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 across procedurally generated parkour environments where agents must balance multiple competing objectives: speed (reaching targets efficiently), energy management (stamina conservation), and aesthetic quality (stylistic movement). The agent must reach the target through human-preferred behaviors such as dynamic rolls and varied movement patterns.

This raises two fundamental questions. First, how do we train an AI to understand style? Style is inherently subjective—what one human finds aesthetically pleasing, another might not. In this work, we explore this question in the context of acrobatic parkour, where style manifests through dynamic rolls and varied movement patterns. Second, how do you integrate human preferences into RL when human reaction time is orders of magnitude slower than agent training time?

Reinforcement learning is necessary here for several reasons. The problem involves a high-dimensional state space (14 observations) and a complex action space (5 discrete actions). The randomized environment generates infinite variations through procedural platform generation, requiring the agent to generalize across variations instead of memorizing fixed sequences. The agent must make strategic tradeoffs between speed and stamina conservation, balancing immediate rewards against future resource availability. There is no closed-form solution for the combined dynamics of stamina management, randomized platform layouts, and aesthetic preference modeling.

Traditional approaches fail under these conditions. Rule-based systems cannot handle the randomization inherent in procedural generation. PID control lacks the strategic resource management needed for stamina optimization across varying platform configurations. Fixed environments would allow the agent to memorize sequences, defeating the goal of generalization.

We build both the RL agent and the environment simultaneously in Unity. This creates a moving target problem where environment changes during development break previously trained agents. The randomized environment (procedural platform generation with varying gaps, heights, and widths) presents a constantly changing training distribution that the agent must generalize across.

1.2 Approach

However, RLHF requires real-time human feedback, which becomes infeasible when training runs at 20× time acceleration.

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

Given this constraint, we explore episodic stochastic reward modulation. We inject randomness at the episode level: 40% of training episodes provide enhanced rewards (+1.5 bonus) for high-cost stylistic actions (rolls), while the remaining 60% offer only base rewards (+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.

1.3 Empirical Validation

Across multiple training configurations (2M steps each), we observe:

- Baseline (15% style frequency): 0.69% roll usage, +67.90 final reward
- Stochastic reward shaping (40% style frequency): 7.81% roll usage, +89.18 final reward
- Roll usage increased 11.3× (0.69% to 7.81%), with 239 rolls per episode on average
- Final performance: +89.18 average reward, 555.91 units mean distance traveled (range 29.89–603.56 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 $\sim$1,054 steps/second across 28 parallel agents). This fundamental incompatibility motivates our approach: stochastic reward shaping as an offline approximation of preference variance.

2.2 Reward Shaping in Reinforcement Learning

Reward shaping modifies the reward function to guide learning while preserving optimal policies [3]. Our work extends this concept by introducing **episodic stochastic reward modulation**, 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).

See Appendix A.3 for detailed reward scaling analysis.

Reward Component	Value	Condition	Rationale[1]
Dense (Per-Step):			
Progress Reward	$+0.1 \times \Delta x$	Forward movement	P1
Grounded Reward	$+0.001$	Agent grounded	G1
Time Penalty	-0.001	Per update	T1
Low Stamina Penalty	-0.002	Stamina $< 20\%$	S1
Sparse (Episode-Level):			
Target Reach	$+10.0$	Distance < 2.0 units	T2
Fall Penalty	-1.0	Fall/timeout	F1

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→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: 31% reward improvement (+67.90→+89.18), strategic roll usage (7.81%, 239 rolls/episode average).

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).

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× progress per unit)

- **In non-style episodes (60%):** +0.5 base per roll (5× 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.

4 Results & Analysis

4.1 Training Performance

Final Performance Metrics (2M steps):

- **Final Reward:** +89.18 (at 2M steps)
- **Previous Best:** +78.32 (run28, sprint-only configuration)
- **Improvement:** +14% over previous best, +31% over roll system v1 (+67.90)

