Agent Environment Cycle Games

ABSTRACT

Partially Observable Stochastic Games (POSGs), are the most general model of games used in Multi-Agent Reinforcement Learning (MARL), modeling actions and observations as happening simultaneously for all agents. We introduce Agent Environment Cycle Games (AEC Games), a model of games based on sequential agent actions and observations. AEC Games can be thought of as sequential versions of POSGs, and we prove that they are equally powerful. We argue conceptually and through case studies that the AEC games model is useful in important scenarios in MARL for which the POSG model is not well suited. We additionally introduce "cyclically expansive curriculum learning," a new MARL curriculum learning method motivated by the AEC games model. It can be applied "for free," and experimentally we show this technique to achieve up to 35.1% more total reward on average.

KEYWORDS

Reinforcement Learning, Multi-Agent Reinforcement Learning, Curriculum Learning

ACM Reference Format:
. 2021. Agent Environment Cycle Games. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), London, UK, May 3–7, 2021*, IFAAMAS, 10 pages.

1 INTRODUCTION

The most common model of games in Multi-Agent Reinforcement Learning ("MARL") is *Partially Observable Stochastic Games* ("POSGs"). In POSGs, all agents take an action simultaneously, then the environment responds and each agent's observation updates and it receives a reward, then all agents act again, and so on [12]. This paradigm of POSGs and "simultaneous agent action" is ubiquitous in MARL outside of strictly turn-based games like chess [3, 11, 12, 16].

While agents are modeled as taking actions simultaneously in a POSG, this is generally slightly untrue in practice. Unless complex parallelization techniques are used, in any environment where multiple agents are interacting, the environment effectively updates one agent at a time, according to some order.

Accordingly, we introduce the *Agent Environment Cycle* ("AEC") games model. It's ultimately a sequential version of the parallel POSG paradigm (and model). In the AEC games model, one agent acts, then the environment may respond (updating the observation), then the next agent steps, and so on. The full definition and mathematical formalization is in section 3. This indefinite cycle of agent and environment steps is the inspiration for the name.

This model was developed after finding a significant unnoticed bug in the *cleanup* environment from Vinitsky et al. [21] and a significant learning inefficiency in the *pursuit* environment from Gupta et al. [7]. Both are very popular MARL environments, and both issues stemmed from treating the environments as if all agents stepped simultaneously, when this was not *quite* the case. They were very easy mistakes to make that could easily occur in any MARL environment, were very difficult to find, and could be easily seen when viewed from a lens of a formal conceptual model of sequential agent stepping.

We argue that, due to their better match-up with game mechanics in practice and their usefulness in finding and preventing serious subtle bugs, AEC games are often a superior model to POSGs in MARL.

Furthermore POSGs, due to treating all agents as acting at once, are a very inelegant model for viewing strictly turn based games like chess or Go. Treating an environment that actually steps in parallel as stepping sequentially isn't nearly as problematic, so AEC games provides a clean, unified model for both POSGs and fully turn based games to be considered with.

Due to these considerations, based on a preprint version of this work, the AEC games model has been used as the basis of the API for the PettingZoo library [18]. PettingZoo is a Python library that's akin to a multi-agent version of OpenAI's Gym library [6], and contains the largest and most diverse collection of multi-agent environments ever under one API.

In this paper, we conceptually and formally introduce the AEC games model and prove its equivalence to POSGs in section 3; we give case studies of the aforementioned bugs in section 4 and section 5. We additionally introduce a new method of curriculum learning termed "cyclically expansive curriculum learning," that's heavily inspired by the AEC games model and can be trivially applied to any MARL environment. We experimentally show it achieves up to 35.1% more total reward on average.

2 BACKGROUND AND RELATED WORKS

2.1 Models of Games For Reinforcement Learning

Reinforcement learning (RL) methods seek to learn a policy (a function which takes an observation and returns an action) that achieves the maximum expected discounted reward for a given environment. Single-agent environments are traditionally modeled as a *Markov Decision Process* ("MDP") or a *partially-observable MDP* ("POMDP") [5]. An MDP models decision making as a process where an agent repeatedly takes a single action, receives a reward, and transitions to a new state (receiving complete knowledge of the state). A POMDP extends this to include environments where the agent may not be able to observe the entire state of the environment.

Multi-agent reinforcement learning similarly seeks to learn a set of optimal policies for each agent in an environment where they interact. The most basic model of environments is the *Multi-agent MDP* ("MMDP"), in which MDPs have multiple agents which act simultaneously, sharing a single reward function [5]. *Decentralized POMDPs* (or "Dec-POMDPs") add partial observablity to MMDPs [4]. *Stochastic Games*, sometimes called *Markov Games*, extend MMDPs to have a unique reward function for each agent [15]. Stochastic

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, London, UK. © 2021 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

games are extended to the partially observable case with *Partially Observable Stochastic Games* ("POSGs"), and are the model most typically used in MARL.

Definition 1. A *Partially-Observable Stochastic Game* (POSG) is a tuple $\langle \mathcal{S}, N, \{\mathcal{A}_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\} \rangle$, where:

- $\mathcal{S}$ is the set of possible *states*.
- N is the *number of agents*. The *set of agents* is $[N]$.
- $\mathcal{A}_i$ is the set of possible *actions* for agent i.
- $P : \mathcal{S} \times \prod_{i \in [N]} \mathcal{A}_i \times \mathcal{S} \to [0,1]$ is the *transition function*. It has the property that for all $s \in \mathcal{S}$, for all $(a_1, a_2, \ldots, a_N) \in \prod_{i \in [N]} \mathcal{A}_i$, $\sum_{s' \in \mathcal{S}} P(s, a_1, a_2, \ldots, a_N, s') = 1$.
- $R_i : \mathcal{S} \times \prod_{i \in [N]} \mathcal{A}_i \times \mathcal{S} \to \mathbb{R}$ is the *reward function* for agent i.
- Ω_i is the set of possible *observations* for agent i.
- $O_i : \mathcal{A}_i \times \mathcal{S} \times \Omega_i \to [0,1]$ is the *observation function*. It has the property that $\sum_{\omega \in \Omega_i} O_i(a, s, \omega) = 1$ for all $a \in \mathcal{A}_i$ and $s \in \mathcal{S}$. $O_i(a_i, s, \cdot)$ specifies the probability distribution over possible observations when agent i takes action a_i, and the POSG transitions to state s.

