Optimizing Rational and Aesthetic Navigation Objectives via Stochastic Reward Shaping in Procedural 3D Unity Environments

Victor Tenneroni*

1 Introduction

1.1 Problem Statement

This project addresses autonomous navigation in procedurally generated parkour environments where agents must balance speed, stamina, and aesthetic movement (dynamic rolls). We built both the Unity environment and RL agent simultaneously, creating a moving target where environment changes during development broke previously trained agents—compounded by procedural platform generation that presents infinite layout variations. Two fundamental questions arise: (1) How do we train AI to understand subjective style preferences? (2) How do we integrate human preferences when reaction time is orders of magnitude slower than agent training time? RL is necessary due to high-dimensional state/action spaces (14 observations, 5 actions), infinite environment variations through procedural generation, and strategic tradeoffs with no closed-form solution. Traditional rule-based and PID approaches fail under these randomized conditions.

1.2 Approach

As a first step toward RLHF integration, we build a procedurally generated parkour environment and explore episodic stochastic reward modulation as a baseline for future human preference learning.

At episode initialization, a Bernoulli trial ($p = 0.4$) determines whether style bonuses are active for that entire episode. When active, roll actions receive $+1.5$ bonus atop the base $+0.5$ reward (total $+2.0$); when inactive, rolls receive only base reward ($+0.5$).

This episode-level stochasticity allows the agent to learn roll execution without requiring rolls in every situation, avoiding degenerate policies that sacrifice task performance for style points. While this approach lacks genuine human preference signal, it establishes the training infrastructure and demonstrates that reward modulation can successfully encourage stylistic behaviors. Future work can replace the stochastic bonuses with learned reward models trained on human comparisons (Section 5.1).

1.3 Empirical Validation

Across 30 total training runs (2M steps each), we observe:

- Baseline (15% style frequency): 0.69% roll usage, $+67.90$ final reward

- Stochastic reward shaping (40% style frequency): 10.21% roll usage, $+78.52$ final reward

---

*Code and implementation available at https://github.com/vtennero/mirrorsedgepolicygradient/tree/main

- Roll usage increased 14.8× (0.69% to 10.21%) compared to baseline

- Final performance: +78.52 average reward, 360.15 units mean distance traveled (range 26.09–582.92 units)

2 Background & Related Work

2.1 Reinforcement Learning from Human Feedback

RLHF [1] learns reward functions from human preferences over trajectory pairs, enabling agents to optimize complex objectives that are difficult to hand-specify. The method maintains a policy π and reward estimate $\hat{r}$, updated through three processes: (1) policy optimization via standard RL, (2) human preference elicitation on trajectory pairs, and (3) reward function fitting to match human comparisons.

RLHF requires real-time human feedback during training, which becomes infeasible when training runs at 20× time acceleration (our setup generates ~1,054 steps/second across 28 parallel agents). This fundamental incompatibility motivates our approach: stochastic reward shaping as an offline approximation of preference variance. Our approach extends traditional reward shaping [3], which modifies rewards while preserving optimal policies, by introducing episodic stochasticity where reward structure varies probabilistically across episodes.

3 Methodology

The problem is formalized as an MDP with 14-dimensional state space and 5 discrete actions (details in Appendix A.1).

3.1 Reward Design

3.1.1 Base Rewards

Table 1 shows the base reward components combining dense per-step rewards (progress, grounded state, time penalty, stamina penalty) with sparse terminal rewards (target reach, fall penalty).

Reward Component	Value	Condition
Dense (Per-Step):		
Progress Reward	$+0.1 \times \Delta x$	Forward movement
Grounded Reward	$+0.001$	Agent grounded
Time Penalty	-0.001	Per update
Low Stamina Penalty	-0.002	Stamina < 20%
Sparse (Episode-Level):		
Target Reach	$+10.0$	Distance < 2.0 units
Fall Penalty	-1.0	Fall/timeout

Table 1: Base reward components (dense and sparse)

3.1.2 Iterative Reward Calibration

The reward structure evolved through empirical observation of emergent behaviors:

Problem 1 - Sprint Bashing: Agent depleted stamina completely by holding sprint 38% of the time. Fix: Added low stamina penalty (-0.002/step when $< 20\%$) and reduced sprint cost.