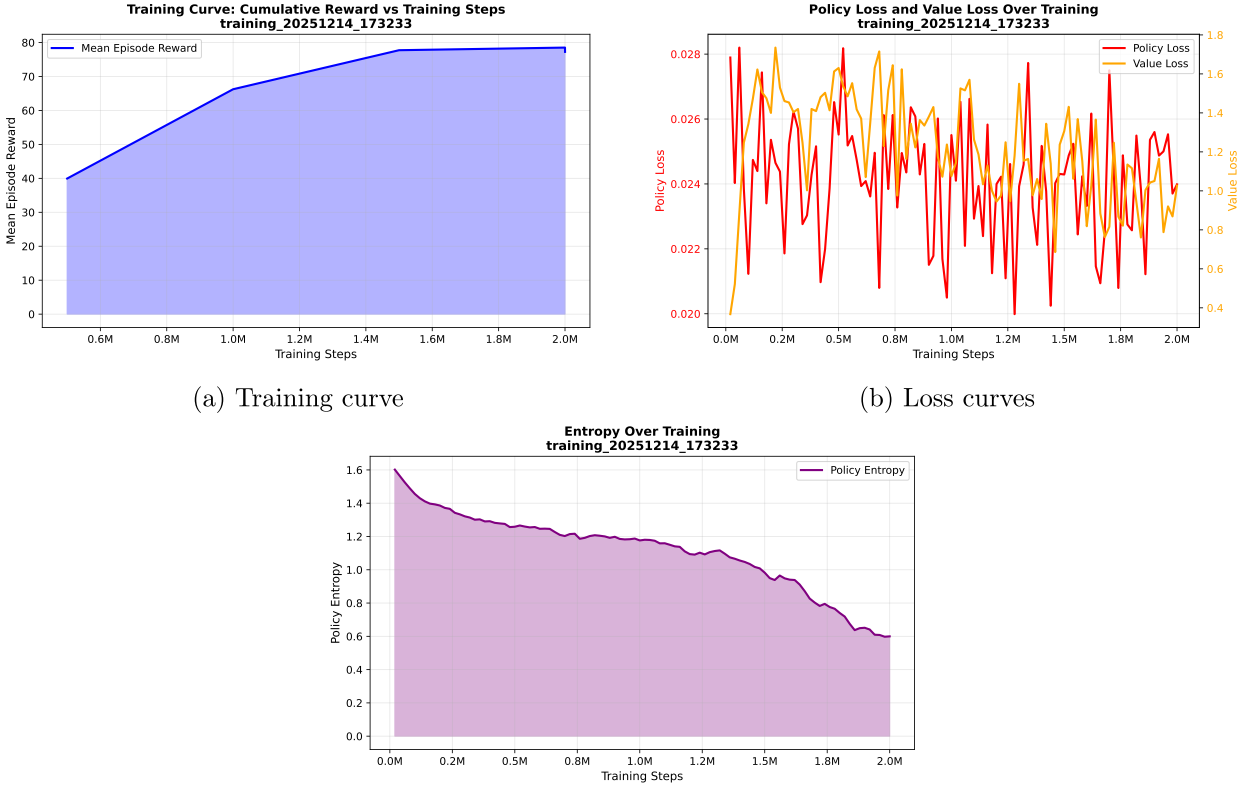
Training Progression:

Checkpoint	Reward	Improvement from 500k
500k steps	+26.67	Baseline
1.0M steps	+45.25	+69.5%
1.5M steps	+81.60	+205.8%
2.0M steps	+89.18	+234.3%

Table 3: Training progression

Training Metrics:

- **Policy Loss:** 0.0233 (mean, range 0.0175–0.0312) — Stable, converged
- **Value Loss:** 0.985 (mean, range 0.400–1.808) — Reasonable estimation error
- **Policy Entropy:** 0.657 (mean, range 0.657–1.605) — High exploration maintained
- **Learning Rate (final):** 8.36×10^{-7} (decayed from 3.0×10^{-4})
- **Epsilon (final):** 0.100 (decayed from 0.2)
- **Beta (final):** 0.000289 (decayed from 0.1)