Note that while O_i doesn't directly depend on the actions of other agents, it does depend on the new state, which in turn depends on the actions of the other agents.

A variety of other models have been introduced in the literature, such as Multiagent POMDPs [13] and Macro-action Decentralized POMDPs (MacDec-POMDPs) [1]. These models are based on modifying or extending existing MDP or POMDP models. As such, they have the property that agents act in unison, and the state transitions in response to their joint action. Our model, presented in section 3, takes a different approach and instead associates the action of only one agent to each "step" of the game.

2.2 Deep Reinforcement Learning Methods

Deep reinforcement learning (DRL) seeks to learn optimal policies for RL environments while representing the policy function as a neural network. This paper uses parameter shared versions of PPO [14] and Ape-X DQN [8], motivated by the empirical results in Terry et al. [19] which showed them to perform very well on the Pursuit environment (described in [7]). Ape-X DQN also regularly achieves state-of the art performance on discrete graphical games in general [8], and PPO can often achieve state-of the art performance and generally requires very little hyperparameter tuning [14].

3 THE AEC GAMES MODEL

3.1 The Model

We begin by defining the model of AEC games in English. The base component of an AEC game is a changeable list of agents. After the first agent in the list acts, the environment can "act" (allowing agents' observations to be updated), or the next designated agent can act (skipping environment turns are how truly simultaneous games are depicted). This process continues indefinitely. In most games, the agents and the order in which they act don't change much, so we've found it very helpful to diagram games as in figures 1 and 2. We call such diagrams "AEC Diagrams."

As for reward, after every agent takes a turn a partial reward is emitted to every other agent. The reward for an agent's action is the total of all rewards following that action and before the agent's next turn (until this point, the reward is not fully defined). Different aspects of a game will be responsible for different portions of reward. As shown in Sections 4 and 6, thinking about rewards in this atomized manner instead of lumping the reward process all together can be very helpful.

3.2 Mathematical Formalism

Accordingly, we formalize AEC games as follows:

Definition 2. An *Agent-Environment Cycle Game* (AEC Game) is a tuple $\langle \mathcal{S}, N, \{\mathcal{A}_i\}, \{T_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\}, \nu \rangle$, where:

- $\mathcal{S}$ is the set of possible *states*.
- N is the *number of agents*. The agents are numbered 1 through N. There is also an additional "environment" agent, denoted as agent 0. We denote the set of agents along with the environment by $\Xi := [N] \cup \{0\}$.
- $\mathcal{A}_i$ is the set of possible *actions* for agent i. For convenience, we further define $\mathcal{A}_0 = \{\varnothing\}$ and $\mathcal{A} := \bigcup_{i \in \Xi} \mathcal{A}_i$. The action $\varnothing$ is a special "null" action which is the only action associated with the environment. This allows us to use the environment as an agent in the functions below, which include both an agent and an action (which is assumed to be part of the given agent's action space) as part of the input.
- $T_i : \mathcal{S} \times \mathcal{A}_i \to \mathcal{S}$ is the *transition function for agents*. State transitions for agent actions are deterministic.
- $P : \mathcal{S} \times \mathcal{S} \to [0,1]$ is the *transition function for the environment*. State transitions for environment steps are stochastic: $P(s, s')$ is the probability that the environment transitions into state s' from state s.
- R_i is the *reward function* for agent i. The set of all possible rewards for each agent is assumed to be finite, which we denote $\mathcal{R}_i \subseteq \mathbb{R}$. We also define $\mathcal{R} := \bigcup_{i \in [N]} \mathcal{R}_i$. Then, the reward function is $R_i : \mathcal{S} \times \Xi \times \mathcal{A} \times \mathcal{S} \times \mathcal{R}_i \to [0,1]$. It is *stochastic*: $R_i(s, j, a, s', r)$ is the probability of agent i receiving reward r when agent j takes action a while in state s, and the game transitions to state s'. The function should only define a nonzero reward $R_i(s, j, a, s', r) \neq 0$ when $a \in \mathcal{A}_j$.
- Ω_i is the set of possible *observations* for agent i.
- $O_i : \mathcal{S} \times \Omega_i \to [0,1]$ is the *observation function* for agent i. $O_i(s, \omega)$ is the probability of agent i observing ω while in state s.
- $\nu : \mathcal{S} \times \Xi \times \mathcal{A} \times \Xi \to [0,1]$ is the *next agent* function. This means that $\nu(s, i, a, j)$ is the probability that agent j will be the next agent permitted to act given that agent i has just taken action a in state s. This should attribute a non-zero probability only when $a \in \mathcal{A}_i$.

