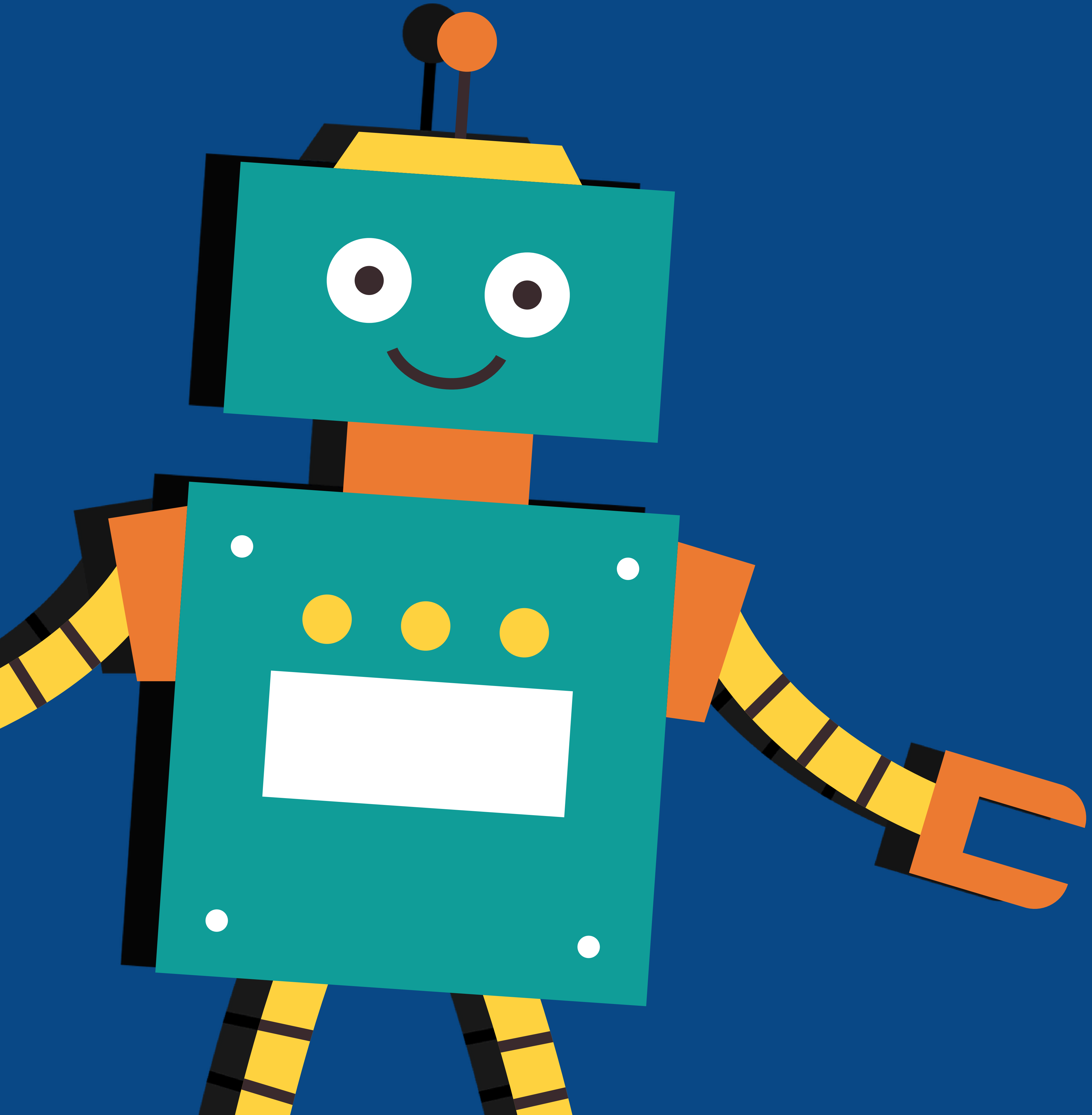
DRL SUMO ROBOT SIMULATION

65340500046 653405000063



CORE CONCEPTS

Train a single PPO agent to control two sumo robots with shared reward, and observe how different reward designs affect cooperative or aggressive behavior.

