A Appendix

A.1 Domain Description

We demonstrate the effectiveness of the proposed algorithms in the following domains:

PyFlags PyFlags (Erceth 2020) is a domain consisting of two adversarial tanks playing a game of capture the flag. This domain was chosen as it provides a standard rule-based player, which can be used as a sub-optimal demonstrator. The domain provides a zero-sum, adversarial game that allows for an intuitive policy comparison (positive score = RL-controlled agent is better than the opponent, and vice versa). This domain also exhibits a continuous control task with sparse rewards, making it challenging for an RL agent to learn an optimized policy. Effective exploration in this domain is especially challenging given that the opponent is actively targeting the RL-controlled agent. Reaching the opponent's base without being shot, stealing the flag, and returning it to the home base is highly unlikely following random exploration.

The PyFlags domain is illustrated in Figure A.1(a). The RL agent controls the blue tank, whereas the red tank is the opponent (a rule-based player provided by the domain). The domain is defined by:

- **State space, S.** The positions and orientations of all the objects in the environment (obstacles, bases, flags, and red tank), relative to the blue tank. The state space also includes the angular and linear velocity of the blue tank and a time-to-reload value for both tanks, representing the earliest time (in the future) at which the tank can shoot. The last 4 observation frames are provided at each time step.
- **Action space, $\mathcal{A}$.** The action space is a Cartesian product of 3 values: angular velocity $\in [-1, 1]$, linear velocity $\in [0, 1]$, and fire $\in \{0, 1\}$. These actions affect the relevant state values for the blue tank (the tank controlled by the RL agent). These actions are available on time steps where the RL-agent is assigned control.
- **Transition function, $\mathcal{P}$.** States evolve based on actions assigned to the blue tank and based on actions assigned to the red tank. The red tank actions are assumed to be drawn from an unknown adversarial policy. If either tank gets shot it re-spawns at its own base after a randomly selected re-spawn period, between 100 and 200 time steps.
- **Target reward function, R^f.** A positive reward (+1) is received when the blue tank returns to its base with the red flag, and a negative reward (-1) is received when the red tank does the same.

FetchPickAndPlace-v1 FetchPickAndPlace (Plappert et al. 2018), is a robotics domain consisting of a 7-DoF Fetch robotics arm, which has a two-fingered parallel gripper. The task in this domain is to use the gripper to hold a box and carry it to the target position, which may be on the table or in the air above it. It is a multi-goal domain since the target position is chosen randomly in every episode. This domain was chosen as it is well suited to demonstrate the algorithm's capability of solving continuous control, hard-exploration sparse reward tasks. The FetchPickAndPlace environment is the most complex of the Fetch environments, as it is difficult to learn to grip the object using random actions. A standard rule-based player (Nair et al. 2018) is also provided, which can be used as a suboptimal demonstrator.

The FetchPickAndPlace domain is illustrated in Figure A.1(b). The domain is defined by:

- **State space, S.** Observations include the Cartesian position of the gripper, its linear velocity as well as the position and linear velocity of the robot's gripper. It also includes the object's Cartesian position and rotation using Euler angles, its linear and angular velocities, its position and linear velocities relative to the gripper, as well as the desired target coordinates it must be moved to. The goals that are achieved by the agent so far can also be obtained, which is required by the HER algorithm.
- **Action space, $\mathcal{A}$.** The action space is 4-dimensional: 3 dimensions specify the desired gripper movement in Cartesian coordinates $\in [-1, 1]$ and the last dimension controls opening and closing of the gripper $\in [-1, 1]$.
- **Transition function, $\mathcal{P}$.** The object and its desired target coordinates are randomly selected at the beginning of every episode. The robotic arm and its gripper change their position after each step, based on the action taken. The same action is applied for 20 subsequent simulator steps (with $\Delta t = 0.002$ each) before returning control to the agent.
- **Target reward function, R^f.** The rewards are sparse and binary. A positive reward (+1) is received when the object is within 5cm of the target coordinates and 0 otherwise.