With this definition, an AEC Game has a state $s_0 \in \mathcal{S}$ which is designated as the initial state, with an agent $i \in \Xi$ designated as the first agent to act. The game then evolves in "turns" where in each turn the game starts in some state s with agent i to act; agent i receives an observation ω_i and then chooses an action $a_i \in \mathcal{A}_i$, and the game transitions into a new state s'; then, a new agent i' is determined who will be the one to act in the next turn. The observation ω_i that is received is random, occurring with probability $O_i(s, \omega_i)$. The new state s' is determined by one

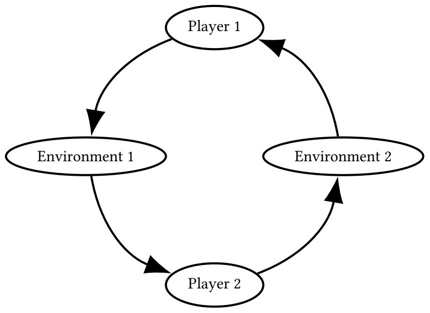


Figure 1: The AEC diagram of Chess

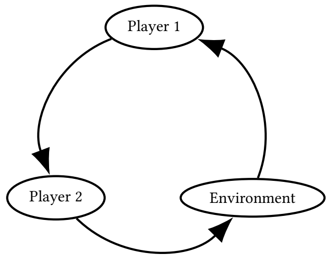


Figure 2: The AEC diagram of Chess naively modeled as a POSG. The missing environment node between Player 1 and Player 2 denotes the simultaneous action as there's no state update without an environment step. This diagram clearly communicates the asymmetry between Player 1 and Player 2 generated by naively reducing an environment to a simultaneous action.

of the transition functions: if $i = 0$, then the current turn is an "environment step" and the next state s' is random, occurring with probability $P(s, s')$; if $i > 0$, then the new state is deterministically $s' = T_i(s, a_i)$. The next agent i' is determined randomly by the next-agent function, with i' being chosen with probability $\nu(s, i, a_i, i')$. Finally, at every turn, every agent receives some (possibly negative) reward. In the preceding example, every agent j will receive a random reward r', with probability $R_j(s, i, a_i, s', r')$. In order for the next-agent and reward functions to be well defined in environment steps, we use the special action $\emptyset$. For example, agent i' will be next to act after an environment step in state s with probability $\nu(s, 0, \emptyset, i')$.

3.3 Equivalence to POSGs

As one would hope, AEC Games are as powerful as POSGs. This is because every POSG can be converted into an AEC game where every $N + 1$ turns of the AEC Game corresponds to one step of the original POSG (each agent will act exactly once, followed by a single environment step to resolve the "joint action").

Theorem 1. *For every POSG, there is an equivalent AEC Game.*

Proof. Let $G = \langle \mathcal{S}, N, \{\mathcal{A}_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\}\rangle$ be a POSG. We define an equivalent AEC Game G_{AEC} as follows:

$$G_{\text{AEC}} = \langle \mathcal{S}', N, \{\mathcal{A}_i\}, \{T_i\}, P', \{R'_i\}, \{\Omega_i\}, \{O_i\}, \nu\rangle$$

where

- $\mathcal{S}' = \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_N$. That is, an element of $\mathcal{S}'$ (and thus, any input to T_i, P, R_i, or O_i representing a state) is a tuple $(s, a_1, a_2, \ldots, a_N)$ where $s \in \mathcal{S}$ and for each $i \in [N]$, $a_i \in \mathcal{A}_i$.
- $T_i((s, a_1, a_2, \ldots, a_i, \ldots, a_N), a'_i) = (s, a_1, a_2, \ldots, a'_i, \ldots, a_N)$. In this sense, the most recent action of agent i is "embedded" in the $(i + 1)$st element of the new state tuple.
- For $\mathbf{s} = (s, a_1, a_2, \ldots, a_N)$ and $\mathbf{s}' = (s', a_1, a_2, \ldots, a_N)$, we define $P'(\mathbf{s}, \mathbf{s}') = P(s, a_1, a_2, \ldots, a_N, s')$. This way, the environment step causes the state to transition in the same way as the original POSG G (ignoring the part of the state tuple that embeds action information). If $\mathbf{s}$ and $\mathbf{s}'$ are such that $a_i \neq a'_i$ for any $i \in [N]$, then $P'(\mathbf{s}, \mathbf{s}') = 0$.
- For $\mathbf{s} = (s, a_1, a_2, \ldots, a_N)$ and $\mathbf{s}' = (s', a_1, a_2, \ldots, a_N)$, and $\mathbf{r} = R_i(s, a_1, a_2, \ldots, a_N, s')$, we let $R'_i(\mathbf{s}, 0, \emptyset, \mathbf{s}', \mathbf{r}) = 1$. We define $R'_i = 0$ for all other cases. In this way, the reward for agent i is emitted only at the environment step, where it is deterministically equal the original reward for agent i in the POSG for the same state transitions and joint action (as embedded in the new state tuple).
- $\nu((s, a_1, a_2, \ldots, a_N), i, a'_i, j) = 1$ if $j \equiv i + 1 \pmod{N + 1}$ (and equals 0 otherwise). In this way, the steps are given by an unchanging deterministic cycle in which agent $i + 1$ follows agent i, with the environment (agent 0) taking a step (causing the first part of the state tuple to transition and a reward to be emitted to all agents) after the last agent, N, takes its action.

