PROJECT REPORT
RL TRADE WARS

The financial markets are complex systems, characterized by volatility, uncertainty, and various participants with varying strategies. "RL Trade Wars: Investments Across Timeframes" aims to explore how different reinforcement learning (RL) techniques can be applied to algorithmic trading strategies to profit across short-term and long-term investments, and addresses “How different RL-based trading algorithms perform and adapt in a competitive market environment as influenced by the interactions among multiple agents?”. This project is exciting as it investigates the adaptability and effectiveness of RL methods in making investment decisions, a domain traditionally dominated by human expertise, sentiment, trends, government policies, and many more. By simulating a competitive environment where agents employ different RL strategies, the project seeks to uncover insights about the performance of different RL techniques across different market conditions.

ENVIRONMENT

Our environment, StockEnvironment, is designed following the gymnasium standards. It is a custom environment designed for simulating a stock trading scenario. Here multiple RL agents interact with the environment, each following a different algorithm.
Here is brief description of the main components of the environment:

- **Initialization:**
  This part initializes the location of data, number of agents, number of days to consider, train and test split data, and other crucial components.

- **Action Space:**
  The environment has Discrete action space with three possible actions: buying(0), sell(1) and hold(2). It is defined as a dictionary mapping each agent to their respective action space.

- **Observation Space:**
  The observation space is also defined as Discrete type with four possible observations:
  - Price increase, no shares held(0)
  - Price increase, shares held(1)
  - Price decrease, no shares held(2)
  - Price decrease, shares held(3)

  This is also defined as a dictionary in the same as the action space. The observation space depends on historical price data based on the number of days selected and the agent’s current holding.

- **‘reset’ method:**
  Resets the environment to the initial state, reinitializing the agents and setting timestep to zero.

- **‘step’ method:**
  Executes one time step at a time, evaluating the actions taken then calculates the rewards and updates the state. Also checks if the simulation has reached its end.

Dataset

Below is how the data is loaded and what type of data it represents:

Loading Data

The load_data() method is responsible for loading the stock data from CSV files located in the specified folder_path.
It iterates over each file in the folder and reads its contents using Pandas' read_csv() function. Each file typically represents historical stock price data for a particular company or financial instrument.
The data from each file is appended to a list named combined_data.
After iterating over all files, the data is concatenated along the rows using Pandas' concat() function, resulting in a single DataFrame containing data from all files.

Data Format

Each CSV file is expected to contain historical stock price data, typically including columns such as:
Date: Timestamp indicating the date of the data entry.
Open: The price of the stock at the opening of the trading day.
High: The highest price of the stock during the trading day.
Low: The lowest price of the stock during the trading day.
Close: The price of the stock at the closing of the trading day.
Volume: The total volume of stocks traded during the trading day.
Symbol: Identifier for the stock or financial instrument.

Preprocessing

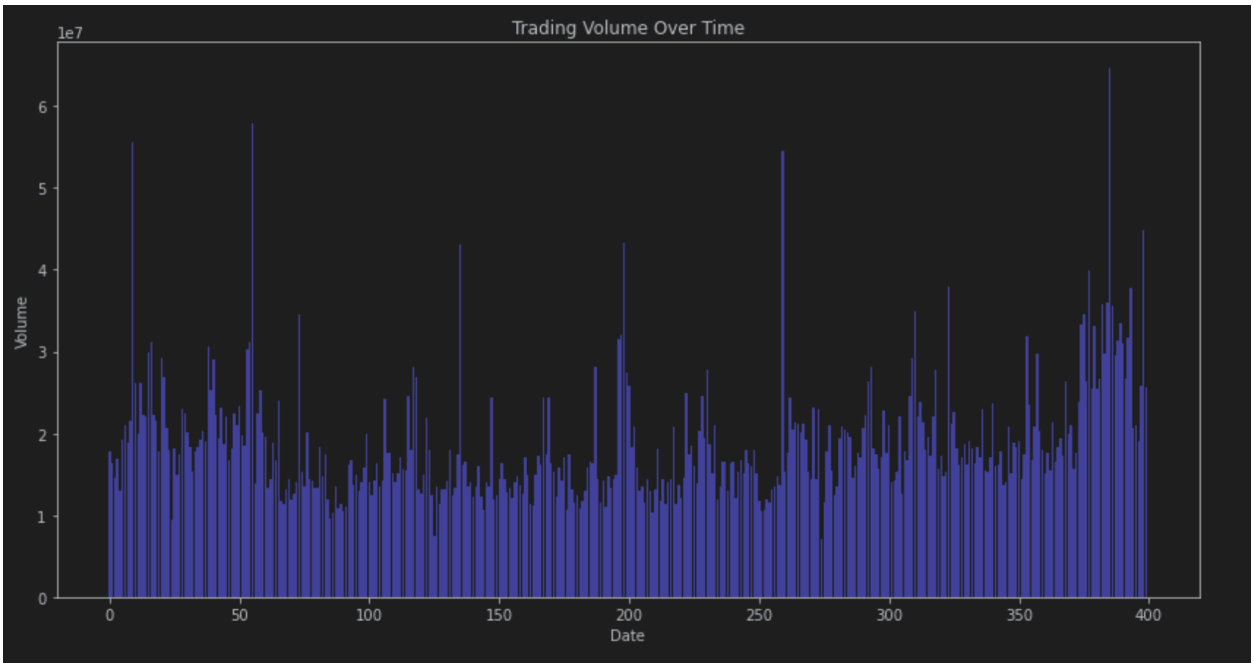
Before returning the combined data, preprocessing steps are applied:
The column names are standardized to ensure consistency.
The StandardScaler() from Scikit-learn is used to scale the numerical features (open, high, low, close, volume). This scaling is essential for many machine learning algorithms as it helps in bringing all features to a similar scale.
The scaled data is returned as the final output.

Now that the data is loaded, this data will be used by reinforcement learning agents within the environment to observe market trends, make decisions (buy, sell, hold), and learn from the consequences of those decisions.

