```python
import gymnasium
from gymnasium.spaces import Box
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from technical_analysis import overlays
from technical_analysis import indicators
from environment.Portfolio import Portiolio
import csv

PERIOD = 20


class SimpleTrader(gymnasium.Env):

    '''
    BASIC STOCK TRADER:
    - passed a list of stock tickers to invest in
    - passed an initial amount of funds to invest
    - for each stock, the action selected by the agent is a float, <0 means
    selling that many shares of a stock, > 0 means buying that many shares of a
    stock and 0 means holding
    - action space -> between -1 (sell all shares) and 1 (buy all shares), 0 is
    hold
    - observation space -> current funds, number of shares owned for each stock
    '''

    # required by gym
    metadata = {"render.modes": ["human", "graph", "review"]}

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

        self.starting_date = starting_date
        self.ending_date = ending_date

        self.ticker_list = ticker_list
        self.num_stocks = len(self.ticker_list)

        self.stock_data = self._get_stock_data()

        self.state_space_dim = observation_metrics * self.num_stocks + 1

        all_days = pd.date_range(
            start=self.starting_date, end=self.ending_date, freq="B")
        actual_days = self.stock_data.index.get_level_values("Date").unique()

        self.trading_days = all_days.intersection(actual_days)

        self.num_trading_days = len(self.trading_days)

        self.action_history = []

        # buy, sell, shares or hold for each stock
        self.action_space = Box(low=-1,
                                high=1,
                                shape=(self.num_stocks,))

        # print(self.action_space)

        self.observation_space = Box(low=-np.inf, high=np.inf,
                                     shape=(self.state_space_dim,))

        self.initial_funds = initial_funds
        self.previous_portfolio = self.initial_funds

        self.total_buy_percents = []
        self.total_sell_percents = []

        self.total_buy_actions = []
        self.total_sell_actions = []

        # Jordans
        self.historic_date = self._get_date()
        self.curr_date = self.trading_days[0]
        self.historic_stock_data = self._get_historic_data()
        self.portfolio = Portiolio(initial_funds, starting_date, ending_date, ticker_list, self.historic_stock_data)

        self.reset()

    def render(self, mode):
        if mode == "human":
            print("=" * 20)
            print(f"Current Step: {self.curr_step} / {self.num_trading_days}")
            print(
                f"Current Profit / Loss: {self.curr_funds - self.initial_funds}")

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

            for stock in range(self.num_stocks):
                print(
                    f"Shares of {self.ticker_list[stock]} owned: {self.owned_shares[stock]}")
                print(
                    f"Opening Price for {self.ticker_list[stock]}: {self.stock_data.loc[(self.ticker_list[stock], curr_date) ,'Open']}")
                print(
                    f"Adjusted Closing Price for {self.ticker_list[stock]}: {self.stock_data.loc[(self.ticker_list[stock], curr_date), 'Adj Close']}")
            print(f"Portfolio Value: {self.portfolio_value}")
            print("=" * 20)

        if mode == 'review':
            self._render_on_completion()

    def set_render_episodes(self, flag):
        self.render_episodes = flag

    def _get_total_action_count(self):
        self.total_buy_actions.append(self.num_buys)
        self.total_sell_actions.append(self.num_sells)

    def _get_total_buy_sell_percents(self):
        if self.num_buys != 0:
            self.total_buy_percents.append(self.buy_percents / self.num_buys)

        if self.num_sells != 0:
            self.total_sell_percents.append(
                self.sell_percents / self.num_sells)

    def _render_on_completion(self):
```

```
        fig, axs = plt.subplots(nrows=2, figsize=(15, 10))

        avg_valuation = [round(sum(values) / len(values), 4)
                         for values in zip(*self.portfolio_history)]

        max_valuation = [(ii + 1, max(episode)) for ii, episode in enumerate(self.portfolio_history)]
        max_valuation.sort(key=lambda x: x[1])

        for _, episode_portfolio in enumerate(self.portfolio_history):
            axs[0].plot(episode_portfolio, color="grey", alpha=0.3, linewidth=2)

        axs[0].plot(avg_valuation, color="red",
                    linewidth=2, marker="o", markersize=4)

        axs[0].set_title(
            f"Average Portfolio Valuation over {len(self.portfolio_history)} Episodes")
        axs[0].set_xlabel("Trading Day")
        axs[0].set_ylabel("Portfolio Valuation ($)")

        episodes, values = zip(*max_valuation)
        col = ["red" if value == max(values) else "grey" for value in values]

        axs[1].bar(range(len(max_valuation)), values, color=col)
        axs[1].set_xticks(range(len(max_valuation)))
        axs[1].set_xticklabels(episodes)
        axs[1].set_title(f"Max Portfolio Valuation per {len(self.portfolio_history)} Episodes")