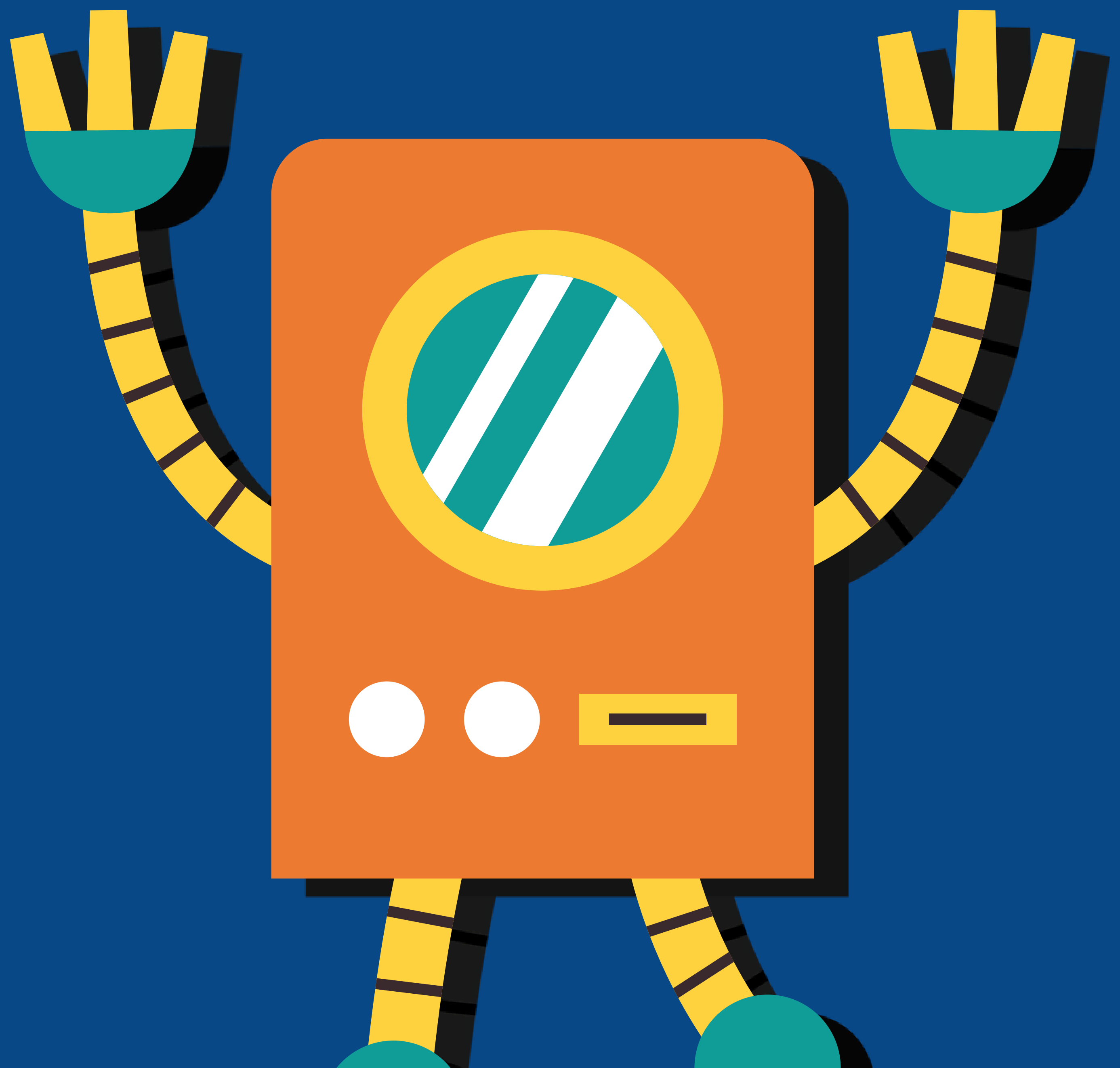
PROJECT GOALS

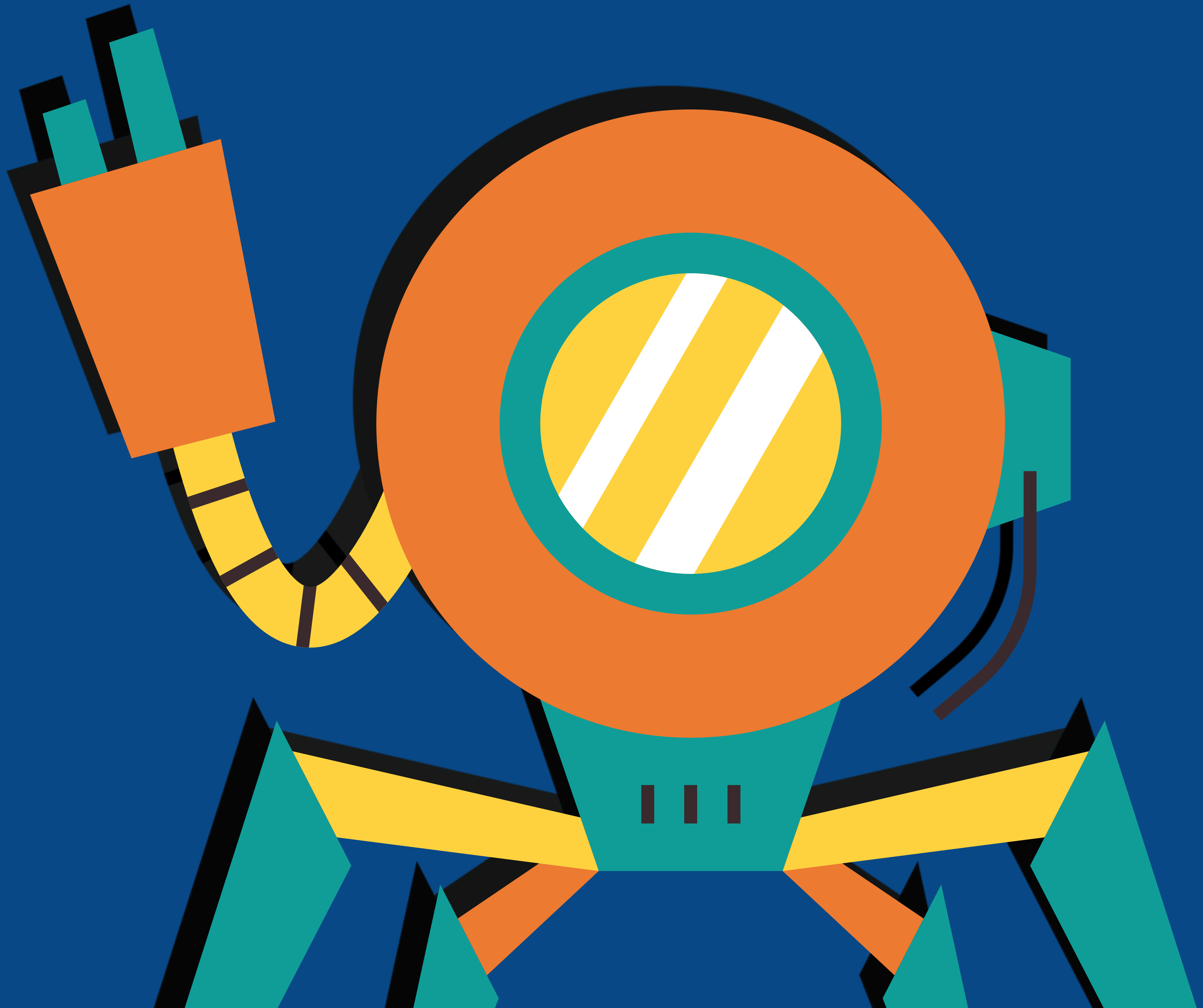
Train a shared PPO agent to control two sumo robots

- using RL
- Observe influence of reward designs competitive behavior in a simulated dohyo.

Observe ENV/Reward designs influence to competitive behavior

- Test with different type and amount of reward
- Test with changed environment (slip floor)
- Evaluate Competitive behavior





SCOPE OF THE PROJECT

Scopes

• Train two robots to compete using a single PPO agent
• Focus on continuous control and shared-reward shaping
• Compare behavior under four different reward strategies
• Evaluate performance through simulation-only (no real robot)

BACKGROUND KNOWLEDGE

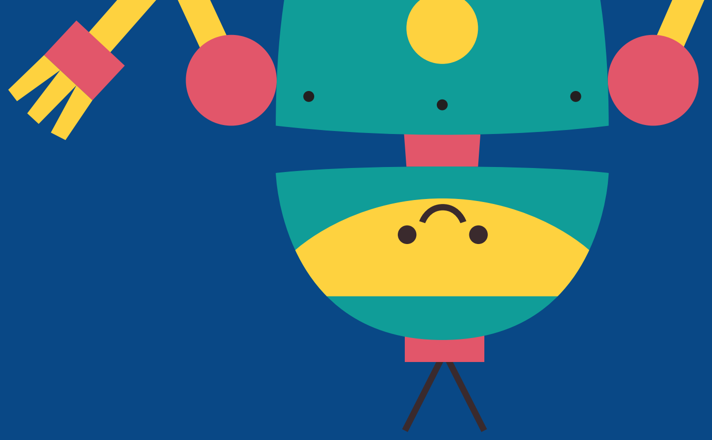
1. PPO (Proximal Policy Optimization)

Definition:
A deep reinforcement learning algorithm that improves policies using clipped objective updates to ensure stability.

2. Shared-Policy Control + Reward Design

Definition:
In our setup, a single PPO agent controls both robots using a shared 4D action space and a common reward signal.

This approach allows us to test how different reward strategies influence cooperative or competitive behavior between the two agents.



TECHNICAL STACK

Simulator

PyBullet + custom URDF

DRL Algorithm

PPO (Stable-Baselines3)

Control Setup

Shared PPO agent controlling two bots

Tools

Python, Gym, OpenCV (for cam)

LITERATURE REVIEW

Selection of PPO

PPO = Proximal Policy Optimization

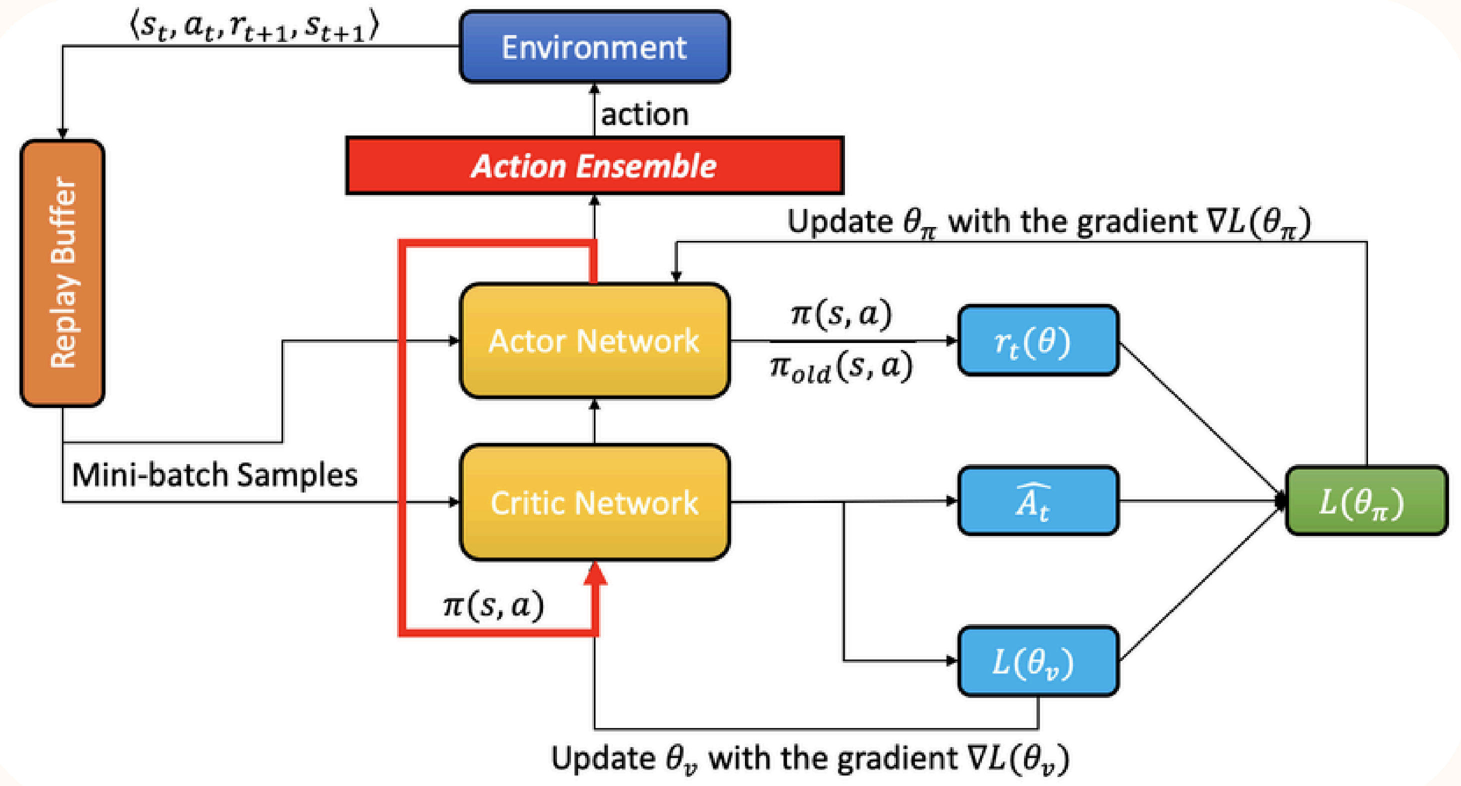
Continuous Control Support
Our bots use continuous motor speeds — PPO handles this natively.