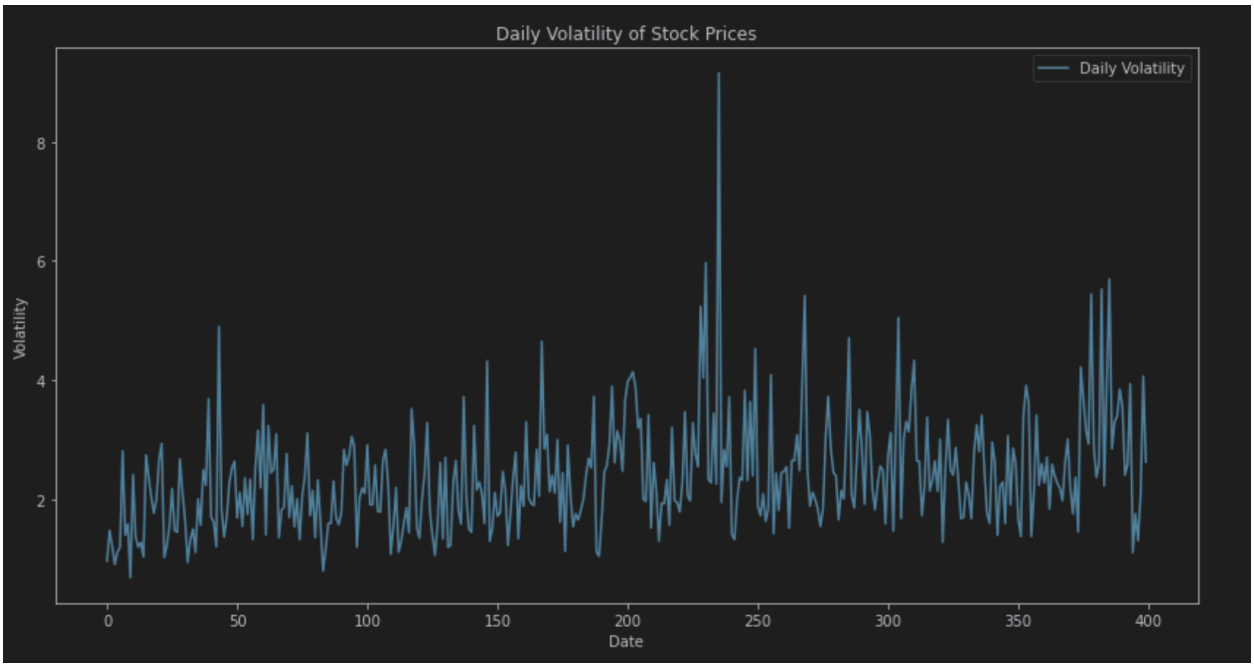
VISUALIZATION GRAPHS

We obtained the following visualization graphs from our dataset:

The above graph represents the stock prices over time with metrics for open, high, low, and close prices. The graph exhibits significant volatility, indicated by visible fluctuations and the variability between the high and low prices of each day. This suggests active trading and responsiveness to market events or news. There is a general downward trend from the start of the period up to around Date 250, followed by a recovery phase, and then another decline, with a subsequent recovery towards the end. This pattern shows both the stock's susceptibility to broader market movements and its inherent volatility.

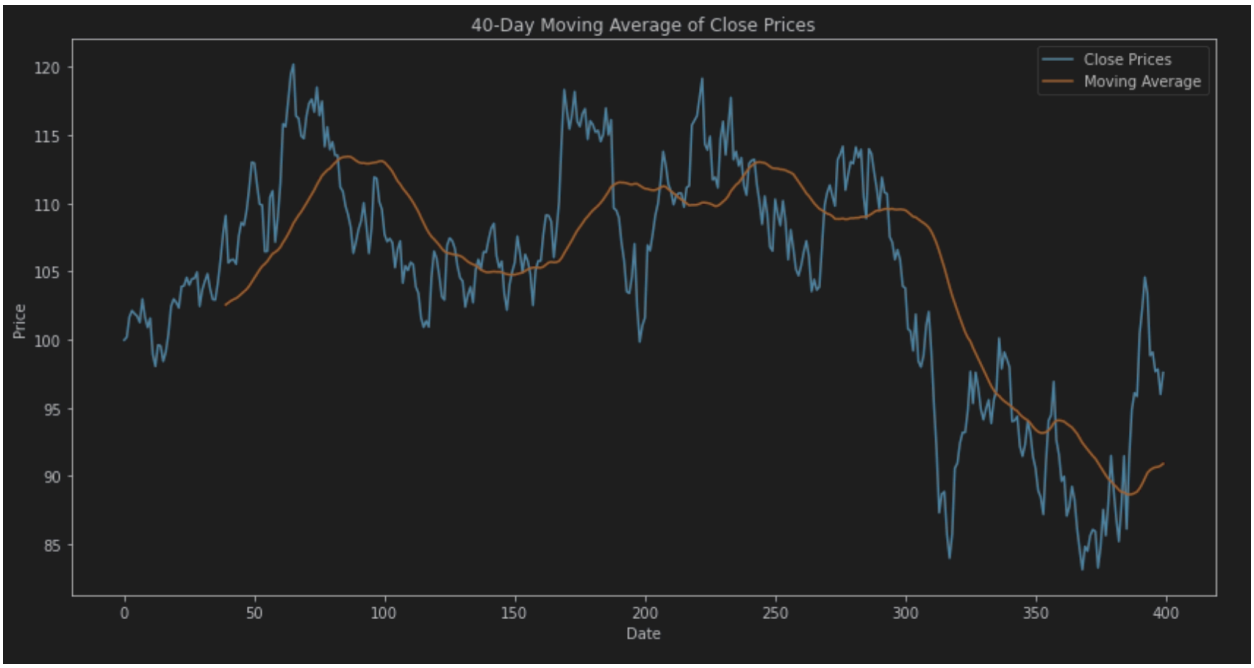


The graph displays the trading volume over time, capturing the total units traded each day. It shows significant fluctuations in volume, with several noticeable spikes that suggest periods of heightened trading activity. These spikes could be tied to specific market events such as news releases, earnings announcements, or other factors that prompt increased buying or selling activity among investors.
Throughout the timeline, the trading volume appears to remain relatively stable but is punctuated by these sharp increases. Notably, from around Date 300 to 400, there's a visible rise in trading volumes, indicating a period of possibly increased investor



