AlegAATr the Bandit

Ethan Pedersen[a] **and Jacob Crandall**[a;*]

[a]Computer Science Department, Brigham Young University, USA
ORCiD ID: Jacob Crandall https://orcid.org/0000-0002-5602-4146

Abstract. One design strategy for developing intelligent agents is to create N distinct behaviors, each of which works effectively in particular tasks and circumstances. At each time step during task execution, the agent, or bandit, chooses which of the N behaviors to use. Traditional bandit algorithms for making this selection often (1) assume the environment is stationary, (2) focus on asymptotic performance, and (3) do not incorporate external information that is available to the agent. Each of these simplifications limits these algorithms such that they often cannot be used successfully in practice. In this paper, we propose a new bandit algorithm, called AlegAATr, as a step toward overcoming these deficiencies. AlegAATr leverages a technique called Assumption-Alignment Tracking (AAT), proposed previously in the robotics literature, to predict the performance of each behavior in each situation. It then uses these predictions to decide which behavior to use at any given time. We demonstrate the effectiveness of AlegAATr in selecting behaviors in three problem domains: repeated games, ad hoc teamwork, and a human-robot pick-n-place task.

1 Introduction

Autonomous agents situated in complex environments often have a set of behaviors (or generators) available to them. Created in any number of ways (e.g., hand-coded or via reinforcement learning), each generator specifies successful agent behaviors under some, but likely not all, circumstances. For example, a cooperative strategy in a prisoner's dilemma when paired with tit-for-tat would be successful, but the same strategy or behavior would struggle when paired with an associate that always defects [2]. Successful agents should be able to effectively determine which behavior generator for each situation they encounter.

Bandit algorithms are designed for such scenarios. At each time step, these algorithms select among N possible choices to maximize expected rewards when the consequences of each choice are stochastic and unknown beforehand [17]. Bandit algorithms are popular for a variety of reasons. First, decision-making in the face of uncertainty is a common challenge and is present in numerous fields, cultures, and applications. Furthermore, several practical uses of bandit algorithms exist in areas such as A/B testing [20], recommendation systems [15], clinical trials [23], and portfolio optimization [21].

Previous work has identified three key limitations of traditional bandit algorithms [17]. First, their performance guarantees typically hold only when the environment is stationary, an assumption that is frequently violated in real-world settings. Second, they often are tuned to asymptotic performance measures (e.g., no regret [1, 7, 4]). While high asymptotic performance is desirable, many real-world tasks require agents to have effective short-term performance as well, a capability that facilitates fast adaptation to changing environments. Third, traditional bandit algorithms typically rely completely on empirical (online) feedback, and do not incorporate useful external information that is available to them. This failure to incorporate external information can limit the agent's ability to learn quickly and to achieve higher short-term performance.

As a step toward overcoming these limitations, we introduce a new bandit algorithm, called AlegAATr (pronounced *aləgādər*). AlegAATr leverages Assumption-Alignment Tracking (AAT) [14, 6], a technique recently introduced in the robotics literature, to predict the future performance of each of the agent's behavior generators at any given time. In AAT, the agent tracks the veracity of the various assumptions that a generator relies on, and in this way forms a *veracity vector*. The resulting vector space forms a state space over which AlegAATr can continually reason about the effectiveness of its generators. Accordingly, AlegAATr does not assume its world is stationary. Rather, it uses external information, encoded by the veracity vector, to inform its selection of generators based on the agent's current environment.

To investigate both the applicability and effectiveness of AlegAATr, we implemented it in three separate problem domains: repeated games, ad hoc teamwork, and a human-robot pick-n-place task. Our results illustrate that the performance of AlegAATr meets or exceeds the state-of-the-art in each of these problem domains. In addition to demonstrating the effectiveness of AlegAATr itself, these results provide two additional contributions. First, they demonstrate that AAT produces a state space that allows simple decision-making rules, encoded in this case in AlegAATr, to produce effective choices in various bandit settings. Second, these results also indicate that AAT can be used effectively in a broader set of tasks and environments than those in which it was initially evaluated. Hence, it has potential use in a wide variety of AI applications.

2 Assumption-Alignment Tracking

Before defining AlegAATr, we review Assumption-Alignment Tracking (AAT) [14, 6], which AlegAATr uses to predict the performance of a generator G at any given time. At each time step t, the generator G takes as input an encoding of the state of the world, denoted s_t^G, and outputs some action a_t for the agent to execute in that time step. In producing the action a_t, we assume that G's planner produces an estimate of the expected (or purported) performance for running the generator, denoted $U_G(s_t^G)$. For example, if the generator uses value iteration or reinforcement learning, the purported

[*] Corresponding Author. Email: crandall@cs.byu.edu

performance is given by the value function (when taking action a_t in state s_t^G).

AAT is based on the idea that a generator is created under certain assumptions about the environment and the agent itself (e.g., its actuators and sensors). As such, the performance of the generator depends to a large degree on these assumptions being met. Thus, to predict the future effectiveness of a generator, the agent must be aware of the extent to which these assumptions are true. This can be accomplished by continually tracking the veracity of the assumptions upon which the generator relies, and then using these assessments to correct predictions of a generator's performance.

To track the veracity of the assumptions made in the construction of generator G, veracity assessments, made both before and during task execution, are calculated using *alignment checkers*. An alignment checker is a program that continually tracks the veracity of an assumption. Note that there may actually be multiple checkers for a single assumption (in fact, the implementations in the next sections demonstrate this), but for the sake of simplicity we will assume there is a bijection.

Formally, let $\Phi = \{\phi_1, \ldots, \phi_M\}$ denote the set of M assumptions upon which the generator relies. Then, let $x_t^{\phi_j} \in [0, 1]$ be the assessment, made by an alignment checker, of the veracity of assumption ϕ_j at time t. If the assumption is evaluated to be true, then $x_t^{\phi_j} = 1$; otherwise, $x_t^{\phi_j} = 0$. Real-valued assessments between 0 and 1 can reflect uncertainty in the truth of the assumptions. Given the individual assessments of assumption veracity, let

$$\mathbf{x}_t = \langle x_t^{\phi_1}, \ldots, x_t^{\phi_M} \rangle \tag{1}$$

denote the *veracity assessment vector* at time t.

