Deep Reinforcement Learning for the Unity "Tennis" Multi-Agent Environment

1. Introduction 3
2. Methodology 3
   1) MADDPG agent 3
   1) Team Reward 3
   2) Replay buffer 4
   3) Search for the best hyperparameters via grid search 4
   4) Network architectures (actor + central critic) 5
      a) Shared actor FCDP 5
      b) Central critic 6
      c) Target networks initialization and update 6
   5) Training process 7
      a) Training actions (Gaussian noise injection) 7
      b) greedy actions 8
      c) Online data collection 9
      d) Optimisation 9
      e) Online Critic update 10
      f) Online Actor update 11
      g) Early stopping 11
      h) Evaluation 12
         i. Online evaluation during training 12
         ii. Final evaluation 12
      a) Training sequence 12
      b) Network Usage and Reward Flow Across Training Phases 13
   6) Bootstrap CI 13
3. Results 14
   1) Best hyperparameters 14
   2) Solved environment 15
      a) Best model over seeds 15
      b) Plot of performance of during training 16

c) Bootstrap CI result 17
4. Suggestions for future work 17
1) Replace simple Gaussian exploration with adaptive exploration 17
2) Make the team reward adaptive 17
3) Use double critic 18

1. Introduction

The current report aims at solving the **Unity environment "tennis"** as described in the accompanying README file.

This is a **multi-agent** environment where both agents receive individual rewards that encourage keeping the ball in play (longer rallies), making the setup effectively cooperative.

The current solution implements the Multi-Agent Deep Deterministic Policy Gradient (**MADDPG**) agent on the basis that the environment is **non stationary**. With MADDP, the environment becomes **stationary** given the joint action of both agents/players.

Additionally , in order to improve cooperation, we reshape the reward used for training , by shifting it from a per-agent environment reward to a shaped **team reward**. The max over per-agent return per episode required by the project assignment will continue to be used as a performance metric and early stopping condition.

2. Methodology

1) MADDPG agent

The agent trains a **single shared deterministic actor** (one policy network used by both agents) with a **centralised critic** that sees the **joint state and joint action** and outputs **two Q-values (one per agent)**.

The **shared actor** (parameter sharing) is used by both agents , given they are identical (they have the same observation/action spaces and pursue the same long term objective).

This design is called Centralised Training, Decentralised Execution (**CTDE**), where the critic sees everything (the joint state and joint action) and produces **future value for each agent** while during execution, each agent still acts from its **local observation** through the shared actor.

1) Team Reward

The **team reward** per environment step is the sum of agents' rewards, decreased by a penalty in case of termination. This encourages both players to avoid letting the ball hitting the ground or hitting the ball out of bounds. A soft **team lambda** of 0.5 is used.

$$\mathbf{r}_t \in \mathbb{R}^2$$
$$r_t^{team} = (r_t^{(1)} + r_t^{(2)}) - 0.5\, d_t$$

$$d_t \in \{0, 1\} \text{ where } d_t = \mathbf{1}[\text{any agent done}]$$

Note that the critic is trained as if both agents share a cooperative shaped reward, but the assignment's metric for reporting and early stopping is still computed from the raw environment returns.

2) Replay buffer

At each time step, vectors of joint state and joint action are built and used in a **tuple** that is populated into the replay buffer. Note that the shape of a state is 24 and not 8 as mentioned in the assignment (positions, velocities of ball and racket stacked over 3 frames). The replay buffer is implemented with arrays of size 48+4+2+1 = 55.

$$x_t = [s_t^{(1)};\ s_t^{(2)}] \in \mathbb{R}^{48} \qquad u_t = [a_t^{(1)};\ a_t^{(2)}] \in \mathbb{R}^4$$
$$(x_t,\ u_t,\ \mathbf{r}_t,\ x_{t+1},\ d_t)$$
$$\mathbf{r}_t = (r_t^{team},\ r_t^{team})$$

The replay buffer makes the training off-policy, decorrelated (random mini batches from the replay buffer almost make the training data **independent and identically distributed**), stable, and sample efficient. Without replay, the critic would chase its own moving target and explode.