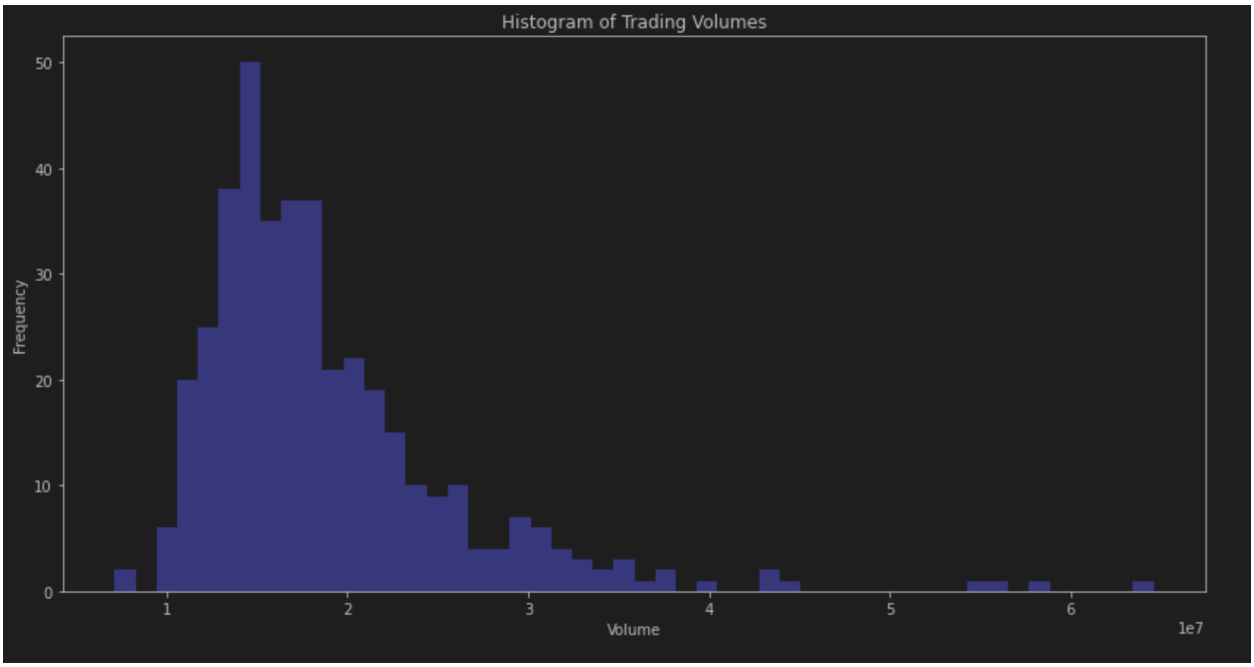
interest or market volatility. This sustained high volume may imply strong market reactions to external events or changes in investor sentiment.

Volatility is measured here as a percentage change from one day to the next, shows how much stock prices fluctuate on a day-to-day basis.
The volatility is relatively constant with occasional spikes, suggesting that while the market conditions are generally stable, there are periods of heightened activity that could be driven by external events such as economic announcements, changes in market conditions, or significant corporate actions (like mergers or earnings releases). Notably, the largest spikes occur around Date 150 and just before Date 250, which are significantly higher than typical daily movements, indicating extraordinary market events or news impacting stock prices.



We are taking a 40 day period to train data based on the availability of complete and consistent data. It's long enough to provide a meaningful dataset but not so long as to include too much noise or irrelevant information. In financial markets, moving averages are a common tool used to smooth out price data over a specific period and help identify trends. A 40-day period is close to a one-and-a-half-month moving average, which could be significant for medium-term trend analysis.

The close price line (in blue) shows the actual price at which the stock closed each trading day. The 40-day moving average (in orange) averages these close prices over the last 40 days and is plotted to help identify the trend direction. The moving average smooths out short-term fluctuations, making it easier to view the overall direction of the price movements.
Initially, the close prices and the moving average both rise, indicating a bullish trend where prices are increasing. However, around Date 100, the close prices begin to exhibit more volatility, and the moving average starts to level off and then decline. This change suggests a transition from a bullish to a bearish trend.



The histogram shows a right-skewed distribution, with the majority of trading days having lower volumes, depicted by the peak in the lower volume range. The frequency of trading days decreases as the volume increases, which is typical in many financial markets where high-volume days are less common than moderate or low-volume days.
The high frequency of lower volume days suggests that under normal market conditions, trading activity remains relatively subdued. The presence of bars at higher volumes, although significantly lower in frequency, indicates occasional days of very high trading activity.

DESCRIPTION OF ALGORITHMS

The simulation was conducted with two agents operating independently within the environment.
Each agent operates autonomously within the environment, making decisions based on its observations and internal state. Despite operating in the same environment, the agents do not communicate or coordinate directly with each other.
Each agent receives its own set of observations, which may include information about current market conditions, historical price data, and the agent's own state (e.g., investment capital, number of shares held).These observations provide the agents with information about the state of the market and guide their decision-making process.

DQN

Deep Q-Learning (DQN) is a reinforcement learning algorithm that combines Q-Learning with deep neural networks to handle high-dimensional state and action spaces. The core idea is to approximate the Q-function using a neural network, allowing the agent to learn optimal policies from complex environments.

Our DQN algorithm is explained below:

DQN Model: The DQN class represents the neural network used to approximate the Q-function. It has three layers: input (observations), a hidden layer, and an output (actions). The forward method defines the computation through the network. It applies ReLU activations on the hidden layers and provides the output values for each possible action.

DQN Agent: The DQNAgent class represents the agent using DQN for decision-making.
Key attributes include:
Epsilon-Greedy Exploration: epsilon, epsilon_min, and epsilon_decay control the exploration rate.
Experience Replay: memory stores past experiences with a maximum length of 30,000.
Discount Factor: gamma is used to discount future rewards.
Methods:
act: Chooses an action based on the current state and epsilon-greedy strategy.
remember: Stores a transition (state, action, reward, next_state, done) in the replay buffer.
replay: Samples from the replay buffer and updates the model using the Q-learning update rule.

DDQN

Double Deep Q-Learning, or DDQN, is an improvement over the traditional Deep Q-Learning (DQN) approach to reinforcement learning. The primary goal of DDQN is to address the overestimation bias present in DQN, leading to more stable and robust learning. DDQN takes advantage of the target network in a different way to reduce overestimation bias. Instead of using the same network to select and evaluate the action for the next state, DDQN uses the current network to select the best action but evaluates it using the target network. This decoupling helps in avoiding overestimation bias.

For each training step, the current network is used to select the best action for the next state. The target network then evaluates the Q-value for that selected action. This value is used to compute the target Q-value, which is used in the loss function to update the current network's weights. The target network is updated periodically.

Below is the breakdown of our DDQN algorithm.