        axs[1].set_xlabel("Episode")
        axs[1].set_ylabel("Portfolio Valuation ($)")

        plt.tight_layout()
        plt.show()
        fig.savefig("model_perf.png", bbox_inches="tight")

        print(
            f"AVERAGE VALUATION: {round(sum(avg_valuation) / len(avg_valuation), 3)}")

        print(
            f"AVERAGE MAX VALUATION: {round(sum(values) / len(values), 3)}")

        print(
            f"ON AVERAGE, AGENT \033[1m\033[92mBOUGHT\033[0m {int(sum(self.total_buy_actions) / len(self.total_buy_actions))} TIMES AND \033[1m\033[91mSOLD\033[0m {int(sum(self.total_s
        print(f"Buy actions: {self.total_buy_actions}")
        print(f"Sell actions: {self.total_sell_actions}")
        print(f"Average buy percents: {[round(value, 3) for value in self.total_buy_percents]}")
        print(f"Average sell percents: {[round(value, 3) for value in self.total_sell_percents]}")

        with open("output/model_performances.csv", "a", newline="") as f:
            writer = csv.writer(f)
            writer.writerow(avg_valuation)

    def _get_stock_data(self):

        stock_data_list = []

        for ticker in self.ticker_list:
            ticker_data = yf.download(
                ticker, start=self.starting_date, end=self.ending_date, interval="1d")
            ticker_data = ticker_data[["Open", "Adj Close", "Volume"]]
            stock_data_list.append(ticker_data)

        stock_data = pd.concat(
            stock_data_list, keys=self.ticker_list, names=["Ticker", "Date"])

        return stock_data

    def _get_date(self):
        start_date = datetime.strptime(self.starting_date, '%Y-%m-%d')
        start_date = start_date.replace(day=1)
        lastMonth = start_date - timedelta(days=182)
        return lastMonth.strftime("%Y-%m-%d")

    def _get_historic_data(self):
        # Prepare a list for the data
        stock_data_list = []

        # Iterate over each of the different stocks
        for ticker in self.ticker_list:

            # Using yahoo finance as the database, pull all the data for the stock between the date range interval
            ticker_data = yf.download(
                ticker, start=self.historic_date, end=self.starting_date, interval="1d")

            # Choose the data worthy for analysis
            ticker_data = ticker_data[["Open", "Adj Close", "Volume"]]

            # Simply append this to a list which contains each stocks data
            stock_data_list.append(ticker_data)

        # Adjust the column headers of the dataframe
        # stock_data = pd.concat(stock_data_list, keys=self.ticker_list, names=["Ticker", "Date"])

        return stock_data_list

    """
    def _add_to_historic_data(self):
        for index, ticker in enumerate(self.ticker_list):
            v = self.stock_data.loc[(ticker, self.curr_date)]
            self.historic_stock_data[index] = pd.concat([self.historic_stock_data[index], v.to_frame().T])
    """

    def _get_ema(self):
        ema90, ema20, ema9 = [], [], []
        for stock in self.portfolio.shares:
            df_9, df_20, df_90 = stock.get_historic_closing_df()
            temp_ema9 = overlays.ema(df_9, 9)
            temp_ema20 = overlays.ema(df_20, 20)
            temp_ema90 = overlays.ema(df_90, 90)
            ema9.append(temp_ema9['Open'].iloc[-1])
            ema20.append(temp_ema20['Open'].iloc[-1])
            ema90.append(temp_ema90['Open'].iloc[-1])
        return ema9, ema20, ema90

    def _get_bband(self):
        bb_low, bb_upper = [], []
        for stock in self.portfolio.shares:
            df_9, df_20, df_90 = stock.get_historic_closing_df()
            temp_lower, temp_upper = overlays.bbands(df_20, period=PERIOD)
            bb_low.append(temp_lower['Open'].iloc[len(temp_lower) - 1])
            bb_upper.append(temp_upper['Open'].iloc[len(temp_upper) - 1])
```

```
        return bb_low, bb_upper

    def _get_rsi(self):
        rsi = []
        for stock in self.portfolio.shares:
            df_9, df_20, df_90 = stock.get_historic_closing_df()
            temp_rsi = indicators.rsi(df_90, period=PERIOD)
            rsi.append(temp_rsi['Open'].iloc[len(temp_rsi) - 1])
        return rsi

    def _get_roc(self):
        roc = []
        for stock in self.portfolio.shares:
            df_9, df_20, df_90 = stock.get_historic_closing_df()
            temp_roc = indicators.roc(df_20, period=PERIOD)
            roc.append(temp_roc['Open'].iloc[len(temp_roc) - 1])
        return roc

    def _get_macd(self):
        fast_window = 30
        slow_window = 90
        signal = 20
        macd = []
        for stock in self.portfolio.shares:
            df_9, df_20, df_90 = stock.get_historic_closing_df()
            temp_macd = indicators.macd(df_90, slow_window, fast_window, signal)
            macd.append(temp_macd['Open'].iloc[len(temp_macd) - 1])
        return macd

    def _get_sharpe_ratio(self, daily_returns):
        sharpe_ratio = 0
        if len(daily_returns) > PERIOD:
            ret = np.array(daily_returns[-PERIOD:])
            sharpe_ratio = np.sqrt(PERIOD) * np.mean(ret) / np.std(ret)
        return sharpe_ratio
```