Policy Gradient-Based
Learns directly from reward signals — no need to model the environment.

Stable Training
Uses clipped updates to avoid drastic changes
- helps with shared-agent learn.

Compatible with Shared Control
We use one PPO agent to control both bots
- PPO is simple to implement with shared observation→action mapping.

Widely Tested & Tunable
Supported by Stable-Baselines3, easy to configure and train with Gym + PyBullet.



Summary:
"PPO gives us stability, simplicity, and flexibility — perfect for training sumo bots with shared control and continuous actions."

PROBLEM FORMULATION

Train a robot to push its opponent out of the ring without falling, using reinforcement learning.

Problem Setup:

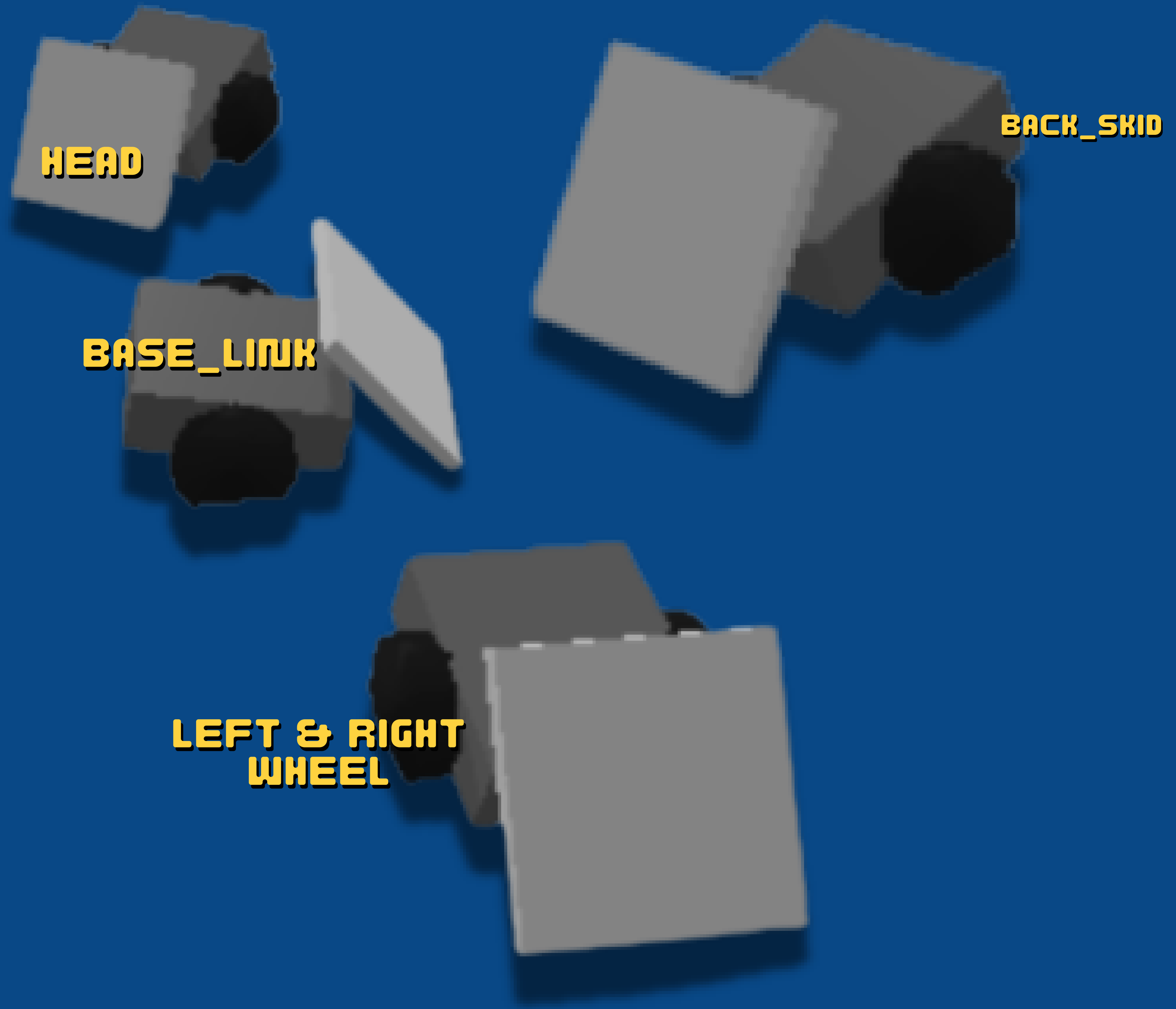
- Continuous state and action space (14D state, 4D action)
- Single PPO model controls both bots
- Both bots receive shared reward per step
- Reward design is the main variable (affects behavior)

Constraints:

- No curriculum learning or dynamic difficulty
- Physics-only simulation, no external supervision
- Bots spawn with randomized orientation (±90° yaw)

DESIGN

ROBOT DESIGN & URDF STRUCTURE



URDF Component

• base_link – main body (0.18 × 0.16 × 0.06 m, 1.0 kg)

• left_wheel & right_wheel – radius 0.05 m, motorized (continuous joints)

• head – angled front plate (0.18 × 0.18 × 0.015 m)
↳ Mounted with RPY: 0, -0.7, -π to tilt down