The vector $\mathbf{x}_t$ provides an encoding of world state, which stands in contrast to s_t^G (the encoding of world state used by G). AAT uses $\mathbf{x}_t$ to predict the performance of generator G at time t by learning a *correction term* $\eta(\mathbf{x}_t, s_t^G)$ which encodes how violations in assumptions impact the performance of G. Specifically, the AAT prediction of G's performance on the task at time t (denoted $\hat{U}(\mathbf{x}_t, s_t^G)$) is computed as follows:

$$\hat{U}(\mathbf{x}_t, s_t^G) = \eta(\mathbf{x}_t, s_t^G) U(s_t^G). \tag{2}$$

The correction term is learned by a model (of choice) using training data that contains both normal (assumptions are met) and abnormal (assumptions are violated) instances. Each sample in the training data encodes the generator's current purported performance $U(s_t^G)$, the veracity assessment vector $\mathbf{x}_t$, and the actual performance in the training run. Once the learning process is complete, the model outputs the correction term $\eta(\mathbf{x}_t, s_t^G)$ for any condition (i.e, $\langle \mathbf{x}_t, s_t^G \rangle$-pair). The type of model is up to the designer; for the implementations described in this paper, we used kNN [11] (with $k = 15$ neighbors) and mixture models to learn the correction terms.

3 AlegAATr

While AAT evaluates the expected future performance of a single generator, we consider the case in which an agent has N generators at its disposal. Denote $\Gamma = \{G_1, \cdots, G_N\}$ as this set of generators. AlegAATr (*Al*gorithm that *e*valuates *g*enerators via *A*ssumption-*A*lignment *Tr*acking) uses the predictions of expected future performance made by AAT (Eq. 2) to select which generator $G_i \in \Gamma$ to use at any given time.

Algorithm 1 gives an overview of AlegAATr. Step 1 specifies the creation of generators available to the agent. The various generators could provide different ways to perform the same task, or they could each perform different sub-tasks of the task. Given the set of generators, Steps 2-4 specify the implementation of AAT for each generator as described in Section 2. The remaining steps of Algorithm 1 describe how AlegAATr selects generators at each time step t. First, AlegAATr uses the assumption checkers to generate the veracity assessment vector (Step 6). Then, based on this vector, AlegAATr predicts the performance of each generator $G_i \in \Gamma$ (Step 7). Finally, AlegAATr uses the predictions to determine which generator to select (Step 8).

Algorithm 1 AlegAATr

```
1: Create $\Gamma$, a set of $N$ generators
2: Determine the set of assumptions, $\Phi$, made in the creation of the
   generators. For each $\phi_j \in \Phi$, create a checker
3: $\forall G_i \in \Gamma$, identify a metric for computing $U_i(s_t^{G_i})$
4: $\forall G_i \in \Gamma$, train a model to compute $\eta(\mathbf{x}_t, s_t^{G_i})$
5: for $t = 1$ to $n$ do
6:     $\forall \phi_j \in \Phi$, compute $x_t^{\phi_j}$ (Eq. 1)
7:     For each $G_i \in \Gamma$, compute $\hat{U}_i(\mathbf{x}_t, s_t^{G_i})$ (Eq. 2)
8:     Select a generator $G^* \in \Gamma$ to follow (Eq. 3)
9: end for
```

In Step 8, AlegAATr selects the generator with the highest expected utility to following during time step t. Formally, let $\hat{P}_i(\mathbf{x}_t, s_t^{G_i})$ be the expected future performance of generator G_i. Then, AlegAATr determines G^* as follows:

$$G^* = \arg\max_{G_i \in \Gamma} \hat{P}_i(\mathbf{x}_t, s_t^{G_i}) \tag{3}$$

We consider two different methods for estimating $\hat{P}_i(\mathbf{x}_t, s_t^{G_i})$. In the first method, $\hat{P}_i(\mathbf{x}_t, s_t^{G_i})$ is based solely on the AAT prediction (Eq. 2). That is:

$$\hat{P}_i(\mathbf{x}_t, s_t^{G_i}) = \hat{U}_i(\mathbf{x}_t, s_t^{G_i}). \tag{4}$$

This approach is used under the assumption that AAT predictions are adequate for deciding the effectiveness of a generator at any given time. We call this method *AAT selection.*

However, because AAT predictions may not always be correct, we also consider a second method for computing $\hat{P}_i(\mathbf{x}_t, s_t^{G_i})$. This method instead uses AAT predictions to guide exploration. In this method (called *AAT exploration*), AlegAATr also tracks the (historical) empirical rewards obtained by each generator in the time periods in which it is used. In general, it selects generators with the highest empirical assessment. However, with some probability, it evaluates generators with respect to AAT predictions.

Formally, let $\bar{R}_t^{G_i}$ be the observed reward obtained when the agent has used generator G_i in prior time periods. Then,

$$\hat{P}_i(\mathbf{x}_t, s_t^{G_i}) = \begin{cases} \bar{R}_t^{G_i} & \text{with probability } \lambda^{n_i} \\ \hat{U}_i(\mathbf{x}_t, s_t^{G_i}) & \text{otherwise} \end{cases} \tag{5}$$

where $\lambda \in (0, 1)$, n_i is the number of time steps since G_i was used by AlegAATr (initially, $\forall i, n_i = \infty$), and $\bar{R}_1^{G_i} = 0$. The evolving probability λ^{n_i} makes it so that the AAT prediction for a generator is more likely to be considered if that generator has not been used for a while, thus encouraging occasional use (exploration) of generators that have recently been neglected and are predicted by AAT to perform well.

AlegAATr is intended to be general, such that it can be used in many different domains. To study the flexibility and effectiveness of AlegAATr, we conducted case studies in three distinct problem domains: repeated games (Section 4), ad hoc teamwork (Section 5), and a human-robot pick-n-place task (Section 6).

Table 1. Generators used in Implementation I for playing repeated games, the assumptions upon which each generator is based, and the number of checkers used to evaluate the veracity of these assumptions. Details about each of the generators and checkers are given in Appendix A.1.

