The Beer Game — Precise Rules & Implementable Math (Python-ready)

Below is a complete, code-oriented spec for a classic four-echelon Beer Game: **Retailer → Wholesaler → Distributor → Factory**. It's written so you can hand it to Codex and implement directly. All symbols and arrays are consistent across roles and time.

---

1) Core Entities & Indices

- Discrete time: weeks t=0,1,2,…,T−1t = 0,1,2,\dots,T-1.
- Roles r∈{Retailer,Wholesaler,Distributor,Factory}r \in \{\text{Retailer}, \text{Wholesaler}, \text{Distributor}, \text{Factory}\}.
- Each role operates the same state machine; only the **external demand source** differs:
    - Retailer's **customer demand** DtD_t arrives exogenously.
    - Wholesaler's demand is **Retailer's order**; Distributor's demand is **Wholesaler's order**; Factory's demand is **Distributor's order**.

---

2) Parameters (per role unless noted)

- LshipL^{\text{ship}} — **shipment lead time** (integer weeks, e.g., 2). Shipments take LshipL^{\text{ship}} periods to arrive.
- LprodL^{\text{prod}} — **production lead time** for Factory only (integer weeks, e.g., 2). Often modeled as an extra pipeline like shipping.
- hh — **holding cost** per unit per week (non-negative float).
- pp — **backlog/penalty cost** per unit per week (non-negative float).
- I0I_0 — initial on-hand inventory (non-negative int).
- B0B_0 — initial backlog (unfulfilled demand; usually 0).
- Q0Q_0 — initial **incoming shipment pipeline** (length LshipL^{\text{ship}}); array of future arrivals each week.
- P0P_0 — initial **production pipeline** (Factory only; length LprodL^{\text{prod}}).
- Policy parameters (optional) for an order policy (e.g., base-stock SS, safety stock, etc.).

- Demand series $\{D_t\}_{t=0}^{T-1}$ for Retailer (random or scripted). Upstream roles **do not** see D_t directly—only the orders they receive from their downstream partner.

---

3) State Variables (per role, each week t)

- I_t — on-hand inventory **at start** of step t.
- B_t — backlog **at start** of step t (unmet demand/orders from downstream).
- A_t — arrival from upstream **at start** of step t (pops from shipment pipeline).
- X_t — amount **available to ship** to downstream during step t after arrivals and before shipping (see sequence below).
- S_t — **shipment sent** to downstream at step t (bounded by available stock and downstream total demand).
- O_t — **order placed** to upstream at step t.
- Pipeline arrays:
    - **Ship pipeline** SHIPPIPE_t — length L^{ship}; S_t is pushed to the back, and the front element is A_{t+1}.
    - For Factory with production lead L^{prod}: **Prod pipeline** PRODPIPE_t; production M_t is pushed in, and the front element becomes **available finished goods** F_{t+1} next step.
- Costs:
    - **Holding cost**: $C^{\text{hold}}_t = h \cdot \max(I_{t+1}, 0)$.
    - **Backlog cost**: $C^{\text{back}}_t = p \cdot B_{t+1}$.
    - Total role cost $C_t = C^{\text{hold}}_t + C^{\text{back}}_t$. (You may add order or production costs if desired.)

---

4) Weekly Step Sequence (exact order of operations)

For **each role** r, in **each week** t:

1. **Arrivals land**

- Pop the front of SHIPPIPE_t → A_t units arrive from upstream.
- Update on-hand: $I_t \leftarrow I_t + A_t$.
- For Factory only: pop the front of PRODPIPE_t → finished goods F_t also land; $I_t \leftarrow I_t + F_t$.