The AEC game G_{AEC} begins with agent 1. If the initial state of the POSG G was s_0, then the initial state of G_{AEC} is $(s_0, \cdot, \cdot, \ldots, \cdot)$; that is, all but the first element of the tuple can be chosen arbitrarily. When agent 1 (more generally, agent $i > 0$) takes any action a_1 (more generally, action a_i), the $(i + 1)$st element of the state tuple is set to a_1 (a_i) and all other elements of the tuple are unchanged. Once agent N for the first time (with action a_N), the state tuple will be $(s_0, a_1, a_2, \ldots, a_N)$ and the environment will then step. Thus, after one full cycle (i.e., after the first environment step), the system will correspond precisely to one step of the original POSG: all agents will have taken one action, and the state will have transitioned and rewards will have been emitted to all agents according to their joint action. Each new cycle of the AEC game corresponds to one step of the original POSG. Notice that in the environment step of each cycle, the state transition depends on the full joint action consisting of each action taken by all agents in the current cycle. The reward is only nonzero during the environment step, at which point it is deterministically equal to the original POSG reward given the same state transition and joint action. □

As you would also expect, every AEC Game can be converted into a POSG, by using a transition function (in the POSG) which ignores all components of a joint action except for one, corresponding to the agent whose turn it is. The details of this construction and the formal proof that every AEC game has an equivalent POSG is included in Appendix A. This means that AEC games and POSGs are truly equivalent.

4 CASE STUDY 1: REWARD PRUNING IN SISL PURSUIT

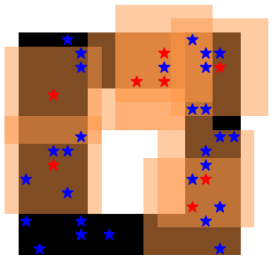


Figure 3: The *pursuit* environment from Gupta et al. [7].

Per subsection 3.1, each agent's reward updates after every other agent's turn. Therefore, an agent's reward is only fully defined right before it acts again. If some of these sources of reward are unable to be learned (in theory due to randomness or in practice due to near-randomness or complexity), it's plainly superior to "ignore" those rewards when they're emitted from the cycle. This can be thought of as "reward pruning." This can easily accidentally happen in a MARL game where reward from many sources are present, and when all rewards from all sources are bundled together, this proves very hard to notice or fix.

Pursuit is a popular MARL benchmark environment from Gupta et al. [7], shown in Figure 3. In it, 8 red pursuer agents must work together to surround and capture 30 randomly moving blue evader agents. The action space of each agent is discrete (cardinal directions or do nothing), and the observation space is a 7×7 box centered around a pursuer (depicted by the orange box). When an evader is surrounded on all sides by pursuers or the game boundaries, each contributing pursuer gets a reward of 5. Pursuers also receive a reward of 0.01 every time they touch an evader.

In *pursuit*, pursuers move first, and then evaders move randomly, before it's determined if an evader is captured and rewards are emitted. Thus an evader that "should have" been captured is not actually captured. Having the evaders move second isn't a bug: it's just way of adding complexity to the classic genre of pursuer/evader multi-agent environments [20], and is representative of real world problems. When *pursuit* is viewed as an AEC game, we're forced to attribute rewards to individual steps, and the breakdown becomes pursuers receiving deterministic rewards from surrounding the evader, and then random reward due to the evader moving after. Removing this random component of the reward (the part caused by the evaders action after the pursuers had already moved), should then lead to superior performance. In this case the problem was so innocuous that fixing it required switching two lines of code where their order made no obvious difference. Bugs of this family could easily happen in almost any MARL environment, and analyzing and preventing them is made much easier with the AEC games model.

We validated this experimentally by training parameter shared Ape-X DQN [8] (the best performing model on *pursuit* [19]) four times using RLLib [11] with and without reward pruning, achieving better results with reward pruning every time and 18.3% more total reward on average. Results with PPO are included in Appendix B. Hyperparameters are included in Appendix C.

5 CASE STUDY 2: RACE CONDITIONS IN SEQUENTIAL SOCIAL DILEMMA GAMES

A very common scenario in multi-agent environments is for two agents to be able to take conflicting actions (i.e. occupy the same space). This discrepancy has to be resolved by the environment (i.e. collision handling); here we call this "tie-breaking." In an environment's code, in general, every agent will step according to some sort of internal agent order in an environment. Ensuring tie-breaking is not biased towards agents earlier in an internal agent order is *very* hard in practice, and modeling environments with biased tie-breaking as strictly simultaneous actions can be very misleading.

Consider an environment with two agents, Alice and Bob, in which Alice steps first and tie-breaking is biased in Alice's favor. If such an environment were assumed to have simultaneous actions, then observations for both agents would be taken before either acted, causing the observation Bob acts on to potentially no longer be an accurate representation of the environment. This is also a true race condition—the result of stepping through the environment can inadvertently differ depending on the internal resolution order of agent actions. In scenarios where it makes sense to do so, adding observation delay makes this problem go away. Otherwise, sequentially handling observations and actions (modeling the environment as an AEC game) is the sensible way to solve this, and using APIs based around AEC games has the benefit of being able to prevent subtle bugs of this variety.

A real world example of a major MARL environment with a parallel API and inadvertently imperfect tie-breaking is the cleanup environment in the open source implementation [21] of Sequential Social Dilemma Games from.

The Sequential Social Dilemma Games, introduced in [10], are a kind of MARL environment where good short-term strategies for single agents lead to bad long-term results for all of the agents. New SSD environments, including the *Cleanup* environment, were introduced in Hughes et al. [9]. The states of these games are represented by a grid of tiles, where each tile represents either an agent or a piece of the environment. In the *Cleanup* environment, the environment tiles can be empty tiles, river tiles, and apple tiles. Collecting apple tiles results in a reward for the agent and the agents must clean the river tiles with a "cleaning beam" for apple tiles to spawn. The cleaning beam extends in front of agents, one tile at a time, until it hits a dirty river tile ("waste") or extends to its maximum length of 5 tiles. Additionally, two more beams extend in front of the agent—one starting in the tile directly to the agent's left, and one from the tile on the right—until each hits a "waste" tile or reaches a length of 5 tiles. The cleaning beam is shown in [6c]. Note that while beams stop at "waste" tiles, they will continue to extend past clean river tiles.