Generator	**Description**	**Assumptions**	**# Checkers**
CFR	Plays the agent's portion of a one-shot Nash equilibrium	1. Associate plays a myopic best response to the agent's strategy	3
Maxmin	Plays the agent's maximin strategy	1. Associate tries to make the agent's payoffs as small as possible	2
Coop	Plays the agent's portion of the Nash Bargaining Solution	1. Associate wants to cooperate (insists on mutual cooperation) 2. Associate does not harm the agent if agent cooperates	4
Coop-Punish	Plays the agent's portion of the Nash Bargaining Solution as long as associate also does so. Plays the attack strategy if associate benefits from deviating	1. Associate is willing to cooperate 2. Punishing associate for defecting will cause associate to cooperate	3
Bullied	Plays the agent's portion of a Pareto optimal solution that most benefits the associate	1. Associate insists on the bully payoff 2. Associate does not harm the agent if agent conforms	3
Bully-Punish	Plays the agent's portion of a Pareto optimal solution that most benefits the agent as long as associate also does so. Plays the agent's attack strategy if associate benefits from deviating	1. Associate is willing to be bullied 2. Punishing associate for defecting will cause associate to cooperate	4

4 Case Study 1: Repeated Games

Over the last several decades, playing repeated games (RGs) has remained a challenge problem for agent development (e.g., [5, 9]). RGs are challenging because the environment is non-stationary as players adapt to each other. In addition to being theoretically interesting, RGs model relevant scenarios, including wireless networks (e.g., [16]) and human-robot relationships (e.g., [19]).

In this paper, we consider general-sum two-player RGs. An RG consists of a sequence of T rounds (where T may or may not be known to the players). In each round, the players play a sequence of joint actions until a terminal state is reached. The players then each receive a payoff for that round dependent on the sequence of joint actions played. The goal of each player is to select actions that lead to the maximization of their payoffs over all rounds of the RG. In this case, we consider that a player selects a generator at the start of a round which dictates the agent's actions throughout that round.

To better study AlegAATr, we analyze two different implementations within this domain. Due to space limitations, we only give a high-level overview of both implementations in this section. Full details are given in Appendix A.

4.1 *Implementation I*

In this implementation of AlegAATr for RGs, we seek to accurately predict the performance of each generator. To do this, we carefully evaluate each of the assumptions made in the creation of the generators available to the agent. We also provide the agent with extensive training data.

4.1.1 *Design Choices*

Design choices for this implementation for each step of Algorithm 1 are as follows:

Steps 1-2: The agent is given the six generators summarized in Table 1. Each of these generators computes strategies that are effective given certain assumptions about the agent's associate. For example, CFR [18] assumes the associate is myopic and will try to maximize its strategy in the current round. On the other hand, Coop-Punish, a generalized tit-for-tat strategy, assumes the associate is less myopic and willing to cooperate if it is in its best interest. Assumption checkers are detailed in the appendix.

Step 3: For CFR, $U(s_t^{G_i})$ is the value to the agent of the computed Nash equilibrium. For Maxmin, $U(s_t^{G_i})$ is the agent's maximin value. For other generators, $U(s_t^{G_i})$ is the expected payoff assigned to the agent in the computed target solution.

Table 2. Payoff matrices for two matrix games.

(a) Prisoner's Dilemma

	c	**d**
C	60, 60	0, 100
D	100, 0	20, 20

(b) Chicken

	a	**b**
A	84, 84	33, 100
B	100, 33	0, 0

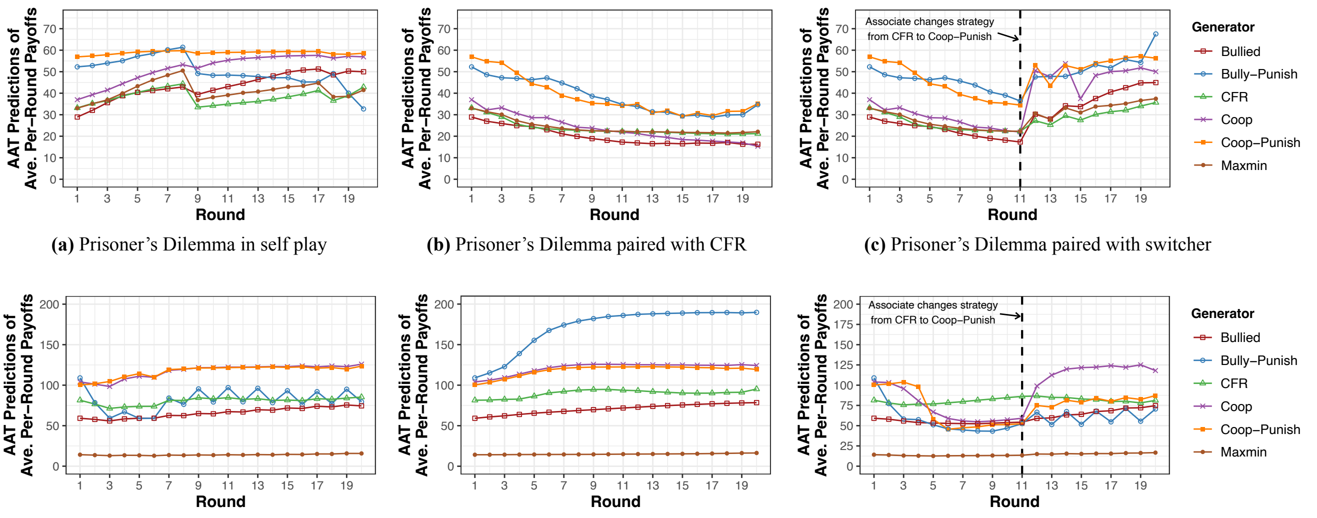
Step 4: AlegAATr was trained by simulating play of each generator against three sets of hypothetical associates: (1) each of the six generators (Table 1), (2) a set of adaptive agents that reactively select among the six generators in each round based on knowledge of the generator being played by their associate, and (3) and an agent that followed a (partially trained) version of AlegAATr. Given this data, AlegAATr estimates $\eta(\mathbf{x}_t, s_t^G)$ using the mixture model described in Appendix A.1.4.

