Mirror’s Edge: Policy Gradient

AI Parkour Agent

Victor Tenneroni | vt2435@columbia.edu

Problem Statement

Problem: Navigate randomized 3D platforms with resource constraints

- 20 platforms, gaps 2.5-4.5 units, widths 20-84 units
- Stamina system (100 max): Sprint costs 20/s, Roll costs 60, Jump costs 20

Why RL?

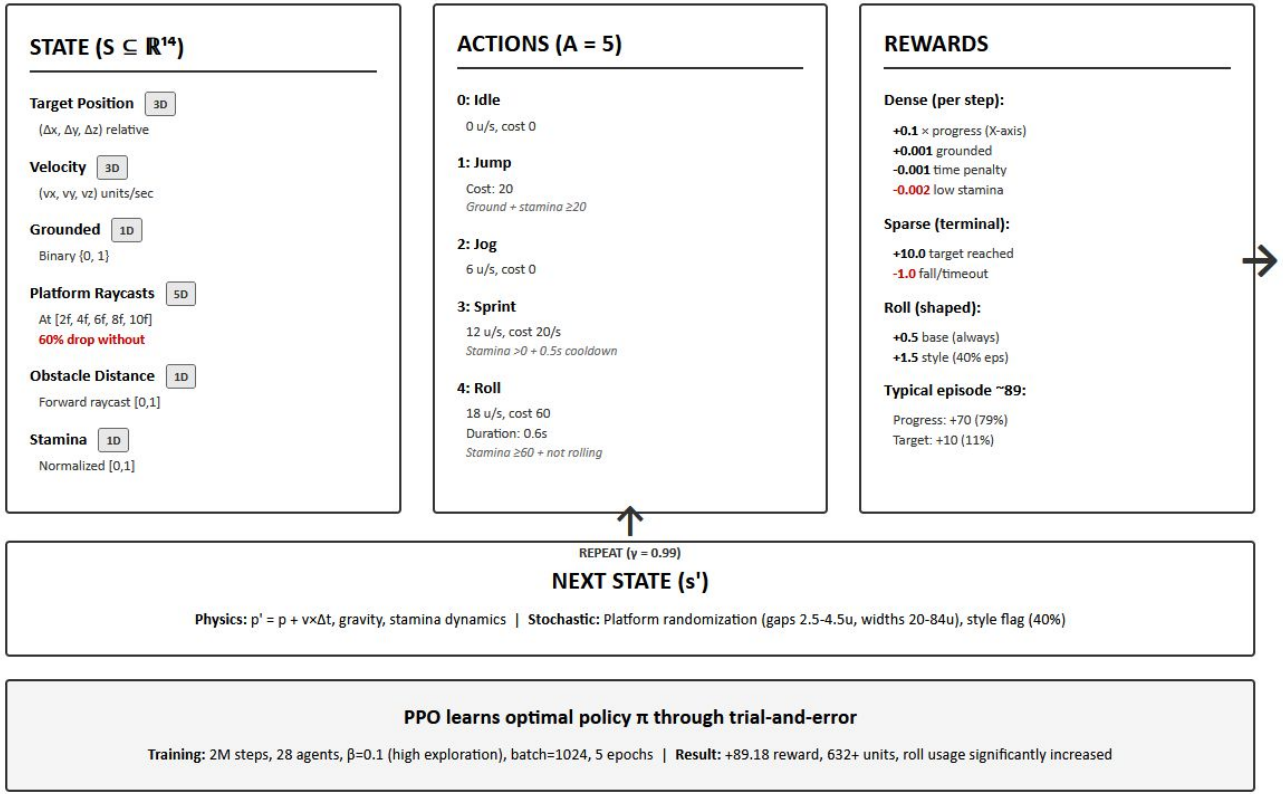
- High-dimensional state (14 obs) + complex action space (5 discrete)
- Infinite environment variations → must generalize, not memorize
- Strategic tradeoffs: speed vs stamina conservation
- No closed-form solution for stamina dynamics + randomization