3) Search for the best hyperparameters via grid search

The grid of parameters :

- Hidden layer's dimensions: (64, 64), (128,128), (256,256),
- optimizer: Adam,
- learning rate : 1e-4, 3e-4, 5e-4,
- buffer_size: 50000,

- batch_size: 64, 128, 256, 500,
- exploration_noise_ratio: 0.05, 0.1, 0.2,
- n_warmup_batches: 5, 10,
- target_update_every: 2, 5, 10,
- tau: 0.001, 0.005, 0.01.

The experiment begins with a hyperparameter search conducted over 10 random trials. The trial achieving the highest performance in the final evaluation is selected as the best configuration.

These optimal hyperparameters are then used for multiple runs with 10 seeds (seeds 33,41,66,39,8,77,21,20,44,22).
Among these runs, the final reported model is the online actor network from the run that reaches the early stopping criterion in the smallest number of training episodes.
The objective is to achieve convergence in fewer than 1500 training episodes.

4) Network architectures (actor + central critic)

Note that the hidden dimensions in the below networks are determined by the search for the best parameters.

a) Shared actor FCDP

It is the Fully Connected Deterministic Policy (FCDP).

$$\pi_\theta : \mathbb{R}^{24} \to [-1,1]^2$$

Architecture:

- Fully-connected Multi Layer Perceptron (MLP) with 2 hidden layers (64,64).
- ReLU hidden activations.
- Tanh output to land in $[-1,1]$. Although the network output is already bounded to [-1,1], clipping is added as a safety.

So, per agent *I* and step *t*:

$$a_t^{(i)} = \pi_\theta(s_t^{(i)})$$

b) Central critic

A joint critic takes joint states and joint action (vector of size 48+4 = 52) as input and outputs **two Q-values** at once:

$$Q_\phi : (\mathbb{R}^{48}, \mathbb{R}^4) \to \mathbb{R}^2$$

$$Q_\phi(x_t, u_t) = \begin{bmatrix} Q_\phi^{(1)}(x_t, u_t) \\ Q_\phi^{(2)}(x_t, u_t) \end{bmatrix}$$

Each scalar is an estimate of the **expected return** of one agent.

Architecture:

- MLP with 2 hidden layers (64,64).
- joint state–action input ($x_t \in \mathbb{R}^{48}$, $u_t \in \mathbb{R}^4$).
- One fully connected layer applied to the joint state x_t. The encoded state is then concatenated with the joint action u_t, followed by additional fully connected layers with no output activation, allowing unbounded value estimates.

$$(x, u) \mapsto \mathrm{MLP}_\phi([\, f(x), u\,])$$

$$h_1 = \sigma(W_1 x + b_1)$$

$$h_2 = \sigma(W_2[h_1; u] + b_2)$$

$$Q_\phi(x, u) = W_3 h_2 + b_3 \in \mathbb{R}^2$$

- ReLU activation functions on all hidden layers.
- A final linear fully connected layer with 2 outputs, producing the joint action-value vector

c) Target networks initialization and update

The Temporal Difference (TD) targets are slow-moving copies (parameters are not learned) of the shared actor and the central critic. If the same rapidly-changing networks were used to build the TD targets, the targets would move every gradient step and the critic would often diverge.

$$\text{target actor } \pi_{\theta^-}$$

$$\text{target critic } Q_{\phi^-}$$

At the start, a hard copy is initialised.

$$\theta^{-} \leftarrow \theta, \qquad \phi^{-} \leftarrow \phi$$

Every "**target_update_every**" environment steps (Hyperparameter, which best value was found to be 10), using a smoothing factor "**tau**" (Hyperparameter, which best value was found to be 0.01), a **soft update** (**Polyak** averaging / exponential moving average) is applied to the copies :

$$\theta^{-} \leftarrow (1-\tau)\theta^{-} + \tau\theta$$
$$\phi^{-} \leftarrow (1-\tau)\phi^{-} + \tau\phi$$

Only 1% of the new online parameters are blended into the target at each update; 99% of the old target is retained.