2. **Observe incoming demand from downstream**
   - Retailer: demand is D_t (exogenous).
   - Others: demand is **the downstream role's order** from the same week t: $\text{dem}_t^{(r)} = O_t^{(\text{downstream}(r))}$.
   - Compute **total demand to fulfill** this step: $\text{TOTDEM}_t = \text{dem}_t^{(r)} + B_t$ (new demand plus backlog).
3. **Ship to downstream (satisfy demand FIFO)**
   - Units available to ship: $X_t = I_t$.
   - Shipment sent: $S_t = \min(X_t, \text{TOTDEM}_t)$.
   - Update inventory immediately: $I_t' = I_t - S_t$.
   - Update backlog: $B_{t+1} = \text{TOTDEM}_t - S_t$. (All unmet becomes backlog.)
   - Push S_t into the back of **downstream's** SHIPPIPE; it will arrive after L^{ship} weeks.
4. **Place order upstream**
   - Decide O_t according to the chosen policy (see §7):
     - The **order is a request**, not guaranteed to be fulfilled immediately.
   - Push O_t into **upstream's incoming demand stream** (that role will see it as its dem_t).
5. **Factory production decision (Factory only)**
   - Factory chooses **production start** M_t. (Often $M_t = O_t$ or policy-driven.)

    - Push MtM_t into **PRODPIPE**; finished goods become available after LprodL^{\text{prod}} weeks.

6. **Costs are assessed at end of week**
    - Set It+1=It'I_{t+1} = I_t'. (No negative inventory; it's already non-negative because we cap shipments.)
    - Compute costs for the role this week:
      Cthold=h·It+1C^{\text{hold}}_t = h \cdot I_{t+1},
      Ctback=p·Bt+1C^{\text{back}}_t = p \cdot B_{t+1},
      Ct=Cthold+CtbackC_t = C^{\text{hold}}_t + C^{\text{back}}_t.

Notes
• **No partial backorders by choice**: backlog always carries forward; there is no lost sales in the classic game.
• **Conservation**: You can never ship more than you have on hand; all unmet demand becomes backlog.
• **Lead times** are implemented purely via FIFO queues (pipelines).

---

5) Initialization

At t=0t=0 for each role:

- I0=I_0 = given (e.g., 12).
- B0=0B_0 = 0.
- SHIPPIPE0=\text{SHIPPIPE}_0 = array of length LshipL^{\text{ship}}, each element often initialized to a steady-state flow (e.g., 4 units each), or zeros.
- Factory only: PRODPIPE0=\text{PRODPIPE}_0 = array of length LprodL^{\text{prod}} (zeros or steady-state).
- Retailer: demand stream D0,…,DT−1D_0,\dots,D_{T-1} given or generated.
- Upstream/downstream links wired: retailer's upstream is wholesaler; wholesaler's upstream is distributor; distributor's upstream is factory; factory's upstream is **none**.

---

6) Cost Calculations (explicit)

At the **end** of week tt:

- Holding cost:

Cthold=h·It+1.C^{\text{hold}}_t = h \cdot I_{t+1}.

- Backlog (shortage) cost:

Ctback=p·Bt+1.C^{\text{back}}_t = p \cdot B_{t+1}.

- Total role cost this week:

Ct=Cthold+Ctback.C_t = C^{\text{hold}}_t + C^{\text{back}}_t.

- Horizon totals:

Crole=∑t=0T−1Ct,Csystem=∑rolesCrole.C^{\text{role}} = \sum_{t=0}^{T-1} C_t, \quad C^{\text{system}} = \sum_{\text{roles}} C^{\text{role}}.

Optional extras you may include:

- **Order cost** k·1[Ot>0]k \cdot \mathbb{1}[O_t>0] or linear per-unit order cost.
- **Capacity limits** on shipping St≤SmaxS_t \le S^{\max} and/or production Mt≤MmaxM_t \le M^{\max}.
- **Lost sales**: replace backlog with lost demand; change cost accordingly (not standard).

---

7) Order/Production Policies (plug-and-play)

You can implement any policy. Two common ones:

A) Base-Stock (Order-up-to) Policy

- Define **inventory position** at time tt:

IPt=It'−Bt+1+∑(on the way to me)⏟SHIPPIPE incoming.\text{IP}_t = I_t' - B_{t+1} + \underbrace{\sum \text{(on the way to me)}}_{\text{SHIPPIPE incoming}}.

