# Mini Project 2 – Part 1

## Overview

In this assignment, you will train a Reinforcement Learning (RL) agent to manage an inventory system effectively. The agent will learn how to:
- Buy raw materials.
- Convert raw materials into finished products.
- Optimize decision-making to balance costs, revenue, and cash flow.

The environment simulates dynamic market conditions, where product prices and demand fluctuate randomly. Your goal is to train an agent using Deep Q-Learning (DQN) to maximize profits while minimizing penalties.

This mini project challenges you to apply reinforcement learning in a realistic, decision-making environment.

This environment has been developed by me and as far as you are concerned it works exactly like the gymnasium environments you have worked with so far. Please look at the usage example provided to understand how it works and how to use it.

## Task

Your task is to train a reinforcement learning agent in the **InventoryManagementEnv** environment. The agent should learn a policy that effectively manages inventory, buys raw materials at the right time, converts materials into products to fulfill demand, and minimizes costs.

The environment ("InventoryManagementEnv") is provided to you as a class in the file called "Inventory_env_class.py". Do not open and make changes to this file as it will get corrupted and will not work.

## Understanding the Environment

***Observation Space (6 Features) is continuous***:

- raw_inventory: Number of raw materials in stock.

- product_inventory: Number of finished products in stock.

- raw_price: Current price of raw materials.

- product_price: Current price of finished products.

- demand: Current customer demand.

- cash: Available cash.

***Action Space (Discrete Choices):***

- 0: No operation (do nothing).

- 1: Buy one unit of raw material.

- 2: Convert two raw materials into one finished product.

***Reward Structure:***

The reward at each timestep is computed as:

Revenue from Sales: Finished products are automatically sold to meet demand.

Penalties: Applied for stockouts (when demand exceeds available product inventory), holding costs for raw and finished products, and invalid actions.

Conversion Bonus: A positive bonus is provided when a valid conversion (action 2) occurs.

Bankruptcy Penalty: Applied if cash drops to zero or below, terminating the episode.

## **Initializing the Environment** (see usage example provided)

Apart from importing the other libraries like gymnasium, etc. you will import the class and some of the methods within it from the file provided.

***from Inventory_env_class import InventoryManagementEnv, NormalizeObservation, ReplayBuffer, DQN***

```
# Create environment instance
env = InventoryManagementEnv()

# Normalize the observation space for better training performance
env = NormalizeObservation(env)

# Reset environment before starting
obs, _ = env.reset()
```

```
# Initialize the DQN networks (Q and Target networks)

policy_net = DQN(env.observation_space.shape[0], env.action_space.n).to(device)
target_net = DQN(env.observation_space.shape[0], env.action_space.n).to(device)
target_net.load_state_dict(policy_net.state_dict())
target_net.eval()

# Initialize Replay Buffer

replay_buffer = ReplayBuffer(capacity=10000)
```

**Render the Environment (give you information)**

Observe the environment's state updates. **Do not use this while training** as it will just print a lot of information, and you do not need it. Use it when you run the test on the policy after training.

```
env.render()
```

Rendering gives out the following information as an example –

> Raw Inventory: 1.0
>
> Product Inventory Before Sale: 0.0, After Sale: 0.0
>
> Raw Price: 5.47, Product Price: 19.44
>
> Demand: 4.49, Cash: 994.95

## Implementing the RL Agent

Key components to implement:

- Experience Replay Buffer: Stores past experiences for training.

- Deep Q-Network (DQN): Neural network using PyTorch for Q-learning.

- Training Loop: Uses experience replay and target network updates.

## Training the Agent

Define hyperparameters like - number of episodes, gamma, learning rate, epsilon decay, etc.

Initialize the environment and networks as mentioned before.

The environment can take anything between 5000 to 50000 episodes to converge. Convergence here means, the total rewards per episode starts at a high value negative value and stabilizes at a lower negative value (may never go to a positive value) and the average MSE per episode of the DQN network being trained (Q Net) starts at a higher value and stabilizes at a lower value.

To check this is happening, the training loop, while running, displays the total rewards per episode, the epsilon value per episode, the average MSE loss of the network being trained (Q network).

## Evaluating the Trained Agent

After training, test the trained policy to see if the agent is managing the inventory well. Test it by running for 10 episodes and see if it grows the initial $1000 in every episode.

## Submission Requirements

Work the whole solution in a Jupyter Notebook.

The assignment will be due on October 28th by midnight.

### Questions

1. What were the biggest challenges in training the agent and what were your key takeaways?
2. How well did your policy perform?
3. Considering that you all are students of science and engineering, what are some areas in your field you see can benefit from using RL and describe 1 or 2 of those possible applications may look like.

### Grading Criteria

1. Environment is setup correctly and the execution runs flawlessly – 30%
2. The training loop is correctly implemented as specified – 30%
3. The Policy testing function is correctly implemented and displays the results per episode using the render method – 20%
4. Questions are answered in a meaningful way with reasoning – 15%
5. Code is structured well, commented generously and the comments gives the reader an idea of what is going on in individual cells – 5%