Traditional Approaches Fail

- Rule-based: Cannot handle randomization
- PID control: No strategic resource management
- Fixed environments: Agent memorizes sequences

https://youtu.be/2PWbs4OHtTM

MDP


STATE (S ⊆ ℝ¹⁴)
Target Position 3D
(Δx, Δy, Δz) relative
Velocity 3D
(vx, vy, vz) units/sec
Grounded 1D
Binary {0, 1}
Platform Raycasts 5D
At [2f, 4f, 6f, 8f, 10f]
60% drop without
Obstacle Distance 1D
Forward raycast [0,1]
Stamina 1D
Normalized [0,1]
ACTIONS (A = 5)
0: Idle
0 u/s, cost 0
1: Jump
Cost: 20
Ground + stamina ≥20
2: Jog
6 u/s, cost 0
3: Sprint
12 u/s, cost 20/s
Stamina >0 + 0.5s cooldown
4: Roll
18 u/s, cost 60
Duration: 0.6s
Stamina ≥60 + not rolling
REWARDS
Dense (per step):
+0.1 × progress (X-axis)
+0.001 grounded
-0.001 time penalty
-0.002 low stamina
Sparse (terminal):
+10.0 target reached
-1.0 fall/timeout
Roll (shaped):
+0.5 base (always)
+1.5 style (40% eps)
Typical episode ~89:
Progress: +70 (79%)
Target: +10 (11%)
REPEAT (γ = 0.99)
NEXT STATE (s')
Physics: p' = p + v×Δt, gravity, stamina dynamics | Stochastic: Platform randomization (gaps 2.5-4.5u, widths 20-84u), style flag (40%)
PPO learns optimal policy π through trial-and-error
Training: 2M steps, 28 agents, β=0.1 (high exploration), batch=1024, 5 epochs | Result: +89.18 reward, 632+ units, roll usage significantly increased

Algo: PPO with high exploration

Why PPO?

- vs DQN: Better for continuous observations (14-dim)
- vs A2C: Clipped objective prevents destructive updates
- vs TRPO: Simpler, computationally efficient

Key Hyperparameters:

Parameter	Value	Rationale
Learning rate	3e-4 (decay)	Standard PPO
Batch size	1024	Balance efficiency/noise
β (entropy)	**0.1 → 0.00074**	**High exploration (6.7× test_v11)**
ε (clipping)	0.2	Prevent large updates
λ (GAE)	0.95	Bias-variance tradeoff
Epochs	5	Multiple passes

Network Architecture:

- Actor-critic
- Policy: 2×256 hidden → 5 action logits
- Value: 2×128 hidden (separate network)

Tweaks:

1. **Episodic style bonus** (40% episodes): +1.5 roll reward
2. **Dual roll rewards**: +0.5 base (always) + 1.5 style (conditional)
3. **Low stamina penalty**: -0.002/step when <20%
4. **Action constraints**: Environment blocks invalid actions

Iterative Reward Design

Goal Alignment: Forward progress (primary) + time efficiency + stamina management + strategic risk-taking

Iterative Reward Design

- test_v11 (baseline): +0.1 progress, +10 target → +11.15 reward, no roll usage
- =run28 (sprint added): 12 u/s, 33.33/s cost → +78.32 reward (+603%), but sprint bashing (38% usage, stamina=0)
- training_171550 (roll v1): Cost 150, bonus +0.1 (15% eps) → +67.90 reward, rolls rarely used (7.5s regen too slow)
- training_210205 (final): Cost 60, base +0.5 (always), style +1.5 (40% eps) → +89.18 reward (+31%), rolls significantly increased

Key Improvements

1. Dual Roll Rewards: Base (+0.5 always) prevents learned aversion, style (+1.5 episodic) creates exploration
2. Multi-timescale Structure: Dense (progress, time, stamina) guide learning, sparse (target, fall) define success
3. Balanced Penalties: Time -0.001 (efficiency), low stamina -0.002 (conservation), fall -1.0 (moderate, not catastrophic)
4. Stamina Rebalancing: Sprint 20/s (reduced), regen 30/s (increased) → allows buildup for rolls/jumps

State Space Design

Critical: Platform raycasts [2f, 4f, 6f, 8f, 10f] - test_v9 (no raycasts) +3.43 vs test_v10 (raycasts) +9.85 (187% improvement)
Encoding: Target position (raw $\mathbb{R}^3$), velocity (raw $\mathbb{R}^3$), stamina [0,1], raycasts [0,1], grounded {0,1}
Why raycasts essential: Randomized environments prevent memorization → agent must perceive terrain

Reward structure drives strategy: Per-unit-time rewards → sprint bashing.
Solution: Cost-benefit balance via stamina penalties + achievable action costs + dual reward structure for underutilized actions.

