Korean 4-Ball Billiards: A Continuous, Deterministic, Sparse-Reward Benchmark Solved by Inference-Time Search

Doyeol Oh
KAIST
ohdoyoel@kaist.ac.kr

Byungmo Kang
KAIST
kbm03@kaist.ac.kr

Seojun Park
KAIST
tjwnsl8@kaist.ac.kr

Abstract

We study Korean 4-ball billiards as a continuous, deterministic, sparse-reward RL problem with a fast exact simulator. Plain off-policy RL outperforms PPO, and random-start continue-on-miss training generalizes best, but the bare task still plateaus below one point per inning. Structure-free aids such as an action curriculum and learned reward-model bonus do not break this ceiling. Explicit geometry does: a first-contact aim constraint plus four carom features lifts SAC from 0.487 to 6.460 points/inning. The final jump comes from inference-time verification rather than reward design: greedy depth-2 lookahead uses the simulator to turn a $\sim$6-point policy into chains up to $9{,}392$ consecutive scoring shots at 99.8% per-shot success. In this domain, learning supplies the prior, geometry makes it competent, and simulator search scales it.

1 The Korean 4-ball domain and learning environment

Game. Korean 4-ball (*sagu*) is a pocketless carom game played on a rectangular table with four balls: two cue balls (a white *player* cue and a yellow *opponent* cue) and two red object balls. A player scores exactly **one point** when their cue ball, struck once, caroms off **both** red balls in a single shot; contacting the opponent's cue ball is a **foul**. After a scoring shot the player continues from the resulting table state, so play is a streak of shots terminated by the first miss or foul. The game is a pure single-agent control problem—there is no adversarial interaction through the table state—which makes it a clean testbed for continuous-control RL with a physically grounded, sparse objective.

Environment. We implement a NumPy physics simulator and expose it as a Gym environment (`Billiards4BallInningEnv`). The simulator is a from-scratch reimplementation of standard billiards-physics models: the cue-tip impact follows Marlow's level-cue instantaneous-point model [1], free flight uses the classic rigid-sphere slip/roll friction transition (reproduced in [1,7]), ball–cushion rebound uses the Han carom model [2], and ball–ball contact is the standard equal-mass restitution exchange; coefficients and the closed forms follow the open-source PoolTool simulator [6], which we port and strip down to the four modules 4-ball play requires (no pockets, jumps, or cloth effects) for a $\sim$20$\times$ speedup. The **observation** is a 28-dimensional vector (4 balls $\times$ 7 state features: planar position, linear velocity, and angular velocity). The **action** is a 4-dimensional continuous control (θ, p, a, b): aim angle $\theta \in [0, 2\pi)$, relative power $p \in [0, 1]$, and side/vertical cue-tip offsets $a, b \in [-1, 1]$ producing spin (projected to the unit disk to forbid miscues). The **reward** is the rule-defined carom score $\{0, 1\}$ per shot; no shaping is used unless stated. An episode is an *inning* of up to `max_shots`$= 10$ shots; with *continue-on-miss* the inning always runs the full budget, exposing the policy to a diverse stream of mid-rack states. One environment step equals one shot, so a budget of, e.g., 10^6 steps means one million simulated shots. Unless noted we evaluate the **mean points per inning** over 100 innings with `max_shots`$= 10$, and additionally report per-shot foul and success rates. One shot costs ≈ 5 ms on a single CPU core, so 2×10^5 transitions take $\approx$ 15–20 minutes on a laptop.

Preprint.

2 Algorithms and learning paradigms on the sparse task

2.1 SAC vs. PPO vs. TD3

We first compare three standard algorithms on the *plain* environment (random start, continue-on-miss, no domain knowledge, reward = raw $\{0, 1\}$ score). All are trained to 400k steps—the learning curves are already in their knee region by then, so the budget does not change the ranking—over 5 seeds.

Table 1: Algorithm comparison on the plain sparse task (5 seeds, 400k steps, eval = 100 innings, `max_shots`= 10). *mean*= points/inning; *max*= best single inning over all seeds. **Take-away: off-policy methods beat PPO by $\sim 2.5\times$ on score; SAC and TD3 are statistically indistinguishable but SAC is more stable.**

algo	**mean±std**	max	foul/inn (%)
TD3	**0.460±0.114**	4	84
SAC	**0.418±0.038**	4	90
PPO	**0.170±0.040**	3	70

Off-policy SAC/TD3 reach 0.42–0.46 mean points, about $2.5\times$ PPO's 0.17, and only the off-policy methods ever reach a 4-point inning. PPO plateaus by 200k (curve flat at ~ 0.15) while the off-policy pair keeps climbing in a separate band, so no ranking reversal is possible at larger budgets: every transition PPO collects is used once and discarded, which is fatal when scoring transitions are rare, whereas SAC/TD3 replay them. TD3's mean is marginally higher but its seed variance is large (per-seed 0.33–0.62); SAC is markedly more stable (± 0.038). All methods stay weak overall—with `foul_penalty`= 0 fouls are unpenalized, so 70–90% of innings contain at least one foul and the best innings remain short—exposing the ceiling of the plain sparse task and motivating Sections 3–4.