Note that because the actor is shared between agents, the target actor is also shared:

- one online actor π_θ
- one target actor π_{θ^-}

Each agent simply feeds its own local observation into the same target actor to compute its part of the joint action.

5) Training process

a) Training actions (Gaussian noise injection)

The exploration noise is used to generate the training data. The noisy transitions are then fed into the replay buffer. This avoids on-policy greedy trajectories and prevents the buffer from collapsing into local optima very quickly.

At every training step, Gaussian noise is added to the deterministic actor output to produce the executed noisy action:

$$\tilde{a}_t^{(i)} = \mathrm{clip}\left(\pi_\theta(s_t^{(i)}) + \varepsilon_t,\ [-1,1]^2\right)$$
$$\varepsilon_t \sim \mathcal{N}(0,\sigma^2)$$
$$\sigma = \mathrm{exploration\_noise\_ratio} \times a_{\max}$$

The exploration noise is a hyper parameter in the grid search. The best value was found to be 0.2. The environment bounds are a_{max} = 1.

Clipping is required to keep the value within the environment boundaries.

$$\tilde{a}_t = \begin{cases} -1 & \text{if } a_t + \varepsilon_t < -1 \\ a_t + \varepsilon_t & \text{if } -1 \leq a_t + \varepsilon_t \leq 1 \\ 1 & \text{if } a_t + \varepsilon_t > 1 \end{cases}$$

The exploration noise is not used in the optimisation process to avoid polluting gradients.

From the per-step noisy actions, a step-level ratio "**ratio_noise_injected**" is calculated to measure how much exploration is actually happening relative to the agent's intended action. It is the relative size of the injected noise.

$$\text{ratio\_noise\_injected}(t) = \frac{1}{d} \sum_{j=1}^{d} \left| \frac{a_{t,j} - \tilde{a}_{t,j}}{h_j - \ell_j + \delta} \right|$$

t = Environment step at which the action is taken.
d : Dimension of the action vector (number of action components). In this environment, 2 per agent.

$a_t \in \mathbb{R}^d$: Greedy action produced by the current online actor.

$\tilde{a}_t \in \mathbb{R}^d$: Executed action after noise injection and clipping.

$\ell \in \mathbb{R}^d$: Lower bound of the action space (-1)

$h \in \mathbb{R}^d$: Upper bound of the action space (1).

δ: Small positive constant for numerical stability (1e-12).

In the log file, the ratio "**mean_exploration_ratio_over_100_episodes_from_training**" is an accumulation this per-step ratio over the episode :

$$\rho_{\text{episode}} = \frac{1}{T} \sum_{t=1}^{T} \text{ratio\_noise\_injected}(t)$$

T = number of environment steps in the episode.

b) greedy actions

They are the actions directly output by the actor.

Evaluation uses a greedy (no-noise) policy every episode.

$$a_t^{(i)} = \text{clip}(\pi_\theta(s_t^{(i)}), [-1,1]^2)$$

Greedy actions are also directly used in the optimisation.

c) *Online data collection*

Transitions are collected online (with noisy actions during training), storing **joint** transitions into replay buffer :

$$(x_t,\ u_t,\ \mathbf{r}_t,\ x_{t+1},\ d_t)$$

d) *Optimisation*

Optimisation starts after the size of the replay buffer D has reached the below threshold (warm-up):

$$|\mathcal{D}| > (\text{batch\_size}) \times (\text{n\_warmup\_batches})$$

$$\mathcal{D} = \{(x_t, u_t, \mathbf{r}_t, x_{t+1}, d_t)\}$$

"**batch_size**" and "**n_warmup_batches**" are hyperparameters , which best values are found to be 64 and 10. So the warmup threshold is 640 transitions.

