AMRITA VISHWA VIDYAPEETHAM

21AIE401
DEEP REINFORCEMENT LEARNING

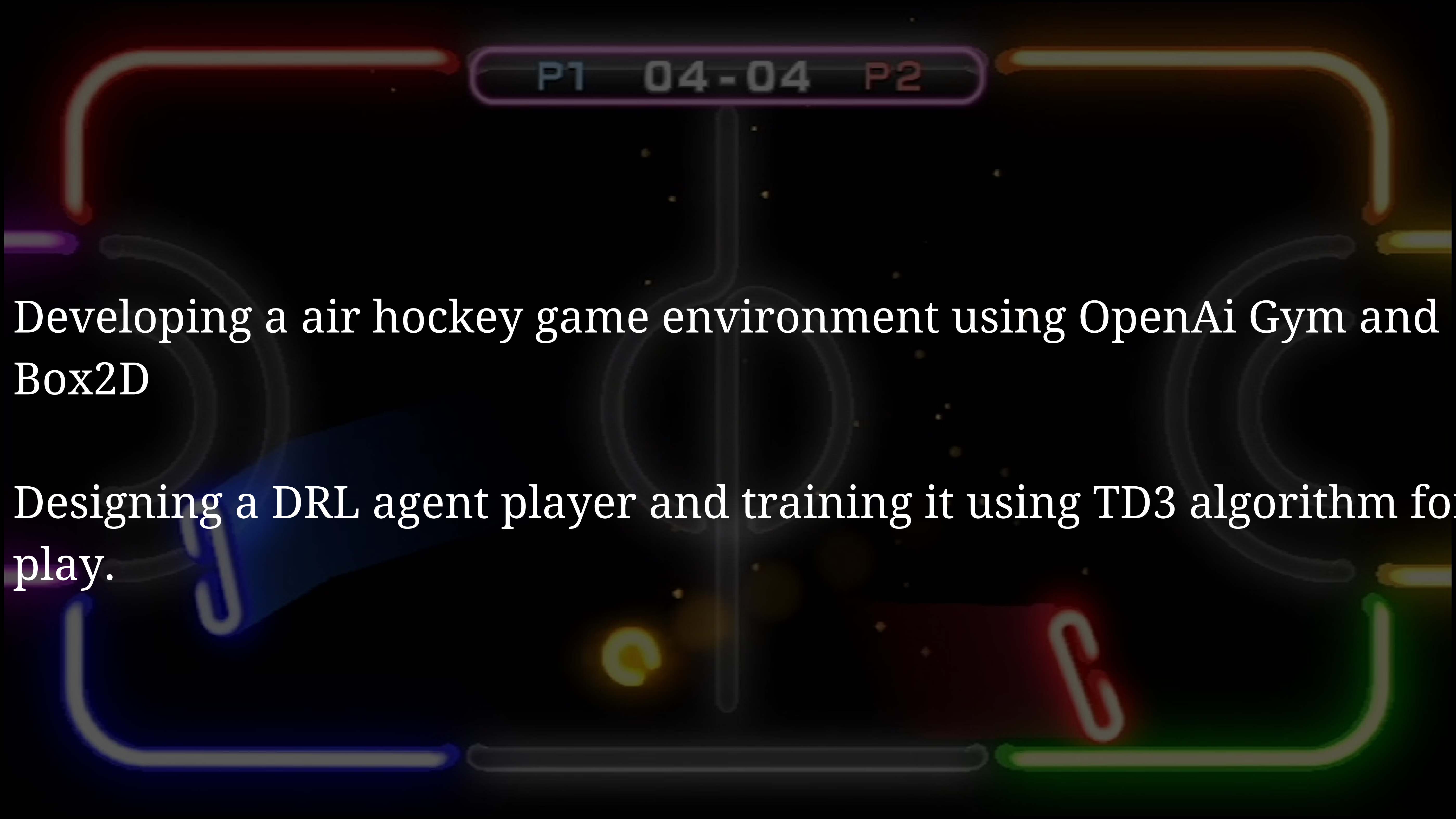
DEVELOPMENT OF AIR HOCKEY GAME USING DEEP REINFORCEMENT LEARNING

GUIDED BY:
Dr. Amudha J
Professor

TEAM INFINITY

Apoorva M	BL.EN.U4AIE19007
Tanuj M	BL.EN.U4AIE19041
Aishwarya V	BL.EN.U4AIE19068

PROBLEM STATEMENT



- Developing a air hockey game environment using OpenAi Gym and Box2D
- Designing a DRL agent player and training it using TD3 algorithm for self play.

LITERATURE SURVEY

- Research has been conducted and published on the application of deep reinforcement learning to the sports of table tennis [6, badminton [7], sword fighting [8], and air hockey [9].

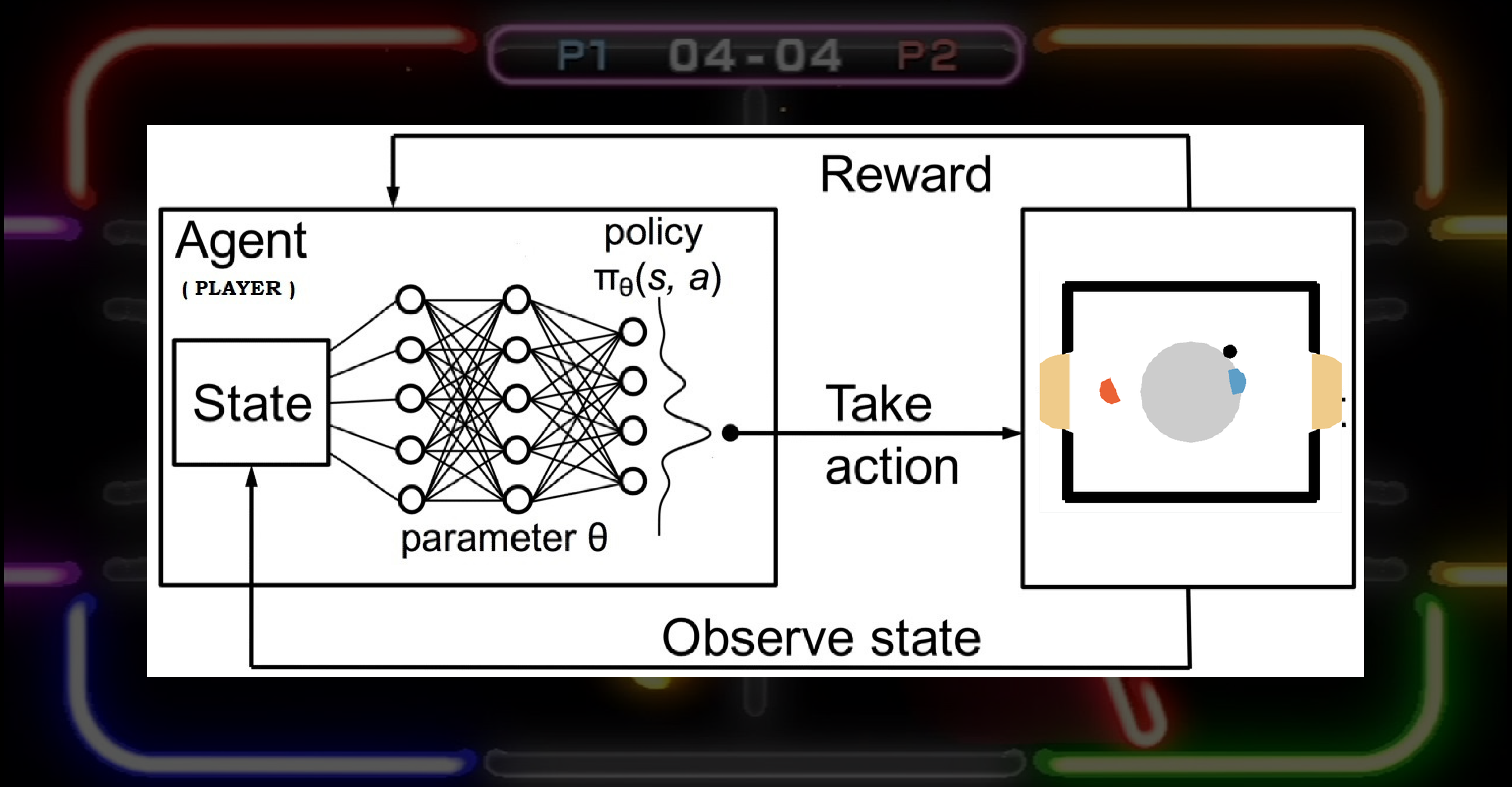
- [2] Solves the laser-hockey gym environment with Reinforcement Learning (RL) using the Twin Delayed Deep Deterministicpolicy gradient algorithm (TD3) .

- [4] illustrates how to create a policy for all purpose robotic manipulators for the game of air hockey using two Kuka Iiwa 14.