2.2 Four training paradigms: start × miss handling

The environment admits four training regimes from the cross of *canonical* vs. *random* starting rack and *reset-on-miss* vs. *continue-on-miss*. We train SAC (400k, 3 seeds) under each and re-evaluate *all four under one unified protocol* (continue-on-miss, `max_shots`= 10, 100 innings) so they are comparable. The random-start eval measures generalization and is the primary metric.

Table 2: Training-paradigm comparison (SAC, 3 seeds, unified eval). **Take-away: random-start + continue-on-miss generalizes best and most stably; continue-on-miss is the dominant factor; canonical+reset overfits and collapses off its training rack.**

method	random-start mean±std	canonical mean	foul%/shot	success%/shot
canonical, reset	0.073±0.045	1.00	9.3	0.7
canonical, continue	0.353±0.101	1.33	20.4	3.5
random, reset	0.407±0.046	0.67	18.6	4.1
random, continue	**0.453±0.012**	0.67	20.3	4.5

Random + *continue* wins on the generalization metric (0.453) and is the most stable (± 0.012). Continue-on-miss is the dominant lever: switching reset→continue helps under both start regimes by exposing many mid-rack states per episode. Canonical+reset overfits its single rack and collapses to 0.073 on random racks. A key caution emerges from the training curves: ranked by trainer-reported episode return, *canonical*+*continue* (1.44) appears to dominate *random*+*continue* (0.37), but the unified eval *reverses* this—training-protocol return is a misleading model-selection signal across paradigms. We adopt random-start + continue-on-miss for all subsequent experiments.

3 Baseline and two attempts to improve it

We fix **SAC** (plain environment, random-start + continue-on-miss, 1M steps, 3 seeds) as the baseline because it is the most stable strong algorithm in Section 2. More budget does *not* break the ceiling: the 3-seed training curve knees near $\sim$250k and stays in a 0.40–0.47 band, while deterministic evaluation

reaches only 0.487 ± 0.097 points/inning. The bottleneck is missing structure, not training time, so we test two structure-free ways to help the baseline.

3.1 Attempt 1: staged action curriculum (failed)

We first release the 4-D action space in stages: aim only, aim plus power, then the full action (θ, p, a, b), warm-starting the same SAC policy across stages for 1M total steps. This does not help.

Table 3: Curriculum and RM-guided SAC vs. the plain SAC baseline (1M steps, 3 seeds). **Take-away: the curriculum underperforms the baseline; the RM lifts the mean somewhat but at a large extra data cost.**

method	mean±std	max	rand. eval	extra cost
plain baseline	0.487±0.097	4	0.053	—
staged curriculum	0.407±0.052	4	0.053	none
RM-guided	**0.840±0.000**	4	0.087	10^7 offline shots

The curriculum is a clear *failure*: mean falls to 0.407 ± 0.052, below the plain baseline, and the random-start eval stays at 0.053. The fixed release order spends budget on constrained sub-problems whose optima are not useful warm-starts for the full action space.

3.2 Attempt 2: learned reward-model (RM) guidance

We also train a small state–action model $V_{\mathrm{rm}}(s, a) \approx P(\text{score} = 1 \mid s, a)$ from 10^7 random offline shots and add it as a dense SAC bonus; evaluation still uses the true binary carom score. The bonus raises train mean to 0.840, but the best inning remains 4 and random-start eval is only 0.087. More importantly, the gain is expensive and does not compound with real geometry: RM plus aim constraint reaches 2.357 ± 0.139, below the constraint alone at 2.570. We therefore treat RM guidance as another negative result and turn to explicit domain knowledge.

4 Adding domain knowledge

Sections 2–3 establish that neither more budget, a curriculum, nor a learned reward model breaks the plain-RL ceiling. We now add *explicit* domain knowledge—a geometric action constraint and a few hand-chosen observation features—to the same SAC baseline (1M steps, 3 seeds, random-start + continue-on-miss). Table 4 reports the cumulative effect.

