Great—here's a clean, unambiguous, **week-by-week cookbook** you can use for **any role** (Retailer, Wholesaler, Distributor, Factory). It matches the classic Beer Game timing (orders and shipments both have delays) and is spreadsheet-friendly.

Key notation (per role r)

- Lead times (choose your game's values; classic uses both = 2):
    - **L□** = shipment/transport lead time (downstream receives after L□ weeks)
    - **L_o** = information/order lead time (upstream receives your order after L_o weeks)
- State at **end of week t** (after all actions in week t):
    - **I□** = on-hand inventory (≥ 0)
    - **B□** = backlog / unmet demand (≥ 0)
    - **ShipPipe□[k]** = units scheduled to **arrive to you** in k weeks (k = 1…L□)
    - **OrderPipe□[k]** = units your **upstream will receive** from you in k weeks (k = 1…L_o)
- Per-week flow variables:
    - **A□** = shipments **arriving to you** this week
    - **R□** = **incoming order** you must satisfy this week
      • Retailer: R□ = customer demand D□
      • Others: R□ = order placed by your downstream L_o weeks ago
    - **S□** = **shipment you send** downstream this week
    - **O□** = **order you place** to your upstream this week
- Costs:
    - **Hold□ = h · I□**, **BacklogCost□ = b · B□**, **Cost□ = Hold□ + BacklogCost□**, **CumCost□ = $\Sigma_{1..□}$ $Cost_i$**

- Useful control quantity:
    - **Inventory position**: **$IP_t = I_t + (\Sigma\ ShipPipe_t[k]) - B_t$**

Initialization (week 0)

Pick sensible starting values (common classroom defaults):

- I_0 = initial stock (e.g., 12)
- $B_0 = 0$
- $ShipPipe_0[1..L_t]$ with some steady flow (often all zeros or a small constant)
- $OrderPipe_0[1..L_o]$ likewise (zeros to start is fine)

Weekly update order (do these steps in this exact sequence for each role)

Assume you are computing **week t = 1, 2, …**. All “previous week” values refer to week t−1.

1) Receive inbound shipment

- **$A_t = ShipPipe_{t-1}[1]$**
- Shift shipment pipeline down one:
    - For k = 1..L_t−1: **$ShipPipe_t[k] = ShipPipe_{t-1}[k+1]$**
    - Set **$ShipPipe_t[L_t] = 0$**
- Compute **available stock at start of week**:
    - **$Avail_t = I_{t-1} + A_t$**

2) Observe inbound order (demand to satisfy)

- If you're the **Retailer**: **R□ = D□** (exogenous customer demand).
- Otherwise (W/D/F): **R□ = OrderPipe□₋₁[1]** (the order your downstream placed L_o weeks ago).
- Shift order pipeline down one (your orders traveling upstream):
    - For k = 1..L_o−1: **OrderPipe□[k] = OrderPipe□₋₁[k+1]**
    - Set **OrderPipe□[L_o] = 0**

At this point, the total demand you must try to ship this week is **Need□ = B□₋₁ + R□**.

3) Ship to downstream (fill backlog first)

- **S□ = min(Avail□, Need□)**
- Update on-hand after shipping:
    - **I′ = Avail□ − S□**
- Update backlog:
    - **B□ = max(Need□ − S□, 0)**

Interpretation: you first clear any backlog B□₋₁, then the current order R□, limited by what you physically have after arrivals.

4) Decide this week's order to upstream

Choose **one** policy (examples):

- **Naïve echo (classic "dumb" agent):**
    - **O□ = max(0, R□)**
      (Order exactly what was ordered from you this week.)
- **Base-stock:**

- **Target** = chosen base-stock level
- Compute **IP′ = I′ + Σ ShipPipe□[k] − B□**
- **O□ = max(0, Target − IP′)**

- **PI controller (smooths oscillations):**
  - With gains k□, k_i and integral state Z:
    - Error **e = Target − (I′ + Σ ShipPipe□[k] − B□)**
    - **Z ← Z + e**
    - Control **u = k□·e + k_i·Z**
    - **O□ = max(0, round(R□ + u))** (optionally clamp to [0, Omax])

Use the **I′** and **B□** just computed (i.e., *after* shipping) so your order reflects the latest state.

5) Schedule your outbound order (to arrive upstream after L_o)

- Put this week's order at the **end** of your order pipeline:
  - **OrderPipe□[L_o] += O□**
    (From your perspective this tracks what will *arrive to upstream* in L_o weeks.)

6) (Factory only) Start production to replenish its own inventory

Two common options:

- **Simple/infinite source factory** (most classroom games): skip this: the Factory ships from on-hand like any other role and uses its **O□** as "start production," which returns as an **inbound arrival** after **L□**:
  - **ShipPipe□[L□] += O□** (this models production lead time feeding the Factory's *own* future arrivals)
- **Capacity-limited production**: cap O□ by capacity before adding to ShipPipe□[L□].

Non-factory roles do **not** add to their own ShipPipe; their upstream partner will generate your future arrivals when *they* ship.

7) Finalize end-of-week inventory & costs

- **$I_t = I'$** (ensure non-negative)
- **$Hold_t = h \cdot I_t$**, **$BacklogCost_t = b \cdot B_t$**, **$Cost_t = Hold_t + BacklogCost_t$**
- **$CumCost_t = CumCost_{t-1} + Cost_t$**

Repeat for week t+1.

Spreadsheet-ready column formulas ($L_t = L_o = 2$ case)

For each role r, set up columns (per week t row):

1. **Arrivals A_t** = value from your “Shipments due this week” column (the first element of your shipment pipeline)
2. **InventoryStart = $I_{t-1} + A_t$**
3. **IncomingOrder R_t** =
    - Retailer: customer demand D_t
    - Others: “Orders due this week” (first element of your order pipeline)
4. **DemandToSatisfy Need$_t$ = $B_{t-1} + R_t$**
5. **Shipment S_t = MIN(InventoryStart, Need$_t$)**
6. **InventoryEnd I_t = InventoryStart − S_t**
7. **Backlog B_t = MAX(Need$_t$ − S_t, 0)**
8. **PipelineOnHand** = **SUM(ShipmentsDueIn1..L_t)** (the rest of your shipment pipeline)
9. **InventoryPosition IP′ = InventoryEnd + PipelineOnHand − Backlog**

10. **Order O□** = (policy; e.g., **Naïve:** =R□; **Base-stock:** =MAX(0, Target − IP′))
11. **OrdersDueIn2** (end of row) **+= O□** (push to the last slot of the order pipeline)
12. **(Factory only) ShipmentsDueIn2 += O□** (production returns to Factory after L□)
13. **HoldingCost** = **h · InventoryEnd**
14. **BacklogCost** = **b · Backlog**
15. **TotalCost** = **HoldingCost + BacklogCost**
16. **CumCost** = **previous CumCost + TotalCost**

At the **top of next week**, shift both pipelines one column toward “due this week,” zero the far end, and iterate.

Common pitfalls this avoids

- Mixing up **orders** (information upstream) with **shipments** (material downstream). Keep two separate pipelines.
- Shipping with stock **before** adding arrivals. Always add **A□** first, then ship.
- Using old state for the order decision. Decide **after** shipping, based on **I′** and **B□**.
- Factory replenishment: model its **own** inbound via production lead time; downstream roles do **not** fill their own ShipPipes.