GAPS

- In the previous works, there in no much evidence of the usage of Replay buffers.
- In this project we'll be incorporating priorotized replay buffer and observation normalization

RL FORMULATION



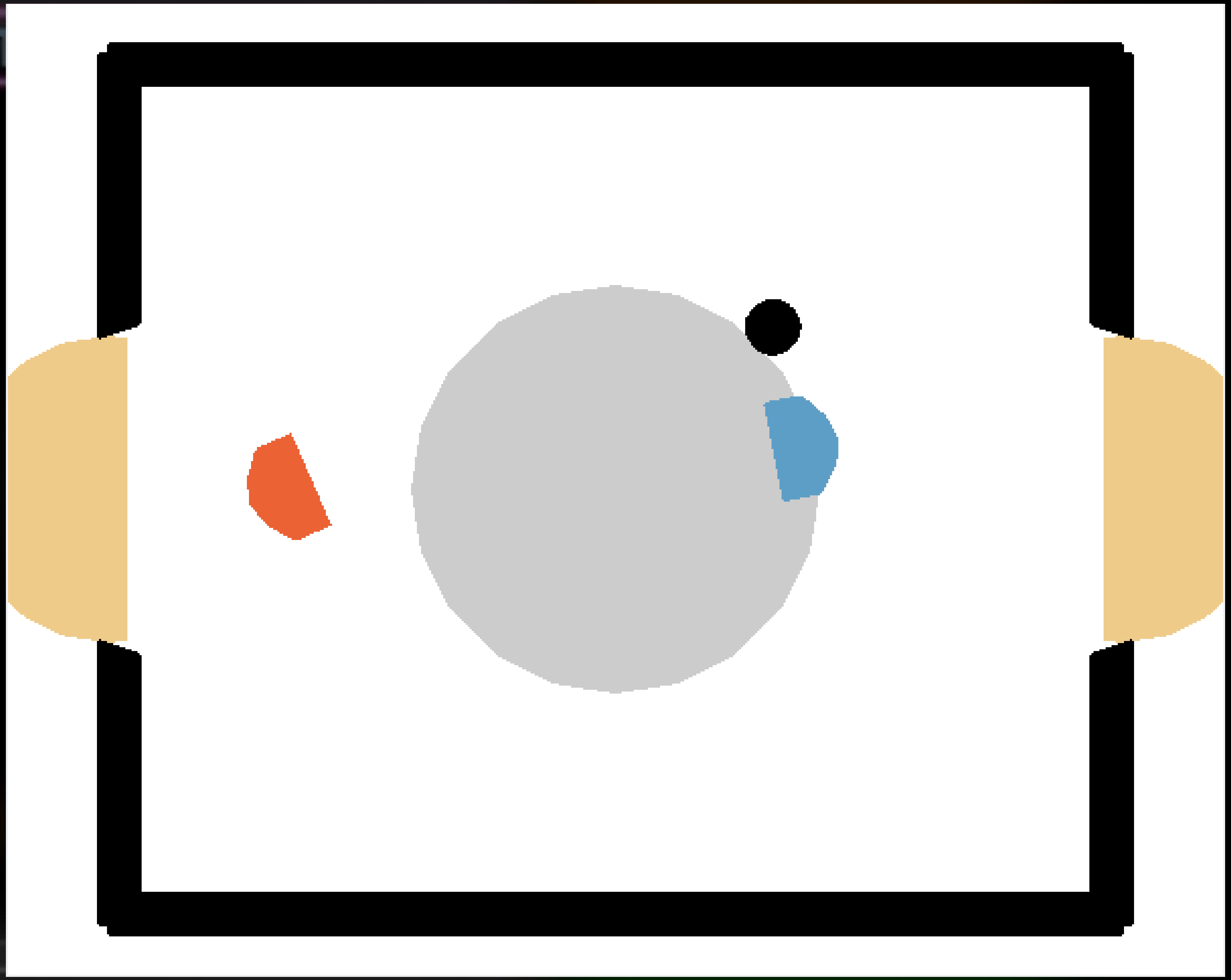
RL FORMULATION

- **Agent :** Air Hockey Player
- **Environment** : Hockey environment created using Gym and Box2D
- **Policy**: Agent scoring maximum goals.
- **State Space** : 18 states
- **Action Space** : 8
- **Rewards** : 10 for win, -10 for lose, 0 for tie
- **Priorotized replay buffer** : Stores intermediate-state,action reward and next according to priority by assigning weights.
- **Terminal state** : when game ends after 100 episodes (default)

AIR HOCKEY ENVIRONMENT DEVELOPMENT

OpenAI Gym

Box2D

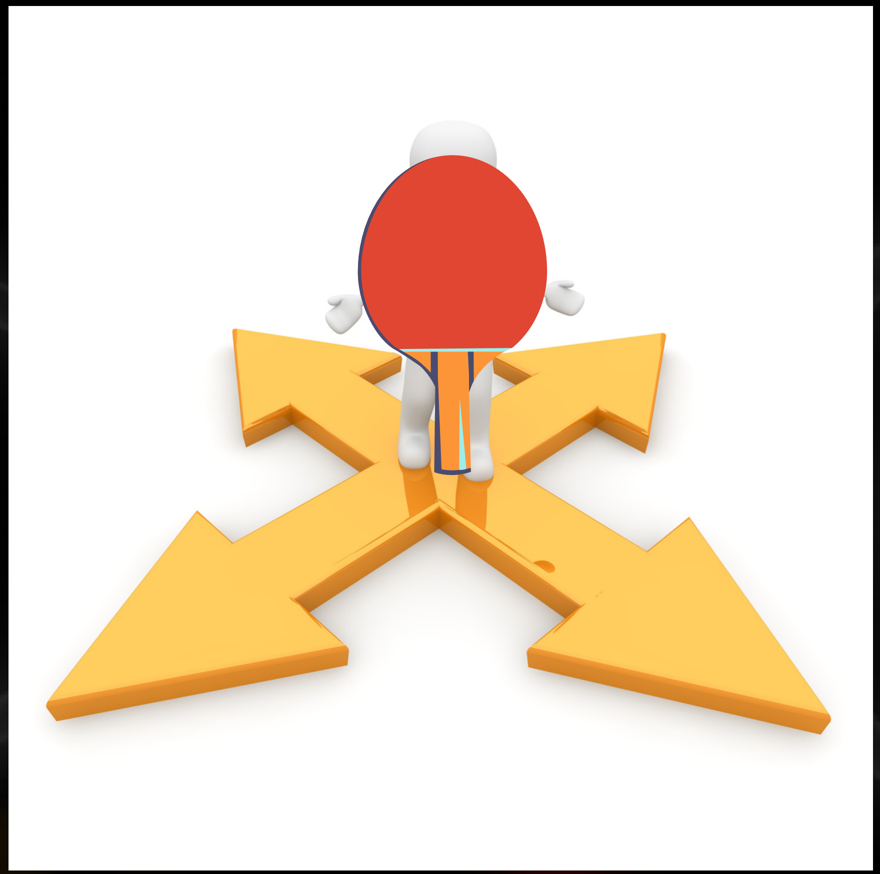


STATE SPACE

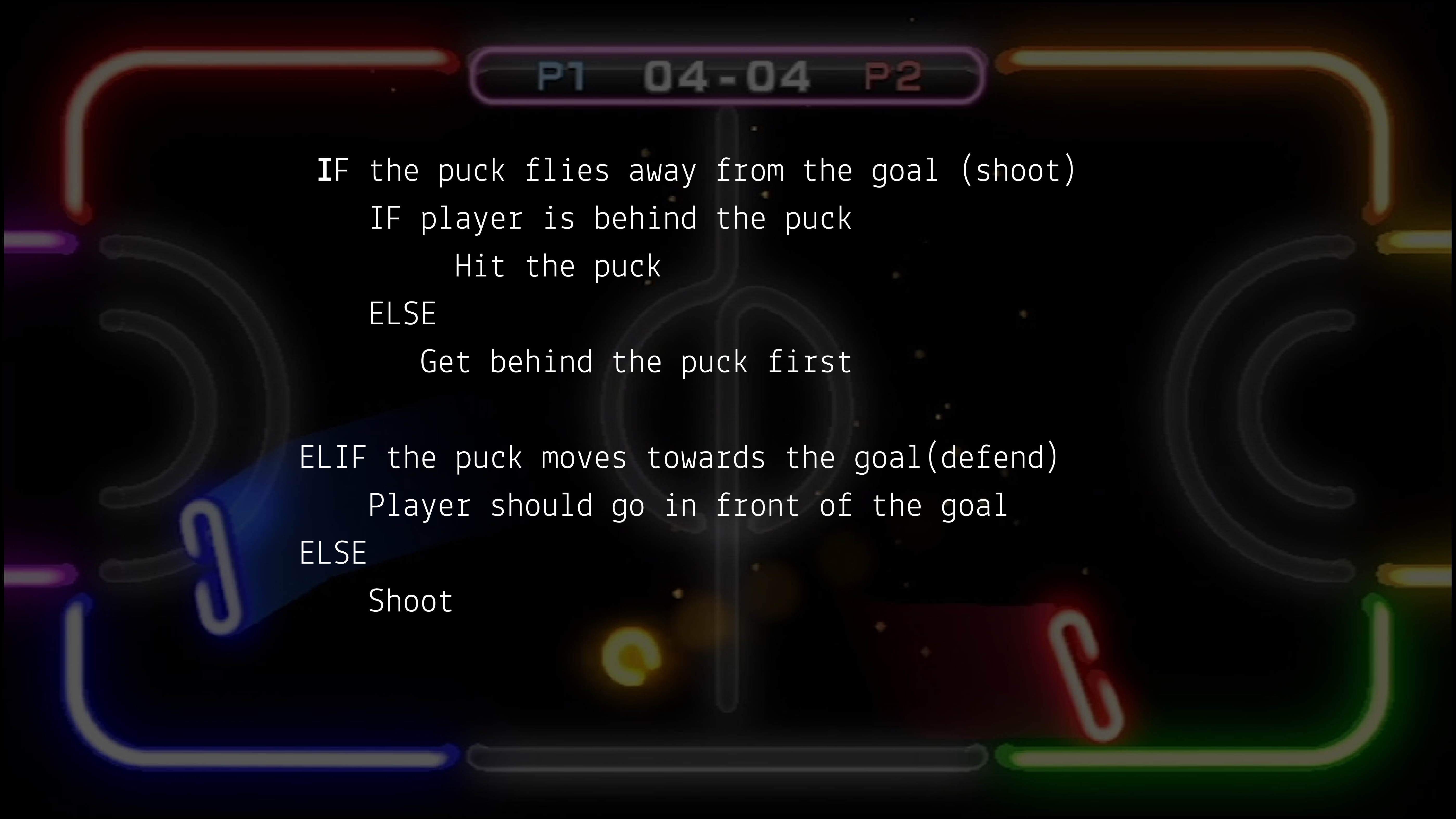
- # 0 x pos player 1
- # 1 y pos player 1
- # 2 Angle player 1
- # 3 x vel player 1
- # 4 y vel player 1
- # 5 Angular vel player 1
- # 6 x pos player 2
- # 7 y pos player 2
- # 8 Angle player 2
- # 9 x vel player 2
- # 10 y vel player 2
- # 11 Angular vel player 2
- # 12 x pos puck
- # 13 y pos puck
- # 14 x vel puck
- # 15 y vel puck
- # 16 time left player has puck
- # 17 time left other player has puck