Table 4: Domain knowledge on the SAC baseline (1M steps, 3 seeds, training-protocol eval). *foul/sh* and *succ/sh* are per-shot foul and scoring probabilities. **Take-away: the aim constraint and the extra geometric features each lift the entire score distribution—together they take the agent from 0.49 to 6.46 points/inning.**

variant	mean±std	max	foul/sh (%)	succ/sh (%)
plain SAC	0.487±0.097	4	20.5	4.9
+ aim constraint	2.570±0.128	6	15.4	25.7
+ **extra feat.**	**6.460±0.123**	10	11.1	64.6

4.1 Aim constraint

The cue's aim θ is the one action dimension where most of the range is wasted: from a random layout the vast majority of directions send the cue ball into empty space, never contacting a red. We therefore reparameterise θ as an *offset* within the angular window $\mathrm{target} \pm \arcsin(2r/d)$ that geometrically *guarantees* first contact with the nearest red ball (radius r, distance d); the policy now chooses *where within* the guaranteed-contact cone to aim rather than whether to make contact at all. This single change is decisive: mean rises $0.487 \to 2.570$ ($5.3\times$), per-shot scoring rises from 4.9%

to 25.7%, and—because most shots now at least touch a red instead of fouling—the per-shot foul rate *drops* ($20.5\% \to 15.4\%$) even though no penalty was added.

4.2 Extra geometric features

On top of the constraint we append four hand-chosen features to the observation, all measured from the red ball nearest the cue: the distances d_1, d_2 from the cue to the two red balls, and $\sin\varphi, \cos\varphi$ of the angle φ between the cue-to-nearest-red direction and the nearest-to-other-red direction. These encode the carom geometry the policy would otherwise have to infer from raw coordinates. The effect is the largest single lever in the paper (Table 4, three-way plain $\to$ constraint $\to$ constraint+extra): mean $2.570 \to 6.460$ ($2.5\times$), per-shot scoring $25.7\% \to 64.6\%$, and the best inning hits the 10-shot cap. Making the relevant geometry explicit matters far more than any change to the reward.

4.3 Reward shaping: foul penalty is the only useful knob

On the strong *constraint+extra* base we first add a fixed foul penalty of -1 (Table 5). It lowers the mean ($6.460 \to 5.553$) but nearly *halves* the per-shot foul rate ($11.1\% \to 6.0\%$), making it a genuine score-vs-safety knob. We then test two dense shaping terms on this foul-penalty base: a *gentle-shot* bonus for low-power scoring shots and a *near-miss* bonus for nearly completing the carom. These do not improve score in a meaningful way: near-miss and gentle+near-miss are essentially tied with the foul-penalty base, while gentle alone lowers fouls further at a scoring cost. Reward shaping therefore does not beat structure; at most, the foul penalty buys safety when fouls are costly.

Table 5: Reward shaping on the constraint+extra base (1M steps, 3 seeds). Dense shaping rows are trained on the foul-penalty base. **Take-away: foul penalty trades ~ 0.9 points/inning for a $\sim 45\%$ foul-rate cut; gentle and near-miss do not provide a reliable scoring gain.**

variant	mean±std	foul/sh (%)	succ/sh (%)
constraint+extra	**6.460±0.123**	11.1	64.6
+ foul penalty (base)	5.553±0.162	6.0	55.5
+ foul penalty + gentle	5.213±0.442	**3.3**	52.1
+ foul penalty + near-miss	5.583±0.370	5.0	55.8
+ foul penalty + both	5.520±0.365	5.0	55.2

5 Inference-time lookahead search

The trained policy outputs a Gaussian over actions, but because our simulator runs a shot in ≈ 5 ms we need not commit to the policy mean: we can sample several candidate actions, simulate each, and keep the one with the best simulated outcome. This is the AlphaZero recipe—learned prior plus inference-time search—specialised to a continuous action space with a known, fast forward model. We use it to chase "infinite billiards": starting from a random rack and terminating on the *first* miss or foul, how long a scoring chain can one inning sustain? The metric is the chain length (scoring shots before the first failure); we cap innings at $10{,}000$ shots, a limit that is rarely reached.

Fixed-engine proposer comparison. To align the search evaluation with the reward-shaping ablation, we use the four 1M proposer ensembles from Table 5: foul-penalty base, gentle, near-miss, and both. The verifier environment is identical for all proposers (`constrain_aim` + `extra_features` + `foul_penalty`$= 1.0$); gentle and near-miss affect only how the proposer policies were trained. We compare greedy depth-2 lookahead against PUCT at the same 400-simulation per-shot budget. We report both a bounded random-start throughput protocol (P1: 100 innings, 100 shots each) and a long-chain protocol (P3: canonical start, terminate on first miss or foul).