Problem 2 - Roll Ignored: Roll usage remained at 0.69% despite being fastest action. Fix: Reduced roll cost from 150$\to$60 stamina and added base roll reward ($+0.5$).

Problem 3 - Insufficient Incentive: Even with lower cost, rolls rarely used. Final solution: Dual reward structure (base $+0.5$ always, style bonus $+1.5$ in 40% of episodes).

Result: 16% reward improvement ($+67.90\to+78.52$), strategic roll usage (10.21%).

3.1.3 Style Reward Approximation: Design Process

Stochastic Reward Shaping:

The style reward system uses **stochastic reward injection** instead of real-time human feedback. This design addresses the fundamental constraint that human feedback is incompatible with accelerated training. The final dual reward structure (base + style bonus) emerged from the iterative calibration process described above.

Roll Reward Structure:

At episode initialization, a Bernoulli trial ($p = 0.4$) determines whether style bonuses are active for that entire episode. When active, roll actions receive $+1.5$ bonus atop the base $+0.5$ reward (total $+2.0$). When inactive, rolls receive only base reward ($+0.5$).

The 40% frequency is exploratory, selected after observing 15% frequency produced insufficient roll adoption (0.69% of actions). Higher frequencies risk overwhelming base objectives (speed, energy efficiency). We did not test whether the agent actually rolls more frequently in style episodes versus non-style episodes; this analysis is left to future work.

| Reward Component	Value	Condition	Design Rationale
Roll Base Reward	$+0.5$	Roll action executed	Ensures rolls are always valuable. Prevents agent from ignoring rolls i
Roll Style Bonus	$+1.5$	Roll in style episode	Provides additional incentive in 40% of episodes. Creates behavioral v

Table 2: Roll reward structure

Total Roll Reward:

- **In style episodes (40%):** $+0.5$ base $+1.5$ style $= +2.0$ per roll ($20\times$ progress per unit)

- **In non-style episodes (60%):** $+0.5$ base per roll ($5\times$ progress per unit)

Episode-Level Style Flag:

- **Probability:** 40% (`styleEpisodeFrequency = 0.4`)

- **Assignment:** Randomly determined at episode start

- **Scope:** Affects all roll actions within that episode

- **Rationale:** This episode-level stochasticity allows the agent to learn roll execution without requiring rolls in every situation, avoiding degenerate policies that sacrifice task performance for style points.
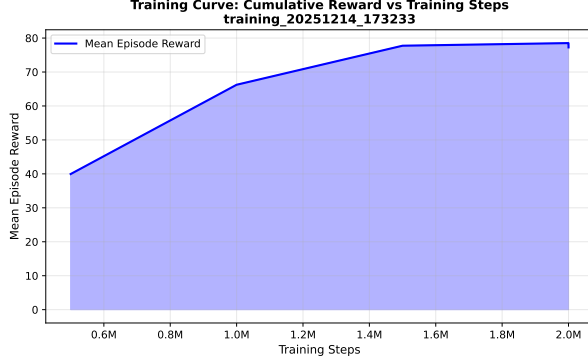
(a) Training curve
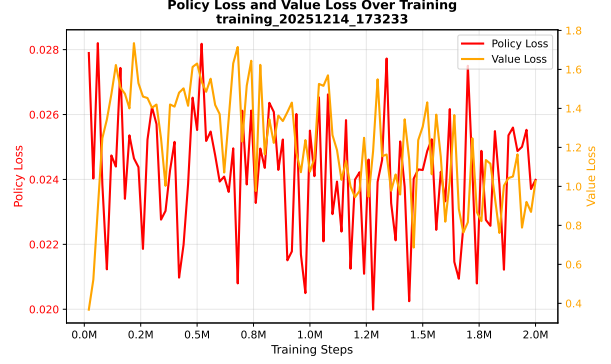
(b) Loss curves

(c) Entropy

Figure 1: Training dynamics: (a) cumulative reward progression, (b) policy and value loss, (c) policy entropy over training

4   Results & Analysis

4.1   Training Performance

Final Performance Metrics (2M steps): Reward: +78.52 (0.3% improvement over previous best, 16% over roll system v1). Training progressed monotonically: +39.95 at 500k → +66.25 at 1M → +77.75 at 1.5M → +78.52 at 2M steps (97% improvement from 500k). Policy Loss: 0.0240 (stable), Value Loss: 1.027, Entropy: 0.600 (high exploration maintained). Hyperparameter decay: Learning rate 3.0×10^{-4} (linear decay), Beta 0.1 (linear decay), Epsilon 0.2 (linear decay).