ACTION SPACE

- UP
- DOWN
- RIGHT
- LEFT
- CLOCKWISE ANGLE
- ANTI CLOCKWISE ANGLE
- SHOOT
- IDLE



BASIC WORKING OF THE DESIGNED OPPONENT




```
IF the puck flies away from the goal (shoot)
    IF player is behind the puck
         Hit the puck
    ELSE
       Get behind the puck first

ELIF the puck moves towards the goal(defend)
    Player should go in front of the goal
ELSE
    Shoot
```

TD3 ARCHITECTURE

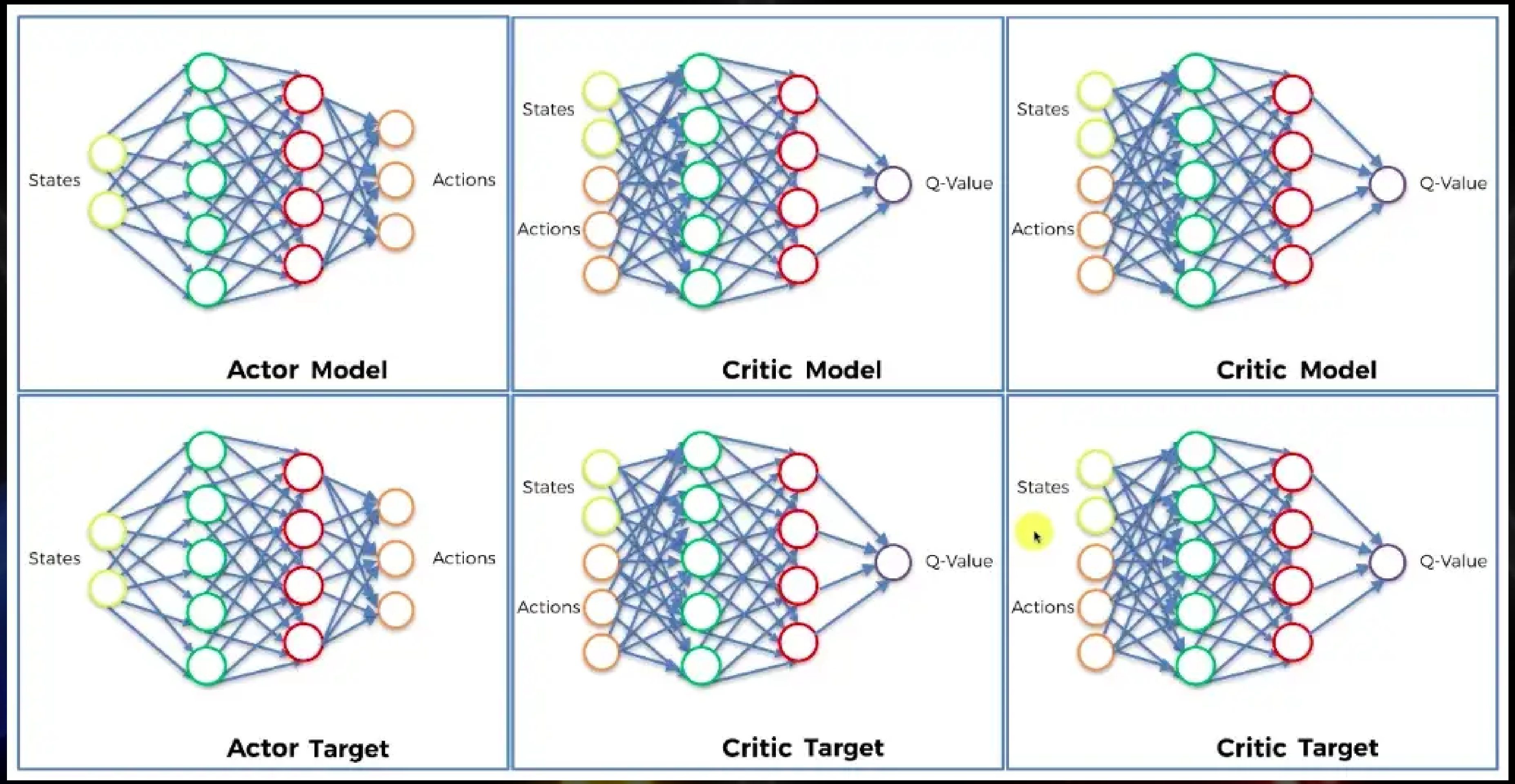
- Twin Deep Deterministic Policy Gradient (TD3) is the successor to the Deep Deterministic Policy Gradient (DDPG)

- DDPG can be unstable and heavily reliant on finding the correct hyper parameters - over estimating the Q values of the critic (value) network.

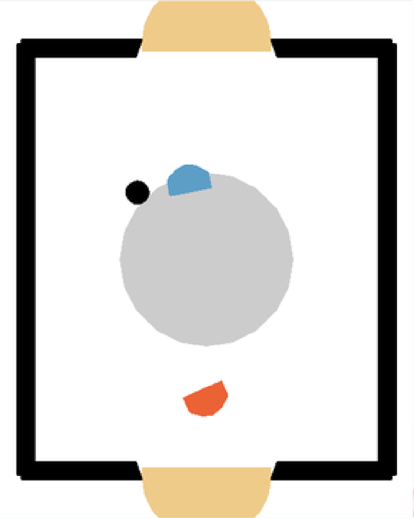
- TD3 solves the drawbacks of DDPG.
    a. Using a pair of critic networks
    b. Delayed updates of the actor
    c. Action noise regularisation

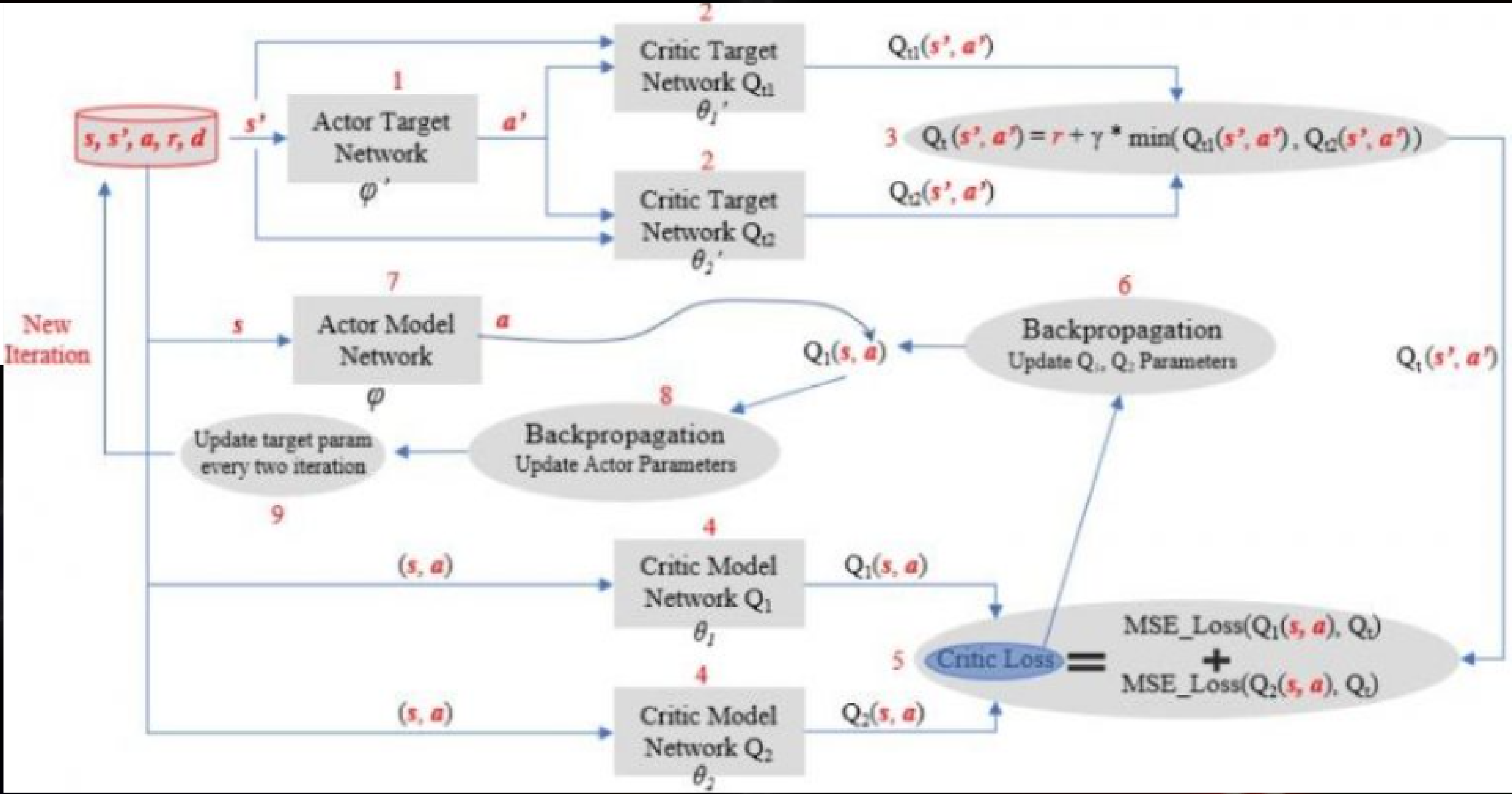
- Off policy algorithm
- Used for environments with Continuous Action Spaces