Step 8: Uses AAT selection (Eqs. 3-4) to determine which generator to follow in each round.

4.1.2 *Experiments and Results*

To begin to understand the strategies produced by this implementation of AlegAATr, we first consider several scenarios in a repeated Prisoner's Dilemma (Table 2a). Figure 1a shows the AAT predictions made by AlegAATr for each of its six generators throughout a 20-round Prisoner's Dilemma in self-play. Initially, AlegAATr estimates that Coop-Punish will produce the highest per-round payoff, so it selects this generator through the first six rounds of the game. However, given that its associate is likewise cooperating, it begins to predict higher payoffs for Bully-Punish. By round 6, it predicts that Bully-Punish will produce higher payoffs than Coop-Punish, so AlegAATr switches to that strategy in rounds 6 and 7, which causes it to defect. However, when its associate retaliates in round 7, the predicted performance of some of its generators quickly change, such that Coop-Punish again is predicted to be the best generator for the remainder of the RG (Coop becomes a close second).

Alternatively, when paired with an associate that always defects (CFR), AlegAATr's performance predictions for each generator decrease over time (Figure 1b). Throughout the interaction, AlegAATR selects either Coop-Punish or Bully-Punish, which causes it to primarily defect, but to occasionally cooperate in hopes that its associate will understand the hint and also start cooperating. In the third scenario (Figure 1c), this indeed happens, as the associate switches to Coop-Punish in round 11. When this happens, AlegAATr immediately predicts higher payoffs for Coop-Punish and Coop, which leads to high degrees of cooperation.



(a) Prisoner's Dilemma in self play **(b)** Prisoner's Dilemma paired with CFR **(c)** Prisoner's Dilemma paired with switcher

(d) Block Dilemma in self play **(e)** Block Dilemma paired with Bullied **(f)** Block Dilemma paired with switcher

Figure 1. Predictions of per-round payoffs, or $\hat{U}(\mathbf{x}, s_t^{G_i})$, in the Prisoner's Dilemma (above) and the Block Dilemma (below) when paired with self (left), a stationary associate (middle), and an agent that suddenly switches its strategy from CFR to Bully-Punish (right).

Figures 1d-f show that AlegAATr achieves similar results in a repeated stochastic game called the Block Dilemma [19, 24] (alternatively, see Appendix A.4). In this game, AlegAATr's strategy encodes optimism in uncertainty. That is, it selects Bully-Punish in round 1. In self play, it quickly abandons this strategy and converges to mutual cooperation (Figure 1d). However, when associating with Bullied, it learns to continue to use this generator (Figure 1e). Finally, when paired with a myopic associate (CFR), it learns within a few rounds to play its portion of the one-shot Nash equilibrium by likewise playing CFR (Figure 1f). However, when the associate switches in round 11 to Coop-Punish, AlegAATr immediately predicts higher payoffs for Coop, leading it to likewise cooperate.

These results demonstrate that AlegAATr learns to (1) cooperate with like-minded associates, (2) exploit associates that allow it to do so, (3) play myopic best responses when necessary, and (4) quickly adopt new strategies when its associate changes strategies. The ability of AlegAATr to learn these behaviors is intriguing, as producing algorithms with all of these characteristics has been challenging (e.g., [12, 9]).

To further study the effectiveness of AlegAATr, we compare it with other expert algorithms for RGs. We selected three baseline comparisons: BBL, EEE [10], and S++ [8]. BBL and EEE provide comparisons to commonly used (greedy) utility-maximization techniques. BBL, or belief-base learning (e.g., Fictitious Play [13]), forms a probability distribution over the generators its associate could be using and then selects the generator that maximizes its expected payoffs given those probability estimates. On the other hand, EEE encodes an ε-greedy strategy based on the historical performance of each generator. Finally, S++ provides a comparison to the state-of-the-art in RGs, as its was the highest-performing algorithm in a comparison of 25 different algorithms [9].

We paired each of the algorithms against itself and the other three algorithms in three different 20-round RGs: the Block Dilemma, a Prisoner's Dilemma, and Chicken (Table 2). Figure 2 shows the normalized per-round payoffs obtained by each of the four algorithms across all pairings in each game. The results show that AlegAATr performs as well as or better than the other algorithms in each game.

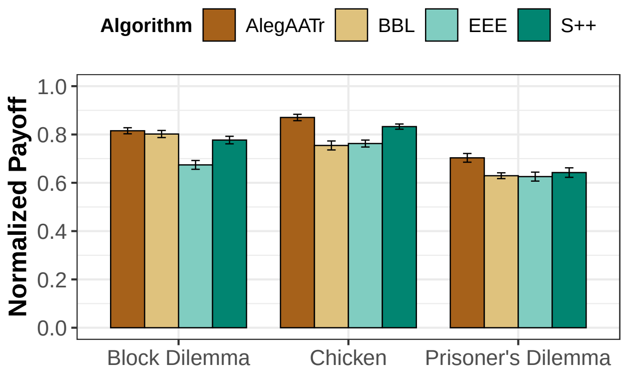


Figure 2. Normalized payoffs when algorithms were paired with each other in three different RGs. We conducted 60 trials for each pairing. Error bars show the standard error of the mean.

Across the three games, AlegAATr obtained higher payoffs than EEE ($p < 0.001$), BBL ($p < 0.001$), and S++ ($p = 0.004$).

4.2 *Implementation II*

In our second implementation of AlegAATr for RGs, we reduced implementation effort and training (compared to implementation 1). This results in a less careful evaluation of the generators, likely resulting in less accurate predictions.

4.2.1 *Design Choices*

Design choices for this implementation for each step of Algorithm 1 are as follows:

Steps 1-2: The agent is given a similar set of generators as in Implementation I (Table 1), but with one additional generator. To reduce implementation complexity, we used just six checkers (as opposed to 19) to evaluate assumptions. See Appendix A.2 for details.

Step 3: $U(s_t^{G_i})$ was computed as in Implementation I.