Results - 30th training loop

Final Reward: +89.18 (at 2M steps)
Previous Best: +78.32 (run28)
Improvement: +14% over previous best

Training Progression:
- 500k steps: +26.67
- 1.0M steps: +45.25
- 1.5M steps: +81.60
- 2.0M steps: +89.18

Policy Loss: 0.0233 (mean, range 0.0175-0.0312)
Value Loss: 0.985 (mean, range 0.400-1.808)
Policy Entropy: 0.657 (mean, range 0.657-1.605)

Learning Rate: 8.36e-07 (final, decayed from 3.0e-4)
Epsilon: 0.100 (final, decayed from 0.2)
Beta: 0.000289 (final, decayed from 0.1)

Episode Statistics
- 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

Action Distribution
- Jog: 67.61% (primary movement)
- Sprint: 14.00%
- Roll: 7.81% (increased from 0.69% in previous run)
- Jump: 3.53%
- Idle: 7.04%

Action Counts (per episode, mean):
- Jog: 2072 actions
- Sprint: 424 actions
- Roll: 239 actions
- Jump: 102 actions
- Idle: 216 actions

Agent Behavior
- Roll Usage: 7.81% of actions (vs 0.69% previous run)
- Roll Count: 239 rolls per episode (mean)
- Roll Improvement: 11.3x increase over previous run

Critical Experiments & Lessons over 30 training loops

Experiment 1: Axis Bug (test_v4)

- **Setup**: Progress reward tracking Z-axis instead of X-axis
- **Result**: -2.497 reward, agent learned to idle (safest strategy)
- **Lesson**: Always verify reward signals track correct objective

Experiment 2: Training Duration (test_v5 vs test_v6)

- **Setup**: 500k steps vs 2M steps, same config
- **Result**: +5.976 vs +8.478 reward (+42% improvement)
- **Lesson**: 500k insufficient for convergence in this domain

Experiment 3: Randomization Requires Perception (test_v9 vs test_v10)

- **Setup**: test_v9 (no raycasts, randomized platforms) vs test_v10 (5 raycasts, randomized)
- **Result**: +3.43 vs +9.85 reward (187% improvement, 60% drop from fixed environment)
- **Lesson**: Fixed environments allow memorization, random environments require sensors
- **Key insight**: Agent without raycasts learned "run 50 steps, jump, repeat" in fixed layout, completely failed in randomized layout

Experiment 4: Beta Coefficient Impact (test_v11 vs =run28)

- **Setup**: β=0.015 (low entropy) vs β=0.1 (high entropy)
- **Result**: +11.15 vs +78.32 reward (603% improvement)
- **Lesson**: Complex action spaces need high exploration to discover optimal strategies
- **Evidence**: test_v11 entropy=0.035 (converged to local optimum), =run28 entropy=0.597 (diverse strategies)