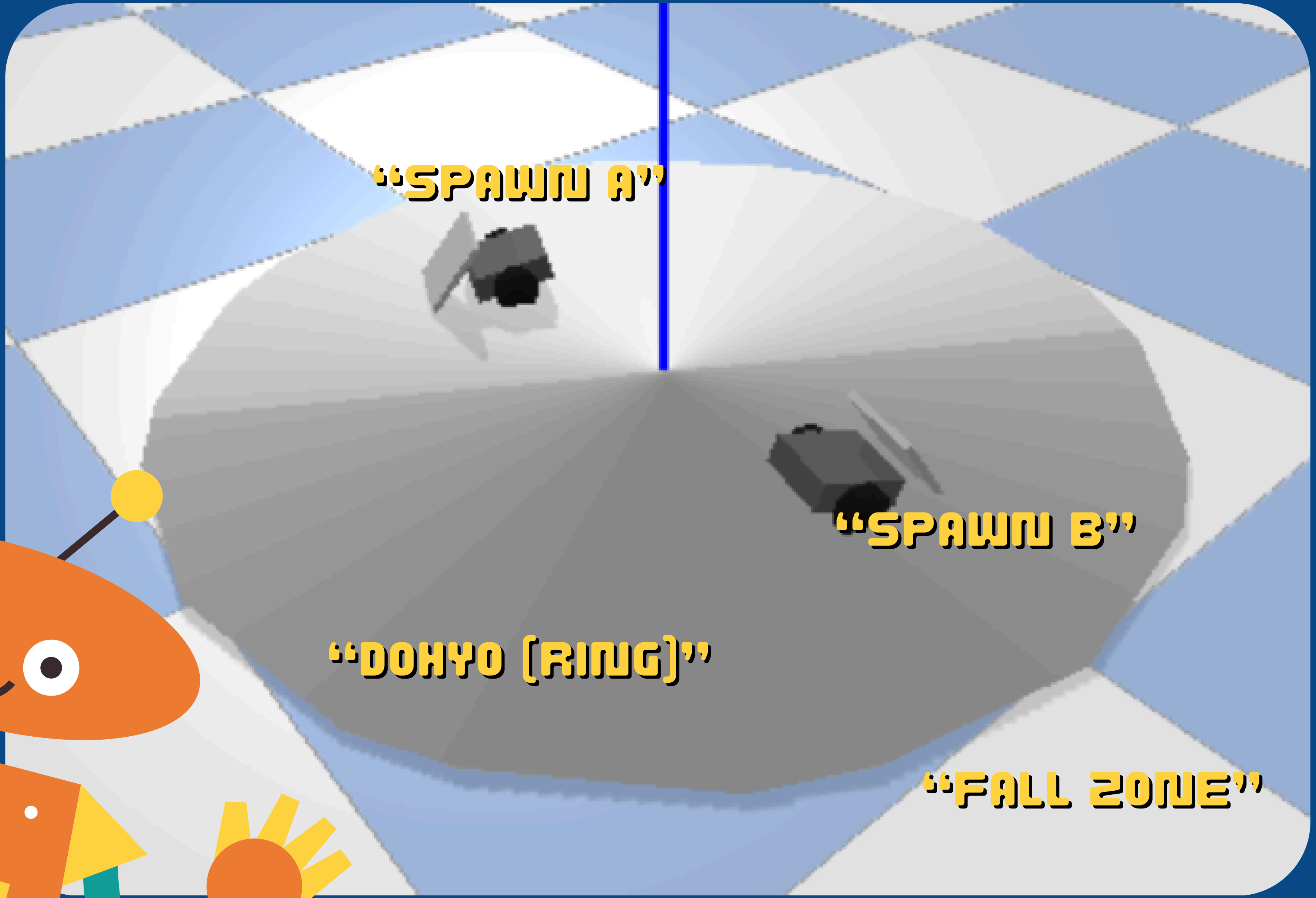
• back_skid – spherical skid (radius 0.01 m) for rear balance

Key Features:

✓ Forward wedge helps lift opponent
✓ Back skid prevents tipping
✓ Simple but realistic dynamics

DESIGN

ENVIRONMENT SETUP – PYBULLET + GYM CUSTOM ENV



"Agents train inside this simulated dohyo arena with custom reset rules."

- Simulator: PyBullet + Gymnasium wrapper
- Arena: Circular ring (radius 1.0m) with black and white layered design
- Agents: 2 identical sumo bots spawned face-to-face with randomized yaw (±90°)
- Reset conditions:
  - – Bot falls (excessive roll or pitch)
  - – Bot exits dohyo boundary
  - – Episode timeout

Camera: FPV camera mounted on each bot's head (OpenCV windows)

DESIGN

OBSERVATION & ACTION SPACE

ACTION SPACE

4-D vector
[lA, rA, lB, rB]
range (-1,1)

- 4-dim vector
- Controls 2 bots (shared agent)
- Scaled ×10 to motor speed

(2 motors per bot)

OBSERVATION SPACE

14-D vector
[Ax, Ay, Avx, Avy,
Bx, By, Bvx, Bvy,
distance, bearing,
rollA, pitchA, rollB, pitchB]

- Provides relative position + angle
- Includes tilt detection (fall check)

(Positions, velocities, relative distance & angle, pitch/roll of each bot)

IMPLEMENTATION
REWARD DESIGN (1)

Learn to push opponent out or avoid falling

+1 → Win (opponent falls or exits ring)
-1 → Loss (self falls or exits ring)
0 → Both fall or timeout

WIN/LOSS REWARDS

+ small bonus → push toward center
- penalty → near edge
- 0.001 → per step cost

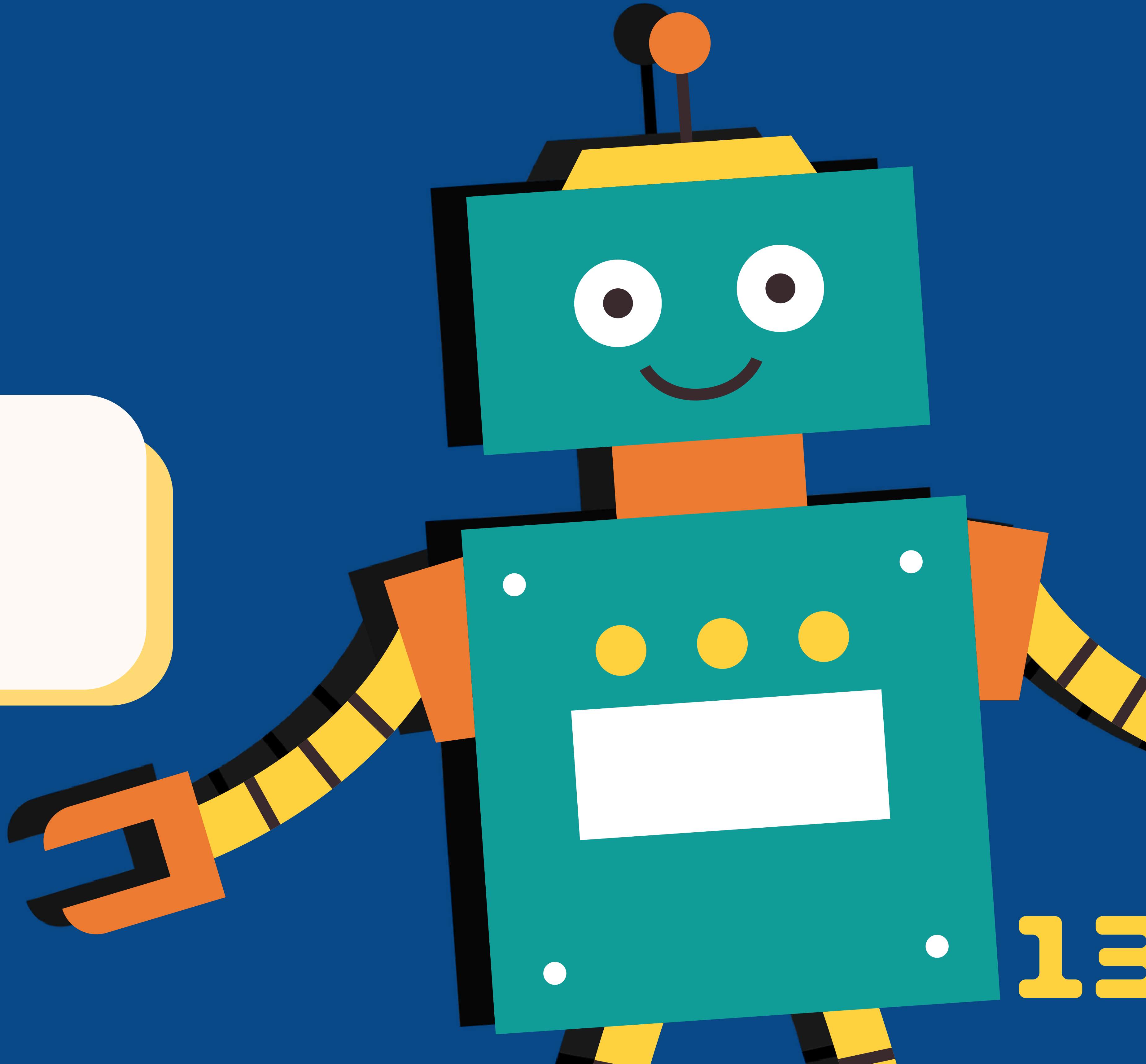
SHAPING

Bias one bot with extra shaping to see if advantage emerges

REWARD DESIGN (2)