Parameter optimisation is interleaved with environment interaction during online data collection, with one **actor–critic update** performed **at every environment step** using **minibatch** samples from the replay buffer.

Target networks are updated according to hyperparameter "**target_update_every**".

During optimisation, **at every environment step**, a minibatch is sampled randomly from the replay and provides the following array :

$x \in \mathbb{R}^{B\times 48}$, $u \in \mathbb{R}^{B\times 4}$
$\mathbf{r} \in \mathbb{R}^{B\times 2}$ (team-shaped duplicated reward)
$x' \in \mathbb{R}^{B\times 48}$
$d \in \mathbb{R}^{B\times 1}$ with values 0/1

This minibatch is used in the networks update described below. This optimisation is called **off-policy pattern**: **collect online, optimise offline** (minibatch from replay).

e) *Online Critic update*

The next state x' from the minibatch is split into two 24-d blocks :

$$x' = [s'^{(1)};\ s'^{(2)}]$$

Then the target actions $a'^{(1)}$ and $a'^{(2)}$ are computed and stacked to give the next joint action u' under the target policy.

$$a'^{(1)} = \pi_{\theta^-}(s'^{(1)}), \quad a'^{(2)} = \pi_{\theta^-}(s'^{(2)})$$
$$u' = [a'^{(1)};\ a'^{(2)}] \in \mathbb{R}^{B\times 4}$$

If the online actor π_θ was used instead of the target one π_θ —, then while training the critic, the policy would change, the value function would change, and the TD target would chase a moving object.

Next, the central critic target is output:

$$Q_{\phi^-}(x', u') \in \mathbb{R}^{B\times 2}$$

Then TD target is computed:

$$\mathbf{y} = \mathbf{r} + \gamma\, Q_{\phi^-}(x', u')\,(1-d)$$

Gamma is set to 0.99 and (1-d) broadcasts across the 2 reward components.

Next, the online critic prediction is calculated.

$$\hat{\mathbf{q}} = Q_\phi(x, u)$$

Then, the critic loss (MSE over both agent's Qs) is output:

$$\mathcal{L}_Q(\phi) = \frac{1}{2}\,\mathbb{E}\big[\|\hat{\mathbf{q}} - \mathbf{y}\|_2^2\big]$$

The old gradients of the loss are cleared from memory before the new ones are computed with respect to the critic parameters via backpropagation. Note that the TD target is detached so that gradients from the loss do not propagate into the target network.

$$\nabla_\phi \mathcal{L}_Q(\phi)$$

Then, the parameters are updated via Adam update rule.

$$\phi \leftarrow \phi - \alpha \, \mathrm{Adam}(\nabla_\phi \mathcal{L}_Q)$$

Alpha α is the learning rate. It is a hyperparameter. Its best value is found to be 0.0003.

f) Online Actor update

It follows the critic update.

First, we compute the current action for each agent with the shared online actor.

$$a^{(1)} = \pi_\theta(s^{(1)}), \quad a^{(2)} = \pi_\theta(s^{(2)})$$

Then, we calculate the 2 critic evaluations (2 agents) averaged across the minibatch, using the counterfactual joint actions.

$$q_1 = \mathbb{E}\big[Q_\phi^{(1)}(x, [a^{(1)}; \mathrm{stopgrad}(a^{(2)})])\big]$$
$$q_2 = \mathbb{E}\big[Q_\phi^{(2)}(x, [\mathrm{stopgrad}(a^{(1)}); a^{(2)}])\big]$$

"stopgrad" for q_1 means that $a^{(2)}$ is detached and the loss gradient cannot flow through the actor network.