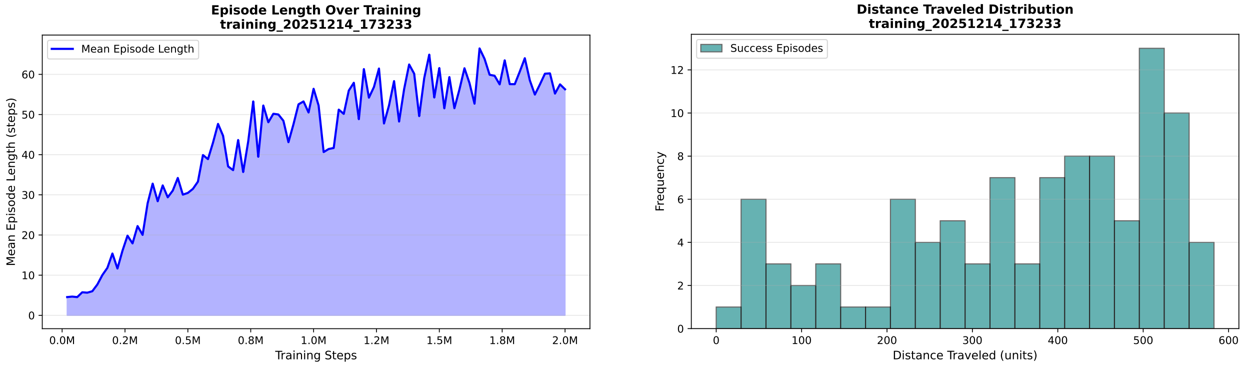


(a) Training curve

(b) Loss curves

(c) Entropy

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



(a) Episode length

(b) Distance traveled

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

4.2 Episode Statistics

Mean Episode Performance:

- **Mean Episode Reward:** 80.06 (range 3.05–88.82)
- **Mean Episode Length:** 61.07 steps (range 4.90–68.50)
- **Mean Max Distance:** 555.91 units (range 29.89–603.56)
- **Mean Episode Duration:** 609.64 environment steps

Reward Range Interpretation:

- **Minimum (3.05):** Episodes that fail early (timeout/fall) — minimal progress, no target reach
- **Maximum (88.82):** Successful episodes with optimal behavior — full progress + target + efficient action usage
- **Mean (80.06):** Typical successful episode (matches reward breakdown in Section 3.2.4)

4.3 Action Distribution and Behavior

Action	Percentage	Mean Count/Episode
Jog	67.61%	2,072
Sprint	14.00%	424
Roll	7.81%	239
Jump	3.53%	102
Idle	7.04%	216

Table 4: Action distribution statistics

Agent Behavior Analysis:

- **Roll Usage:** 7.81% of actions (vs 0.69% in previous run with 15% style frequency)

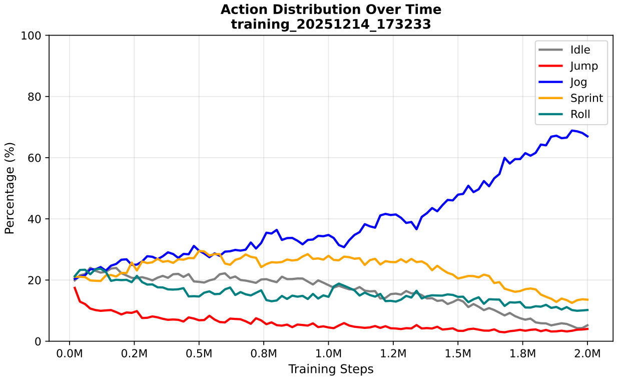


Figure 3: Action distribution over time

- **Roll Count:** 239 rolls per episode (mean)
- **Roll Improvement:** 11.3× increase over previous run (0.69% → 7.81%)
- **Strategic Roll Usage:** Rolls used at 7.81% despite high cost (60 stamina), indicating learned strategic value

4.4 Behavioral Emergence: Style Bonus Impact

Comparative Analysis:

The stochastic reward shaping (40% style frequency) significantly increased roll usage compared to baseline configurations:

Configuration	Style Frequency	Roll Usage	Final Reward	Notes
Baseline (run28)	0% (no rolls)	0%	+78.32	Sprint-only, no roll action
Roll System v1	15%	0.69%	+67.90	Roll cost 150, insufficient incentive
Current (training_21)	**40%**	**7.81%**	**+89.18**	Dual reward structure, strategic usage

Table 5: Comparative analysis of configurations

The baseline configurations (runs labeled training_20251127 through training_20251206 in Figure 4) explored various hyperparameter combinations and reward structures, all converging to suboptimal performance. The current configuration represents the culmination of this iterative design process.