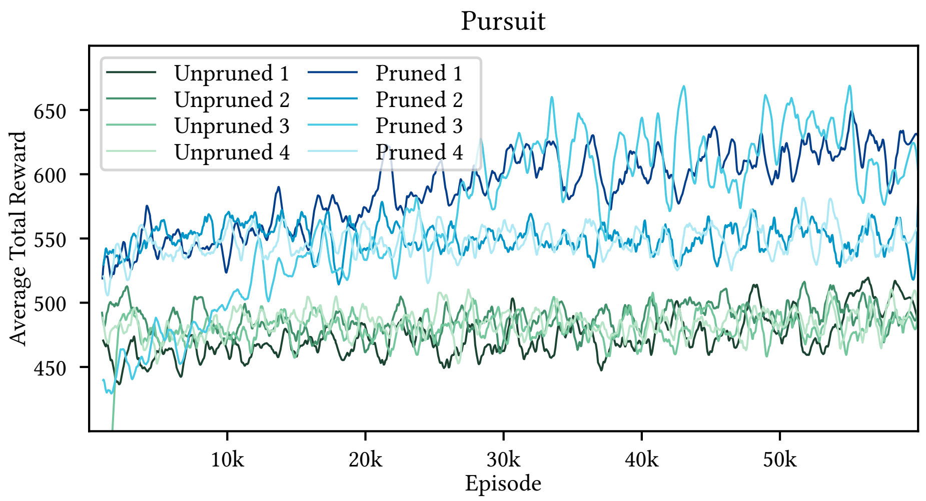
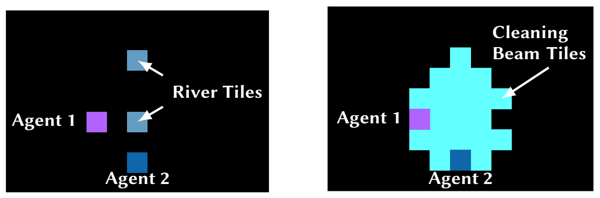


Figure 4: Learning on the *pursuit* environment with and without pruned rewards, using parameter sharing based on Ape-X DQN (the current state of the art method for the environment). This shows an average of an 18.3% improvement by using this method.

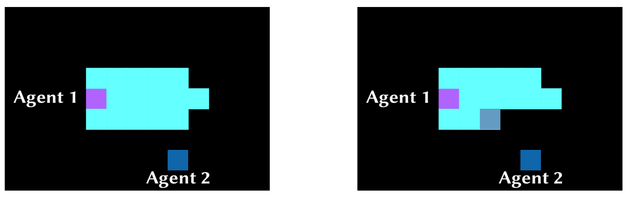


(a) The initial setup with two agents and two river tiles. When the river tiles become dirty, they are shown as a brownish color instead.

(b) The result of both agents perform the "clean" action. Both river tiles can be are cleaned since Agent 1's action is resolved first.

Figure 5: Cleanup, a Sequential Social Dilemma Game from Vinitsky et al. [21].

The agents act sequentially in the same order every turn, including the firing of their beams. In the case of two agents trying to occupy the same space, one is chosen randomly, however the tie breaking with regards to the beams is biased, due to a bug. Consider the setup in Figure 5 where each agent chooses the "clean" action for the next step. This results in Agent 1 firing their cleaning beam first, clearing the close river tile. Next, Agent 2 fires their cleaning beam and they are able to clean the far river tile because the close tile has already been cleared by Agent 1. However, if we keep the same placement and actions but switch the labels of the agents, we get a different result, seen in Figure 7. Now, Agent 1 fires first and hits the close river tile and can no longer reach the far river tile. In situations like these, the observation the second agent's policy is using to act on is going to be inherently wrong, and if it had the true environment state before acting it would very likely wish to make a different choice.



(a) If there are no dirty river tiles in the path of the cleaning beams, the beams will extend to the full length of five tiles.

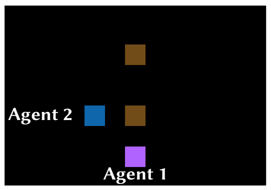
(b) If there is a dirty river tile in the path of a beam, the beam will stop at the tile, changing it to a "clean" river tile.

Figure 6: An example of Agent 1 using the "clean" action while facing East. The beams extend to a length of up to five tiles. The "main" beam extends directly in front of the agent, while two auxiliary beams start at the tiles directly next to the agent (one to the left and one to the right) and also extend up to five tiles. A beam stops when it hits a dirty river tile.

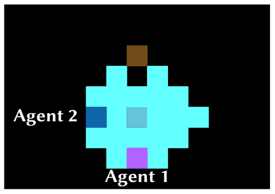
This is a serious class of bug that's very easy to introduce when using parallel action-based APIs, and using AEC games-based APIs prevents the class entirely. In this case the bug had gone unnoticed for years, and was both very subtle and serious.

6 CYCLICALLY EXPANSIVE CURRICULUM LEARNING

Curriculum learning is a technique where progressively more complicated mechanics are added to an environment to simplify learning complex mechanics [2]. Doing this is almost always helpful, but it can be very challenging to implement in practice because it



(a) The same setup as in Figure 5, but with the agent labels reversed.



(b) The result of both agents performing the "clean" action, with this agent assignment.

Figure 7: The impact of switching the internal agent order on how the environment evolves. When both agents clean, agent 1's action is resolved first, and the main beam stops when it hits the near dirty river tile, so the far river tile is not cleaned. In Figure 5, Agent 2's beam was able to reach the far beam because Agent 1's beam cleaned the near tile first.

typically requires making large changes to the environment (which is sometimes impossible, e.g. with Atari games).