More reward for 1 agent

Bot A gets:
  + extra push-center reward
Bot B gets:
  - standard shaping only

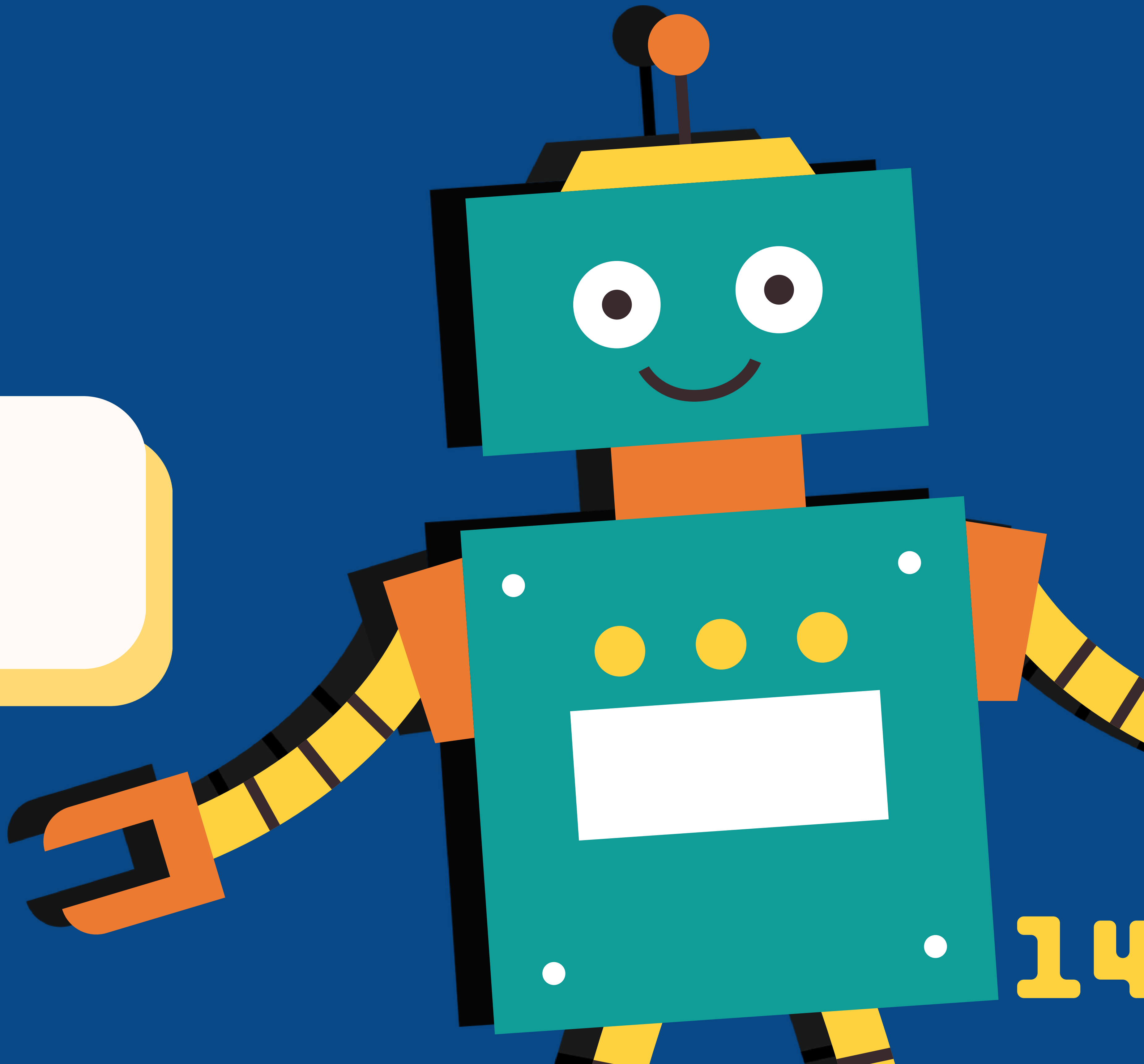


Encourage faster aggression and discourage camping

REWARD DESIGN (3)

Increase Step Penalty

```
shaping *= (1 - 0.005 * self.step_count)
```

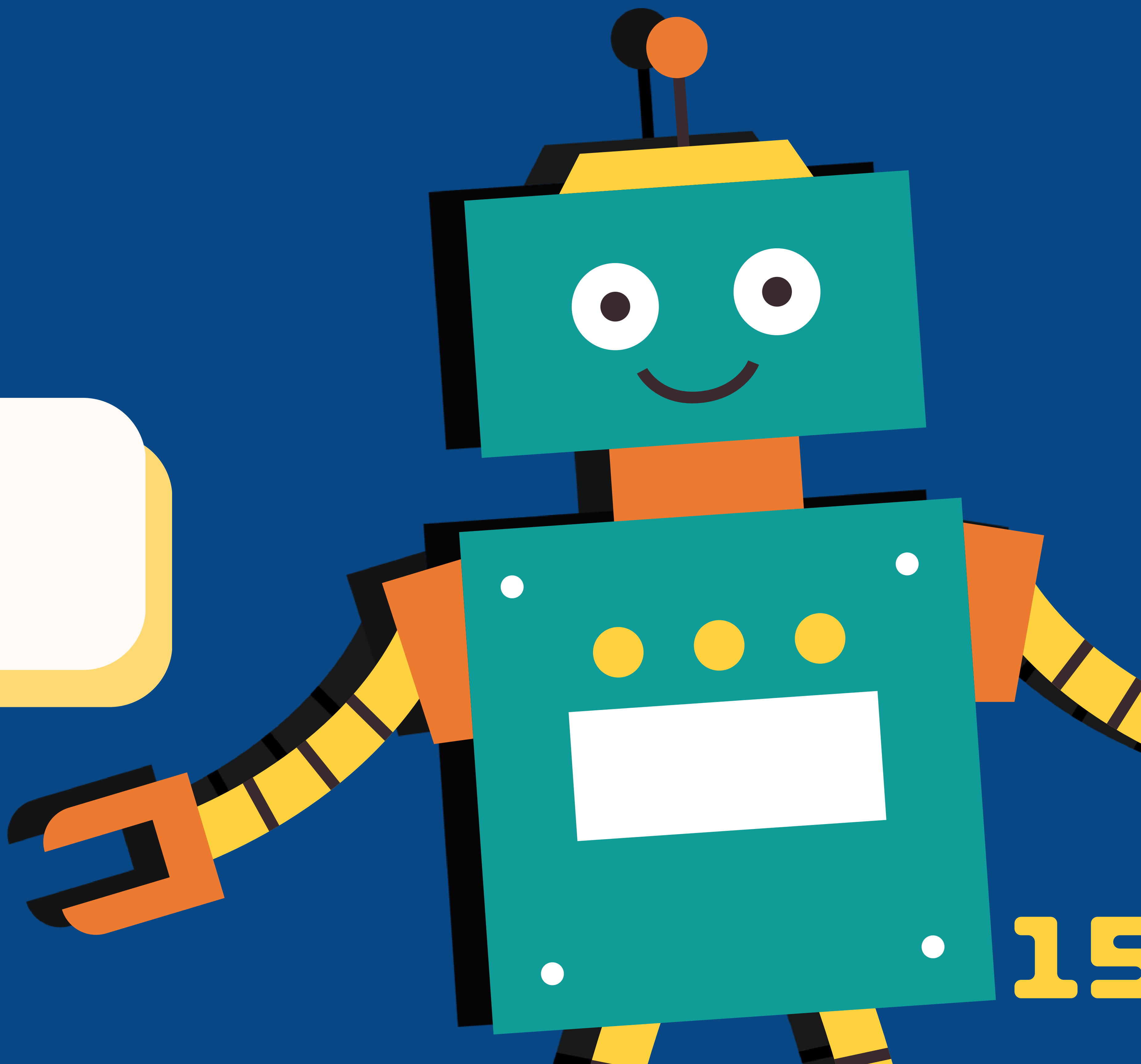


See how behavior changes under low frictio
physics

REWARD DESIGN (4)

Reduces Friction

```
p.changeDynamics(... lateralFriction = 0.001)
```



TRAINING CONFIGURATION

PPO SETTINGS

• Algorithm: PPO (Stable-Baselines3)
• Learning rate: 3e-4
• Batch size: 2048
• Gamma: 0.99
• Entropy coeff: 0.01