(Use It'I_t' **after shipping** this step.)

- Target level SS (per role).
- Order:

Ot=max(0, S−IPt).O_t = \max\big(0,\; S - \text{IP}_t \big).

- Factory production typically set to Mt=OtM_t = O_t (or clip to capacity).

B) Proportional-Integral (PI) Heuristic

- Let et=target_inv−It+1+Bt+1e_t = \text{target_inv} - I_{t+1} + B_{t+1}.
- Ot=α·dem_forecasttt+β·et+γ·∑k=0tekO_t = \alpha \cdot \text{dem_forecast}_t + \beta \cdot e_t + \gamma \cdot \sum_{k=0}^t e_k.
- Clip to non-negative; integers if desired.

For the classic classroom game, base-stock is simplest and deterministic.

---

8) Exact Python-Friendly Step Function

Below is a precise, language-agnostic signature with all inputs/outputs you need for each role per week. (Codex can convert to dataclasses easily.)

Inputs at week tt (for a role):

- I (int), B (int)
- ship_pipe (deque/queue len = L_ship), where ship_pipe[0] arrives **now**
- For Factory also: prod_pipe (len = L_prod)
- downstream_demand (int) ← D_t for Retailer, or the downstream role's O_t
- Parameters: h, p, L_ship, (L_prod for Factory), any policy params/state (e.g., S)
- Optionally: capacity_ship, capacity_prod

Outputs:

- I_next, B_next
- S_t (shipment sent), O_t (order placed), A_t (arrival), (F_t for Factory)
- updated ship_pipe_next, (prod_pipe_next)
- cost_t

Algorithm skeleton (non-Factory):

1. A_t = ship_pipe.popleft(); I += A_t
2. dem = downstream_demand
3. TOTDEM = dem + B

4. S_t = min(I, TOTDEM) (clip by capacity if modeled)
5. I_prime = I - S_t
6. B_next = TOTDEM - S_t
7. **Policy**: compute O_t using I_prime, B_next, and pipeline contents (see base-stock)
8. ship_pipe.append(S_t) (to be received by downstream after L_ship)
9. I_next = I_prime
10. cost_t = h*I_next + p*B_next

Factory adds:

- Step 1a. F_t = prod_pipe.popleft(); I += F_t
- Step 7a. Decide M_t (e.g., M_t = O_t); push prod_pipe.append(M_t)

---

9) Worked Micro-Example (single role, single step)

Assume a non-Factory role, Lship=2L^{\text{ship}}=2, h=0.5h=0.5, p=1.0p=1.0.
At start of week tt:

- It=8 I_t = 8, Bt=3B_t=3
- SHIPPIPEt=[4,2] \text{SHIPPIPE}_t = [4, 2] (so At=4A_t=4 now)
- Downstream demand this week (new) = 5
- Base-stock target S=15S=15

Step:

1. Arrive: At=4A_t=4 → I=8+4=12I = 8+4=12
2. Demand: dem=5\text{dem} = 5, TOTDEM=5+3=8\text{TOTDEM} = 5 + 3 = 8
3. Ship: St=min(12,8)=8S_t = \min(12, 8) = 8; I'=12−8=4I' = 12 - 8 = 4; Bt+1=0B_{t+1} = 0
4. Inventory position:
   Incoming pipeline sum = ship_pipe contents **after** pop but **before** appending this week's outbound to downstream **(careful: IP counts inbound to me, not outbound I send)**. For this role, inbound is whatever **its upstream** has already shipped. Implementation detail: store **my** inbound pipeline separately; outbound pipe

belongs to the downstream role.
So \text{IP}_t = I' - B_{t+1} + \text{sum(inbound_to_me)}. Suppose inbound_to_me (future arrivals) now totals 22. Then IPt=4−0+2=6\text{IP}_t = 4 - 0 + 2 = 6.