In the AEC games model, a reward is emitted to every agent after each agent acts. This argue above that this is beneficial by more clearly modeling the origins of reward. It also creates a natural curriculum, where the learning method for each agent could begin by only considering the reward attributed to the agent that acts after it, progressively adding following agents to the reward considered. We term this *cyclically expansive curriculum learning*. While this is a very simple method, it's powerful in that it can be implemented via a simple wrapper any environment with an AEC game based API. This means that it can be used by researchers "for free," with almost no engineering required required. Based on a prepublication release of this work, this method has been included in the SuperSuit library as a simple wrapper anyone can use for all PettingZoo spec environments [17].

We experimentally demonstrate that this technique can be helpful by comparing the performance of fully parameter shared PPO, motivated by it's performance on pursuit in Terry et al. [19] and general popularity. Using RLLib [11], we learned pursuit 4 times with and without curriculum learning, achieved 35.1% more total reward on average. Each run is shown in Figure 8. All hyperparameters used for PPO and the curriculum learning scheme are included in Appendix C. Results for ApeX DQN on pursuit are also included in Appendix B. We used a version of this environment with reward pruning for these experiments because we feel it is the fairest representation of the environment.

This optimization could be applied via a very simple wrapper for any environment, making this a nearly "free" method of curriculum learning to use in any scenario, though it could only be reasonably applied to an environment with an AEC games based API as a POSG based API wouldn't expose the needed rewards.

7 CONCLUSION

In this paper, we introduced the AEC games model, an alternative to the dominant POSG model with sequential stepping, which we argue is more reflective of real games in MARL while being equally powerful mathematically. We present a major learning inefficiency in section 4 and present a significant bug in the *pursuit* environment in section 5, both of which are illustrative of large subtle issues that can occur in nearly any MARL environment. Both problems stemmed from treating the environment has a POSG with agents stepping fully in parallel when this was not *quite* the case, and despite being very subtle were more reasonably understandable from the lens of AEC games. Motivated by this, and the ease of cleanly modeling both strictly turn based games and those where agents truly step in parallel, we argue that the model of AEC games is often a superior model in multi-agent reinforcement learning.

Perhaps the greatest testament to the AEC games model is it serves as the basis of a remarkably clean API for PettingZoo [18], which includes the largest and most diverse collection of multi-agent environments ever under one API. The clean modeling of so many games is something no library has ever been able to do before.

We additionally use AEC games to pose a new method of curriculum learning we term "cyclically expansive curriculum learning" that can be implemented for any MARL environment via a simple environment wrapper, and show it to improve performance in the *pursuit* environment by 35.1% on average.

The one limitation we are aware of with our model is that, when an environment is truly a perfect POSG, the performance characteristics of certain environments can may it such that POSG based APIs can be faster than AEC games APIs in practice because it allows for neural network computations for each agent to be parallelized (though this is not a common thing to do).

Several options for future exploration remain. We feel that the most interesting are to seek out new flaws this model elucidates in other large MARL works, to explore the practical utility of cyclically expansive curriculum learning, and to explore using AEC games as an more wieldly alternative to extensive form games in theoretical MARL research. We hope the model of AEC games stays in researchers' "back pockets," allowing them an alternative perspective to find new and unknown things about environments during the course of MARL and multi-agent systems research.

REFERENCES

[1] Christopher Amato, George D Konidaris, and Leslie P Kaelbling. 2014. Planning with macro-actions in decentralized POMDPs. (2014).

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. 41–48.

[3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv preprint arXiv:1912.06680* (2019).

[4] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research* 27, 4 (2002), 819–840. https://doi.org/10.1287/moor.27.4.819.297 arXiv:https://doi.org/10.1287/moor.27.4.819.297

[5] Craig Boutilier. 1996. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*. Morgan Kaufmann Publishers Inc., 195–210.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[7] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. 2017. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 66–83.

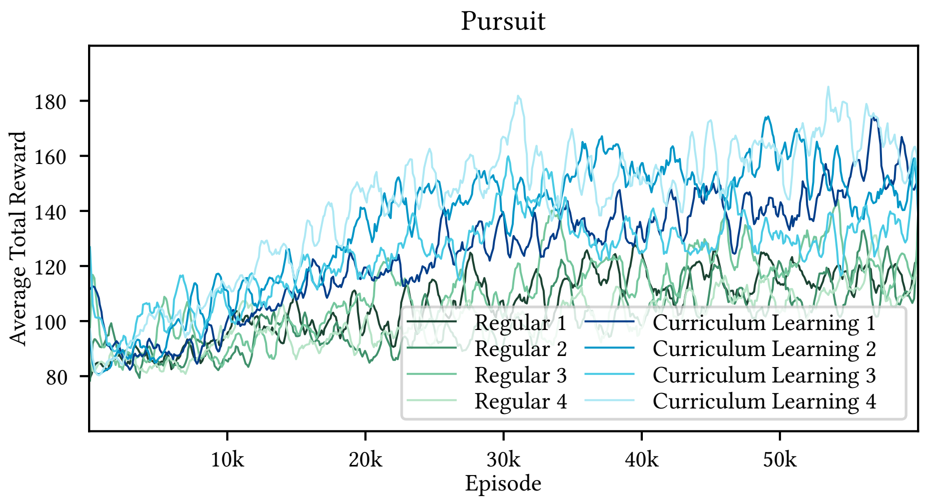


Figure 8: Learning on the *pursuit* environment with and without curriculum learning, using parameter sharing based on PPO. Curriculum learning increased the total reward by 35.1% on average.

[8] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. 2018. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933* (2018).