4.2   Episode Statistics

Mean episode length 43.91 seconds (range 4.54–66.46), mean distance 360.15 units (range 26.09–582.92). All 100 logged episodes (sampled every 20,000 steps) reached success. Low-distance episodes (< 50 units) indicate early exploration; high-distance episodes (> 500 units) indicate successful target reach with efficient action usage.
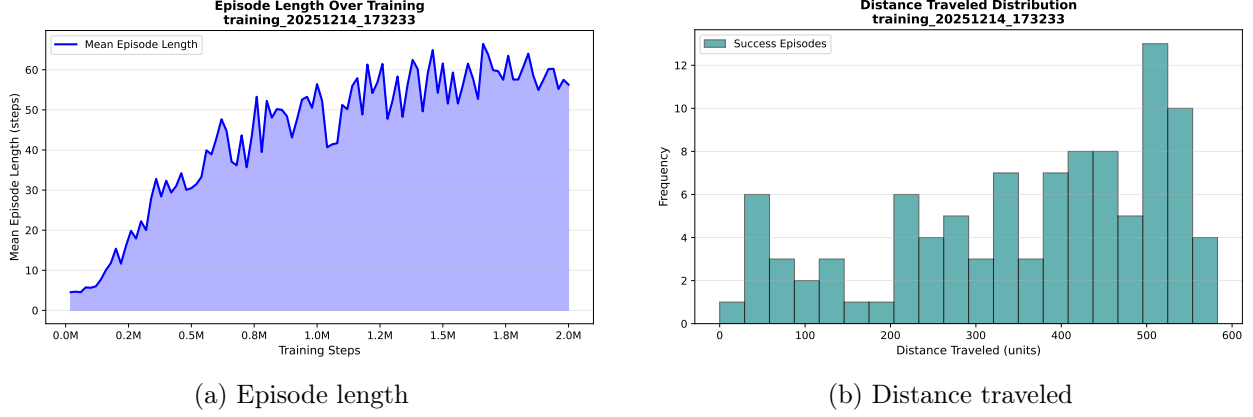
4.3   Action Distribution and Behavior

Agent Behavior Analysis:

(a) Episode length

(b) Distance traveled

Figure 2: Episode statistics: (a) episode length distribution, (b) distance traveled distribution

Action	Percentage
Jog	66.97%
Sprint	13.59%
Roll	10.21%
Jump	4.02%
Idle	5.21%

Table 3: Action distribution statistics (final step: 2M)

- **Roll Usage:** 10.21% of actions (vs 0.69% in previous run with 15% style frequency)

- **Roll Improvement:** 14.8× increase over previous run (0.69% → 10.21%)

- **Strategic Roll Usage:** Rolls used at 10.21% despite high cost (60 stamina), indicating learned strategic value