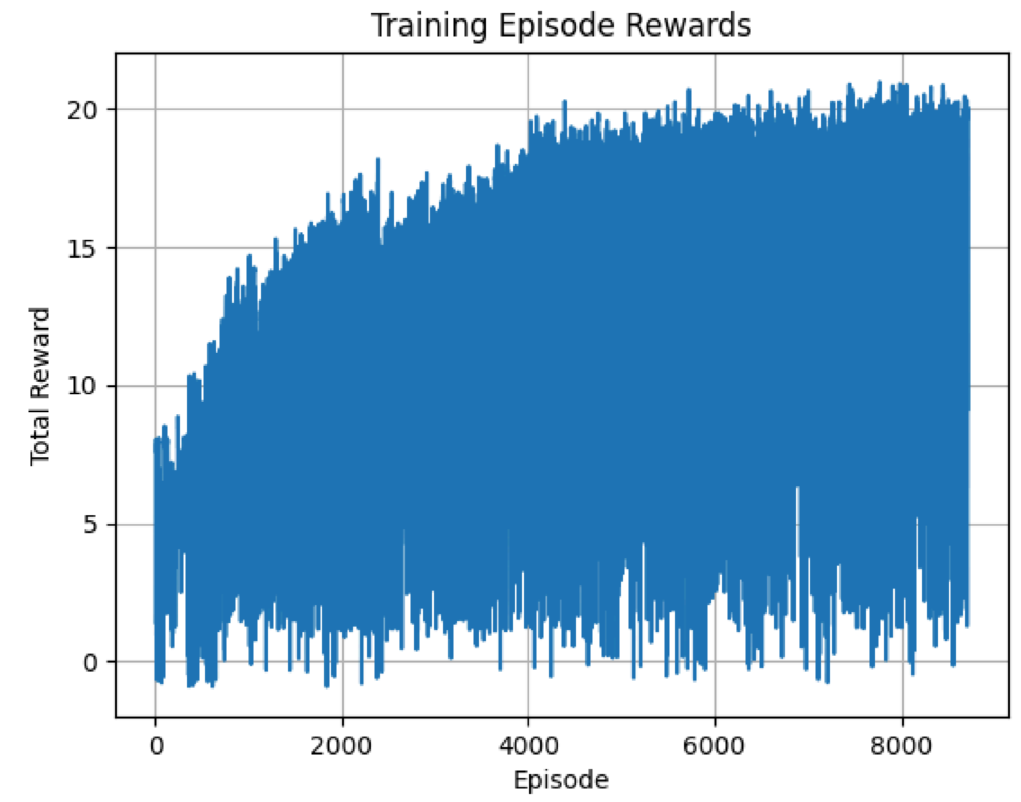
TRAINING LOOP

Shared-Agent Training Loop
• One PPO model controls both bots
• Reward is shared across both agents
• Bots spawn with random yaw (±90°)
• Trained for ~X steps (fill in actual)
• Final model: ppo_sumo.zip

10,000,000 steps

TRAINING PHASE

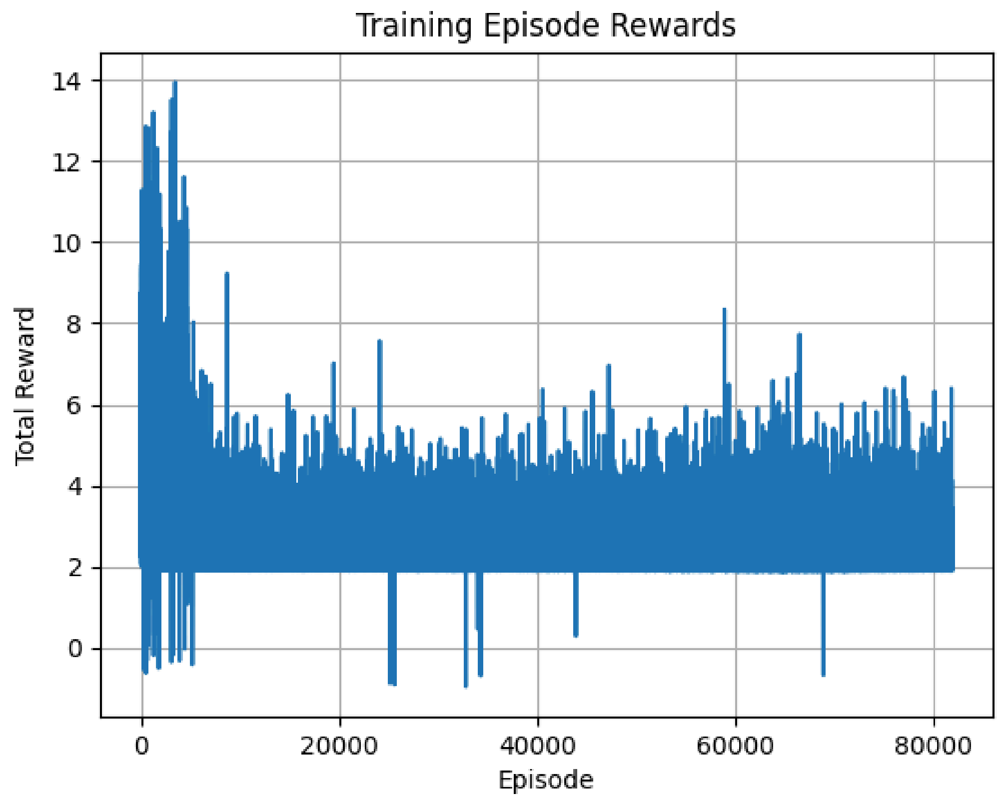
TRAINING EPISODE REWARDS — COMPARISON ACROSS REWARD DESIGNS



Base Setting

Peak reward/ep :
- 20

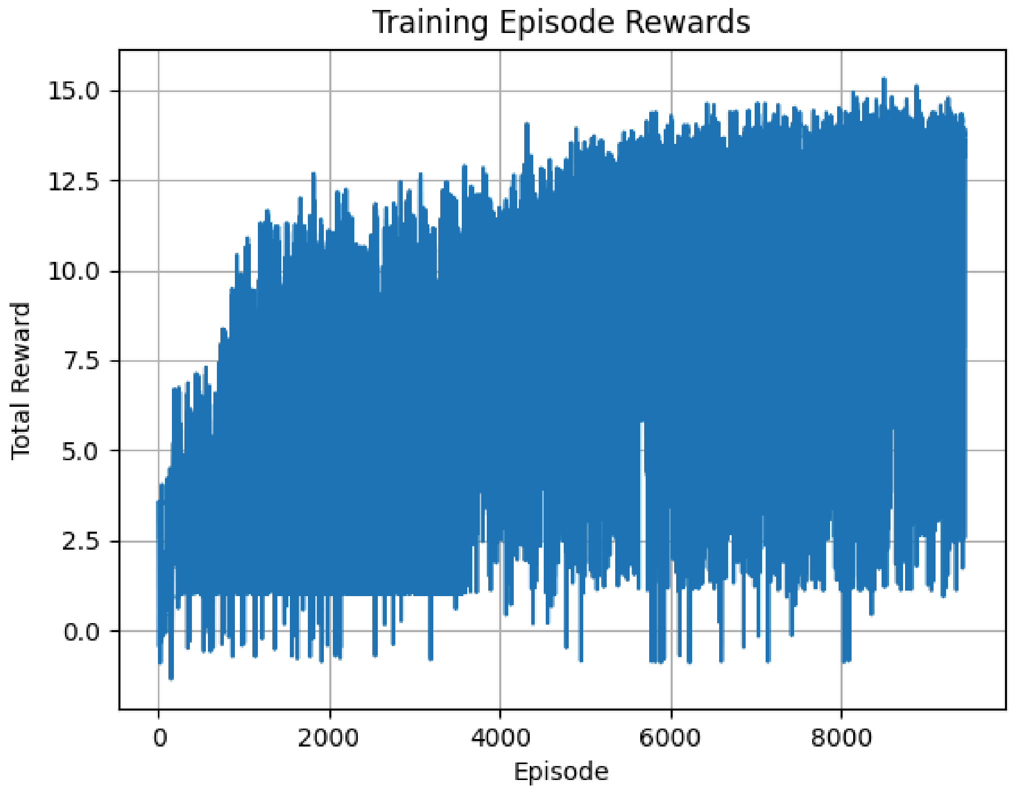
Behavior :
- Learn steadily but slow
- Settles with small reward by sticking to edge
- Lack aggressiveness



More Winning Reward A

Peak reward/ep :
- 14
- (end with 5 avg. reward)

Behavior :
- Explore sticking to edge strategy
- Exploit with new strategy despite lower reward



