Deal-or-No-Deal: Environment, Prompting, and RL Training Summary

October 26, 2025

1 Environment

Env ID: DealOrNoDialog-v0
Implementation: `deal_or_no_deal_env/env.py` (`NegotiationEnv`).

1.1 Game Description

This environment simulates a two-agent, multi-issue bargaining problem based on Lewis et al. (2017). The two agents (the learning agent and a partner) must negotiate the division of a fixed pool of items, which are available in three distinct types (e.g., books, hats, balls) with given counts.

Each agent has its own private **utility function** (a vector of values, one for each item type), which determines their preference and potential score. The agents negotiate over a limited number of turns (`max_turns`) by exchanging dialogue acts (PROPOSE, INSIST, AGREE, DISAGREE, END).

A successful negotiation—a "deal"—is reached if both agents AGREE on an allocation (o_A, o_B) that perfectly conserves the total item pool i (i.e., $o_A + o_B = i$). In this case, the agent receives a reward equal to the dot product of its utility vector and the items it receives: $r_A = u_A \cdot o_A$. The partner receives its own score $r_B = u_B \cdot o_B$. The episode terminates (often with zero reward) if an agent decides to END the negotiation, if a DISAGREE occurs, or if the turn limit is reached. The agent's goal is to maximize its own utility r_A while ensuring the partner also agrees to the deal.

1.2 State (Observation)

Dictionary with:

- **counts** (MultiDiscrete, size 3): total item counts.
- **my_utilities** (MultiDiscrete, size 3): agent utilities $u_A \in \{0..10\}^3$.
- **partner_utilities** (MultiDiscrete, size 3): included if `reveal_partner_utilities=true`, else zeros.
- **last_partner_act** (Discrete(5)): last partner dialogue act.
- **last_partner_offer_for_me** (MultiDiscrete, size 3): last partner offer for agent.
- **turns_remaining** (Discrete): remaining turns up to `max_turns`.

1.3 Actions

Dict: {`act_type` (Discrete(5)), `oA` (MultiDiscrete, size 3)}. Acts:

- 0: PROPOSE(`oA`) 1: INSIST(`oA`) 2: AGREE 3: DISAGREE 4: END

Action masking provided in `info[action_mask]` (AGREE valid only if partner proposed/insisted). Per-dimension caps for `oA` are exposed via `info[oA_max]`.

1.4 Rewards and Termination

- If both agree on a valid allocation with conservation $o_A + o_B = i$: reward $r_A = u_A \cdot o_A$, partner receives $u_B \cdot o_B$.
- Episodes terminate on AGREE, END (final selection), or when `turns_remaining` reaches zero.

A simple heuristic partner policy accepts offers meeting a utility ratio threshold (0.55 default; 0.5 if the agent used INSIST), else counter-proposes greedily.

1.5 Context Sampling and Dataset

By default, contexts are sampled uniformly over counts/utilities. If `use_dataset=true`, contexts are loaded from `deal_or_no_dialog.py` (`self_play` or `dialogues`); optional normalization enforces a points budget.

2 LLM Prompt and Hinting Mechanism

2.1 Prompt Template

Path: `configs/llm_prompt.txt`. The template enforces a strict single-line JSON output with keys `act_type`, `oA`, and `confidence`, and embeds current observation/context fields. Example lines from the template:
"Return ONLY one JSON object on a single line with keys: act_type (0–4), oA ([3 ints]), confidence (0–1)."

Template variables include: `counts_csv`, `my_utils_csv`, `last_partner_act_token`, `last_offer_csv`, `turns`, `p`, `history_str`. Prompt rendering performed by `llm/prompt.py` with optional history length capping.

2.2 Hint Injection Wrapper

Wrapper: `env_wrappers/hint_injector.py` (`HintInjectorEnv`). Every k steps (and on reset), a hint is requested and converted to features:

- **act_onehot** (5), **oA_norm** (3, normalized by caps), **confidence** (1), **h_avail** (1)

These are placed into `info[hint_features]` and concatenated to base features by the `HintAdapter`. Providers:

- **random**: uniformly legal actions and capped `oA`.
- **expert**: uses a PPO or supervised expert checkpoint if available; greedy fallback otherwise.
- **llm**: uses GROQ or local HF client; strict JSON parsing and legality checks vs. `action_mask`.

Retry/backoff with exponential delays; illegal or failed hints yield neutral features.

3 RL Training Setup

3.1 Algorithms

PPO and **REINFORCE** implementations consume concatenated base + hint features via a shared policy network with heads for act type, three `oA` categoricals, and value.

3.2 Configs and Hyperparameters

Default config files: `configs/ppo_config.yaml`, `configs/reinforce_config.yaml`. Key fields:

- **Env**: `id=DealOrNoDialog-v0`, `max_turns=10`.
- **Hints**: `mode` $\in \{none, random, llm, expert\}, cadencek$, `prompt path, provider, and models.`
- `Training (PPO): lr=3e-4,` $\gamma = 0.99$, $\lambda = 0.95$, `clip=0.2, epochs=4, minibatch=64, ent=0.0, vf=0.5, grad_norm=0.5, rollout=128; steps via num_train_steps.`
- `Training (REINFORCE): lr=3e-4,` $\gamma = 0.99$, `ent=0.01; steps via num_train_steps.`
- `Logging:  output dir, CSV/TensorBoard, save cadence.`

3.3 Feature Wiring

The `HintAdapter` builds base features from observation: counts (3), my utilities (3), optional partner utilities (3), last act one-hot (5), last partner offer (3), turns (1). Hint features add 10 dims. Action masking and `oA` caps are applied in both sampling and loss.

3.4 Opponents and Curriculum

The built-in partner is a heuristic (accepts above a ratio threshold, else greedy counter-offer). No staged curriculum is implemented by default; curricula can be emulated by varying `max_turns`, dataset usage, hint mode/provider, or switching opponent to `expert`.

3.5 Typical Runs

- PPO example: `--algo ppo --hint llm --config configs/ppo_config.yaml`
- REINFORCE example: `--algo reinforce --hint none --config configs/reinforce_config.yaml`

4 Prompt Examples

Template includes inlined examples; e.g., when partner proposes the full allocation and it matches the agent's utilities, the model should output AGREE with high confidence. The system additionally adds a strict system message in remote calls to force a single-line JSON.

5 References

Main: 2303.00001v1 (attached). See also Lewis et al. (2017), Kwon et al. (2021).