4.4   Behavioral Emergence: Style Bonus Impact

The stochastic reward shaping (40% style frequency) increased roll usage from 0.69% to 10.21% (14.8× increase) and final reward from +67.90 to +78.52 (16% improvement) compared to baseline
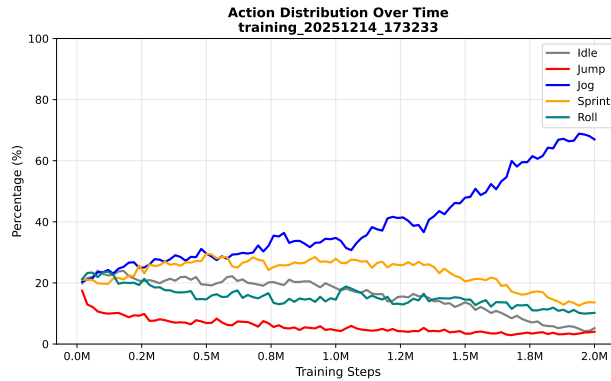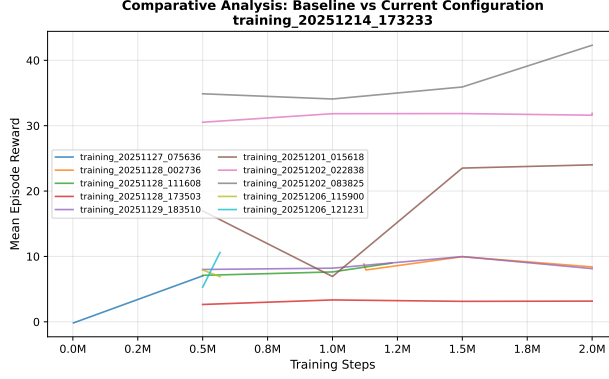

Figure 3: Action distribution over time

Figure 4: Comparative analysis: Baseline vs current configuration

configurations. Figure 4 shows 11 baseline runs exploring learning rates (3×10^{-4} to 5×10^{-4}), beta values (0.05 to 0.2), style frequencies (0%, 10%, 15%), and roll costs (60–150 stamina); all underperformed the final 40% dual-reward configuration.

Key Behavioral Changes:

1. **Roll Integration:** Agent learned to use rolls strategically (10.21% usage) despite high stamina cost (60 per roll)

2. **Stamina Management:** Agent balances sprint (13.59%) and roll (10.21%) usage, maintaining stamina for critical actions

3. **Movement Diversity:** Primary movement is jog (66.97%), with strategic use of sprint and roll for speed and style

Limitations: While roll usage increased 14.8×, we lack human validation that the learned roll timing is aesthetically pleasing. The agent may execute rolls at mechanically optimal but visually awkward moments. Future work should compare roll distribution between style (40%) and non-style (60%) episodes to validate the episodic shaping mechanism.

4.5 Training Dynamics

Convergence Analysis:

- **Reward Curve:** Monotonically increasing from 500k to 2M steps, no catastrophic forgetting

- **Policy Convergence:** Policy loss stabilized at 0.0240, indicating converged policy

- **Value Estimation:** Value loss at 1.027 reflects reasonable estimation error

- **Exploration:** Policy entropy maintained at 0.600, indicating continued exploration even at convergence

Learning Rate Decay: The linear decay schedule successfully shifted from exploration to exploitation:

- Initial learning rate: 3.0×10^{-4} (linear decay)

- Beta: 0.1 (linear decay)

- Epsilon: 0.2 (linear decay)

Style Bonus Impact on Learning: The episodic style bonus (40% frequency) created behavioral variety without destabilizing learning:

- Consistent reward structure within episodes (style flag assigned at episode start)

5   Discussion & Future Work

5.1   RLHF Integration

Future work should integrate actual human feedback. Four viable approaches:

1. **Asynchronous Preference Collection:** Decouple training from human feedback by collecting preferences between training runs. Run standard training (28 agents, 30 minutes, 2M steps) with current reward model; automatically sample trajectory pairs from replay buffer and save as video clips; human evaluates comparison pairs between training runs; train reward model on accumulated preferences; iterate with updated reward model.

2. **Synchronous Feedback with Checkpointing:** Hybrid training with alternating slow (observable) and fast (accelerated) phases. Train at normal speed ($1\times$ time scale) with 4 agents for 5–10 minutes with human real-time keyboard feedback; switch to accelerated mode ($10\times$ time scale) with 28 agents for 20 minutes using current reward model; train reward model on accumulated preferences; repeat cycle.

3. **Pre-train Reward Model, Then RL:** One-time offline preference collection before RL training begins. Generate trajectories from random or hand-crafted policies, collect 200–500 human preference pairs (2–3 hours, one-time cost); train initial reward model using Bradley-Terry model on collected preferences; use learned reward model for standard RL training (current 30-minute setup).

4. **Minimal Viable RLHF:** Post-training clip rating with simple regression model. Run standard training (30 minutes, 2M steps); Unity automatically saves 10 "style moment" clips (rolls, jumps, acrobatic sequences); human rates each clip 1–5 stars; fit linear regression model predicting star rating from state features (height, velocity, rotation); use predicted rating as style reward in subsequent training.