- DDQN Model: The DDQN class defines a neural network with three layers. The input layer size (observations) and output layer size (actions) correspond to the state and action spaces, respectively. The forward method defines the forward pass of the neural network, using ReLU activation functions.

- DDQN Agent: The DDQNAgent class defines the agent that uses the DQN model. It has an exploration-exploitation mechanism through the epsilon variable and supports storing experiences in a replay memory. The update_target_model method copies the weights from the policy model to the target model. The act method chooses actions based on the policy model, using an epsilon-greedy strategy. The replay method samples from the replay memory and performs the DDQN update using the target network to evaluate Q-values for the next state.

- DDQN Training: The train_ddqn function sets up multiple agents and trains them over a specified number of episodes. During training, it handles agent-environment interaction, stores transitions in the replay memory, and applies the replay mechanism to update the model weights. It also includes code for updating the target network and recording the rewards for plotting.

A2C

Advantage Actor-Critic (A2C) algorithm combines benefits of policy and value based methods. This algorithm has 2 components: an actor and a critic, as the name suggests. Actor learns the policy and decides which action to take given the current state of the environment. Whereas a critic learns the value function, which in turn is used to evaluate how good or bad the action taken by the actor is, given the state from the policy. The "advantage" part comes from estimating how much better an action is compared to a baseline, typically the value of the current state. The actor is updated based on the gradient of the log probability of the selected action, adjusted by the advantage, which measures how much better the action was compared to the critic's value estimate. The critic is updated to minimize the mean squared error between its estimate and the actual reward plus the discounted future value

- Actor and Critic Models: The Actor and Critic classes are neural networks designed to represent the actor and critic components of A2C. Both classes use an embedding layer to transform the observation into a feature space. This transformation might be necessary if your observations are categorical. The Actor network outputs a probability distribution over the actions, using a softmax activation. The Critic network outputs a single value, which is the estimated value of the state.

- Environment setup: An environment is initialized, and the actor and critic models are created and moved to the appropriate device GPU else to CPU. Optimizers for both the actor and critic networks are initialized, with learning rates set to 0.001. A gamma value of 0.99 is used for discounting future rewards.

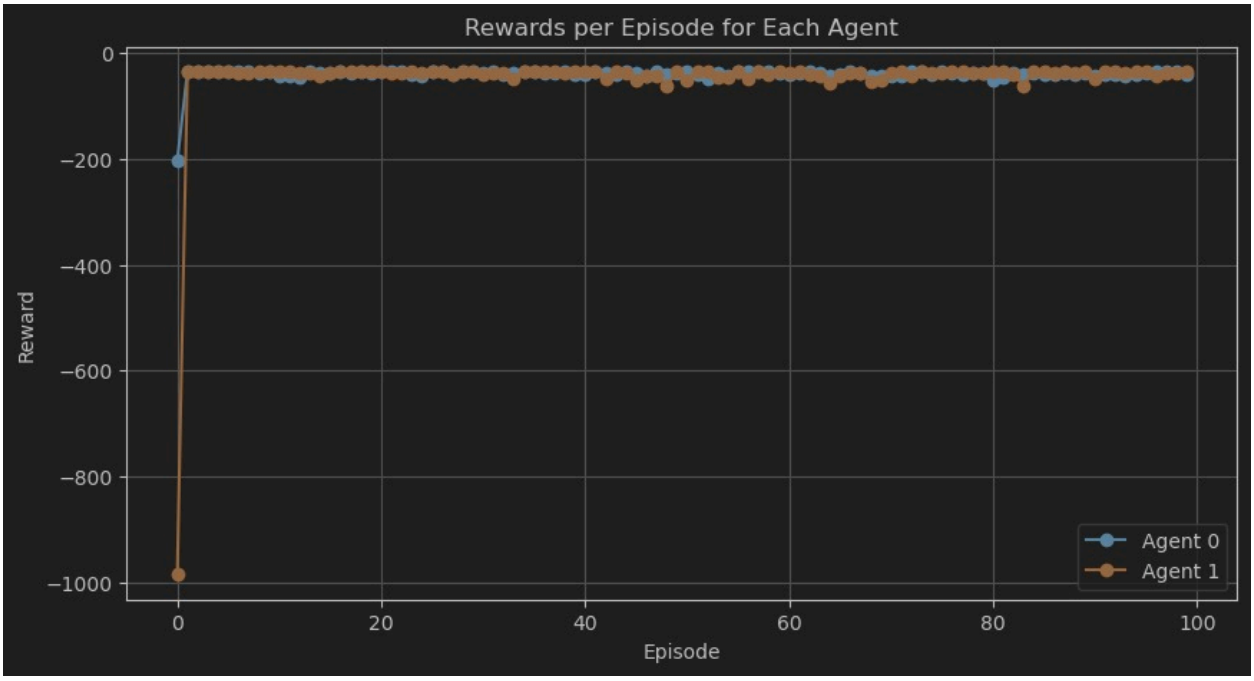
- Actor Critic Training: The training loop consists of a number of episodes, where in each episode, the agent interacts with the environment to collect experience. For each agent in the environment, the following steps occur:
  The actor decides an action based on the current state.
  The environment processes the action and returns the next state, reward, and whether the episode is done.
  The critic estimates the value of the state.
  The advantage is calculated, representing the difference between the actual reward and the critic's estimate.
  The actor and critic networks are updated:
  The actor is updated using the advantage to compute the loss.
  The critic is updated to minimize the mean squared error between its estimated value and the actual reward + discounted future value.
  Rewards are accumulated for plotting and analysis after training.

RESULTS AND EVALUATION

We ran each algo for single stock dataset and evaluated their performances to check which one is better and then used it for top 25 stocks by weightage from S&P 100 index.