5. Order: Ot=max(0,S−IPt)=max(0,15−6)=9O_t = \max(0, S - \text{IP}_t) = \max(0, 15 - 6) = 9

6. Costs: It+1=4⇒Cthold=0.5·4=2.0 I_{t+1} = 4 \Rightarrow C^{\text{hold}}_t = 0.5 \cdot 4 = 2.0; Bt+1=0⇒Ctback=0B_{t+1}=0 \Rightarrow C^{\text{back}}_t = 0; total Ct=2.0C_t=2.0.

7. Pipe updates: push St=8S_t=8 into **downstream's inbound** pipeline (they'll get it in 2 weeks).

Implementation tip: **Keep inbound and outbound pipelines separate** per arc to avoid double-counting. The upstream's shipment becomes your inbound in LshipL^{\text{ship}} weeks.

---

10) System Wiring (message passing per week)

At each week tt:

1. **All roles** pop arrivals, update inventory, fulfill, compute S_t.

2. **Push shipments**: each role's S_t is appended to the **downstream role's inbound pipeline queue**.

3. **Compute orders** O_t, then **deliver orders** to upstream as that role's **demand input** for week tt.

4. **Factory** also pushes production M_t into its prod pipeline.

5. Compute costs and log KPIs.

To avoid simultaneity issues, do it in **two passes** each week:

- **Pass A** (receive & ship): arrivals pop → ship & compute post-ship states (but don't mutate other roles yet)

- **Pass B** (commit): push shipments to downstream pipelines; pass orders upstream; Factory pushes production.

---

11) KPIs & Outputs

Per role and system:

- Weekly: I_t, B_t, A_t, S_t, O_t, costs, pipeline snapshots.
- Totals: cumulative holding, backlog, total cost; service level ∑tSt∑tdemt\frac{\sum_t S_t}{\sum_t \text{dem}_t}; bullwhip indicators (var(Ot)(O_t)/var(Dt)(D_t)).

---

12) Determinism & Randomness

- If you want repeatable runs, fix a RNG seed and pre-draw the Retailer's demand series.
- Upstream roles **must not** see the future Dt+kD_{t+k}.

---

13) Reference Defaults (classic classroom settings)

- T=50T = 50 weeks
- Lship=2L^{\text{ship}} = 2 for all arcs
- Factory Lprod=2L^{\text{prod}} = 2
- h=0.5h = 0.5, p=1.0p = 1.0 (per unit-week)
- Initial I0=12I_0 = 12, B0=0B_0 = 0
- Inbound pipelines initialized with steady 4 units/week (optional)
- Retailer demand DtD_t: 4 units/week for weeks 0–3, then 8 thereafter (the "shock") — or use your own series.

---

14) Minimal Data Structures (Python hint)

- RoleState: I, B, inbound_pipe: deque[int], optional prod_pipe: deque[int] (Factory only), policy params, logs.
- Edge: downstream inbound pipeline is a deque residing with the **downstream** role.
- One step(role_state, downstream_demand) function per role that returns (state_next, S_t, O_t, cost_t).
- Week loop orchestrator that:

1. collects arrivals for all roles,
2. calls step (compute but **stage** shipments and orders),
3. commits staged shipments and orders to the appropriate queues/inputs,
4. accumulates costs and logs.

---

15) Common Pitfalls to Avoid

- **Shipping more than on-hand**: always cap S_t by available inventory after arrivals this week.
- **Using outbound pipeline in IP**: inventory position includes **inbound to me**, not what I've promised to someone else.
- **Order timing**: orders placed at week tt become the upstream role's **demand in the same week tt** (information lead time = 0), but **physical shipments** respect LshipL^{\text{ship}}.
- **Factory creation of goods**: production enters prod_pipe and only becomes inventory after LprodL^{\text{prod}} weeks.
- **Backlog arithmetic**: always B_{t+1} = previous_backlog + new_demand - shipped (never negative).

If you want, I can turn this spec into a clean Python scaffold (dataclasses + deterministic simulator + pluggable policies) in a single file.