Step 4: AlegAATr was trained against two kinds of associates. First, each generator $G_i \in \Gamma$ played against every other $G_j \in \Gamma$. Second, we again trained on each generator $G_i \in \Gamma$, but randomly changed the associate's strategy to another $G_j \in \Gamma$ after each round. Given this data, AlegAATr estimates $\eta(\mathbf{x}_t, s_t^G)$ using a kNN algorithm ($k = 15$).

Step 8: Uses AAT exploration (defined by Eqs. 3 and 5) to determine which generator to follow in each round. To give the associate time to adapt to AlegAATr's strategy, we only considered switching generators every two rounds. Additionally, AlegAATr played Coop for the first two rounds to signal to its associate that it was willing and able to cooperate.

4.2.2 Experiments and Results

To evaluate the effectiveness of this implementation of AlegAATr, we conducted a series of experiments in five RGs: Block Dilemma, Chicken, Coordination, Matching Pennies, and Prisoner's Dilemma (see Appendix A.4 for details). Similar to Implementation I, we compared the payoffs achieved by AlegAATr with those of BBL, EEE, and S++ in 50-round interactions.

We first compared these algorithms when they are paired with four different sets of associates: (1) *Train*: Associates that follow a single generator $G_i \in \Gamma$ (drawn from the same set of generators as AlegAATr used) for the entirety of the RG; (2) *Test*: Other non-learning algorithms that are not in Γ (these agents are described in Appendix A.3); (3) *Self-play*; and (4) *Learners*: BBL, EEE, and S++. For each game and agent pairing, we ran 500 simulations. Table 3 compares the normalized payoffs obtained by AlegAATr with those of BBL, EEE, and S++ in each category. Across all five games, AlegAATr outperforms the other algorithms in every category, though it was not statistically superior to BBL against associates in the *Test* category ($p = 0.831$).

Finally, we conducted a user study in which we paired the algorithms with humans in the Block Dilemma. In this study, each participant played eight 20-round games, two against each algorithm. Participants were paid according to the payoffs they received in the games. The ordering of the match-ups was randomized, such that participants did not know the identify of their associates. To encourage variation in playing styles, we biased participants in three different ways: (1) *No bias* (13 participants): We gave no strategic suggestions to participants; (2) *Bully bias* (10 participants): We suggested to participants that they may make more money by bullying their associate.; and (3) *Coop bias* (10 participants): We suggested to participants that they may make more money by cooperating with their associate.

Results of this study are summarized in Table 4. When paired with people in the Block Dilemma, AlegAATr typically does at least as well as the other algorithms (though it was only statistically better than EEE). Combined with the superior scores when paired with non-human associates, AlegAATr indeed seems to be a fairly robust algorithm.

4.3 Discussion

Implementations I and II embody two different approaches. Implementation 1 used a more detailed set of checkers and a more thorough training process. These checkers and training process take more time to create, but lead to accurate AAT predictions that adapt quickly to changes in the associate's strategy (Figure 1). On the other hand, Implementation 2 used fewer checkers and a less thorough training process. The subsequent predictions made by AAT in this implementation were not as accurate, but were still good enough to effectively guide exploration. Interestingly, both implementations produce similar results: they consistently perform as well as or better than the baseline algorithms (BBL, EEE, and S++).

Table 3. Differences between AlegAATr's normalized rewards and those of BBL, EEE, and S++ when paired with other agents. A positive difference is in favor of AlegAATr. * indicates statistical significance from Tukey-Kramer comparisons ($p < 0.05$).

	BBL	**EEE**	**S++**
Train	$0.02^*, p < 0.001$	$0.17^*, p < 0.001$	$0.20^*, p < 0.001$
Test	$0.00, p = 0.596$	$0.11^*, p < 0.001$	$0.09^*, p < 0.001$
Self-play	$0.21^*, p < 0.001$	$0.28^*, p < 0.001$	$0.14^*, p < 0.001$
Learners	$0.02^*, p < 0.001$	$0.02^*, p < 0.001$	$0.03^*, p < 0.001$

Table 4. Differences between AlegAATr's normalized rewards and those of BBL, EEE, and S++ when paired with people in the Block Dilemma. A positive difference is in favor of AlegAATr. * indicates statistical significance from Tukey-Kramer comparisons ($p < 0.05$).

	BBL	**EEE**	**S++**
No Bias	$1.45, p = 0.846$	$4.92^*, p = 0.033$	$2.17, p = 0.615$
Bully Bias	$0.03, p = 0.999$	$4.79, p = 0.133$	$1.67, p = 0.869$
Coop Bias	$-0.68, p = 0.981$	$5.24^*, p = 0.024$	$0.81, p = 0.969$
Overall	$0.41, p = 0.986$	$4.98^*, p < 0.01$	$1.62, p = 0.512$

5 Case Study 2: Ad Hoc Teamwork

We next consider an ad hoc teamwork setting [22], in which an agent must coordinate its behavior with unknown teammates. In particular, we consider a predator-prey domain [3] where four predators work together to capture a prey (by surrounding it on all four sides) in as few moves as possible. AlegAATr selects generators to control one of these agents.

5.1 Design Choices

Design choices for each step of Algorithm 1 are as follows:

Steps 1-2: We selected three algorithms for this domain from [3]: Team Aware, Greedy, and Greedy Probabilistic. Results from Barrett *et al.* showed that Team Aware had high performance, whereas Greedy and Greedy Probabilistic had lower performance. We identified five assumptions upon which these generators are based, and then created one checker for each of these assumptions.

Step 3: Purported performance $U(s_t^{G_i})$ was determined as the average number of rounds required to surround the prey on a 5×5 grid when all predators used the generator. For example, it takes 5.003 rounds (on average) for Team Aware predators to surround the prey in a 5 x 5 grid.