DQN

- Both agents show a substantial negative reward in the first episode, with Agent 0 experiencing a particularly sharp drop. This indicates that the agents are learning from a significant loss due to their initial actions in the trading environment.
- After the initial episodes, both agents appear to stabilize around a reward level close to zero, with Agent 1 showing slightly better consistency. This stabilization could be indicative of the agents learning to avoid actions that result in significant losses,

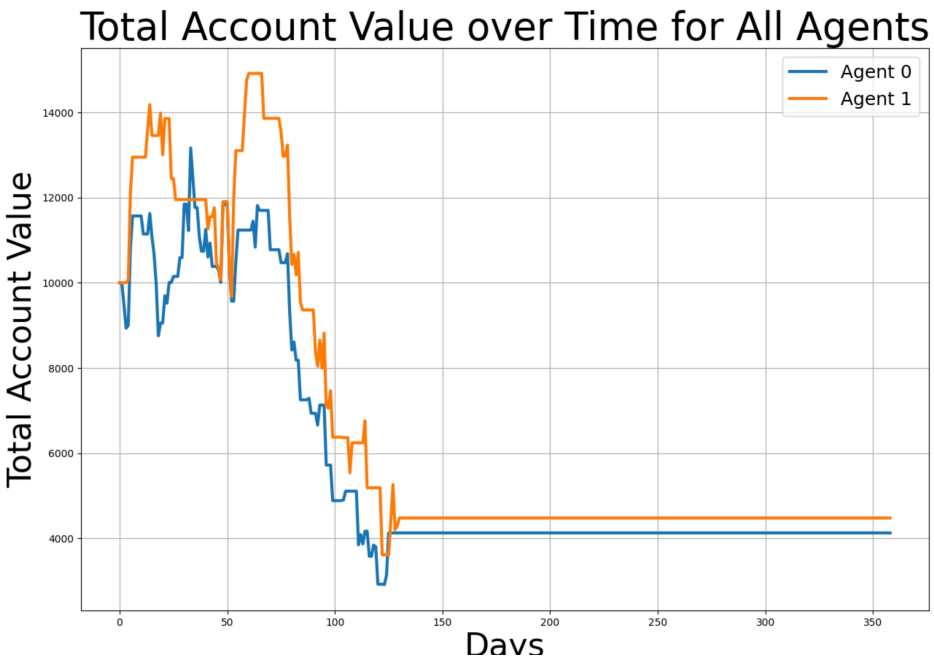


possibly converging towards a safer strategy that minimizes losses but doesn't necessarily optimize for high gains.

- Agent 1 consistently performs slightly better than Agent 0 throughout the episodes, maintaining a reward level closer to zero. This might suggest that Agent 1 has either learned a more effective strategy or is less prone to taking risky actions that result in large losses.
- The general low reward levels close to zero suggest that while the agents have learned to avoid catastrophic losses, they haven't mastered strategies to achieve higher gains

A2C

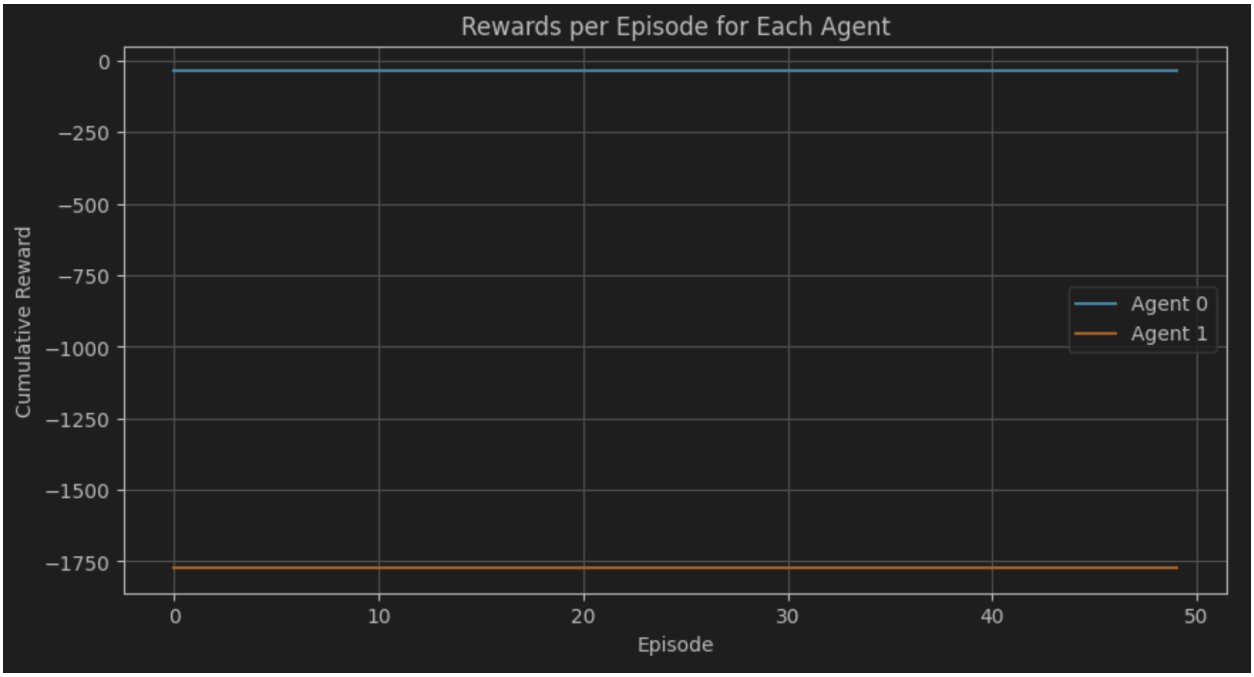
The reward fluctuations indicate that the agents are exploring different strategies or are sensitive to initial conditions or specific market scenarios presented in each episode.
The graph displays the performance of two agents (Agent 0 and Agent 1) in a stock trading simulation over 100 episodes, as measured by total rewards per episode. Both agents exhibit significant volatility in their performance, with rewards generally oscillating between -1100 and -1500. Notably, there isn't a clear upward or downward trend, indicating that neither agent consistently improves or degrades in performance over time. The rewards of both agents frequently cross paths, suggesting that there is no consistent outperformer between the two.
The peak performance appears sporadically, such as near episode 90 for Agent 0, showing a short-term gain in learning or favorable conditions, while the overall pattern shows that the agents might be exploring different strategies or facing a dynamic environment that affects their decision-making.



Initially, both agents start with similar account values and show a tendency to fluctuate together, suggesting either a shared trading strategy or similar responses to market conditions. Notably, after the first 50 days, both agents experience a pronounced and consistent decline in account value, continuing until around day 250 where the values plateau.

This decline in value indicates that the agents are not successfully adapting to the market or that their trading strategies are inherently unprofitable in the given simulation environment. The similar trajectories of both agents suggest that they might be exposed to the same types of risks or inefficiencies in their trading algorithms

DDQN



This constant level of rewards indicates that the agents are not learning from their interactions with the environment or that the learning has stagnated. The rewards are uniformly low without improvement over time, it suggests issues such as ineffective learning strategies, or inadequate exploration.