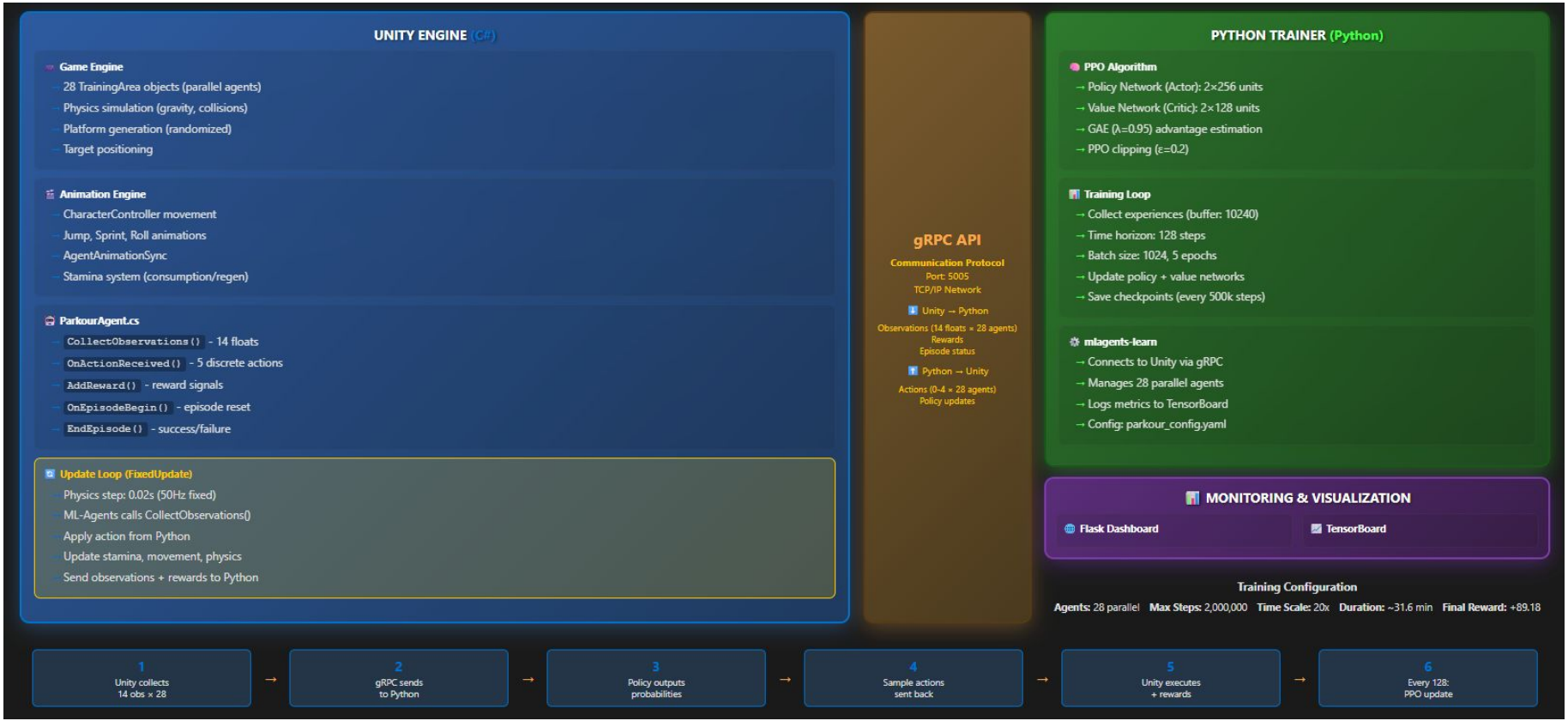
Experiment 5: Roll Cost Tuning (training_20251207_171550 vs training_20251207_210205)

- **Setup**: Roll cost 150 (1.5× max) vs 60 (0.6× max)
- **Result**: +67.90 vs +89.18 reward (+31% improvement)
- **Lesson**: Action costs must be achievable mid-episode, not just theoretically available
- **Evidence**: Roll cost 150 requires 7.5s regeneration from 0 stamina (unrealistic during gameplay)

Demo (inference)

https://youtu.be/geAkthimaI0

Architecture & Stack



Future Work

SHORT-TERM IMPROVEMENTS

1. Hyperparameter Optimization
   - Current: Beta 0.1 (linear), entropy 0.635 (over-exploring), LR 3e-4
   - Proposed: Beta 0.05→0.001, LR 5e-4→1e-5, Epsilon 0.15→0.05 (all exponential), Lambda 0.98, Epochs 3
   - Rationale: Exponential decay matches learning curves; current linear too slow
   - Expected: +95-100 reward (vs 89.18), faster convergence, lower entropy ~0.2-0.3
2. Movement Smoothing
   - Issue: Sprint stuttering, no flow penalty
   - Solution: Momentum system - continuous sprint 12→14 u/s, interruption penalty -0.005

LONG-TERM DIRECTIONS

1. Full 3D Movement - Multi-axis platforms, turning

2. Expanded Action Space - Add slide, wall jump, vault (5→9+ actions or continuous control)

3. (Dynamic) Obstacles - Moving platforms, time-dependent physics, partial observability