TD3 ARCHITECTURE



SYSTEM ARCHITECTURE





TD3 ALGORITHM

Algorithm 1 TD3

1 Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$

2 Initialize replay buffer $\mathcal{B}$

for $t = 1$ **to** T **do**

3 Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'

4 Store transition tuple (s, a, r, s') in $\mathcal{B}$

5 Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
$\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
$y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$

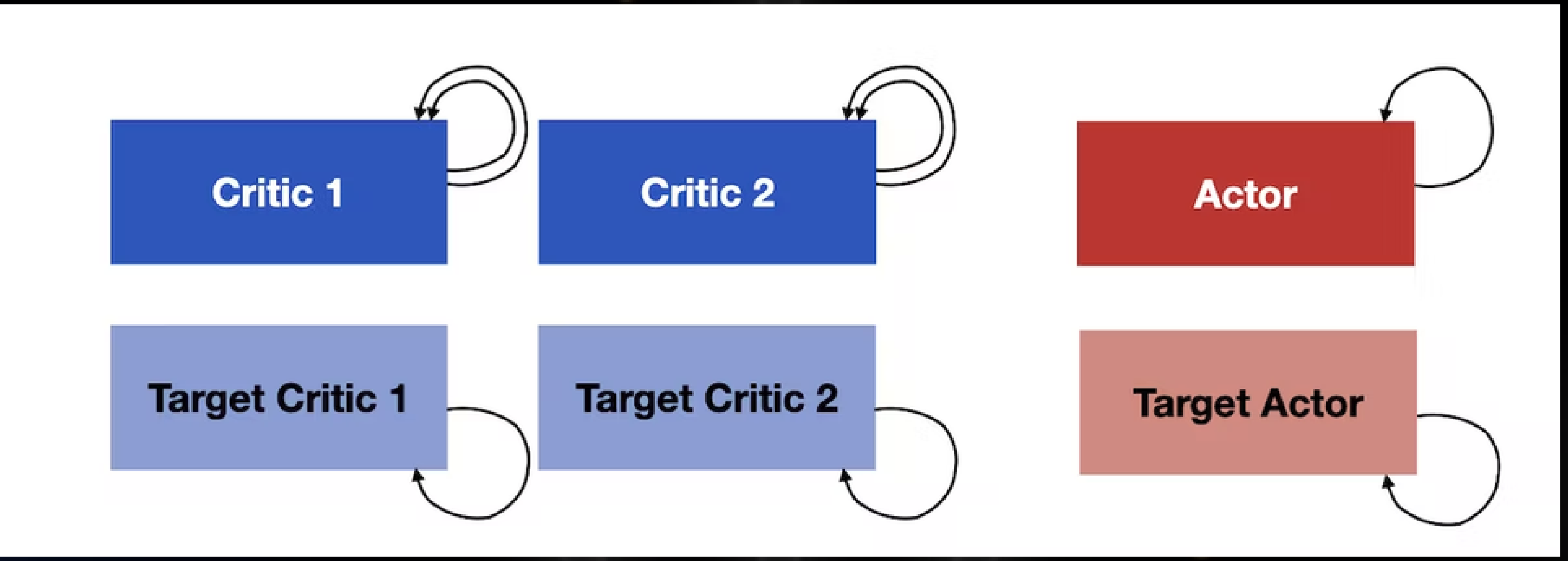
6 **if** t mod d **then**
Update ϕ by the deterministic policy gradient:
$\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$

7 Update target networks:
$\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
$\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$

end if
end for

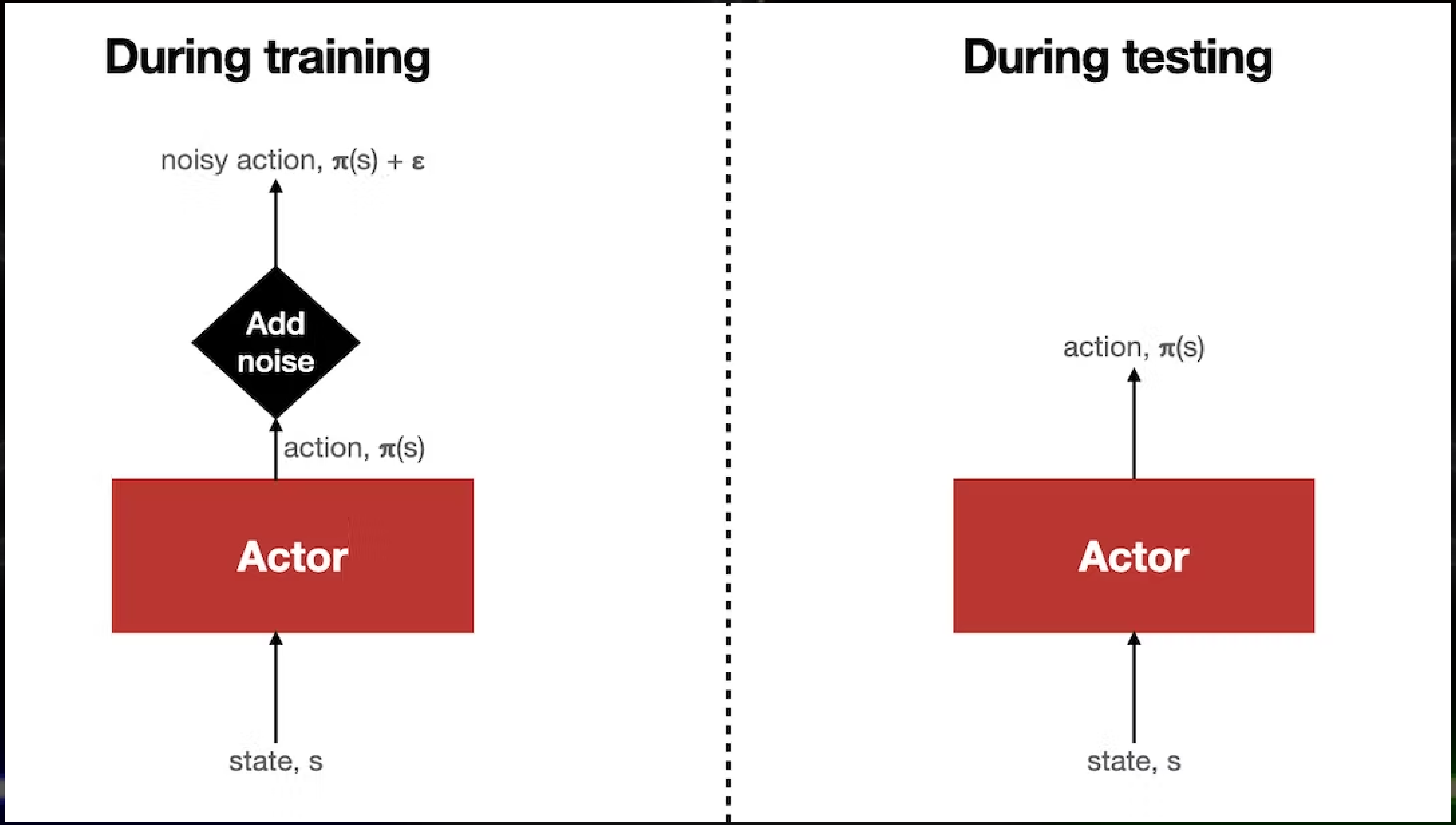
1. Initialise networks
2. Initialise replay buffer
3. Select and carry out action with exploration noise
4. Store transitions
5. Update critic
6. Update actor
7. Update target networks
8. Repeat until sentient