Search. Our search is a greedy depth-2 K-expansion: at state s_t the three policies emit K_1 candidates jointly, we simulate all from a snapshot and keep the top M_1 by immediate reward, expand each with K_2 children, simulate M_2, and execute $\arg\max_{a_1}(r_1 + \gamma \max r_2)$ with γ=0.99. Because the environment is deterministic a candidate is fully described by its action—a second visit reproduces an identical trajectory—so there is no value in the visit-count revisiting or tree reuse that PUCT [5]

relies on; freshly sampling K candidates at every shot uses the simulator budget far more effectively. The only knob that matters is therefore how many candidates we can afford per shot.

Results: h=2 dominates PUCT, proposer differences are noisy. Lookahead transforms the agent. In the bounded P1 protocol, h=2 nearly saturates the 100-shot horizon for every proposer (99.7–99.8 points), while PUCT remains far lower (62.8–81.9). In the stricter P3 chain protocol, h=2 produces thousand-shot chains and PUCT remains below 100 shots on average (Table 6). The longest observed chain is **9,392 shots** from the near-miss proposer, but its variance is enormous; we therefore do not claim near-miss is the best proposer. The robust result is method-level: with the same simulator budget, shallow verified lookahead is far stronger than PUCT, and reward-shaping variants do not change that conclusion.

Table 6: Fixed-engine proposer comparison at 400 simulations/shot. P1 is random-start throughput (100 innings $\times$ 100 shots); P3 is canonical-start chain length before the first miss/foul. **Take-away: h=2 dominates PUCT across proposers and reaches a 9,392-shot chain; proposer differences are high variance.**

proposer	P1 h=2	P1 PUCT	P3 h=2 mean (max)	P3 PUCT mean (max)
base	99.80	67.93	1383 (2634)	85.7 (404)
gentle	99.71	79.11	955 (1750)	32.4 (157)
near-miss	**99.83**	**81.85**	**2052 (9392)**	77.4 (345)
both	99.77	62.84	1194 (3272)	50.4 (191)

6 Conclusion: a bitter lesson

Korean 4-ball is an unusual RL target. Unlike the discrete games where search was born or the dense-reward, model-free benchmarks where learning shines alone, it is *continuous, deterministic, and sparsely rewarded*, and it comes with a cheap, exact forward model. We set out to solve it with reinforcement learning, and the arc of our attempts reads as a small re-enactment of Sutton's bitter lesson.

The hand-built, knowledge-injecting ideas disappointed. Plain RL plateaued below 0.5 points/inning regardless of budget (Section 3); a staged action curriculum did *worse* than no curriculum; and a learned reward model —precisely the kind of human-designed scaffolding the bitter lesson warns against—cost an extra 10^7 simulated shots to build yet added little and vanished entirely once a real lever was present. Even the explicit domain knowledge that *did* help (the aim constraint and geometric features, which lifted the agent from 0.5 to 6.5 points/inning) behaves like classic hand-engineering: a large but *bounded* one-time gain (Section 4).

What actually broke the ceiling were the two general methods that scale with computation. Learning gave us a competent prior; *search* did the rest. A simple greedy depth-2 lookahead—no learned value function, no tree, just the fast simulator used as its own verifier—turned a $\sim$6-point policy into chains of thousands of consecutive scoring shots, up to $9{,}392$ shots, while PUCT remained far shorter under the same per-shot budget (Section 5). The lesson we draw is domain-specific but pointed: when an exact, fast forward model is available, the highest-leverage move is not a cleverer reward but spending compute at inference time. We tried to make 4-ball an RL problem; in the end, search heuristics solved it.

Contribution

Doyeol Oh: Proposed the idea; built the simulator, environments, and training/evaluation pipeline; implemented the lookahead search; gave the talk.

Byungmo Kang: Proposed the key ideas — aim constraint, geometric features, reward shaping, and the inference-time search direction.

Seojun Park: Implemented the near-miss reward-shaping variant; curated and validated experimental results; organized ablation tables and narrative; wrote the initial report draft.

All three ran the experiments together and refined the final report jointly.

References

[1] Marlow, W.C. (1995). *The Physics of Pocket Billiards*. MAST Publications.

[2] Han, I. (2005). Dynamics in carom and three cushion billiards. *J. Mechanical Science and Technology*, 19(4):976–984.

[3] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S. (2018). Soft actor-critic. *ICML*.

[4] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal policy optimization. arXiv:1707.06347.

[5] Silver, D., Hubert, T., Schrittwieser, J., et al. (2017). Mastering chess and shogi by self-play with a general RL algorithm. arXiv:1712.01815.

[6] Kiefl, E. (2024). Pooltool. *JOSS*, 9(101):7301.

[7] Alciatore, D.G. (2004). *The Illustrated Principles of Pool and Billiards*. Sterling Publishing.