Step 4: Training data was obtained in three training phases on 5×5 and 10×10 grids. In phase one, we paired each generator $G_i \in \Gamma$ with three other predators that used the same generator. In phase two, we paired each $G_i \in \Gamma$ with three other predators that each used the same randomly-selected $G_j \in \Gamma$. In phase three, we paired each $G_i \in \Gamma$ with three predators who each used their own generator $G_j \in \Gamma$ (randomly-selected). We produced 100 training runs for

Table 5. Differences between the (normalized) time to surround the prey between AlegAATr and other top-performing algorithms. A negative difference favors AlegAATr (fewer rounds to surround the prey is better). p-values are from Tukey-Kramer comparisons.

	Greedy Planner	**Min Sum**	**Team Aware**
Deterministic	0.01, $p = 0.825$	0.02, $p = 0.363$	0.00, $p = 0.999$
Non-Determ.	0.01, $p = 0.999$	0.01, $p = 0.998$	0.00, $p = 0.999$
Mixed	0.00, $p = 0.999$	0.02, $p = 0.905$	-0.01, $p = 0.999$
Overall	0.01, $p = 0.918$	0.01, $p = 0.270$	0.00, $p = 0.999$

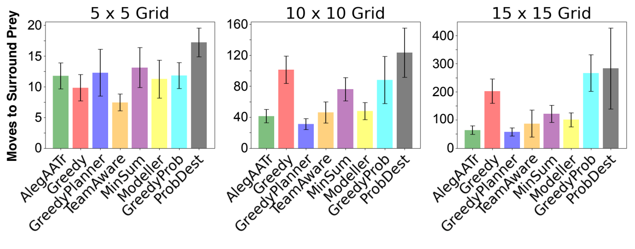


Figure 3. Moves to surround the prey with mixed teams on different grid sizes. Note the change in scale of the y-axis across the three figures. Error bars show the standard error.

each generator in each phase. AlegAATr estimated $\eta(\mathbf{x}_t, s_t^G)$ using a kNN algorithm ($k = 15$).

Step 8: Uses AAT exploration (defined by Eqs. 3 and 5); a randomly-selected generator is used in the first two rounds.

Implementation details are provided in Appendix B.

5.2 *Experiments and Results*

To evaluate AlegAATr's ability to select a good generator given the behavior of its teammates, we compare its performance with that of seven other algorithms: Greedy, Greedy Planner, Team Aware, Min Sum, Modeller, Greedy Probabilistic, and Probabilistic Destinations. Details for each of these algorithms is given in Appendix B. Tests were conducted on 5×5, 10×10, and 15×15 grids. On each grid size, we paired each of the eight algorithms with three teammates. These three teammates were constructed in three different ways: (1) *Deterministic*: Teammates all used the same deterministic (Greedy, Greedy Planner, Team Aware, or Min Sum) strategy; (2) *Non-Deterministic*: Teammates all used the same non-deterministic strategies (Greedy Probabilistic, Modeller, or Probabilistic Destinations); (3) *Mixed*: Each teammate randomly used one of the seven strategies.

Given that Team Aware, Greedy Planner, and Min Sum all perform well in this domain [3], we were interested to determine whether AlegAATr would coordinate with its teammates as well as these high-performing algorithms. Comparisons over all grid sizes between AlegAATr and these high-performing algorithms are shown in Table 5. The results (averaged over 30 trials in each condition) show that there was little difference between AlegAATr and these high-performing algorithms. Additionally, AlegAATr performed statistically better than the other four algorithms for every category of teammate ($p < 0.01$ in all cases).

Figure 3 shows more detailed results for *Mixed* teams, or teams in which AlegAATr's teammates all used different (randomly selected) behaviors. Ideally, AlegAATr should perform at least as well as its best generator (Greedy, Team Aware, and Greedy Probabilistic). AlegAATr takes a few more moves, on average, than if it had just always followed Team Aware on 5×5 grids, an outcome that could brought

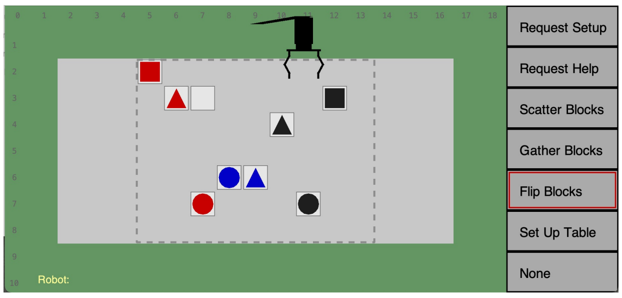


Figure 4. A robot is tasked with arranging blocks on a table.

about by a single misstep. However, on the larger grid sizes, its performs better on average (though error bars overlap) than if it had just stuck with any one of its generators. This is true even on 15×15 grids in which it was not trained on. Overall, these results indicate that AlegAATr is able to learn to effectively choose between its generators in this problem domain.

6 Case Study 3: Robot Pick-n-Place Task

In the third scenario, we consider a simulated robot, equipped with a gripper, which is tasked with arranging blocks on a table (Figure 4) as quickly as possible. A human bystander, who is potentially more efficient at performing the task, may also be available to (help) perform the task. However, the robot does not know if the human bystander is willing and able to do so. Thus, the robot must decide to do the task itself or to ask the human bystander to do it.

6.1 *Design Choices*

Design choices for each step of Algorithm 1 are as follows:

Steps 1-2: The robot has six generators (Table 6). In *Set Up Table*, the robot attempts to place the blocks in their correct locations. However, this generator only fully succeeds when blocks are not too close to each other (so it can grip them), are facing upward in the middle of the table, and are reachable (the robot's arm is not long enough for it to reach all places on the table). *Flip Blocks*, *Gather Blocks*, and *Scatter Blocks* are behavior generators available to the robot that are designed to resolve these anomalies, though there are cases when they fail. Alternatively, the robot has generators in which it either asks a human bystander to resolve the anomalies (*Request Help*) or to just set the blocks in the correct locations on the table (*Request Setup*). The generators, assumptions, and checkers are detailed in Appendix C.

In case studies 1 and 2, each generator was designed to perform the full task. However, in this task, some of the available generators only perform certain sub-tasks (e.g., resolve a particular anomaly). Thus, to be successful in this case study, AlegAATr must (1) sequence its generators to complete the task (e.g., scatter, flip, and gather blocks before arranging them) while (2) deciding between multiple ways of carrying out the task (robot manipulation vs. asking the human bystander to do it).