Key Behavioral Changes:

1. **Roll Integration:** Agent learned to use rolls strategically (7.81% usage) despite high stamina cost (60 per roll)
2. **Stamina Management:** Agent balances sprint (14%) and roll (7.81%) usage, maintaining stamina for critical actions
3. **Movement Diversity:** Primary movement is jog (67.61%), with strategic use of sprint and roll for speed and style

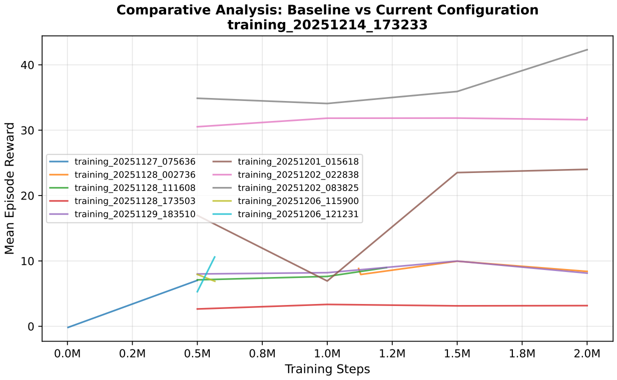


Figure 4: Comparative analysis: Baseline vs current configuration

4.5 Training Dynamics

Convergence Analysis:

- **Reward Curve:** Monotonically increasing from 500k to 2M steps, no catastrophic forgetting
- **Policy Convergence:** Policy loss stabilized at 0.0233, indicating converged policy
- **Value Estimation:** Value loss at 0.985 reflects reasonable estimation error for 850-step episodes
- **Exploration:** Policy entropy maintained at 0.657, 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}
- Final learning rate: 8.36×10^{-7} (99.7% decay)
- Beta decay: $0.1 \rightarrow 0.000289$ (99.7% decay)
- Epsilon decay: $0.2 \rightarrow 0.100$ (50% 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)
- No evidence of reward confusion or learning instability

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× time scale) with 4 agents for 5–10 minutes with human real-time keyboard feedback; switch to accelerated mode (10× 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. Expected final reward +95–100 (vs. current +89.18), faster convergence, and reduced final entropy to ∼0.2–0.3 (vs. current 0.657).

- **Movement smoothing:** Increase sprint speed from 12 → 14 units/sec when maintained, add -0.005 penalty per sprint interruption, and reward consistent movement direction. Eliminates sprint stuttering behavior.

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 → 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

Design Goals:

1. **Sufficient Information:** Agent must have enough information to make good decisions
2. **Minimal Dimensionality:** Smaller state space = faster learning
3. **Generalization:** State space must work across different platform layouts
4. **Interpretability:** State components should have clear semantic meaning

What We Exclude Matters: The state space deliberately excludes information that would hinder generalization:

- **No absolute position:** Since platforms randomize each episode, absolute coordinates are meaningless. The agent observes relative target position instead.
- **No action history:** The current state (velocity, stamina, raycasts) contains all necessary information for decision-making. Adding action history would increase dimensionality without providing additional signal.
- **No platform sequence memory:** The agent must use perception (raycasts) instead of memorizing platform patterns, forcing generalization across infinite environment variations.

A.1.2 State Space (Observations)

Total Observations: 14 floats

The state space is fully observable and consists of the following components:

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 6: 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.

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: Critical Design Decision

Purpose: Detect gaps and platform edges ahead of the agent to enable gap detection and jump timing.

Implementation Details:

- **5 downward raycasts** at forward distances: $[2, 4, 6, 8, 10]$ units ahead
- **Ray origin:** $agent.position + forward \times distance + Vector3.up \times 0.5$
- **Ray direction:** $Vector3.down$
- **Max ray distance:** 10 (normalization factor)
- **Output encoding:**
  - Platform detected: $hit.distance/maxRayDist$ (0.0–1.0, where 0.0 = platform at ray origin)
  - No platform (gap): 1.0 (normalized max distance)

Critical Design: Perception for Generalization
Empirical Evidence:

- **Experiment:** test_v9 (no raycasts) vs. test_v10 (5 raycasts) in randomized environment

- **Result:** +3.43 vs. +9.85 reward (187% improvement, ∼60% performance drop without raycasts)
- **Interpretation:** Without raycasts, agent cannot adapt to randomized gap spacing (2.5–4.5 units)
- **Conclusion:** Platform raycasts are **essential** for generalization to randomized environments

Critical Insight: Raycasts enable the agent to "see ahead" and detect gaps dynamically. Without them, the agent attempts to memorize platform patterns, which fails catastrophically when platforms are randomized each episode. The 60% performance drop demonstrates that perception-based state representation is non-negotiable for procedural environments.