Then the actor loss is computed:

$$\mathcal{L}_\pi(\theta) = -\frac{q_1 + q_2}{2}$$

The parameters are then updated using backpropagation on the loss and the Adam update rule.

g) Early stopping

Training stops when any of these triggers:

- The goal performance is achieved when the 100-episode moving average (MA) of the max-over-agents episodic return score exceeds +0.5.
- The wall-clock elapsed time exceeds 180 minutes.
- The total number of episodes exceeds 3000.

h) Evaluation

i. Online evaluation during training

After *every* training episode, exactly 1 greedy episode (no noise) is run using the shared online actor and the max-over-agents episodic return is stored.

$$\mathbf{r}_t^{env} = (r_t^{(1)},\ r_t^{(2)})$$

$$\text{Score}_{\text{episode}} = \max\left(\sum_t r_t^{(1)},\ \sum_t r_t^{(2)}\right)$$

Then a 100-episode MA of those results is computed.

This MA is used for early stopping if it reaches or exceeds +0.5 and if there are at least 100 values in the MA calculation.

This MA is used in the plot together with the episodic max-over-agents score.

ii. Final evaluation

After training finishes, the final model is evaluated over 100 episodes without noise.

For each episode, the same maximum score is computed.
The **mean and standard deviation** of these 100 episode scores are then calculated, providing a low-variance and reliable estimate of the policy's performance.

This mean performance is used as the comparison metric:

- first, to select the best hyperparameters across 10 trials, and
- once the optimal hyperparameters are fixed, to identify the best overall performance across 10 seeds.

a) Training sequence

Per training episode:

- Run one episode in Unity environment using noisy actions from the shared actor.
- Store each step into replay (with team-shaped reward).
- Once warm-up is passed, do minibatch updates every step.
- Update targets every "target_update_every" steps.

- End episode when Unity environment signals done for either agent.
- Evaluate immediately**:** run 1 greedy episode (no noise), calculate the max-over-agents episodic return, and record that evaluation score.
- Save an actor checkpoint online_policy_model.<episode>.tar.

After training ends:

- Run final evaluation of 100 greedy episodes with the final model, return mean ± std.

b) Network Usage and Reward Flow Across Training Phases

The below table highlights that :

- The team reward is directly used for data collection (into the replay buffer) and the optimisation phase, while the evaluation process uses the raw environment rewards.
- Explorative actions are used for the data collection while greedy actions are used by the optimisation and evaluation phases.

Phase	Shared Online Actor	Online Critic	Target Actor	Target Critic	Minibatch from replay	Reward type
Data collection (explorative strategy) into replay buffer	X					Team reward
Optimisation	X (greedy strategy)	X	X	X	X (explorative strategy)	Team reward
Evaluation (greedy strategy)	X					Environment reward

6) Bootstrap CI

The experiment uses a bootstrap 95% confidence interval to answer the question :

Given the final evaluation scores from multiple random seeds, is the mean performance very likely to be above the goal threshold (+0.5)?

The mean and the std are calculated over the list of final evaluation scores (10 seeds in our experiment).

Each bootstrap iteration creates a new list of 10 scores by randomly drawing from the original 10 scores allowing repeats. Then a mean is created for each resampled set. The script repeats that resampling 5000 times, ending up with a list of 5000 bootstrap means.

Then, the script takes quantiles:

- low = 2.5th percentile of bootstrap means.
- high = 97.5th percentile of bootstrap means.

That becomes the **bootstrap 95% confidence interval** for the **mean final score**.

The interpretation can be derived as follows:

- If the lower bound of the 95% CI is above the goal score of +0.5, then the script claims: “Significant (mean > goal @95%)? Yes”
- Otherwise: No (insufficient evidence that the mean is above the goal at 95% confidence)

For meaningful inference, the bootstrap CI is only relevant when using multiple seeds.