More penalty

Peak reward/ep :
- 15

Behavior :
- Reward rise quick but unsteady tills 6k+ eps
- Explore for longer

Low Friction Floor

Peak reward/ep :
- 22

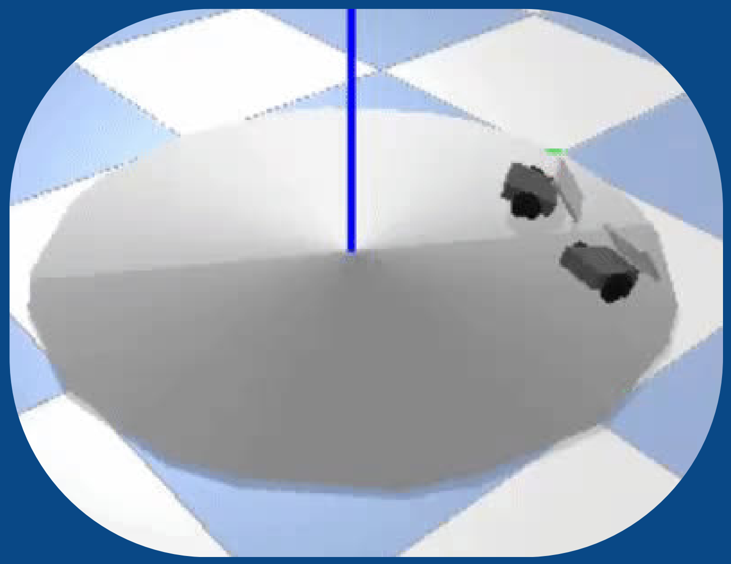
Behavior :
- The most steadily learned
- Learn/Exploit faster

BEHAVIORAL ANALYSIS

Reward Design (1) – Standard Observation

Both bots show symmetric behavior they aiming to push each other or just stay in the circle.

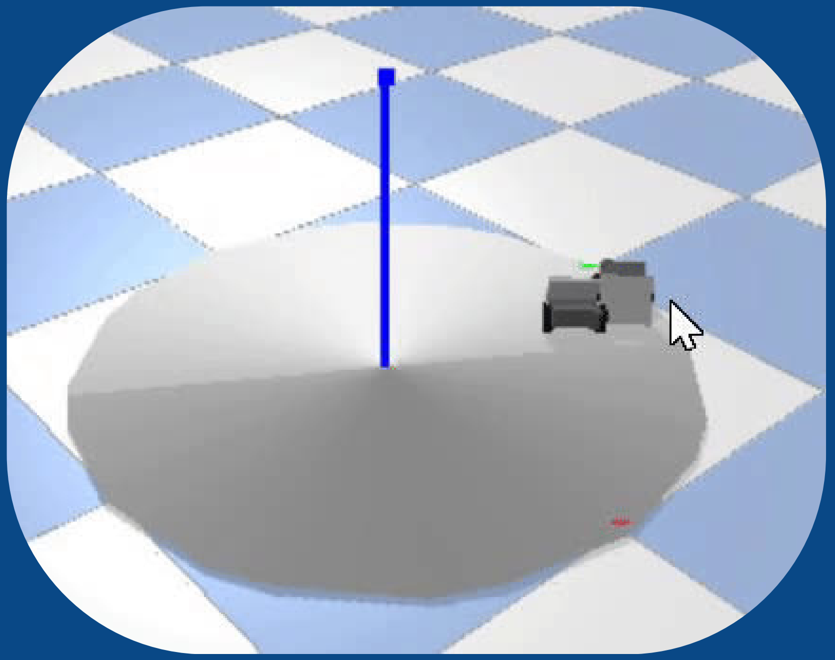
No clear role split. Strategy appears mirrored and balanced throughout.



Reward Design (2) – Biased Observation

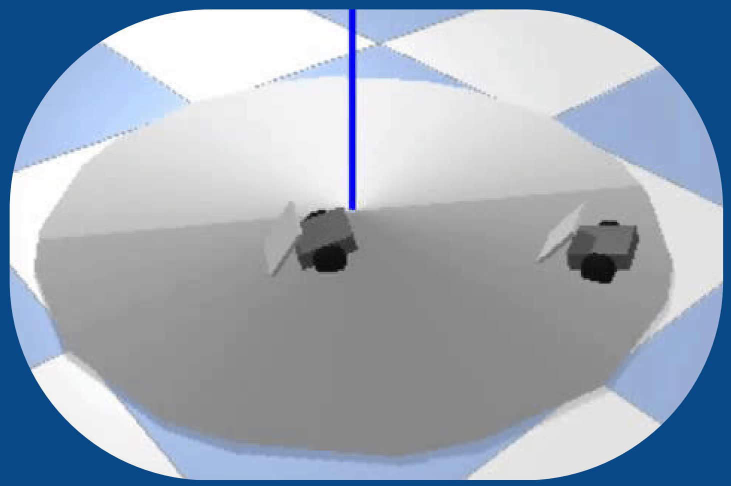
One bot consistently chased, while the other tried to hold still.
Clear role split emerged even though both bots use the same policy.

despite lower reward
Agent still choose to be agressive



Reward Design (3) – Decay Over Time Observation
Bot highly focus on Exploring early on, trying to end match fast before reward dropped.

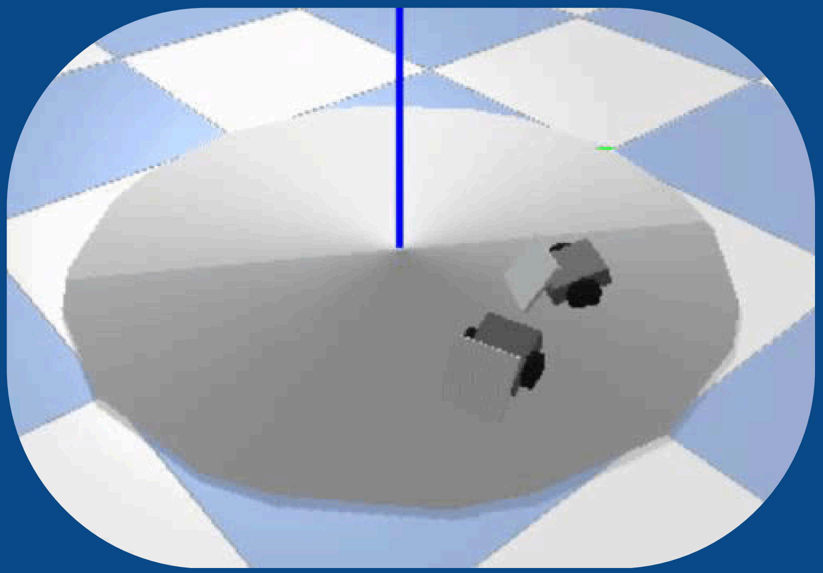
Ended up settling with less risky reward
Good for exploration boost



Reward Design (4) – Slippery Floor Observation

Bots became less aggressive.
Most of the behavior was focused on staying upright and avoiding sliding out of the ring.

Settles with non-agressive/safe behavior



EVALUATION

Rewards

- Too low “Completion reward”
- Too high “Behavior guidance rewards”
- Acceptable timestep penalties (could be reduced for exploration)

Environment/Agent

- Agent’s pushing power is too weak to simulate agressive combat
- Should reduce floor friction or increase Agent torque

CHALLENGES

URDF Design Issues

Building the robot in URDF required precise joint placement and orientation.

- Wheels initially rotated on the wrong axis due to incorrect rpy setup
- Front wedge (head) often floated or clipped into the ground due to mismatched origin/rotation

OpenGL Driver Errors

Our machine used an Intel HD Graphics 2500, which does not support OpenGL 3.3.