[9] Edward Hughes, Joel Z Leibo, Matthew Phillips, Karl Tuyls, Edgar Dueñez-Guzman, Antonio García Castañeda, Iain Dunning, Tina Zhu, Kevin McKee, Raphael Koster, et al. 2018. Inequity aversion improves cooperation in intertemporal social dilemmas. In *Advances in neural information processing systems*. 3326–3336.

[10] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. 2017. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037* (2017).

[11] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. 2017. RLlib: Abstractions for distributed reinforcement learning. *arXiv preprint arXiv:1712.09381* (2017).

[12] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *Neural Information Processing Systems (NIPS)* (2017).

[13] Joao V Messias, Matthijs TJ Spaan, and Pedro U Lima. 2013. Asynchronous execution in multiagent POMDPs: Reasoning over partially-observable events. In *AAMAS*, Vol. 13. 9–14.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[15] L. S. Shapley. 1953. Stochastic Games. *Proceedings of the National Academy of Sciences* 39, 10 (1953), 1095–1100. https://doi.org/10.1073/pnas.39.10.1095

[16] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. 2018. *DeepMind Control Suite*. Technical Report. DeepMind. https://arxiv.org/abs/1801.00690

[17] Justin K Terry, Benjamin Black, and Ananth Hari. 2020. SuperSuit: Simple Microwrappers for Reinforcement Learning Environments. *arXiv preprint arXiv:2008.08932* (2020).

[18] Justin K Terry, Benjamin Black, Mario Jayakumar, Ananth Hari, Luis Santos, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Ryan Sullivan, Caroline Horsch, and Praveen Ravi. 2020. PettingZoo: Gym for Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2009.14471* (2020).

[19] Justin K Terry, Nathaniel Grammel, Ananth Hari, Luis Santos, and Benjamin Black. 2020. Revisiting Parameter Sharing In Multi-Agent Deep Reinforcement Learning. *arXiv preprint arXiv:2005.13625* (2020).

[20] Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. 2002. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE transactions on robotics and automation* 18, 5 (2002), 662–669.

[21] Eugene Vinitsky, Natasha Jaques, Joel Leibo, Antonio Castenada, and Edward Hughes. 2019. An Open Source Implementation of Sequential Social Dilemma Games. https://github.com/eugenevinitsky/sequential_social_dilemma_games/. GitHub repository.

A CONVERSION OF AEC GAMES TO POSGS

As a warm up, we first show this only for the case of deterministic rewards.

Definition 3. An AEC Game

$$G = \langle \mathcal{S}, N, \{\mathcal{A}_i\}, \{T_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\}, \nu \rangle$$

is said to have *deterministic rewards* if for all $i, j \in \Xi$, all $a \in \mathcal{A}_j$, and all $s, s' \in \mathcal{S}$, there exists a $R_i^*(s, j, a, s')$ such that $R_i(s, j, a, s', r) = 1$ for $r = R_i^*(s, j, a, s')$ (and 0 for all other r).

Notice that an AEC Game with deterministic rewards may still depend on the new state s' which can itself be stochastic in the case of the environment ($j = 0$).

Theorem 2. *Every AEC Game with deterministic rewards has an equivalent POSG.*

Proof. Suppose $G = \langle \mathcal{S}, N, \{\mathcal{A}_i\}, \{T_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\}, \nu \rangle$ is an AEC game with deterministic rewards. We define $G_{\text{POSG}} = \langle \mathcal{S}', N, \{\mathcal{A}_i\}, P', \{R_i'\}, \{\Omega_i\}, \{O_i\} \rangle$ as follows.

- $\mathcal{S}' = \mathcal{S} \times \Xi$
- $P'((s, i), a_1, \ldots, a_N, (s', i')) = \nu(s, i, a_i, s', i') \cdot \Pr(s' \mid s, i, a_i)$, where

$$\Pr(s' \mid s, i, a_i) = \begin{cases} 1 & \text{if } i > 0 \text{ and } T(s, a_i) = s' \\ P(s, s') & \text{if } i = 0 \\ 0 & \text{o/w (i.e., if } i = 0 \text{ and } T(s, a_i) \neq s') \end{cases}$$

- $R_i'((s, j), a, (s', j')) = R_i^*(s, j, a, s')$

In this construction, the new state in the POSG encodes information about which agent is meant to act. State transitions in the POSG therefore encode both the state transition of the original AEC game and the transition for determining the next agent to act. In each step, the state transition depends only on the agent who's turn it is to act (which is included as part of the state).

This construction adapts POSGs to be strictly turn-based so that it is able to represent AEC Games. □

We now present the full proof.

THEOREM 3. *Every AEC Game has an equivalent POSG.*

PROOF. Suppose $G = \langle \mathcal{S}, N, \{\mathcal{A}_i\}, \{T_i\}, P, \{R_i\}, \{\Omega_i\}, \{O_i\}, \nu \rangle$ is an AEC game with $\mathcal{R}$ being the (finite) set of all possible rewards. We define $G_{\text{POSG}} = \langle \mathcal{S}', N, \{\mathcal{A}_i\}, P', \{R'_i\}, \{\Omega_i\}, \{O_i\} \rangle$ as follows.