Step 3: $U(s_t^{G_i})$, determined empirically, is the amount of work done (blocks placed or anomalies resolved) over 20 seconds.

Step 4: Each generator was trained in 16 random worlds as well as five custom worlds. The responsiveness of the (simulated) bystander was systematically varied in these training runs to be unresponsive

Table 6. Generators available to the robot.

Generator	Description
Set Up Table	Robot attempts to put blocks in the middle of the table into place
Flip Blocks	Robot attempts to flip overturned blocks that are in the middle area of the table
Gather Blocks	Robot attempts to move blocks to the middle
Scatter Blocks	Robot attempts to separate blocks that are next to each other
Request Help	Robot asks bystander to resolve anomalies
Request Setup	Robot asks human bystander to set up table

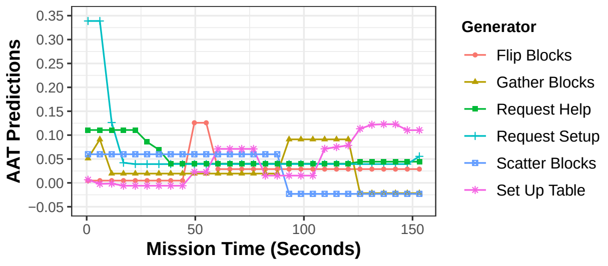


Figure 5. Predictions of work done per unit time, or $\hat{U}(\mathbf{x}_t, s_t^{G_i})$, for each generator in a pick-n-place scenario.

(does not respond to any robot requests), fully responsive (does whatever the robot asks), or just helpful (solves anomalies when asked). Additionally, in some training scenarios, the bystander was malicious in that it moved blocks to hinder the robot's progress. The speed at which the human bystander acted was also varied. Given this data, AlegAATr estimates $\eta(\mathbf{x}_t, s_t^G)$ using a mixture model, the details of which are given in Appendix C.4.

Step 8: Uses AAT selection (Eqs. 3-4) to determine which generator to use. AlegAATr made a new selection every five seconds.

6.2 Experiments and Results

Figure 5, which shows $\hat{U}(\mathbf{x}_t, s_i^{G_i})$ for each generator over time in a scenario with an unresponsive bystander, illustrates the strategy chosen by AlegAATr. For the first 15 seconds, AlegAATr predicts that the *Request Setup* generator will have the highest utility. Thus, it selects this behavior, which lobbies the bystander to set up the table. When the bystander does not respond to these requests, the utility of using this generator decreases, such that the *Request Help* generator is then predicted to have the highest utility. Thus, AlegAATr selects this generator, which causes the robot to ask the bystander to correct anomalies for the next 20 seconds. When the bystander still does not respond, generators in which the robot manipulates the blocks itself are estimated to have the highest predicted performance. These behaviors are subsequently selected and sequenced by AlegAATr in a way that allows it to complete the task in just over 150 seconds.

To evaluate how well AlegAATr selects generators in this task, we compared its performance with that of people. Specifically, we compared how well AlegAATr and human participants selected generators in three different conditions, determined by the responsiveness of the simulated human bystander. A *fully responsive* bystander adhered to the requests of the robot (resolve anomalies or set up the table), whereas a *helpful* bystander only resolved anomalies (upon request). An *unresponsive* bystander did not respond to any request

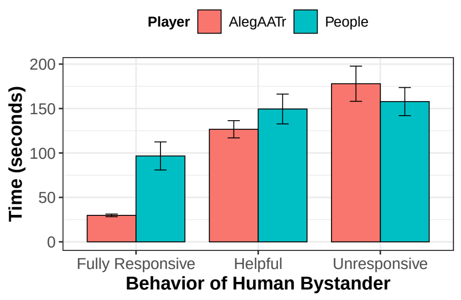


Figure 6. Time to complete the task when generators are selected by AlegAATr and People given different human bystanders.

from the robot. The pace at which the human bystander acted was set at a pace that was distinct from the training scenarios.

Twelve people participated in the study. These participants selected generators by clicking buttons on a GUI (Figure 4). Each human participant performed the task under each of the three conditions. The order the subjects experienced the conditions was counterbalanced across participants. While strategies differed across human participants, participants were in general not inclined to have the robot ask the bystander to do any work. In general, they tasked the robot with setting up the blocks. This behavior stands in contrast to the behavior learned by AlegAATr (illustrated in Figure 5).

The difference in strategy between AlegAATr and the typical human participant produced different performance profiles across the three conditions of the study (Figure 6). Given an unresponsive bystander, AlegAATr took slightly longer on average to complete the task than people (not statistically different; $p = 0.436$) since AlegAATr spends the first 30 seconds or so asking the (unresponsive) bystander for help (to no avail). However, when the bystander is responsive, this strategy produces better results. AlegAATr was, on average, faster than people when the bystander was helpful (not statistically significant; $p = 0.251$) or fully responsive ($p < 0.001$). These results indicate that AlegAATr learned to both successfully stitch together incomplete generators while also adapting to different environmental conditions.

7 Conclusion

In this paper, we introduced a new bandit algorithm called AlegAATr. AlegAATr leverages Assumption-Alignment Tracking (AAT), a technique proposed in the robotics literature to perform proficiency self-assessment [14, 6], to predict the performance of each of the N behaviors that are available to it. It then uses these predictions to select generators at any given time. To evaluate the effectiveness of AlegAATr, we tested it in three distinct problem domains. Results presented in this paper, when taken as a whole across all experiments conducted in all three domains, show that AlegAATr selects generators as well as or better than baseline comparisons. These results indicate that the veracity assessment vector encoded by AAT (and used by AlegAATr) provides a robust and flexible encoding of state that leads to effective decision-making in a variety of bandit settings. Additionally, these results also indicate that AAT can be used effectively in a broader set of tasks and environments than those in which it was initially evaluated. Hence, we believe that it has potential use in a wide variety of AI applications.

Acknowledgments

This work was supported in part by the U.S. Office of Naval Research (ONR) under Grant #N00014-18-1-2503.