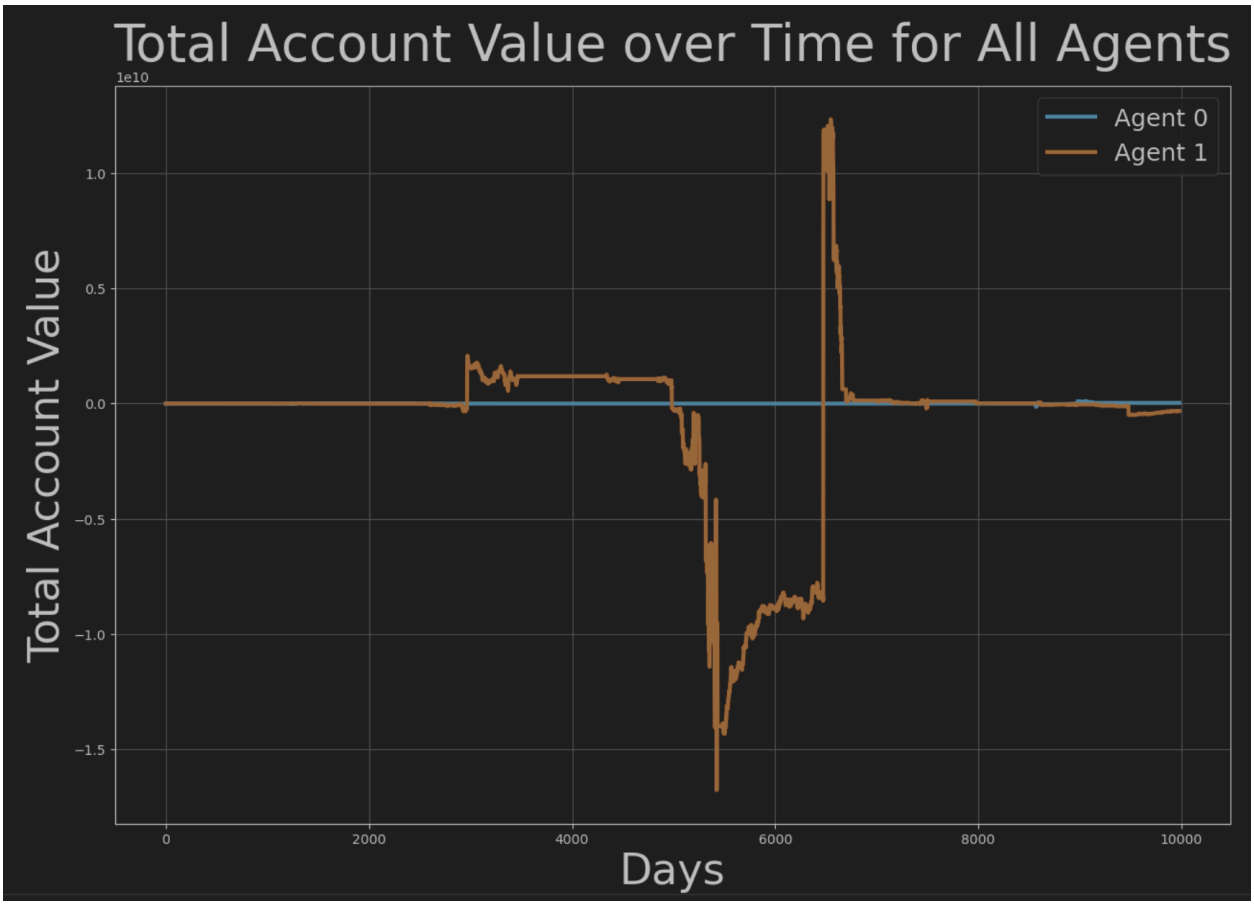
Now we will run the Advantage Actor-Critic (A2C) model to trade the top 25 stocks by weightage from the S&P 100 index. This approach focuses on the most influential stocks within the index, aiming to capitalize on their market impact.

A2C 25 stocks

Training Data

The graph depicts the total rewards per episode for two agents (Agent 0 and Agent 1) over 100 episodes. The reward values are negative throughout, indicating that both agents are consistently incurring losses. The rewards fluctuate within a range, mostly between -32,000 and -38,000, with occasional spikes, such as seen around episodes 20 and 80 for Agent 0, and a similar pattern for Agent 1 though with less intensity.

The general stability of the reward levels, albeit negative, suggests that both agents have reached a relatively stable strategy that, unfortunately, does not yield positive outcomes. The presence of sudden spikes might indicate episodes where the agents' strategies momentarily aligned more effectively with the market conditions, only to revert back to less profitable strategies in subsequent episodes.



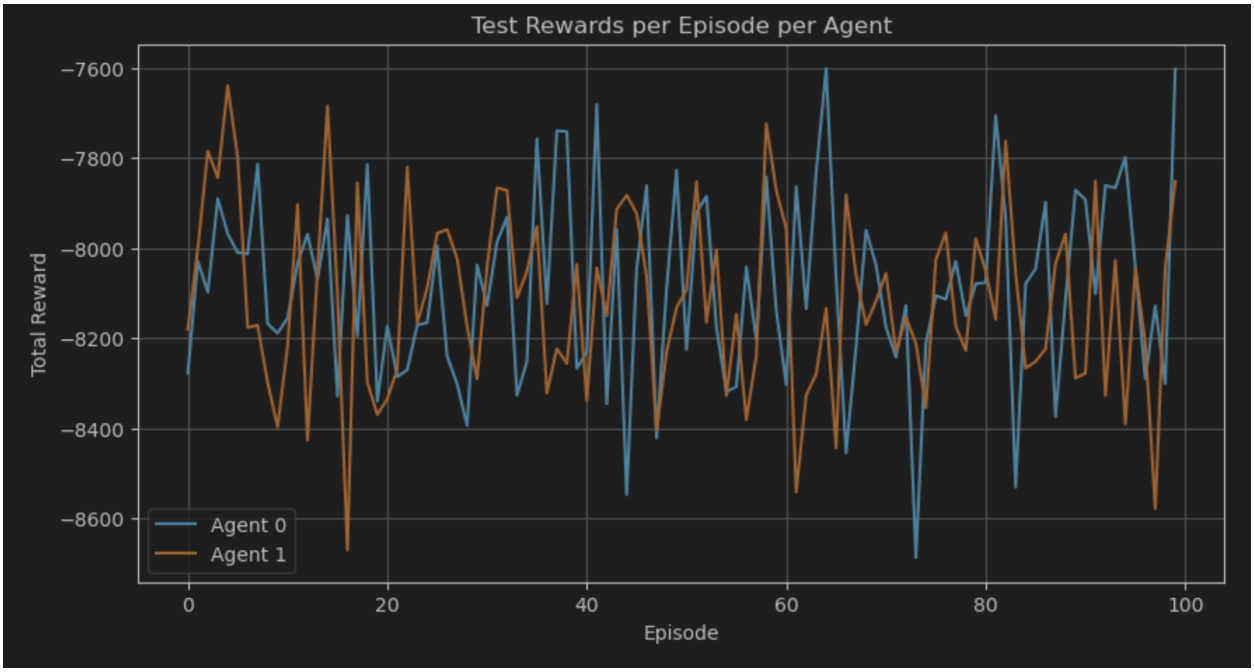
The graph shows the total account value over time for two agents (Agent 0 and Agent 1) over a span of approximately 10,000 days, as part of their A2C training in a trading simulation. The graph exhibits some dramatic fluctuations in account values, especially notable are the sharp declines and subsequent recoveries in both agents' account values.