Delayed Policy and Target Updates



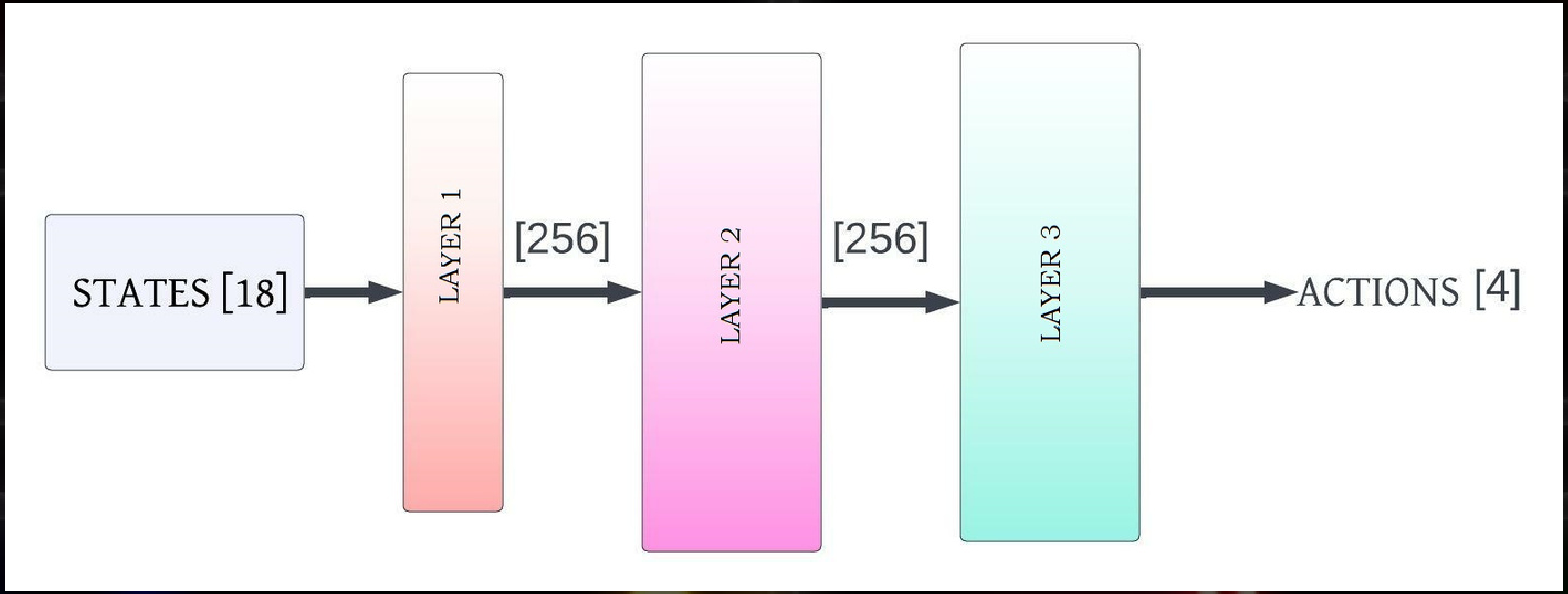
The two critic networks are more frequently updated than the actor and the three target networks. The critics are updated at every step, while the actor and the targets are updated every second step.