FetchSlide-v1 FetchSlide (Plappert et al. 2018), is a robotics domain consisting of a 7-DoF Fetch robotics arm, which has a two-fingered end effector used for pushing a block on a long table top. The reach of the robotic arm is limited to only a small portion of the table. The task in this domain is to use the end effector of the robotic arm to push the box in a precise manner, such that it comes to a stop at a target location on the table. The target location may not be within reach of the robotic arm, requiring it to push the block with an exact force and angle such that it slides exactly to the target location. It is a multi-goal domain since the target position is chosen randomly in every episode. This domain was chosen as it is well suited to demonstrate the algorithm's capability of solving continuous control, hard-exploration sparse reward tasks. The FetchSlide environment is a more complex version of the FetchPush environment, as it requires greater precision. A standard rule-based player (Nair et al. 2018) is also provided, which can be used as a suboptimal demonstrator.

The FetchSlide domain is illustrated in Figure A.1(c). The domain is defined by:

- **State space, S.** Observations include the Cartesian position of the gripper, its linear velocity as well as the position and linear velocity of the robot's gripper. It also includes the object's Cartesian position and rotation using Euler angles, its linear and angular velocities, its position and linear velocities relative to the gripper, as well as the

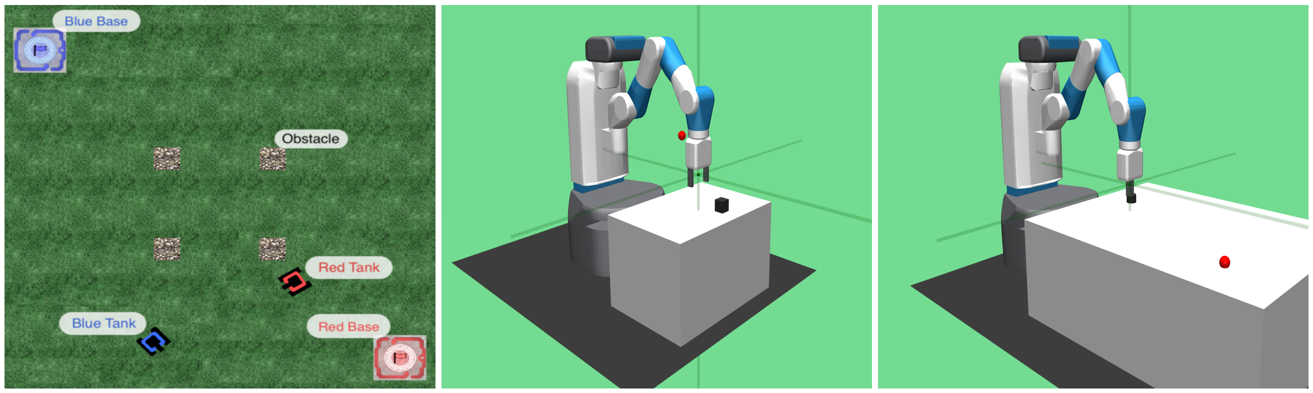

Figure A.1: (a) The PyFlags domain; (b) FetchPickAndPlace-v1 domain; (c) FetchSlide-v1 domain

desired target coordinates it must be moved to. The goals that are achieved by the agent so far can also be obtained, which is required by the HER algorithm.

- **Action space, $\mathcal{A}$.** The action space is 4-dimensional: 3 dimensions specify the desired gripper movement in Cartesian coordinates $\in [-1, 1]$ and the last dimension controls opening and closing of the gripper $\in [-1, 1]$. In this domain, the gripper action is disabled.
- **Transition function, $\mathcal{P}$.** The object and its target coordinates are randomly selected at the beginning of every episode. The robotic arm and its gripper change their position after each step, based on the action taken. The same action is applied for 20 subsequent simulator steps (with $\Delta t = 0.002$) before returning control to the agent.
- **Target reward function, R^f.** The rewards are sparse and binary. A positive reward (+1) is received when the object is within 5cm of the target coordinates and 0 otherwise.

A.2 Hyperparameters

The hyperparameters of the algorithms used in the experiments in the PyFlags, FetchPickAndPlace and FetchSlide domains are mentioned below. The network architecture specifies the number of hidden layers in the neural network and the number of nodes each layer is composed of. The $\tanh$ activation function is applied on each node of the hidden layers. They are shared by the actor and critic networks. The learning rate mentioned applies to both the actor and critic networks. The demonstrations for all the algorithms requiring demonstrations are collected from a rule-based policy for each domain.