- $\mathcal{S}' = \mathcal{S} \times \Xi \times \mathcal{R}^N$. An element of $\mathcal{S}'$ is a tuple $(s, i, \mathbf{r})$, where $\mathbf{r} = (r_1, r_2, \ldots, r_N)$ is a vector of rewards for each agent.
- $P'((s, i, \mathbf{r}), a_1, a_2, \ldots, a_N, (s', i', \mathbf{r}')) = \nu(s, i, a_i, s', i') \Pr(s' \mid s, i, a_i) \prod_{j \in [N]} R_j(s, i, a_i, s', \mathbf{r}'_i)$, where

$$\Pr(s' \mid s, i, a_i) = \begin{cases} 1 & \text{if } i > 0 \text{ and } T(s, a_i) = s' \\ P(s, s') & \text{if } i = 0 \\ 0 & \text{o/w (i.e., if } i = 0 \text{ and } T(s, a_i) \neq s') \end{cases}$$

- $R'_i((s, j, \mathbf{r}), a, (s', j', \mathbf{r}')) = \mathbf{r}'_i$

□

B ADDITIONAL REWARD PRUNING AND CURRICULUM LEARNING RESULTS

We ran PPO on pursuit with and without reward pruning. The total reward average for 4 runs increased by 5.4% with pruned rewards compared to unpruned rewards.

We included results for ApeX DQN on pursuit with curriculum learning. The total reward average for 4 runs increased by 13.2% after implementing curriculum learning.

Hyperparameters are included in Appendix C.

C HYPERPARAMETERS

All common hyperparameters are presented in Table 1. Method specific hyperparameters for PPO and ApeX DQN are given in Table 2 and the curriculum learning schedule is in Table 3.

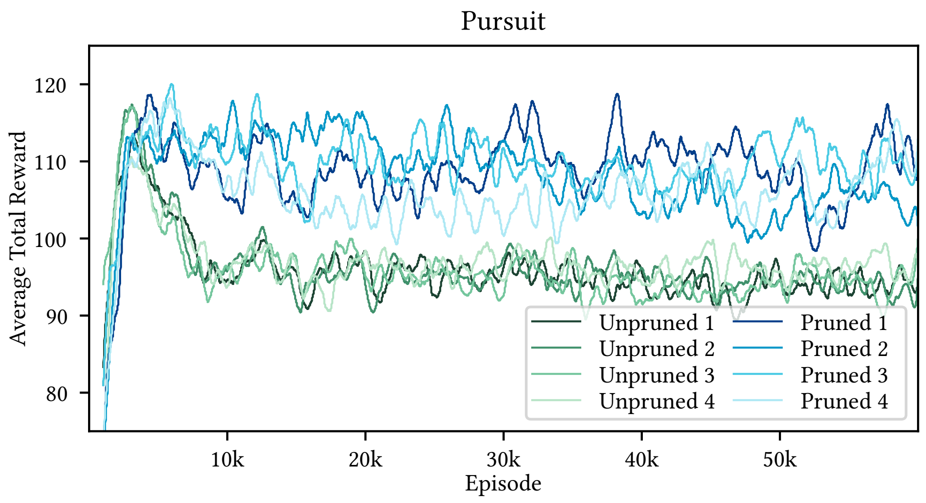


Figure 9: Learning on the *pursuit* environment with and without reward pruning, using parameter sharing based on PPO. Reward pruning increased the total reward by 5.4% on average.

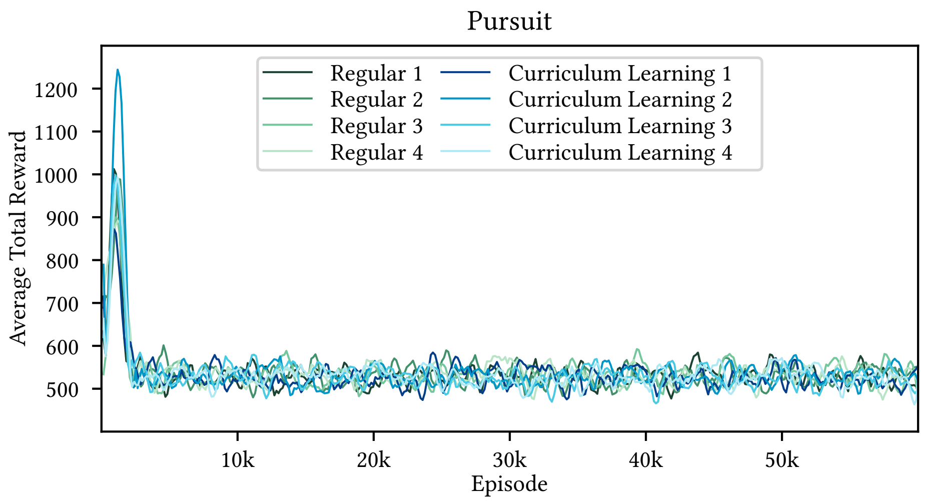


Figure 10: Learning on the *pursuit* environment with and without curriculum learning, using parameter sharing based on ApeX DQN. Curriculum learning increased the total reward by 13.2% on average.

Variable	Value set in all RL methods
# worker threads	8
# envs per worker	8
`gamma`	0.99
MLP hidden layers	[400, 300]

Table 1: Variables set to constant values across all RL methods for all RL games

RL method	Hyperparameter	Value for Pursuit
PPO	`sample_batch_size`	100
	`train_batch_size`	5000
	`sgd_minibatch_size`	500
	`lambda`	0.95
	`kl_coeff`	0.5
	`entropy_coeff`	0.01
	`num_sgd_iter`	10
	`vf_clip_param`	10.0
	`clip_param`	0.1
	`vf_share_layers`	`True`
	`clip_rewards`	`True`
	`batch_mode`	`truncate_episodes`
ApeX DQN	`sample_batch_size`	20
	`train_batch_size`	512
	`buffer_size`	100000
	`learning_starts`	1000
	`compress_observations`	`True`

Table 2: Hyperparameters for Pursuit

Step	Number of reward steps included
0	1
10	2
100	3
1000	8

Table 3: Curriculum learning schedule