Exploration by the TD3 Agent

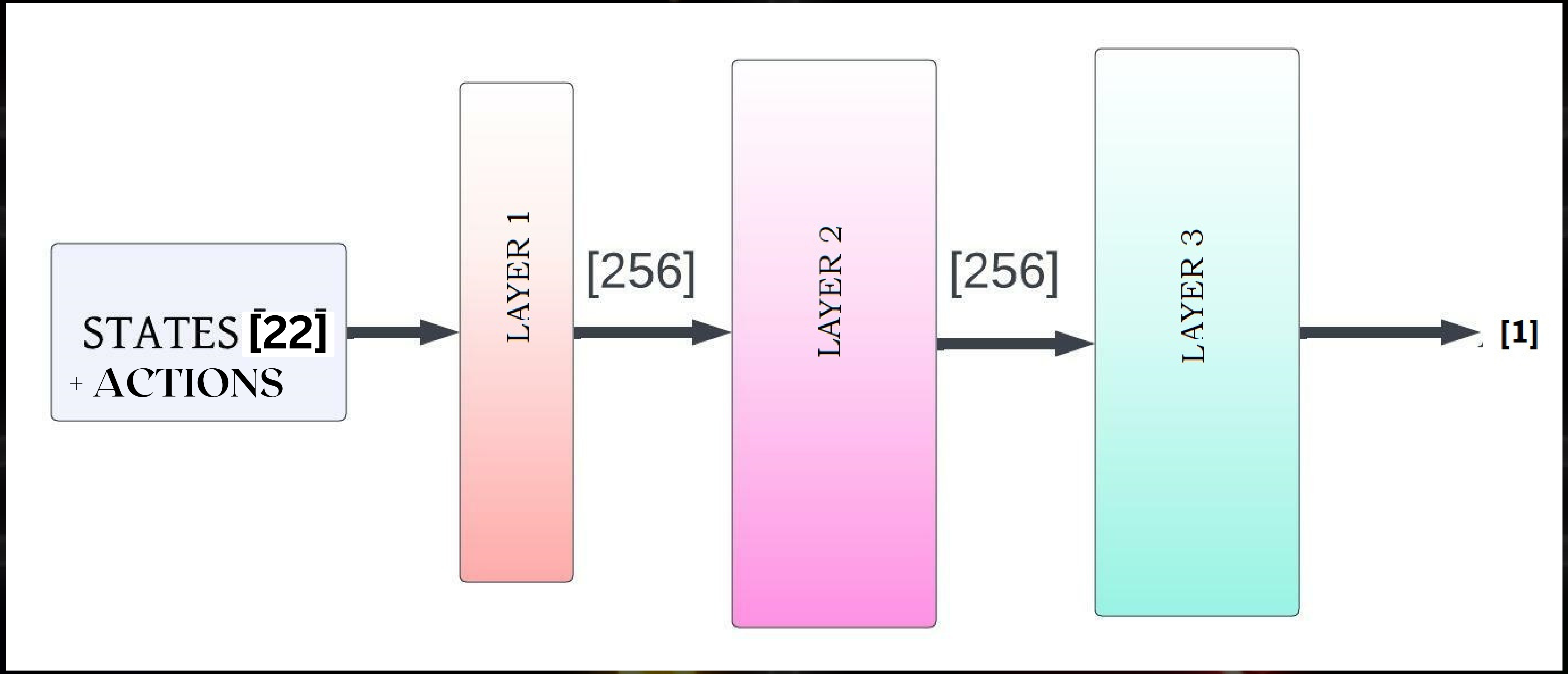


Action prediction during Training vs Testing phase in TD3

ACTOR NETWORK



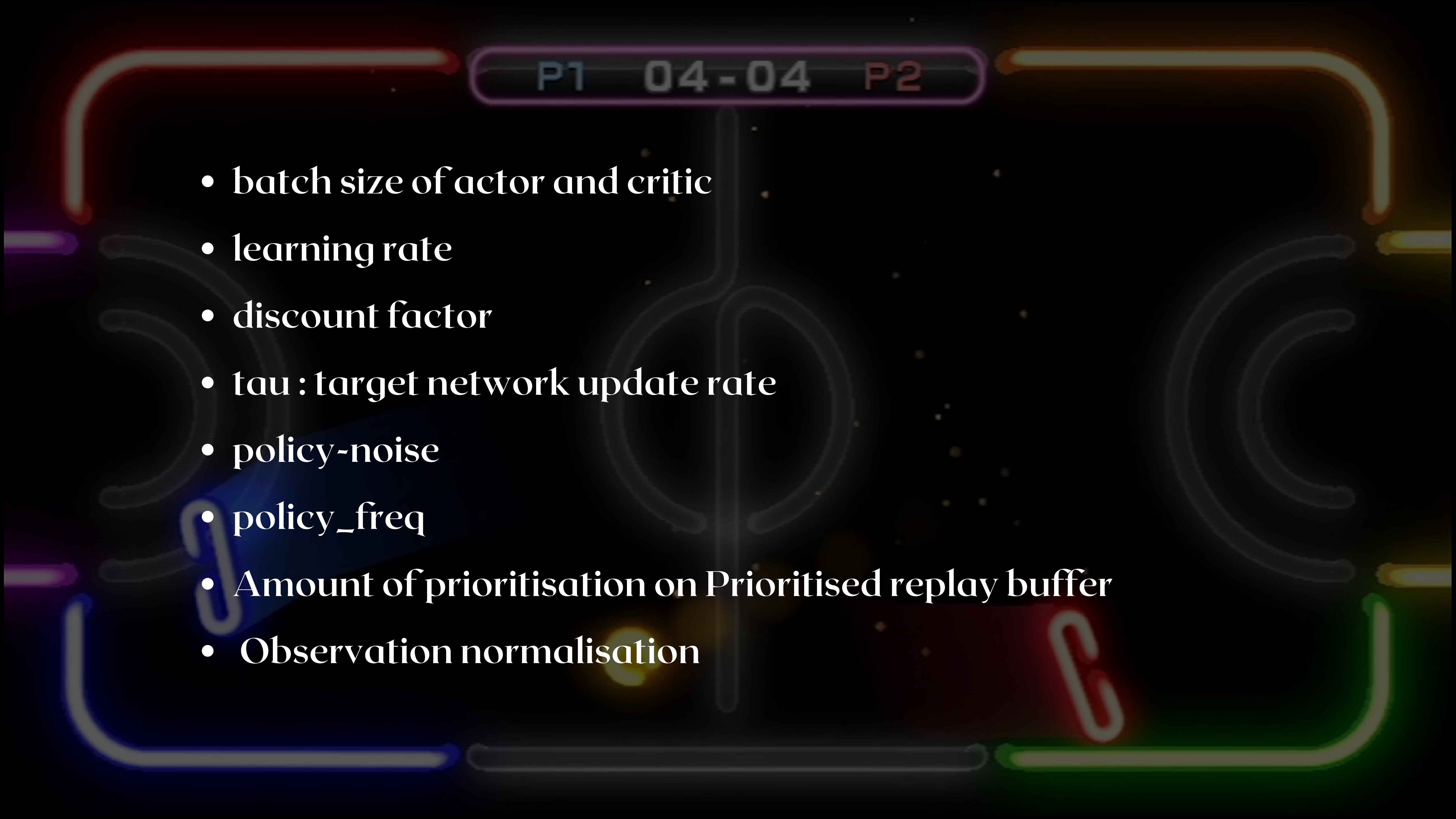
CRITIC NETWORK



PRIOROTIZED REPLAY BUFFER

- When we sample experiences to feed the Neural Network, we assume that some experiences are more valuable than others.
- If we sample with weights, we can make it so that some experiences which are more beneficial get sampled more times on average
- For Prioritized Experience Replay, we do need to associate every experience with additional information, its priority, probability and weight.
- The priority is updated according to the loss obtained after the forward pass of the neural network.
- The probability is computed out of the experiences priorities, while the weight (correcting the bias introduced by not uniformly sampling during the neural network backward pass) is computed out of the probabilities

HYPERPARAMETERS



- batch size of actor and critic
- learning rate
- discount factor
- tau : target network update rate
- policy-noise
- policy_freq
- Amount of prioritisation on Prioritised replay buffer
- Observation normalisation

CODE SNIPPET

```
parser.add_argument("--policy", default="TD3", help='Policy name (TD3)')
parser.add_argument("--env", default="Hockey-v0_NORMAL", help='Gym environment name')
parser.add_argument("--trial", default=0, type=int, help='Trial number')
parser.add_argument("--seed", default=42, type=int, help='Sets Gym, PyTorch and Numpy seeds')
parser.add_argument("--start_timesteps", default=5e4, type=int, help='Time steps initial random policy is used')
parser.add_argument("--eval_freq", default=5e3, type=int, help='How often (time steps) it will be evaluated')
parser.add_argument("--self_play_freq", default=15e5, type=int, help='Add current agent to list of opponents')
parser.add_argument("--max_timesteps", default=1e6, type=int, help='Max time steps to run environment')
parser.add_argument("--max_episode_timesteps", default=500, type=int, help='Max time steps per episode')
parser.add_argument("--max_buffer_size", default=1e6, type=int, help='Size of the replay buffer')
parser.add_argument("--expl_noise", default=0.15, type=float, help='Std of Gaussian exploration noise')
parser.add_argument("--hidden_dim", default=256, type=int, help='Hidden dim of actor and critic nets')
parser.add_argument("--batch_size", default=256, type=int, help='Batch size for both actor and critic')
parser.add_argument("--learning_rate", default=3e-4, type=float, help='Learning rate')
parser.add_argument("--discount", default=0.99, type=float, help='Discount factor')
parser.add_argument("--tau", default=0.01, type=float, help='Target network update rate')
parser.add_argument("--policy_noise", default=0.1, type=float,
                    help='Noise added to target policy during critic update')
parser.add_argument("--noise_clip", default=0.5, type=float, help='Range to clip target policy noise')
parser.add_argument("--policy_freq", default=2, type=int, help='Frequency of delayed policy updates')
parser.add_argument("--prioritized_replay", action="store_true", help='Use prioritized experience replay')
parser.add_argument("--alpha", default=0.6, type=float, help='Amount of prioritization in PER')
parser.add_argument("--beta", default=1.0, type=float, help='Amount of importance sampling in PER')
parser.add_argument("--beta_schedule", default="", help='Annealing schedule for beta in PER')
parser.add_argument("--normalize_obs", action="store_true", help='Use observation normalisation')
parser.add_argument("--only_win_reward", action="store_true", help='Rewards only wins')
parser.add_argument("--early_stopping", action="store_true", help='Use early stopping')
parser.add_argument("--load_model", default="",
                    help='Model load file name, \"\" does not load')
```

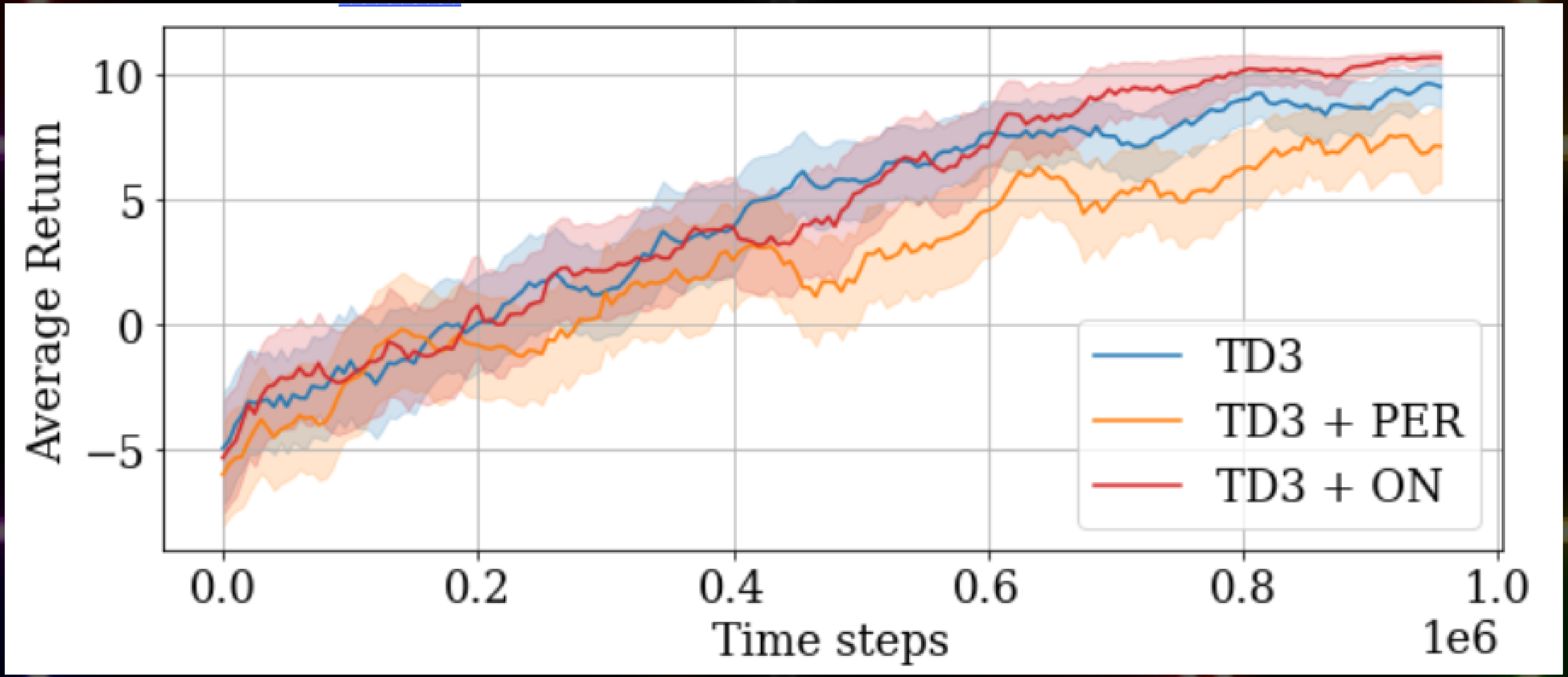
CODE SNIPPET

```python
class Actor(nn.Module):
    def __init__(self, state_dim, action_dim, hidden_dim, max_action):
        super(Actor, self).__init__()

        self.l1 = nn.Linear(state_dim, hidden_dim)
        self.l2 = nn.Linear(hidden_dim, hidden_dim)
        self.l3 = nn.Linear(hidden_dim, action_dim)

        self.max_action = max_action

    def forward(self, state):
        a = F.relu(self.l1(state))
        a = F.relu(self.l2(a))
        return self.max_action * torch.tanh(self.l3(a))
```

```python
class Critic(nn.Module):
    def __init__(self, state_dim, action_dim, hidden_dim):
        super(Critic, self).__init__()

        # Q1 architecture
        self.l1 = nn.Linear(state_dim + action_dim, hidden_dim)
        self.l2 = nn.Linear(hidden_dim, hidden_dim)
        self.l3 = nn.Linear(hidden_dim, 1)

        # Q2 architecture
        self.l4 = nn.Linear(state_dim + action_dim, hidden_dim)
        self.l5 = nn.Linear(hidden_dim, hidden_dim)
        self.l6 = nn.Linear(hidden_dim, 1)

    def forward(self, state, action):
        sa = torch.cat([state, action], 1)

        q1 = F.relu(self.l1(sa))
        q1 = F.relu(self.l2(q1))
        q1 = self.l3(q1)

        q2 = F.relu(self.l4(sa))
        q2 = F.relu(self.l5(q2))
        q2 = self.l6(q2)

        return q1, q2

    def Q1(self, state, action):
        sa = torch.cat([state, action], 1)

        q1 = F.relu(self.l1(sa))
        q1 = F.relu(self.l2(q1))
        q1 = self.l3(q1)

        return q1
```

