Pseudo Code for RL Maturity Indices

```
1. SETUP AND ENVIRONMENT INITIALIZATION
   1.1 Define constants:
       - NUM_SUPPLIERS, NUM_WAREHOUSES, NUM_PRODUCTS
       - STATE_SIZE, ACTION_SIZE
       - TIME_HORIZON, DISCOUNT_FACTOR, STOCKOUT_PENALTY
       - MAX_PRODUCTION_CAPACITY, MAX_TRANSPORTATION_CAPACITY

   1.2 CLASS SupplyChainEnv:
       - Attributes:
         * inventory, lead_times, production_capacities, etc.
         * inventory_buffer (queues for in-transit items)
         * demand, step_count
       - reset():
         * Initialize or randomize environment variables
         * Clear buffers, step_count     0
         * Return current state
       - step(action_idx):
         * Convert action_idx into (order, routing, production_adjustments)
         * Compute reward (compute_reward)
         * Update environment states
         * step_count += 1
         * done = (step_count >= TIME_HORIZON)
         * Return next_state, rewards, done

2. REWARD CALCULATION
   SUBROUTINE compute_reward(state, action, ...):
     - Extract inventory, demand, lead_times, etc.
     - shipped_qty = min(order_qty, production_capacities, transport capacity)
     - service_level = (inventory + shipped_qty) / demand
     - ordering_cost, holding_cost, stockout_cost
     - lead_time_variance
     - Return reward vector [service_level, total_cost, lead_time_variance]

3. TIME-TO-RECOVER (TTR) CALCULATION
   SUBROUTINE calculate_ttr(env, agent, delta, perf_metric):
     3.1 Reset environment, define disruption_time
     3.2 For each time step:
         - Possibly introduce disruption
         - action = agent.act(state)
         - next_state, rewards, done = env.step(action)
         - Evaluate performance (e.g., mean service_level)
         - If performance recovers above delta, return TTR
     3.3 If never recovers, return TIME_HORIZON
```

```

4. TIME-TO-ADAPT (TTA) CALCULATION
   SUBROUTINE calculate_tta(env, agent, epsilon, max_steps):
     4.1 Reset environment, define disruption_time
     4.2 For each time step:
         - Possibly introduce major disruption (permanent capacity
    change)
         - agent.remember(...), agent.replay()
         - Periodically update target network & record model params
         - If param diffs < epsilon for multiple steps, return TTA
     4.3 If never stabilized, return max_steps

5. RL MATURITY EVALUATION
   SUBROUTINE evaluate_maturity_indexes(env, agent, num_runs,
    max_steps):
     5.1 FOR run in [1..num_runs]:
         - (ttr, rewards_hist) = calculate_ttr(env, agent)
         - (tta, param_hist)   = calculate_tta(env, agent)
         - Collect metrics (speedup, convergence_rate, adaptability
    , etc.)
     5.2 Return dictionary of maturity metrics, TTR/TTA lists,
    reward histories

6. DQN AGENT (DQNAgent):
   - model, target_model
   - memory (experience replay), epsilon, batch_size, etc.
   Methods:
     6.1 act(state):
         * epsilon-greedy over predicted Q-values
     6.2 remember(state, action, reward, next_state, done)
     6.3 replay():
         * sample mini-batch, compute target Q, train
         * update epsilon
     6.4 update_target():
         * sync target_model     model weights

7. MAIN SIMULATION:
   - Create env = SupplyChainEnv()
   - Create agent = DQNAgent()
   - metrics = evaluate_maturity_indexes(env, agent)
   - Extract TTR, TTA, reward histories
   - plot_reward_components(...)
   - Print or log final maturity indices

END
```

Listing 1: Pseudo Code for RL Maturity Indices