticker_list])
        adj_closing_price = np.array([self.stock_data.loc[(ticker, curr_date), "Adj Close"] for ticker in self.ticker_list])
        volume = np.array([self.stock_data.loc[(ticker, curr_date), "Volume"] for ticker in self.ticker_list])

        # Create the observational list
        observation = [round(self.portfolio.current_funds, 3)]
        for ii in range(len(self.portfolio.ticker_list)):
            observation.append(round(self.portfolio.shares[ii].num_shares, 3))
            observation.append(round(self.portfolio.shares[ii].aggregated_buy_price, 3))
            observation.append(round(opening_price[ii], 3))  # Add
            observation.append(round(adj_closing_price[ii], 3))
            observation.append(volume[ii])

        return observation

    ####### ATTEMPT 2

    # Deep robust reinforcement learning for practical algorithmic trading (Yang Li, Wanshan Zheng, Zibin Zheng)
    def _get_reward_jordan_1(self, action_list):
        reward = 0

        for asset, action in enumerate(action_list):

            # delta c: change in opening price to the closing price (t - 1)
            # closing price (t-2) - yesterdays closing price (t-1)
            price_2 = self.portfolio.shares[asset].historic_closing_price[-1]
            price_1 = self.portfolio.shares[asset].historic_closing_price[-2]
            delta_c = price_2 - price_1

            # delta p: change in position (t - 1)
            # total funds (t-2) - yesterdays total funds (t-1)
            pos_2 = self.portfolio.shares[asset].historic_positions[-3]
            pos_1 = self.portfolio.shares[asset].historic_positions[-2]
            delta_pos_1 = pos_2 - pos_1

            # delta p: change in position (now)
```

```python
            # yesterdays total funds (t-1) - todays total funds (t)
            pos_0 = self.portfolio.shares[asset].historic_positions[-1]
            delta_pos_0 =  pos_1 - pos_0

            # alpha + beta: aplha = transactional cost, beta = slippage
            alpha = 1
            beta = 0

            reward += delta_pos_1 * delta_c - (alpha - beta) * abs(delta_pos_0)

        #print(f"Reward :{reward}")
        return reward

    # Deep robust reinforcement learning for practical algorithmic trading (Yang Li, Wanshan Zheng, Zibin Zheng)
    def _get_observation_jordan_1(self):

        # Establish the current date/trading day
        curr_date = self.trading_days[self.curr_step - 1]

        # Get the opening price and the adjusted closing price for today and previous day
        opening_price = np.array([self.stock_data.loc[(ticker, curr_date), "Open"] for ticker in self.ticker_list])
        adj_closing_price = np.array([self.stock_data.loc[(ticker, curr_date), "Adj Close"] for ticker in self.ticker_list])
        volume = np.array([self.stock_data.loc[(ticker, curr_date), "Volume"] for ticker in self.ticker_list])
        ema9, ema20, ema90 = self._get_ema()
        rsi = self._get_rsi()
        macd = self._get_macd()
        roc = self._get_roc()
        bb_low, bb_upper = self._get_bband()

        # Private variables
        observation = [round(self.portfolio.current_funds, 3)]
        observation.append(self._get_sharpe_ratio(self.portfolio.historic_daily_return))
        for ii in range(len(self.portfolio.ticker_list)):

            # Private variables
            observation.append(round(self.portfolio.shares[ii].num_shares, 3))
            observation.append(round(self.portfolio.shares[ii].aggregated_buy_price, 3))

            # Market Data
            observation.append(round(opening_price[ii], 3))
            observation.append(round(adj_closing_price[ii], 3))
            observation.append(volume[ii])

            # Technical Analysis
            observation.append(round(ema9[ii], 3))
            observation.append(round(ema20[ii], 3))
            observation.append(round(ema90[ii], 3))
            observation.append(round(rsi[ii], 3))
            observation.append(round(macd[ii], 3))
            observation.append(round(roc[ii], 3))
            observation.append(round(bb_low[ii], 3))
            observation.append(round(bb_upper[ii], 3))

        return observation
```