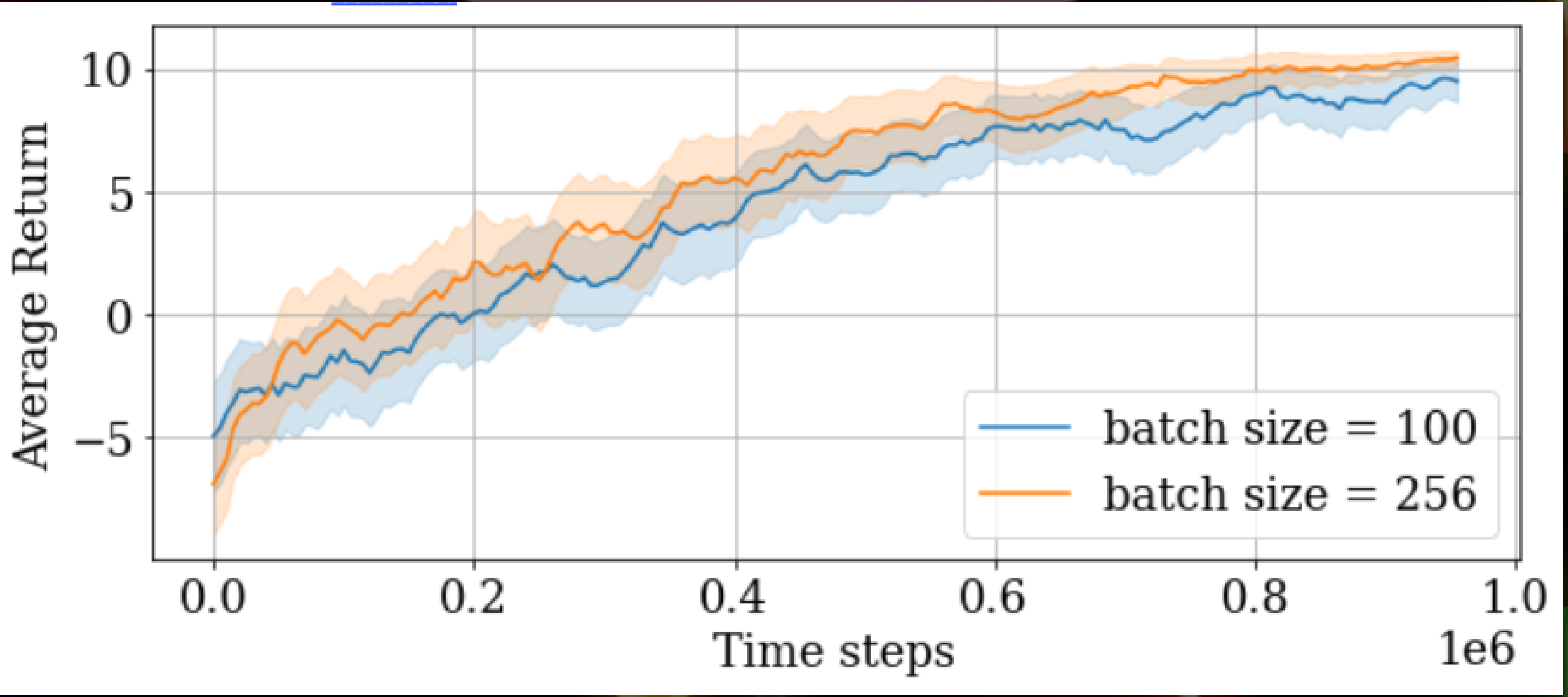
RESULTS

Average episode return in the evaluation during training



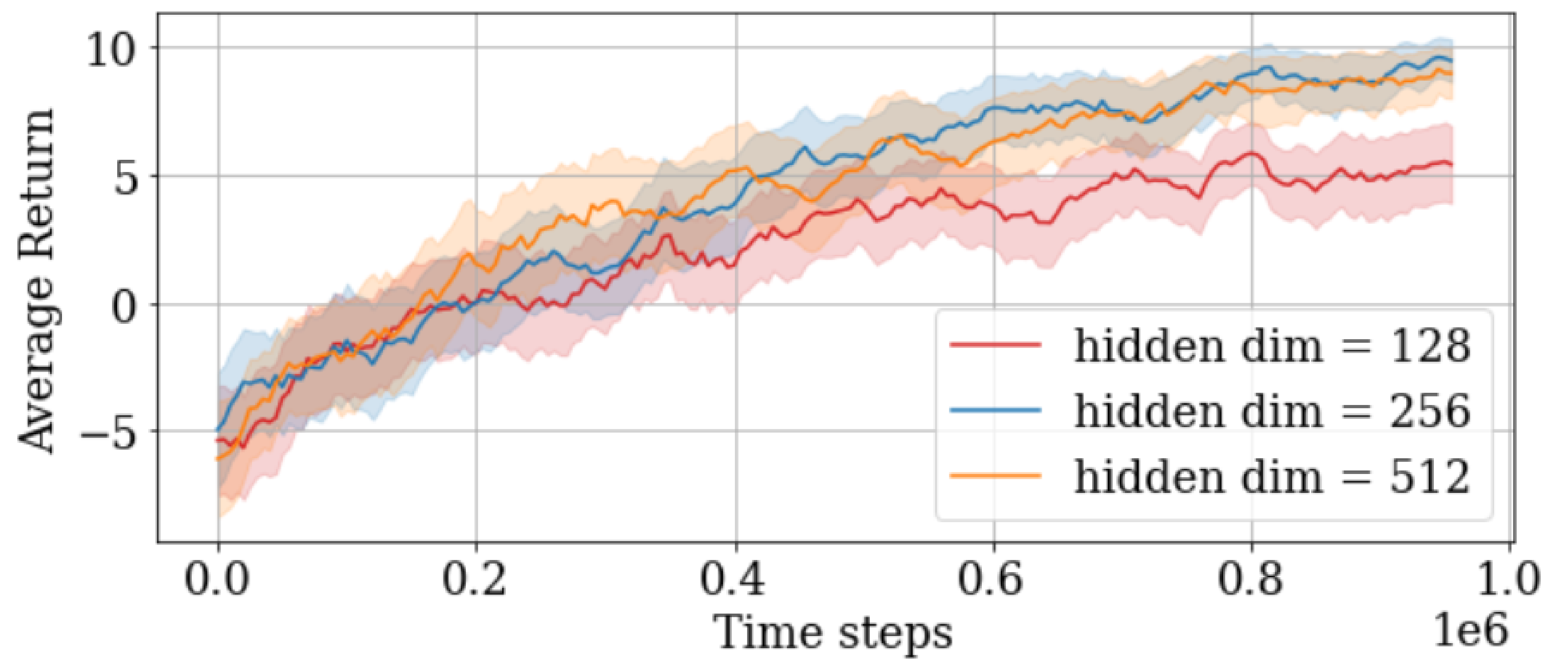

```
TD3: Win-rate = 0.87, Tie-rate = 0.12, Loss-rate = 0.01
TD3 + PER: Win-rate = 0.54, Tie-rate = 0.41, Loss-rate = 0.05
TD3 + ON: Win-rate = 0.94, Tie-rate = 0.04, Loss-rate = 0.02
```

Average episode return in the evaluation during training with different batch sizes