3. Results

1) Best hyperparameters

The hyperparameter search found that trial 7 outperformed other trials with:

- 2823 episodes required to exceed the goal score, with a 100-episode MA of evaluation score 0.5151 at stop during training
- 100-episode MA of the final evaluation performance (post training) of **1.742.**

trial_id	seed	cause_of_early_stopping	total_nbr_of_train_episodes	mean_100_eval_at_stop	final_eval_performance_mean_over_100_episodes	hidden_dims	optimizer	lr	buffer_size	batch_size	exploration_noise_ratio	n_warmup_batches	target_update_every	tau
trial_001	29	goal	2288	0.5088	1.3617	(256, 256)	Adam	0.0005	50000	500	0.05	5	5	0.01
trial_002	29	episodes	3000		0.0447	(64, 64)	Adam	0.0005	50000	500	0.1	5	10	0.01
trial_003	29	goal	1765	0.5101	0.6197	(256, 256)	Adam	0.0005	50000	500	0.1	5	5	0.01
trial_004	29	episodes	3000		0	(128, 128)	Adam	0.0001	50000	128	0.1	10	10	0.005
trial_005	29	episodes	3000		0.2579	(64, 64)	Adam	0.0005	50000	128	0.1	5	5	0.005
trial_006	29	episodes	3000		0	(64, 64)	Adam	0.0003	50000	256	0.05	5	10	0.01
trial_007	29	goal	2823	0.5151	1.742	(64, 64)	Adam	0.0003	50000	64	0.2	10	10	0.01
trial_008	29	goal	2944	0.5179	0.5417	(64, 64)	Adam	0.0001	50000	500	0.2	5	5	0.01
trial_009	29	episodes	3000		0.0203	(256, 256)	Adam	0.0005	50000	500	0.05	5	2	0.001
trial_010	29	episodes	3000		0.0318	(256, 256)	Adam	0.0005	50000	128	0.05	10	2	0.001

Best hyperparameters found:

- Hidden layer's dimensions: (64, 64),
- optimizer: Adam,
- learning rate : 3e-4,
- buffer_size: 50000,
- batch_size: 64,
- exploration_noise_ratio: 0.2,
- n_warmup_batches: 10,
- target_update_every: 10,
- tau: 0.01.

2) Solved environment

a) Best model over seeds

Training across 10 seeds with the best hyperparameters found that seed 41 outperformed with the least episodes:

- **1136** episodes required to exceed the goal score, with a 100-episode MA of evaluation score of 0.5119 at stop during training.
- 100-episode MA of the final evaluation performance (post training) of **1.54±1.24**

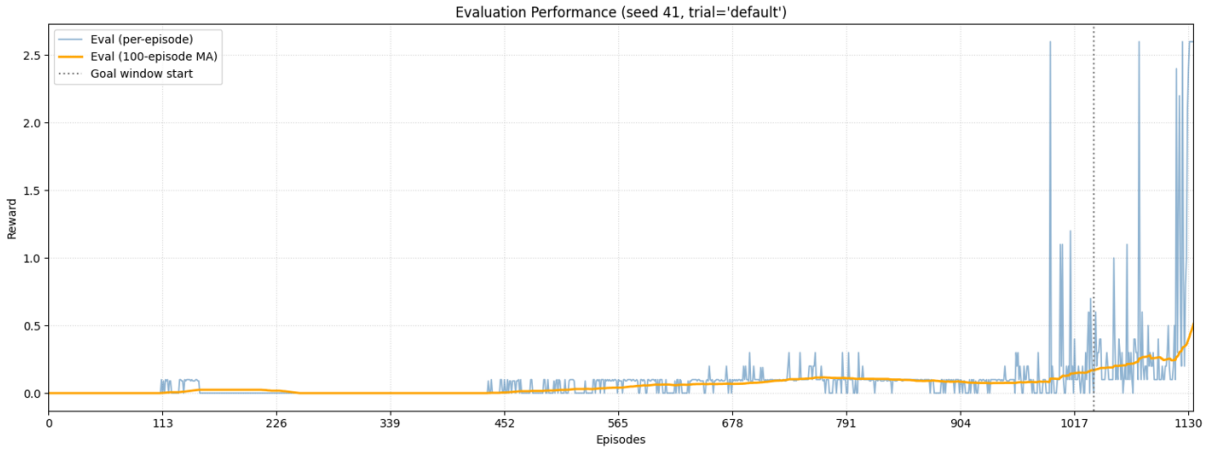
It can be concluded that the current experiment surpasses Udacity's reference (1136 < 1500 episodes).

