Power Grid RL Environment Design

1. Environment Components

- **N main generators**: each auto-scales between min & max output.
- **N emergency generators** (bundled): agent can boot or shut them down.
- **1 battery**: can discharge or recharge (charging adds to demand).
- **1 high-priority zone**
- **2 low-priority zones**: each with different scaled demand curves.
- **24-hour dynamic demand cycle**, repeating every episode.
- The agent operates in a **continuous time-step** simulation (e.g., hourly steps) and must keep demand met while minimizing cost and penalties.
- **Demand** is based on a shared Gaussian peak function, scaled per zone.

---

2. Load Zone Demand Functions

Each zone's demand over time `t ∈ [0, 24]` (in hours):

```
def demand_profile(t, scale=1.0):
    base = 0.5
    morning_peak = 0.4 * np.exp(-(t - 8)**2 / (2 * 2**2))
    evening_peak = 0.6 * np.exp(-(t - 19)**2 / (2 * 2**2))
    return scale * (base + morning_peak + evening_peak)

D_hi = demand_profile(t, scale=1.0)
D_lo1 = demand_profile(t, scale=0.7)
D_lo2 = demand_profile(t, scale=0.5)
```

---

3. Generator Behavior

Each **main generator i**:

- Output auto-scales to meet demand **within limits**:

    - `G[i]_min ≤ output ≤ G[i]_max`
- May fail with probability `p_fail` per step:
    - Goes offline and heals after `T_down` steps.

Each **emergency generator i** (bundled with main gen):

- Controlled by agent:
    - Can be booted (`start_time = T_boot`)
    - Can be **shut down manually** to preserve runtime
- When online:
    - Produces fixed power or is controllable
    - Has limited `runtime_remaining`, decremented when running

---

4. Battery Dynamics

- SoC ∈ [0, max_capacity]
- If agent **discharges**:
    - Draws power, SoC decreases
- If agent **charges**:
    - SoC increases
    - Charging **adds to total demand**, so generator/battery must cover it

Battery control options:

- `a_batt ∈ {0: idle, 1: discharge, 2: charge}`

---

5. Observation Space (Expanded)

Per step, agent sees:

Category	Variables
Time	`t_norm ∈ [0,1]`
Battery	`SoC`, `a_batt_mode`
Load	`D_hi`, `D_lo1`, `D_lo2`
Generator[i]	`G[i]_online ∈ {0,1}`, `G[i]_fail_timer`,

	`G[i]_min`, `G[i]_max`
Emergency[i]	`E[i]_online`, `E[i]_start_timer`, `E[i]_runtime_left`

Observation dimension:

`1 (time) + 3 (loads) + 1 (battery) + N × (3 + 3) = 4 + 6N`

---

6. Action Space

For each step:

- `a_batt ∈ {0,1,2}` — idle, discharge, charge
- `a_em[i] ∈ {0,1,2}` — do nothing, boot emergency, shut down
- `a_shed_lo1 ∈ {0,1}` — shed zone 1
- `a_shed_lo2 ∈ {0,1}` — shed zone 2

Total dimension:

`3 (battery) + N×3 (em control) + 2 (load sheds)`

---

7. Power Allocation Logic

1. **Total demand** =

   `D_hi + D_lo1*(not shed) + D_lo2*(not shed) + battery_charge(if charging)`

2. **Available power** =

   `Σ active main outputs + Σ emergency outputs + battery_discharge(if discharging)`

3. **Balance**:

   - Try to meet demand in this order:
     1. D_hi
     2. D_lo1
     3. D_lo2
     4. battery charge

---

8. Reward Function

At each step:

```
reward = (
    + w_hi * 1[hi_zone powered]
```

```
    + w_lo1 ∗ 1[lo1 powered]
    + w_lo2 ∗ 1[lo2 powered]
    - c_batt_discharge * energy_drawn
    - c_batt_charge * energy_charged
    - Σ c_em_boot * 1[emergency i booted]
    - Σ c_em_run * 1[emergency i running]
    - Σ c_em_idle * 1[emergency i idle but online]
    - Σ c_fail * 1[main gen i failed]
    - c_shed * 1[any load zone shed]
)
```

- Emergency generators have **run cost** and **boot cost**
- Penalty for keeping unused emergency generators online
- Charging battery has a small cost since it adds to demand

✅ Ready to Proceed?

Would you like:

1. A **code scaffold for this environment** (e.g., `PowerGridEnv(gym.Env)`)?
2. A **visual flowchart** of how state → action → reward flows?
3. Or begin with **Q-table / PPO agent plan** for this?

Let's move to implementation.