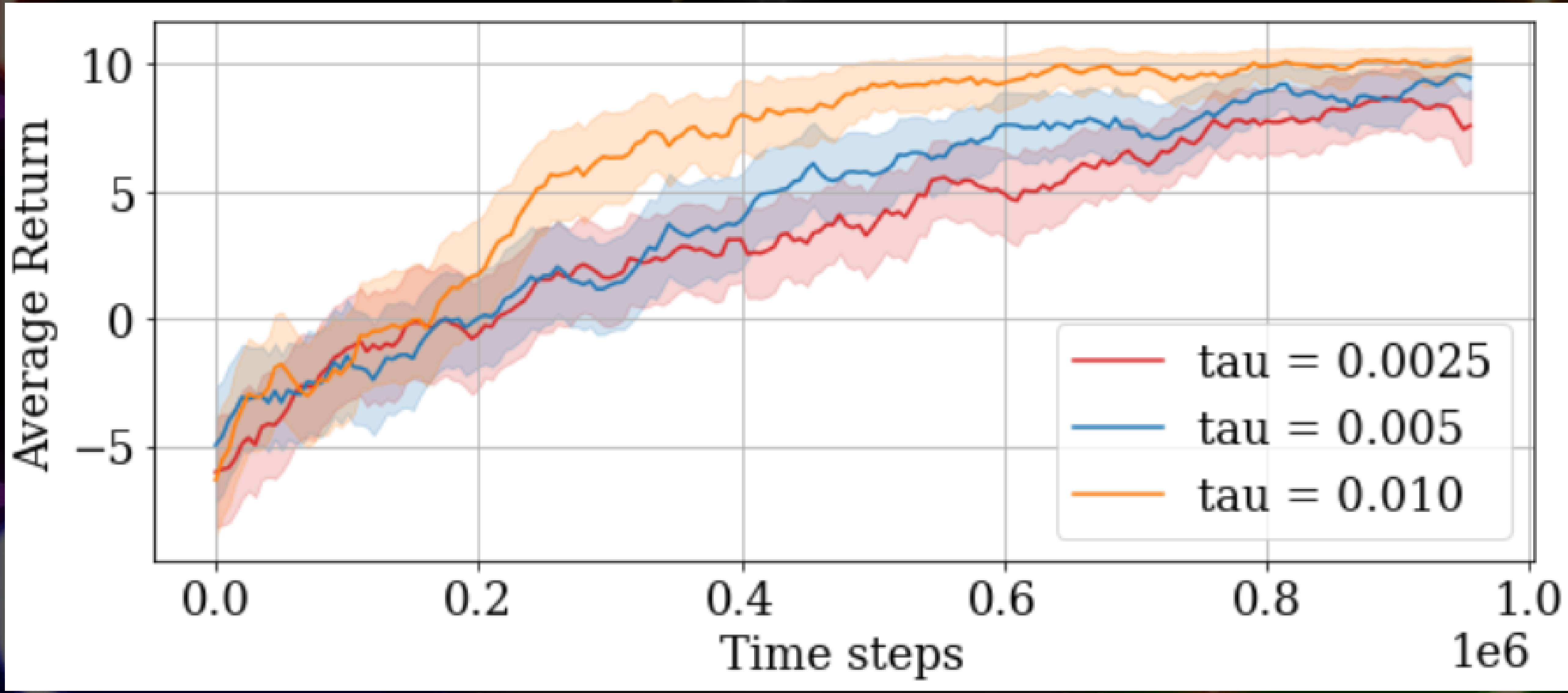
batch size = 100: Win-rate = 0.87, Tie-rate = 0.12, Loss-rate = 0.01
batch size = 256: Win-rate = 0.99, Tie-rate = 0.01, Loss-rate = 0.00

Average episode return in the evaluation during training with hidden dim



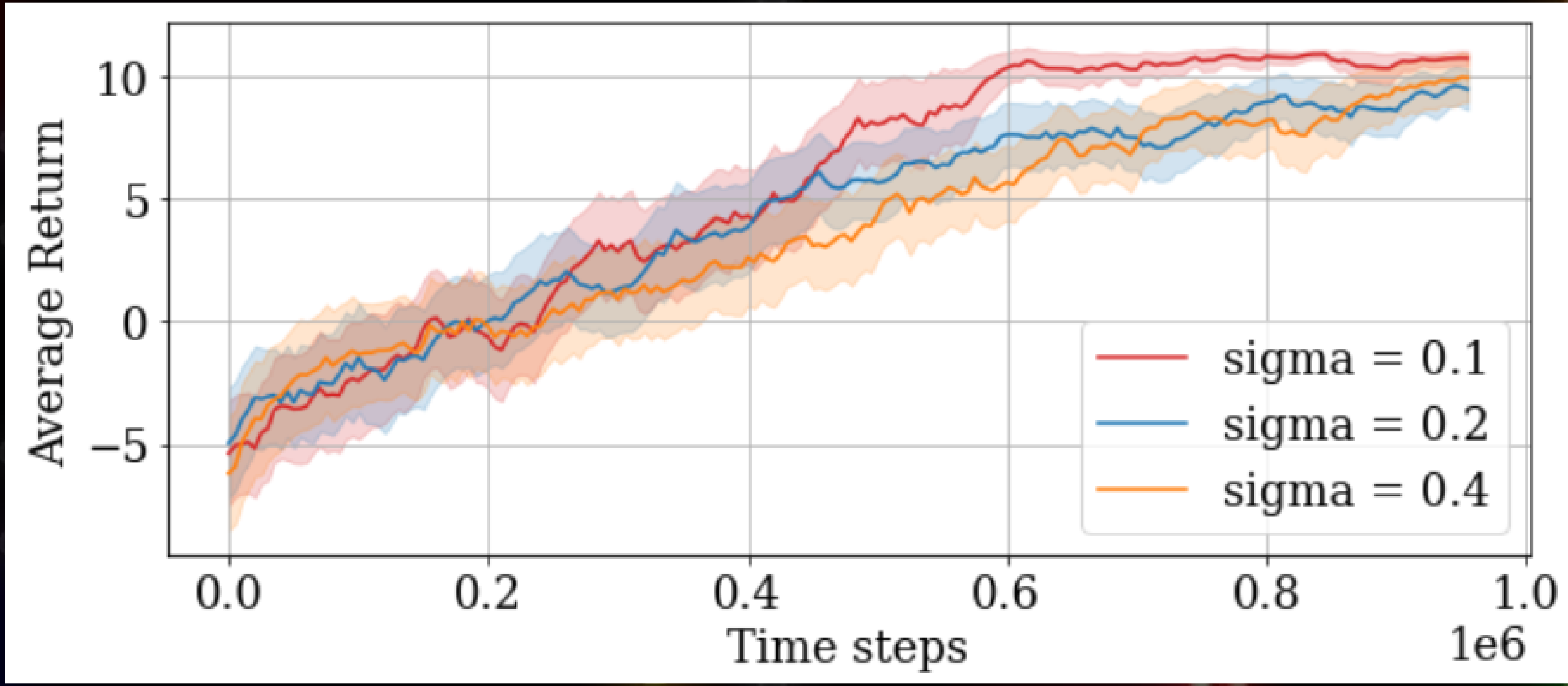

```
hidden dim = 128: Win-rate = 0.40, Tie-rate = 0.53, Loss-rate = 0.07
hidden dim = 256: Win-rate = 0.87, Tie-rate = 0.12, Loss-rate = 0.01
hidden dim = 512: Win-rate = 0.82, Tie-rate = 0.17, Loss-rate = 0.01
```

Average episode return in the evaluation during training with different values of tau (target update proportion)




```
tau = 0.0025: Win-rate = 0.52, Tie-rate = 0.45, Loss-rate = 0.03
tau = 0.005: Win-rate = 0.87, Tie-rate = 0.12, Loss-rate = 0.01
tau = 0.010: Win-rate = 0.96, Tie-rate = 0.03, Loss-rate = 0.01
```

Average episode return in the evaluation during training with different values of sigma (policy noise)




```
sigma = 0.1: Win-rate = 0.99, Tie-rate = 0.01, Loss-rate = 0.00
sigma = 0.2: Win-rate = 0.87, Tie-rate = 0.12, Loss-rate = 0.01
sigma = 0.4: Win-rate = 0.74, Tie-rate = 0.25, Loss-rate = 0.01
```

CONCLUSION

- In this project, Twin Delayed Deep Deterministic policy gradient algorithm (TD3) algorithm has been implemented which can be used to train agent to play the game Air Hockey against the programmed oponent in the laser-hockey gym environment. Therefore the agent plays the game and saves the experience into a replay memory which poses as a data set for training the neural network.

- In addition to TD3, prioritized experience replay (PER) and observation normalization (ON) have been implemented to analyze the effect of these modifications on TD3 in the laser-hockey environment.

- The results include the influence of certain hyperparameters, the influence of the modifications prioritized experience replay and observation normalization.

FUTURE SCOPE

- Various other algorithms could be incorporated adding optimization techniques.
- To train the DRL application, a mobile application that plays air hockey and allows the agent to choose the opponent.
- we can train the agent to play against oponents with different difficulty level
- This technique can be used to train and implement other Arcade games.

REFERENCE

[1] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor- critic methods. In International Conference on Machine Learning, pages 1587–1596. PMLR, 2018..

[2] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algo- rithms and applications. CoRR, abs/1812.05905, 2018.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[4] Wenbo Gao, Laura Graesser, Krzysztof Choromanski, Xingyou Song, Nevena Lazic, Pannag Sanketi, Vikas Sindhwani, Navdeep Jaitly. Robotic Table Tennis with Model-Free Reinforcement Learning, 2020 arXiv:2003.14398

[5] J. Tebbe, L. Krauch, Y. Gao and A. Zell, ”Sample-efficient Reinforcement Learning in Robotic Table Tennis,” 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, pp. 4171-4178, doi: 10.1109/ICRA48506.2021.9560764.

[6] Kumar, Akash, ”Playing Pong Using Q-Learning” (2021). West Chester University Master’s Theses. 226.

[7] T. Tompa, D. Vincze and S. Kov´acs, ”The Pong game implementation with the FRIQ-learning reinforcement learning algorithm,” Proceedings of the 2015 16th International Carpathian Control Conference (ICCC), 2015, pp. 542-547, doi: 10.1109/CarpathianCC.2015.7145139.

[8] S. K. U, P. S, G. Perakam, V. P. Palukuru, J. Varma Raghavaraju and P. R, ”Artificial Intelligence (AI) Prediction of Atari Game Strategy by usingReinforcement Learning Algorithms,” 2021 International Conference on Computational Performance Evaluation (ComPE), 2021, pp. 536-539,doi: 10.1109/ComPE53109.2021.9752304.

THANK YOU