trial_id	seed	cause_of_early_stopping	total_nbr_of_train_episodes	mean_100_eval_at_stop	final_eval_performance_mean_over_100_episodes ± std
best_hparameters	33	episodes	3000		0.00±0.00
best_hparameters	41	goal	1136	0.5119	1.54±1.24
best_hparameters	66	goal	2562	0.5026	0.33±0.56
best_hparameters	39	goal	2615	0.5234	2.07±1.06
best_hparameters	8	goal	2338	0.5066	2.55±0.51
best_hparameters	77	episodes	3000		0.09±0.02
best_hparameters	21	episodes	3000		0.40±0.44
best_hparameters	20	episodes	3000		0.00±0.01
best_hparameters	44	goal	2608	0.5068	2.60±0.35
best_hparameters	22	goal	2727	0.504	1.88±1.16

The weights of the best online policy model are then extracted using the script extract_weights_n_load_into_policy.py.

b) Plot of performance of during training

The below plot shows that the 100-episode MA score (blue curve) starts increasing very fast from episode 1000 before reaching the goal in episode 1136. During this period, a very high variance is observed in the episodic score, reaching a high score around +2.6.



The extracted weights and plot can be found in ./ best_model_and_weights_and_plot/ folder.

./ best_model_and_weights_and_plot/

online_policy_model.1135.tar
online_policy_model.1135.weights.npz
online_policy_model.1135.weights.pt

c) Bootstrap CI result

Regarding the statistical test with bootstrap 95% CI across all seeds , it was found that **the bootstrap mean is significantly above the goal score of +0.5 at 95%,** as shown below:

- list of 10 final seed-based eval scores: [0.0, 1.541, 0.332, 2.067, 2.552, 0.093, 0.403, 0.002, 2.597, 1.877],
- bootstrap statistics : Mean=1.097, Std=0.469,
- bootstrap 95% CI for mean : [0.518, 1.794]

This means that **the models created with the best hyper-parameters found, were able to significantly outperform the threshold score of +0.5.**

4. Suggestions for future work

1) Replace simple Gaussian exploration with adaptive exploration

In the current experiment, the Gaussian noise is blind. Performance could be improved with an **adaptative exploration** to avoid premature exploration or over-exploration in sensitive regions. For example, we could make the noise increase slightly when the 100-episodes MA evaluation score plateaus, or decay it when the score improves.

2) Make the team reward adaptive

The team reward currently integrates a fixed **team lambda of 0.5** at each step.

$$r_t^{team} = r_t^{(1)} + r_t^{(2)} - \lambda\, d_t$$

d_t=1 if the point ends at time t. λ controls how painful termination is.

With a fixed λ (0.5), the performance increases in early training as the agent aims to prevent failures. But in late training, the agent becomes **conservative**.

By making λ adaptive, decreasing with time (e.g. from 0.5 down to 0.05), the agent could shift towards **performance optimisation in late training**.

3) Use double critic

The current critic is currently single-headed (two outputs, but one network). By introducing 2 critics and a TD-target using the min of both values , we could massively reduce **overestimation bias** and improve stability.

$$Q_{\phi_1}, \quad Q_{\phi_2}$$

$$y = r + \gamma \min(Q_{\phi_1^-}, Q_{\phi_2^-})$$