Initially, both agents' values hover around zero, suggesting a neutral or breakeven trading performance. However, a significant drop occurs near 4,000 days for Agent 1, plunging into deep negative values. This indicates a catastrophic trading decision or a series of poor trades that led to substantial losses. Interestingly, Agent 0 experiences a similar, though less dramatic, drop around the same time.

The most striking feature is the sharp spike for Agent 0 around day 6,000, where the account value dramatically increases to positive extremes and then sharply falls back to near zero. This could reflect an exceptionally profitable trade or series of trades, possibly exploiting some anomaly or specific market condition not consistently present throughout the training period.

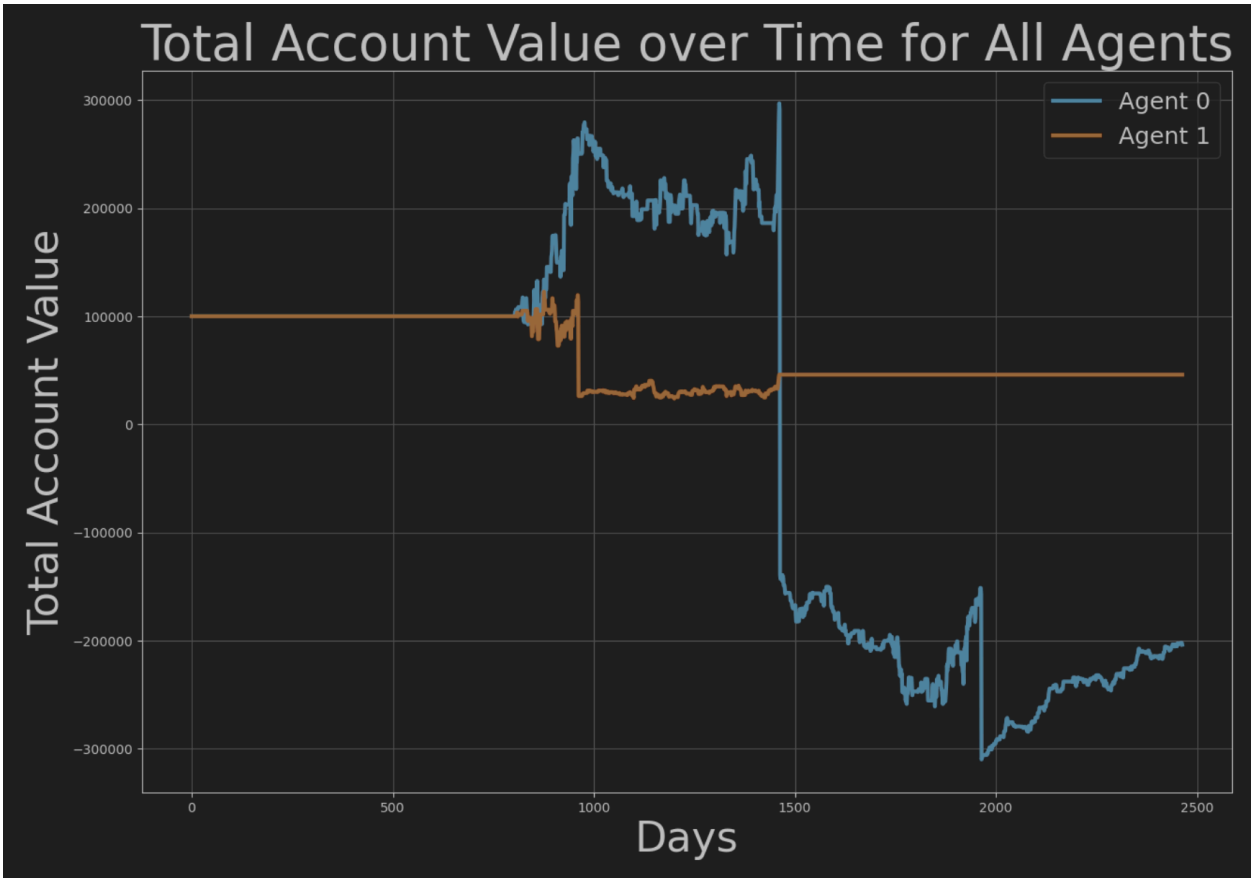
Following these events, both agents' values stabilize somewhat, but Agent 1 shows a gradual improvement in performance, recovering losses and steadily increasing in account value until around day 8,500, after which it stabilizes near zero.

Testing Data



The test rewards per episode for Agents 0 and 1 over 100 episodes using the A2C model on the top 25 stocks by weightage from the S&P 100 index show a range of results with considerable volatility. Both agents consistently receive negative rewards throughout the test, with values generally oscillating between -7600 and -8600. The graph suggests neither agent has developed a profitable trading strategy as both consistently incur losses in each episode.

The overlapping paths of the two agents imply similar performance patterns, which could indicate they are utilizing comparable strategies or are equally ineffective under the market conditions simulated. The rewards do not show any clear upward or downward trend, which typically would indicate learning and adaptation. Instead, the rewards fluctuate within a defined range, suggesting the agents have possibly converged to a suboptimal policy that fails to capitalize on market opportunities.