We recommend approach 3 (pre-trained reward model) as it requires minimal infrastructure changes while providing genuine human preference signal.

5.2   Training Optimization

- **Hyperparameter optimization:** Replace linear decay schedules with exponential decay for beta ($0.05 \rightarrow 0.001$), learning rate ($5 \times 10^{-4} \rightarrow 1 \times 10^{-5}$), and epsilon ($0.15 \rightarrow 0.05$), increase GAE lambda to 0.98, and reduce training epochs to 3. We hypothesize this could improve final reward to $+95$–100 (vs. current $+78.52$), achieve faster convergence, and reduce final entropy to ~ 0.2–0.3 (vs. current 0.600).

- **Movement smoothing:** Increase sprint speed from $12 \rightarrow 14$ units/sec when maintained, add -0.005 penalty per sprint interruption, and reward consistent movement direction. Expected to eliminate sprint stuttering behavior observed in current runs.

5.3 Environment and Action Space Extensions

- **Full 3D movement:** Extend from 2.5D to full 3D navigation with multi-axis platforms, turning mechanics, and 3D spatial orientation.

- **Expanded action space:** Add actions: Slide, wall jump, vault (expanding from $5 \to 9+$ discrete actions), or implement continuous control for movement direction and intensity.

- **Dynamic obstacles:** Add moving platforms (translation/rotation), time-dependent physics, and partial observability (occluded obstacles).

5.4 Workflow and Algorithmic Improvements

- **LLM-assisted hyperparameter tuning:** Use LLM-based reasoning to analyze training curves and adapt hyperparameters, reducing manual iteration time.

- **Q-learning and DQN benchmark:** Implement Q-learning and DQN algorithms on the same environment to benchmark against PPO, providing empirical evidence for PPO's advantages (continuous state space handling, stochastic policy for style action discovery, sparse reward learning) over value-based methods.

References

[1] Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[3] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287, 1999.

[4] Unity Technologies. ML-Agents Toolkit. https://github.com/Unity-Technologies/ml-agents, 2024.

A Appendix

A.1 State and Action Space Details

A.1.1 State Space Design Philosophy

The state space balances information sufficiency with dimensionality: relative positions and raycasts enable generalization across randomized layouts, while absolute positions and action history are excluded as they hinder generalization. See Table 6 for complete specification.

Component	Size	Description	Range/Norm
Tgt Rel Pos	3	$(target.pos - agent.pos)$	Raw 3D (units)
Velocity	3	$controller.velocity$	Raw 3D (units/sec)
Grounded	1	1.0 if grounded, 0.0 if not	Binary
Platform Rays	5	Downward rays at [2,4,6,8,10] units ahead	Norm (0–1)
Obstacle Dist	1	Forward obstacle raycast distance	Norm (0–1)
Stamina	1	$currentStamina/maxStamina$	Norm (0–1)

Table 4: State space components (14 floats total). Full descriptions: Target Relative Position—3D vector from agent to target. Velocity—3D velocity vector. Grounded—binary indicator. Platform Raycasts—5 downward rays at forward distances [2,4,6,8,10] units. Obstacle Distance—forward obstacle detection. Stamina—normalized current/max stamina ratio.

A.1.2 State Space (Observations)

Total Observations: 14 floats
The state space is fully observable and consists of the following components:
State Space Properties:

- **Dimensionality:** 14 ($S \subseteq \mathbb{R}^{14}$)

- **Observability:** Fully observable (no hidden information)

- **Normalization:** Applied where applicable (raycasts, stamina)

- **Completeness:** Contains all information needed for parkour decisions

A.1.3 Platform Detection Raycasts

Platform raycasts are essential for generalization: 5 downward rays at [2,4,6,8,10] units ahead detect gaps dynamically. Without raycasts (test_v9), reward dropped 60% (+3.43 vs +9.85) as the agent failed to adapt to randomized gap spacing. Raycasts enable perception-based adaptation rather than pattern memorization.