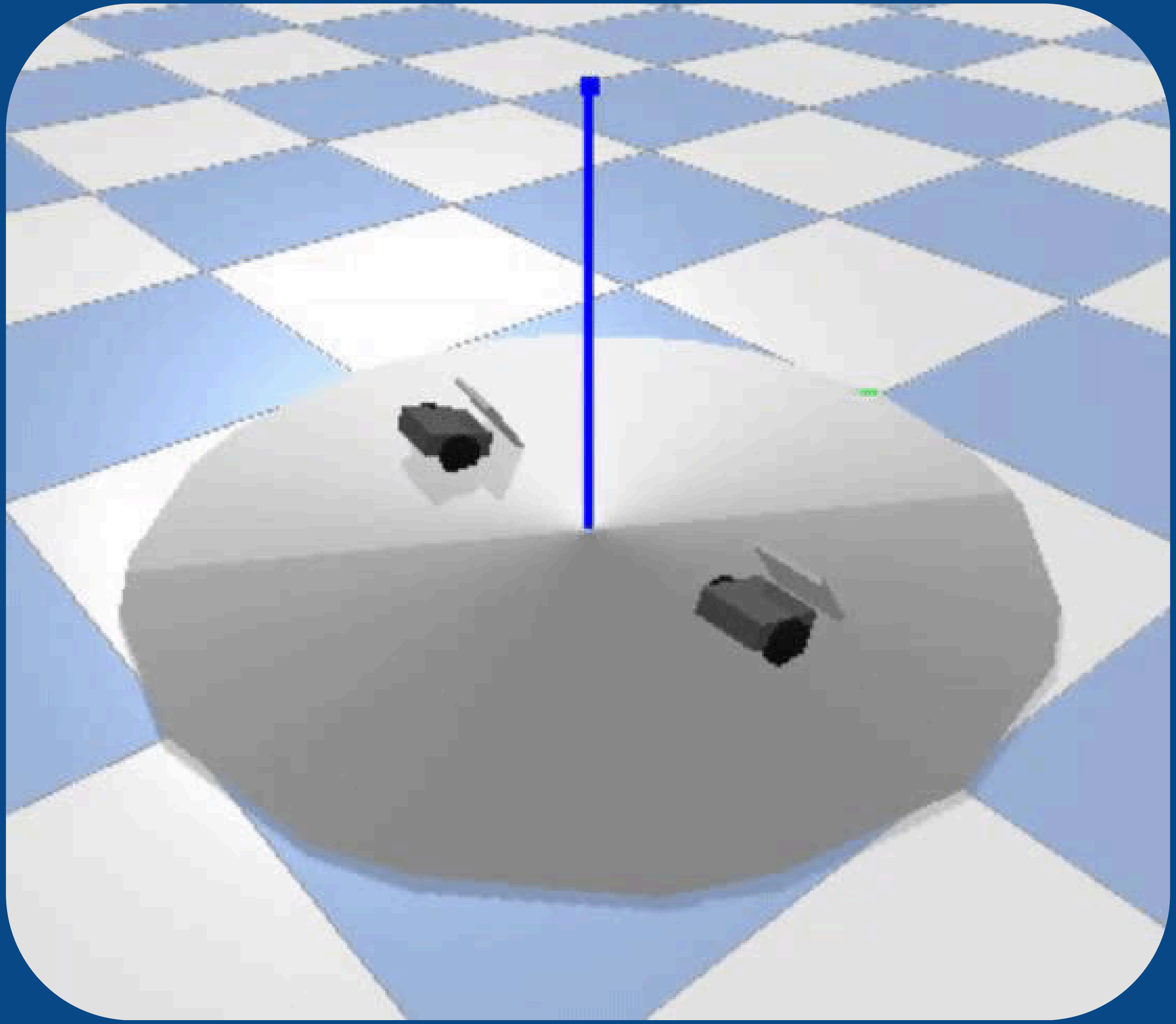
- PyBullet GUI failed with: “Failed to create OpenGL context”
- Required workaround using LD_PRELOAD and LIBGL_DRIVERS_PATH to fix driver errors
- Still ran slowly and limited visual performance

PyBullet Camera View Setup

We wanted to attach a camera to the robot’s head for FPV (First-Person View).

- PyBullet GUI only supports previewing one camera at a time
- Used OpenCV (cv2.imshow) to show two robot cams simultaneously

SELFPLAY (MULTI-AGENT)



Preview of Selfplay Training

- Compete with Previous 50k timestep
- Trained for 2M timestep
- Refactor into only rewarding 1 agent

CURRENT SELFPLAY CHALLENGES

Self-Play Implementation Complexity

Implementing self-play required major changes to the original environment.

- Could not reuse standard environments
- Had to redesign reset() and step() to support two agents in one environment

Failed Self-Play Refactor

Attempted to redesign self-play logic to make it more advanced and modular(waste more times)

- significantly slower training, harder to tune, more bugs
- Eventually reverted to the original simple version and just added a few key lines.

FUTURE DEVELOPMENT PLANS

Rewards Adjustment

- Reward agents more for successful Knockout
- Clearly define the “Player” and “Opponent” and set Clear reward for “Player” Winning
- Slightly increase timestep penalties to raise training speed

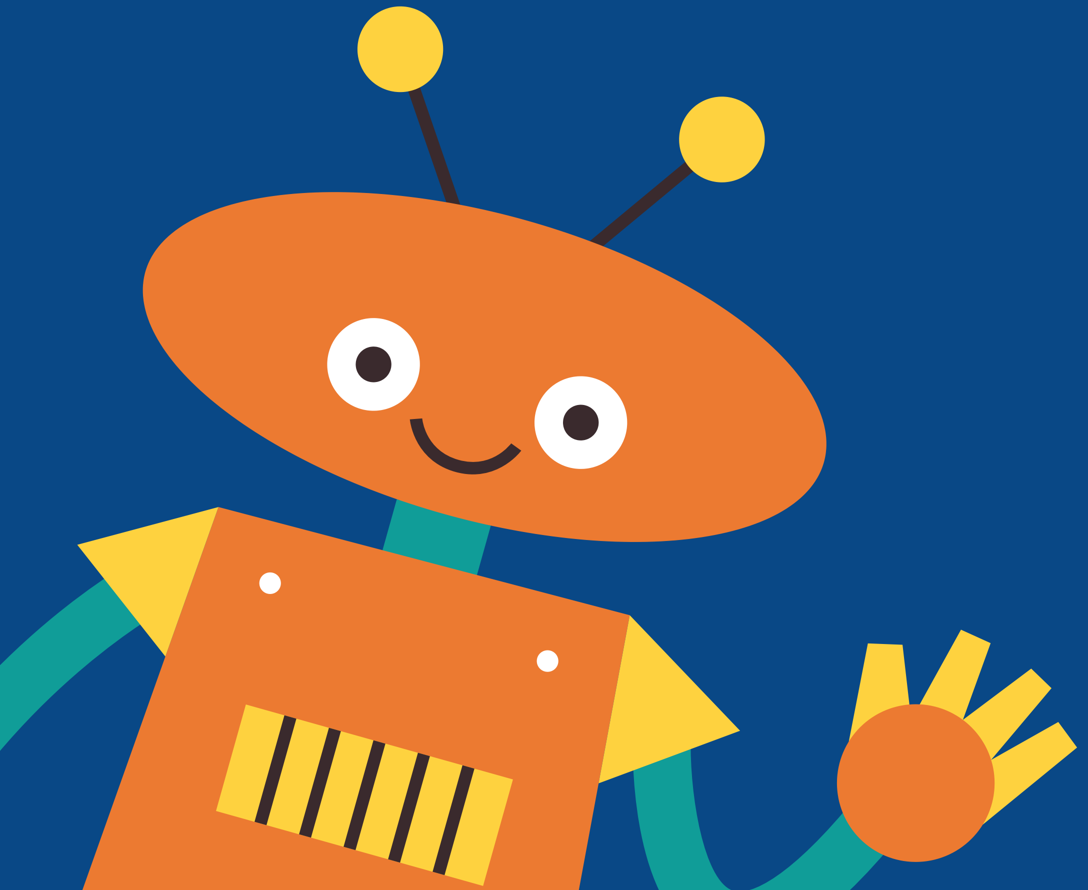
Environment

- Increase impact of “Collision” to promote more “Attack” and “Dodging” behaviors
- Reduce complexity of Self-play Algorithm and Attempts with more training
- Set a Clear Self-play curricullum to make sure “Player” agent gain helpful insight from it’s “Opponent”

THANK YOU

หุ่นซูโม่ ประลองแรง ฝึกแข่งกล
เรียนรู้เอง ผ่านเกม จำลองยืน
ใช้ **PPO** เป็นทาง ฝึกเรียนรู้
ปรับรางวัล พลิกแพลง วิธีมอง

ฝึกฝนตน ไม่หยุด แม้สุดฝืน
ท้าทายคืน วันด้วย มันสมอง
ฝ่าความงง งวยอยู่ กลางดงผอง
แผนจู่จอง ด้วย **SELF-PLAY** อย่างแท้จริง



Q&A