The graph depicts the total account values of two agents (Agent 0 and Agent 1) over approximately 2500 days. Throughout the early stages, Agent 0's account value remains relatively stable, fluctuating slightly around zero before experiencing a significant rise and subsequent sharp decline. This spike suggests a possibly aggressive trading strategy that initially succeeded but ultimately resulted in considerable losses. Following this, Agent 0's value stabilizes again near zero.
In contrast, Agent 1's account value consistently decreases from the onset, indicating a gradual but steady loss over time, with no significant recoveries. This steady decline suggests that Agent 1’s strategy may be fundamentally flawed or excessively conservative, leading to slow erosion of capital without substantial gains.

Conclusion

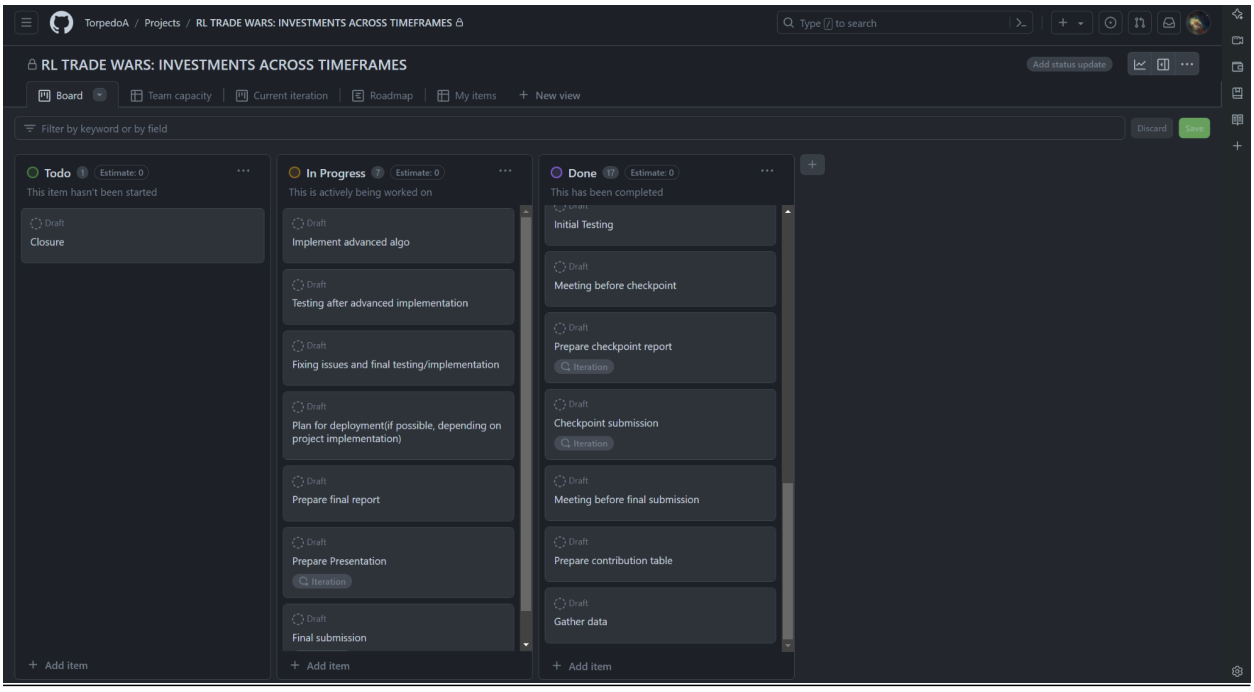
All models (DQN, DDQN, and A2C) exhibited significant variability in performance across different episodes and conditions. None of the models have consistently learned a profitable trading strategy as evidenced by frequent negative rewards and declining account values.

The DDQN and A2C models though, showed episodes of large gains but followed by equally significant losses, suggesting issues with stability and convergence. This might indicate that the models are either overfitting to particular market conditions or that they are too sensitive to outliers or noise in the trading data.
After the reset in the simulations, there was a notable change in the performance of the A2C model, suggesting that some adaptation or learning occurred, but it wasn't sufficient to maintain consistently positive outcomes. This points that the model may work if trained for a long period of time which we couldn’t do due to resource constraints. The model can be improved in future by regular evaluation and adjustment of the models based on performance metrics beyond simple profit and loss, such as the Sharpe ratio or maximum drawdown.

CONTRIBUTION TABLE

Team Member	Contribution
aarya2	33.33%
akapoor5	33.33%
eklavya	33.33%

PM TOOL



REFERENCES

- DQN - Assignment 2
- Financial market data sources such as Yahoo Finance, and Alpha Vantage for historical price data.
- S&P 100
- The Wall Street Journal
- Python libraries for RL (e.g., TensorFlow, PyTorch) and data analysis (Pandas, NumPy, yfinance).
- https://ieeexplore.ieee.org/abstract/document/8786132
- https://www.sciencedirect.com/science/article/abs/pii/S0957417419305822
- https://ieeexplore.ieee.org/abstract/document/935097
- https://ieeexplore.ieee.org/abstract/document/8283307
- https://towardsdatascience.com/deep-reinforcement-learning-for-automated-stock-trading-f1dad0126a02
- https://www.mlq.ai/deep-reinforcement-learning-trading-strategies-automl/
- https://medium.com/@diegodegese/unlocking-the-power-of-reinforcement-learning-how-it-can-improve-stock-trading-decisions-307ff4238532
- https://ieeexplore.ieee.org/abstract/document/8851831
- https://ieeexplore.ieee.org/abstract/document/9031159
- https://github.com/tensorflow/agents/blob/master/tf_agents/agents/dqn/dqn_agent.py
- https://github.com/keras-rl/keras-rl/issues/55
- https://github.com/udacity/deep-reinforcement-learning/blob/master/dqn/exercise/dqn_agent.py
- https://discuss.pytorch.org/t/runtimeerror-one-of-the-variables-needed-for-gradient-computation-has-been-modified-by-an-inplace-operation-torch-floattensor-64-1-which-is-output-0-of-asstridedbackward0-is-at-version-3-expected-version-2-instead-hint-the-backtrace-further-a/171826
- https://github.com/ChristoffelDoorman/Multi-Agent-Reinforcement-Learning/tree/main/DQN
- https://amulyareddyk97.medium.com/coding-multi-agent-reinforcement-learning-algorithms-683394556645
- https://github.com/rilwen/multi-agent-RL-A2C
- Assignment 1 - 3rd part for the environment's reference.