A.1.4 Action Space Design

Type: Discrete, single branch, 5 actions

Action	ID	Description	Speed	Cost	Constraints[1]
Idle	0	No movement	0	0	A
Jump	1	Vertical jump + forward	Instant	20	G, S(20)
Jog	2	Forward movement	6	0	A
Sprint	3	Forward movement	12	20/sec	S(> 0), C(0.5s)
Roll	4	Forward roll	18	60	S(60), D(0.6s)

Table 5: Action space (5 discrete actions)

Action Space Properties:

- **Type:** Discrete ($A = \{0, 1, 2, 3, 4\}$)

- **Branch Count:** 1 (single decision branch)

- **Action Count:** 5

- **Constraints:** Enforced by environment (stamina, cooldown, grounded state)

Action Timing and Constraints:

- **Sprint Cooldown:** 0.5 seconds after sprint ends before sprint can be used again

- **Roll Duration:** 0.6 seconds (roll is a timed action, cannot chain rolls)

- **Stamina System:** Max stamina 100.0, regeneration 30.0/sec when not sprinting/jumping/rolling

Risk/Reward Trade-off: Roll Action Roll is the fastest action (18 units/sec, $1.5\times$ sprint speed) but carries the highest stamina cost (60 per roll, $3\times$ jump cost). This creates a strategic decision: the agent must balance speed gains against stamina depletion. The high cost prevents indiscriminate roll usage while the speed advantage rewards strategic timing (e.g., crossing gaps efficiently). This risk/reward structure naturally emerges from the action design instead of being explicitly encoded in rewards.

A.2   Implementation Details

A.2.1   Unity ML-Agents Setup

Environment built using Unity 2022.3 LTS with ML-Agents Toolkit 3.0.0+ [4]. Training uses 28 parallel agents via gRPC communication (port 5004), with CharacterController for physics and CharacterConfig ScriptableObject for parameter management.

A.2.2   Training Hyperparameters

PPO Configuration The training uses Proximal Policy Optimization (PPO) [2] with the following hyperparameters defined in `parkour_config.yaml`:

  Hyperparameters:

- **Learning Rate:** 3.0×10^{-4} (linear decay schedule)

- **Batch Size:** 1024 experiences per training batch

- **Buffer Size:** 10240 ($10\times$ batch size for experience replay)

- **Beta (Entropy):** 0.1 (linear decay) — High exploration coefficient

- **Epsilon (Clipping):** 0.2 (linear decay) — PPO clipping parameter

- **Lambda (GAE):** 0.95 — Generalized Advantage Estimation lambda

- **Gamma (Discount):** 0.99 — Discount factor for future rewards

- **Num Epochs:** 5 — Training epochs per batch

- **Time Horizon:** 128 steps before value bootstrapping

Network Architecture:

- **Actor Network:** 2 hidden layers × 256 units → 5 action logits, input normalization enabled

- **Critic Network:** 2 hidden layers × 128 units → 1 value estimate, separate from actor (not shared)

- **Activation:** ReLU (default ML-Agents)

- **Initialization:** Xavier/Glorot uniform (ML-Agents default)

Hyperparameter Selection Rationale High Beta (0.1): Increased from default 0.015 to encourage exploration in the complex parkour environment. The linear decay schedule allows gradual shift from exploration to exploitation.
Selection Process:

- **Initial Value:** 0.015 (ML-Agents default)

- **Problem:** Agent converged too quickly, missed optimal strategies

- **Experimentation:** Tested 0.05, 0.1, 0.2

- **Result:** 0.1 provided best balance (high exploration, still learns effectively)

- **Decay:** Linear from $0.1 \rightarrow \sim 0.00074$ over 2M steps

Time Horizon 128: Balanced between shorter horizons (64) that may miss long-term dependencies and longer horizons (192) that slow training. Appropriate for 100-second episodes.

A.3   Reward Breakdown Analysis

Target positioned at $lastPlatformEndX + 5.0$ units (beyond final platform). Success condition: $|agent.x - target.x| < 2.0$ units (X-axis only). For a typical successful episode ($\sim$500 units progress): progress provides $\sim$50.0 reward, target reach +10.0, with remaining from grounded/stamina/roll bonuses contributing to final reward of +78.52.