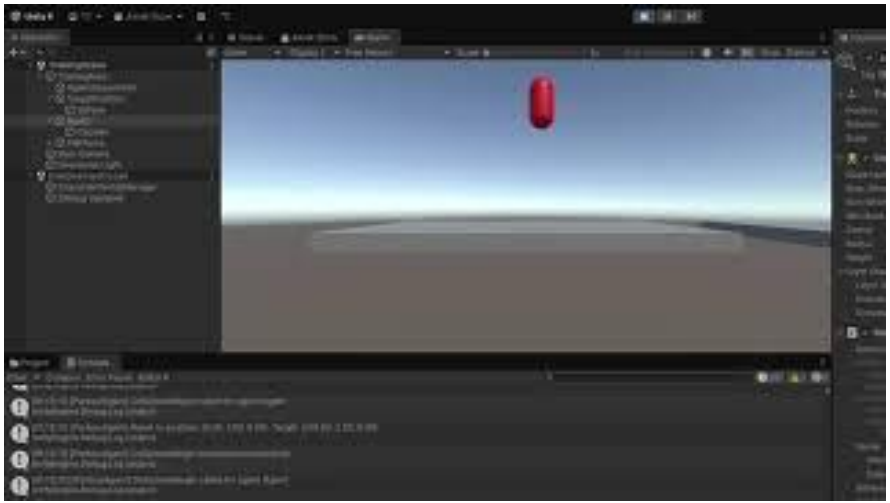
4. LLM-Assisted Tuning - Automated hyperparameter optimization on environment changes

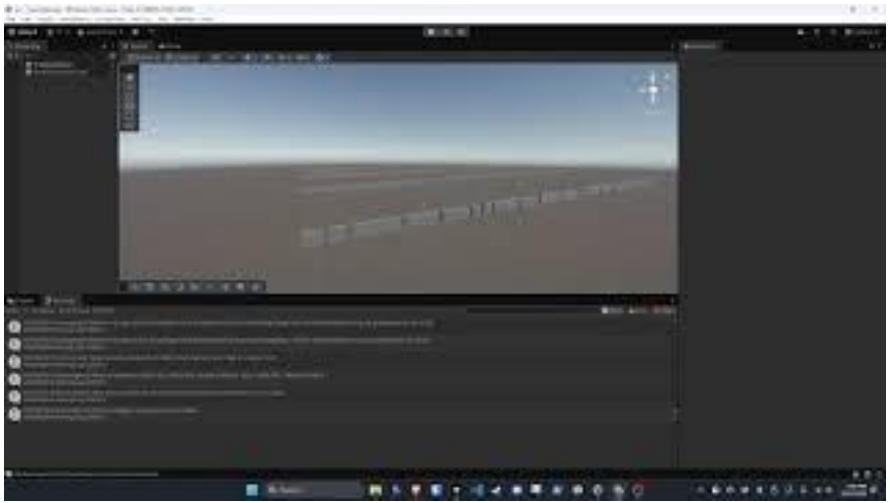
5. Q-Learning, DQN Benchmark

Thank you!

Appendix

Scaling Training in Unity





Sprint Bashing Demo

Prior Open Questions & Initial Development Path

Core Problems:

- Continuous → discrete state space conversion
- 2D → 3D navigation scaling
- Vision-based vs positional observations
- Fast-forward training in Unity

Two Development Tracks:

- Vertical: Add actions (slide, salto, wall-jump)
- Horizontal: Compare RL methods (PPO → Q-Learning → DQN)

Success = Agent navigates A→B with visible human influence on behavior

MDP

State Space S ⊆ ℝ¹⁴:

- Target relative position (3): `(Δx, Δy, Δz)`
- Velocity (3): `(vx, vy, vz)` units/sec
- **Platform raycasts (5)**: Downward at [2f, 4f, 6f, 8f, 10f], normalized [0,1]
    - **Critical**: 60% performance drop without raycasts
- Obstacle distance (1): Forward raycast [0,1]
- Grounded (1): Binary {0,1}
- Stamina (1): Normalized [0,1]

Reward Function:

- Dense: +0.1×progress, +0.001 grounded, -0.001 time, -0.002 low stamina
- Sparse: +10.0 target, -1.0 fall
- Roll: +0.5 base (always) + 1.5 style (40% episodes)

Action Space A = {0,1,2,3,4}:

Action	Speed	Stamina Cost	Constraints
Idle	0	0	Always
Jump	Impulse	20	Ground + stamina≥20
Jog	6	0	Always
Sprint	12	20/s	Stamina>0 + 0.5s cooldown
Roll	18	60	Stamina≥60 + not rolling

Past Experience making Games with AI

Raycasting engine in C with
Stable Diffusion Textures

LLM RPG in BabylonJS

https://drive.google.com/file/d/110FcmJgdU1eOwg5Pv-QXF9nvBX9XceBS/view?resourcekey

Policy Gradient Learning System

Why PPO:

- Trial-and-error learning (no expert demos)
- Handles delayed consequences
- Proven for continuous control in Unity ML-Agents

Stack: Unity 2022.3 + ML-Agents + Python + Cloud GPU