Temporal Phasing In the PyFlags domain we set $\alpha = 0.1$ if the current RL policy achieved a game score ≥ 0 in the last 50 episodes; else we set $\alpha = 0$. Since the demonstrator used is the same as the opponents policy a threshold of 0 implies that the performance must be at least as good as that of the demonstrator. In the P&P domain we set $\alpha = 0.015$ if the current RL policy achieved a game score ≥ 0.2 in the last 50 episodes; else we set $\alpha = 0$. A 0.2 threshold would imply that the agent was able to reach the goal position on average. Similarly, $\alpha = 0.015$ in the FS domain if the game score $\geq$ 0.1 in the last 50 episodes; else we set $\alpha = 0$.

Reward Phasing In the P&P and FS domains, we find that reducing the entropy coefficient to 0.001, the learning rate to 0.00007 and α to 0.001 after 75% of reward phasing results in more stable learning. This may be interpreted as a need to prevent the policy from drifting too far from the current policy as the demonstrators affect on the policy diminishes. A large drift may cause the policy to forget useful behaviours that it learnt while imitating the demonstrator.

Hindsight Experience Replay (HER) with demonstrations The hyperparameters for FetchPickAndPlace and FetchSlide are the same as those mentioned in (Nair et al. 2018). Similarly, most of the hyperparameters for PyFlags are the same as those mentioned in (Nair et al. 2018) with a few adjustments so that the algorithm better adapts to the domain. Each demonstration in the PyFlags domain consists of 20480 timesteps. HER with demonstrations baseline uses DDPG (Lillicrap et al. 2016) as the policy optimization algorithm.

Random Network Distillation (RND) (Burda et al. 2019). Proximal Policy Optimization (PPO) (Schulman et al. 2017) is used as the policy optimization algorithm. The hidden layers of the neural network are shared by an actor, intrinsic value function, and extrinsic value function. The feature predictor neural network as well as the fixed randomly initialized neural network consists of 2 hidden layers, each composed of 32 nodes with $\tanh$ activation function, and an output feature layer consisting of 16 nodes, in all domains.

Self-Adaptive Imitation Learning (SAIL) (Zhu et al. 2022). Twin-delayed deep deterministic policy gradient (TD3) (Fujimoto, Hoof, and Meger 2018) is used as the policy optimizing algorithm, The hyper-parameters are tuned using the default optimizer tool provided in the reproducible code, presented in (Zhu et al. 2022). The network architecture and demo buffer information is provided in Table A.6.

A.3 Phasing Variants

***Temporal-Phasing* variants** Since *Temporal-Phasing* was only successful in the PyFlags domain, we present the comparison of its different variants and the ablation studies in this domain. The learning trends of the *Temporal-Phasing* variants can be observed in Figure A.2. Variant 1 is able to

	Network arch.	γ	λ	lr	$clipping(\epsilon)$	ent_coef	batch_size	α
PyFlags	(512,512)	0.99	0.94	0.00003	0.3	0.005	20480	0.1
P&P & FS	(256,256,256)	0.98	0.95	0.0003	0.2	0.005	1024	0.015

Table A.1: Temporal phasing: PPO hyperparameters for each domain

	Network arch.	γ	λ	lr	$clipping(\epsilon)$	ent_coef	batch_size	α	No. Demos
PyFlags	(512,512)	0.99	0.95	0.0003	0.2	0.005	20480	0.05	50
P&P & FS	(256,256,256)	0.98	0.95	0.0003	0.2	0.005	1024	0.015	50

Table A.2: Reward Phasing: PPO hyperparameters for each domain

	Network arch.	γ	lr	batch_size	activation func.
PyFlags	(128,128)	0.99	0.0003	20480	ReLU
P&P & FS	(128,128)	0.98	0.0003	1024	ReLU