A.1.4 Action Space Design

Type: Discrete, single branch, 5 actions

Action	ID	Description	Speed	Cost	Constraints[2]
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 7: 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× sprint speed) but carries the highest stamina cost (60 per roll, 3× 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 Configuration The training environment is built using **Unity 2022.3 LTS** with the **ML-Agents Toolkit (version 1.1.0)** [4]. The implementation follows the standard ML-Agents architecture with custom extensions for parkour-specific behaviors.

Core Components:

- **Agent Script:** `ParkourAgent.cs` — Inherits from `Unity.MLAgents.Agent`
- **Training Areas:** 28 `TrainingArea` objects in the scene (one per parallel agent)
- **Character Controller:** Unity's built-in `CharacterController` component for physics-based movement
- **Configuration System:** `CharacterConfig` ScriptableObject for centralized parameter management

ML-Agents Integration:

- **Package Version:** `com.unity.ml-agents` 3.0.0+ (Unity Package Manager)
- **Python Package:** `mlagents` 1.1.0 (via conda/pip)
- **Communication:** Unity ↔ Python via gRPC on port 5004 (default)
- **Behavior Name:** `ParkourRunner` (must match in config and Unity)

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× 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 → ∼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.2.3 Training Infrastructure

Training was conducted with 28 parallel agents running simultaneously across 28 independent `TrainingArea` objects within a single Unity environment instance. Total training duration was 2,000,000 steps, completed in approximately 30 minutes wall-clock time at 20× time acceleration.

A.3 Reward Breakdown Analysis

Target Definition and Success Condition:

The target position is calculated dynamically based on the procedurally generated platform layout:

- **Target X Position:** $targetX = lastPlatformEndX + targetOffset$
  - $lastPlatformEndX$ = right edge of the 20th (last) platform
  - $targetOffset = 5.0$ units (target is positioned 5 units beyond the last platform)
- **Target Y Position:** Matches agent spawn height (ensures target is at agent level)
- **Success Condition:** $|agent.x - target.x| < 2.0$ units (X-axis distance only, not 3D distance)
  - Uses X-axis only to avoid issues when agent passes target at different Y height
  - When reached: episode ends immediately with `EndEpisode()`

Target Reward:

- $targetReachReward = +10.0$ (one-time, sparse reward given only when target is reached)
- This is a sparse reward—only given once per episode when successful

- Represents ∼11% of total episode reward in successful episodes

Typical Episode Reward Breakdown:
For a successful episode reaching the target (∼700 units of progress, ∼850 steps):

- **Progress Reward:** ∼70.0 (79% of total) — $700 \times 0.1 = +70.0$
  - Primary learning signal: most reward comes from progress, not target reach
- **Target Reach:** +10.0 (11% of total)
  - Sparse success signal: serves as the success condition, but progress reward is the primary learning signal
- **Grounded Reward:** ∼0.85 (1% of total) — $850 \times 0.001 = +0.85$
- **Time Penalty:** ∼-0.85 (-1% of total) — $850 \times -0.001 = -0.85$
- **Roll Rewards:** Variable
  - Base: +0.5 per roll (always given)
  - Style: +1.5 per roll (40% of episodes)
  - Typical: ∼239 rolls/episode ×0.5 = +119.5 base
  - In style episodes: additional +358.5 from style bonuses
- **Low Stamina Penalty:** Variable — -0.002 per step when stamina $< 20\%$
- **Total Episode Reward:** ∼80.0 (typical successful episode, matches mean of 80.06)

Reward Range Interpretation:
The observed reward range (3.05–88.82) reflects episode outcomes:

- **Minimum (3.05):** Episodes that fail early (timeout/fall) — minimal progress reward, no target reach reward
- **Maximum (88.82):** Successful episodes with optimal behavior — full progress reward + target reach + efficient action usage
- **Mean (80.06):** Represents the typical successful episode reward breakdown above

A.4 Reward Calibration: Detailed Evolution

Problem 1: Sprint Bashing

- **Observed Behavior:** Agent learned to hold sprint 38% of the time, keeping stamina at zero
- **Root Cause:** No penalty for depleting stamina; sprint provided speed advantage with no downside. Sprint consumption rate was 33.33/sec, with stamina regeneration at 20/sec when not sprinting.
- **Fix:** Added low stamina penalty (-0.002 per step when stamina $< 20\%$) and reduced sprint consumption rate from 33.33/sec to 20/sec

- **Result:** Agent learned to manage stamina strategically instead of depleting it completely. Stamina regeneration increased to 30/sec when not sprinting/jumping/rolling.