Table A.3: Reward Phasing: IRL discriminator hyperparameters for each domain

	Network arch.	Q_lr	pi_lr	No. Demos
PyFlags	(512,512)	0.0001	0.0001	50

Table A.4: HER with Demos: Modified HER hyperparameters for PyFlags domain

	Network arch.	γ	λ	lr	$clipping(\epsilon)$	ent_coef	batch_size
PyFlags	(512,512)	0.99	0.95	0.0003	0.2	0.005	20480
P&P & FS	(256,256,256)	0.98	0.95	0.0003	0.2	0.005	1024

Table A.5: RND: PPO hyperparameters for each domain

	Network arch.	γ	buffer size	Demo buffer size	No.Demos	batch_size	train freq.
PyFlags	(512,512)	0.99	1000000	600000	50	2048	20480
P&P & FS	(256,256,256)	0.98	1000000	50000	50	256	1024

Table A.6: SAIL: TD3 hyperparameters for each domain

complete *Temporal-Phasing* (reach $\beta = 1$) and surpass the demonstrator's policy, whereas variants 2 and 3 did not reach $\beta = 1$ following 30,000 episodes. Recall that a dynamic step size was used for these variants hence reaching $\beta = 1$ is not guaranteed.

In Fig. A.3 we provide results for the ablation studies conducted on the temporal phasing approach, which demonstrates that Importance Sampling and a dynamic phase interval play a crucial role in the success of this approach. In the fixed-step phasing approach, the sampling frequency of the RL policy is increased by 10% every 1,000 episodes. Despite providing a long interval between phasing steps, we see that the policy does not perform very well. This is because the policy may often have a long and unstable re-training phase before more control can be given to the RL policy, as can be observed when $\beta = [0.5, 0.7]$ (50-70% RL control) for a dynamic step-size. We attribute this to the fact that the $\{\mathcal{K}_\beta, \pi^*_\beta\}$ space is not continuous. The RL agent is capable of learning to perform the task if a long enough interval is provided to the 're-train' function, (Line 7, in Algorithm 1).

***Reward-Phasing* variants** Since *Reward-Phasing* was only successful in the FetchPickAndPlace domain, we present the comparison of its different variants and the ablation studies in this domain. The results in Figure A.4 and Figure A.5 suggest that V1 (constant phasing) outperforms V2 (random phasing). This indicates that *Reward-Phasing* works well when the magnitude of the dense reward is gradually reduced instead of reducing the frequency of the dense reward.

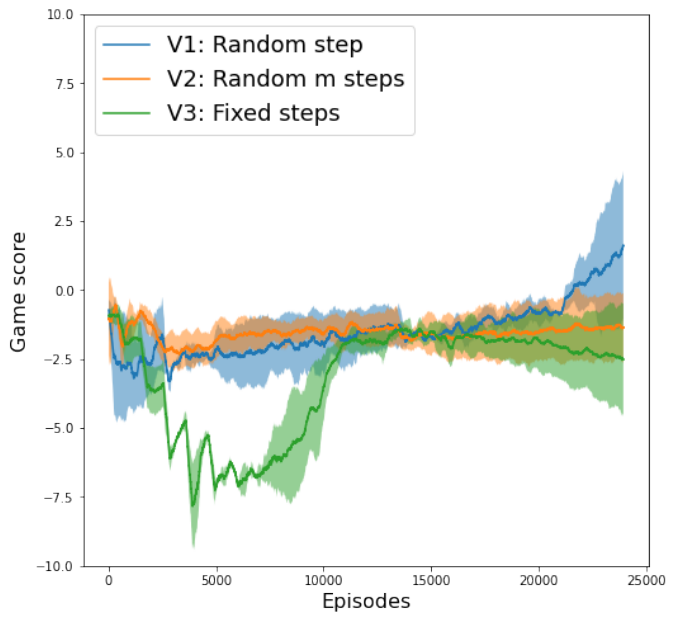

Figure A.2: Learning trend comparisons between the task phasing variants in PyFlags. Curves are averaged over 300 runs.