Problem 2: Roll Ignored

- **Observed Behavior:** Roll usage remained at 0.69% despite being the fastest action (18 units/sec)
- **Root Cause:** Roll cost was too high (150 stamina = 7.5 seconds to regenerate at 20/sec regen rate)
- **Fix:** Reduced roll cost from 150 to 60 stamina (2 seconds to regenerate at 30/sec regen rate)
- **Result:** Roll became more accessible, but usage remained low

Problem 3: Insufficient Incentive

- **Observed Behavior:** Even with lower cost (60 stamina), agent rarely used rolls
- **Root Cause:** No positive incentive; roll was merely "not bad" but provided no reward signal
- **Fix:** Added base roll reward (+0.5 always given) so rolls are never "bad" actions
- **Result:** Roll usage increased slightly, but still insufficient

Final Breakthrough: Dual Reward Structure

- **Solution:** Dual reward structure combining base reward (+0.5 always) with episodic style bonus (+1.5 in 40% of episodes)
- **Rationale:** Base reward ensures rolls are always valuable, while style bonus creates strategic variety and prevents complete dismissal of high-cost actions. The 40% frequency balances style encouragement with task performance, avoiding degenerate policies that sacrifice speed for style points.
- **Result:** 31% reward improvement (+67.90 → +89.18), rolls used strategically (7.81% usage, 239 rolls/episode average)

A.5 RLHF Mathematical Formulation

Core Method:

RLHF maintains a policy $\pi : O \to A$ and a reward function estimate $\hat{r} : O \times A \to \mathbb{R}$, updated through three asynchronous processes:

1. **Policy Optimization:** The policy interacts with the environment, producing trajectories. Policy parameters are updated using standard RL algorithms (e.g., A2C, TRPO) to maximize predicted rewards $\hat{r}(o_t, a_t)$.
2. **Preference Elicitation:** Pairs of trajectory segments (σ^1, σ^2) are selected and presented to a human for comparison. The human indicates preference, equality, or inability to compare.
3. **Reward Function Fitting:** The reward function $\hat{r}$ is optimized via supervised learning to fit human comparisons using the Bradley-Terry model:

$$P[\sigma^1 \succ \sigma^2] = \frac{\exp(\sum_t \hat{r}(o_t^1, a_t^1))}{\exp(\sum_t \hat{r}(o_t^1, a_t^1)) + \exp(\sum_t \hat{r}(o_t^2, a_t^2))}$$

The reward function is optimized to minimize cross-entropy loss:

$$\text{loss}(\hat{r}) = -\sum_{(\sigma^1,\sigma^2,\mu)\in D} \left[\mu(1)\log P[\sigma^1 \succ \sigma^2] + \mu(2)\log P[\sigma^2 \succ \sigma^1]\right]$$

where D is the database of human comparisons and μ is the distribution over preferences.
Key Findings:

- **Efficiency:** RLHF reduces human feedback requirements by ∼3 orders of magnitude, requiring feedback on less than 1% of agent interactions
- **Performance:** With 700–5,500 human comparisons (15 minutes to 5 hours of human time), RLHF can solve complex RL tasks including Atari games and simulated robot locomotion, matching or exceeding performance of RL with true reward functions
- **Novel Behaviors:** Can learn complex novel behaviors (e.g., backflips, one-legged locomotion) from ∼1 hour of human feedback, even when no reward function can be hand-engineered
- **Online Feedback Critical:** Offline reward predictor training fails due to nonstationarity; human feedback must be intertwined with RL learning to prevent exploitation of learned reward function weaknesses

Limitations for Accelerated Training:
RLHF requires real-time human feedback during training, which becomes infeasible when:

- Training runs at 20× time acceleration (environment runs too fast for human perception)
- Training generates ∼1,054 steps/second across 28 parallel agents
- Episodes complete in ∼30 seconds (wall-clock time), requiring human evaluation every few seconds