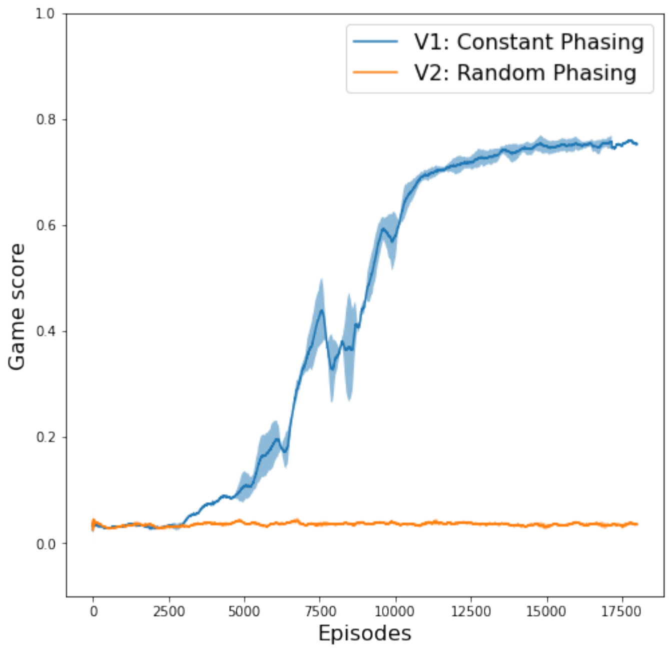

Figure A.4: Reward phasing variants in P&P domain

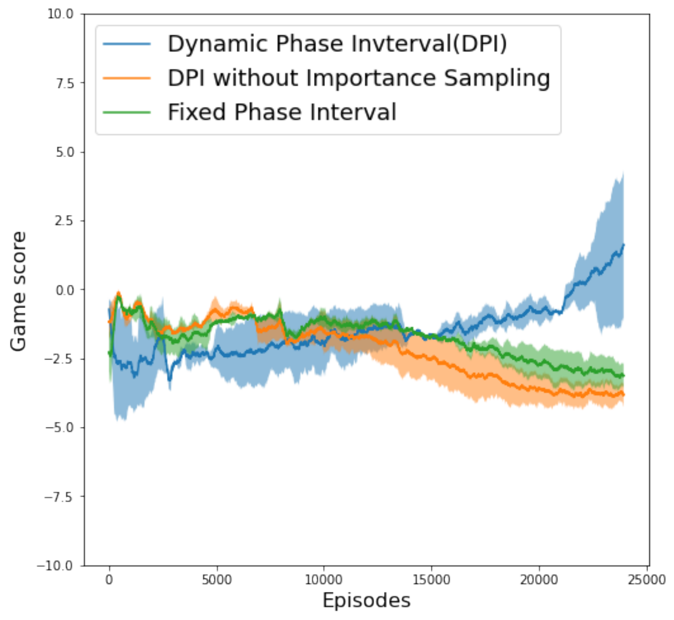

Figure A.3: Temporal phasing with/without dynamic step size and importance sampling in PyFlags

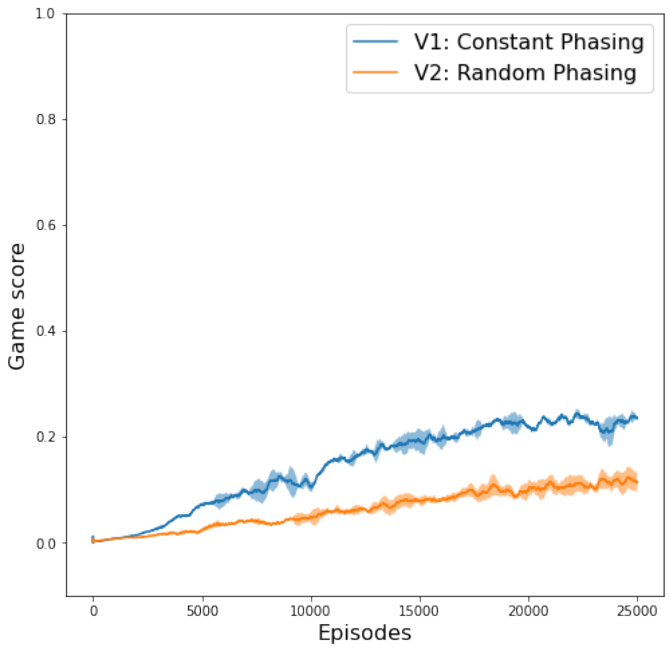

Figure A.5: Reward phasing variants in FetchSlide domain