References

[1] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, 'Gambling in a rigged casino: the adversarial multi-armed bandit problem', in *Proceedings of the 36th Symposium on the Foundations of Computer Science*, pp. 322–331, (1995).

[2] R. Axelrod and W. D. Hamilton, 'The evolution of cooperation', *Science*, **211**(4489), 1390–1396, (1981).

[3] S. Barrett, P. Stone, and S. Kraus, 'Empirical evaluation of ad hoc teamwork in the pursuit domain', in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 567–574, (2011).

[4] M. Bowling, 'Convergence and no-regret in multiagent learning', in *Advamces in Neural Information Processing Systems*, pp. 209–216, (2004).

[5] A. Burkov and B. Chaib-Draa, 'Repeated games for multiagent systems: a survey', *The Knowledge Engineering Review*, **29**(1), 1–30, (2014).

[6] X. Cao, A. Gautam, T. Whiting, S. Smith, M. A. Goodrich, and J. W. Crandall, 'Robot proficiency self-assessment using assumption-alignment tracking', *IEEE Transactions on Robotics*, (2023).

[7] Y. Chang and L. P. Kaelbling, 'Hedge learning: Regret-minimization with learning experts', in *Proceedings of the 22nd International Conference on Machine Learning*, pp. 121–128, (2005).

[8] J. W. Crandall, 'Towards minimizing disappointment in repeated games', *Journal of Artificial Intelligence Research*, **49**, 111–142, (2014).

[9] J. W. Crandall, M. Oudah, F. Ishowo-Oloko, S. Abdallah, J.-F. Bonnefon, M. Cebrian, A. Shariff, M. A. Goodrich, and I. Rahwan, 'Cooperating with machines', *Nature communications*, **9:233**(1), 233, (2018).

[10] D. P. De Farias and N. Megiddo, 'Combining expert advice in reactive environments', *Journal of the ACM (JACM)*, **53**(5), 762–799, (2006).

[11] Evelyn Fix and Joseph Lawson Hodges, 'Discriminatory analysis. nonparametric discrimination: Consistency properties', *International Statistical Review/Revue Internationale de Statistique*, **57**(3), 238–247, (1989).

[12] J. Foerster, R.Y. Chen, M. Al-Shedivat, S. Whiteson, P. Abbeel, and I. Mordatch, 'Learning with opponent-learning awareness', in *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*, pp. 122–130, (2018).

[13] D. Fudenberg and D. K. Levine, *The Theory of Learning in Games*, The MIT Press, 1998.

[14] A. Gautam, T. Whiting, X. Cao, M. A. Goodrich, and J. W. Crandall, 'A method for designing autonomous robots that know their limits', in *International Conference on Robotics and Automation (ICRA)*, pp. 121–127, (2022).

[15] D. Glowacka, 'Bandit algorithms in recommender systems', in *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 574–575, (2019).

[16] D. T. Hoang, X. Lu, D. Niyato, P. Wang, D. I. Kim, and Z. Han, 'Applications of repeated games in wireless networks: A survey', *IEEE Communications Surveys & Tutorials*, **17**(4), 2102–2135, (2015).

[17] T. Lattimore and C. Szepesvári, *Bandit algorithms*, Cambridge University Press, 2020.

[18] T. W. Neller and M. Lanctot, 'An introduction to counterfactual regret minimization', in *Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence (EAAI-2013)*, volume 11, (2013).

[19] M. Oudah, V. Babushkin, T. Chenlinangjia, and J. W. Crandall, 'Learning to interact with a human partner', in *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 311–318. IEEE, (2015).

[20] S. Satyal, I. Weber, H. Paik, C. Di Claudio, and J. Mendling, 'Ab testing for process versions with contextual multi-armed bandit algorithms', in *International Conference on Advanced Information Systems Engineering*, pp. 19–34. Springer, (2018).

[21] W. Shen, J. Wang, Y.-G. Jiang, and H. Zha, 'Portfolio choices with orthogonal bandit learning', in *Twenty-fourth international joint conference on artificial intelligence*, (2015).

[22] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein, 'Ad hoc autonomous agent teams: Collaboration without pre-coordination', in *Proceedings of the 24th Conference on Artificial Intelligence*, (2010).

[23] S. S. Villar, J. Bowden, and J. Wason, 'Multi-armed bandit models for the optimal design of clinical trials: benefits and challenges', *Statistical science: a review journal of the Institute of Mathematical Statistics*, **30**(2), 199, (2015).

[24] T. Whiting, A. Gautam, J. Tye, M. Simmons, J. Henstrom, M. Oudah, and J. W. Crandall, 'Confronting barriers to human-robot cooperation: balancing efficiency and risk in machine behavior', *Iscience*, **24**(1), 101963, (2021).

AlegAATr the Bandit

(Technical Appendix)

Ethan Pedersen[a] **and Jacob Crandall**[a;*]

[a]Computer Science Department, Brigham Young University, USA
ORCiD ID: Jacob Crandall https://orcid.org/0000-0002-5602-4146

A Two-Player Repeated Games – Additional Details

In this section of the appendix, we give implementation details for both implementations of AlegAATr for repeated games. In so doing, we describe the generators, assumptions, and checkers used in the studies. We also detail the training data that was gathered to train both versions of the algorithm. After describing the implementations of AlegAATr, we also describe details related to the experiments conducted in this case study, including descriptions of the games used as well as implementation details for other algorithms.

A.1 Implementation 1

Code for this first implementation can be found at https://github.com/jakecrandall/AlegAATr_rg1.

A.1.1 Generators

We used a similar set of generators in each implementation. However, small differences exist between these implementations, so we detail them each separately. The first implementation used six distinct generators:

1. *CFR*: Computes a Nash equilibrium using counterfactual regret (CFR). The generator plays the agent's strategy in that solution. We used the algorithm in [7].
2. *Maxmin*: Compute the maximin strategy
3. *Coop*: Computes the Nash Bargaining Solution (NBS) by solving a set of MDPs as defined in [4] and [3]. The agent selects the strategy computed for the MDP that maximizes the product of the agents' advantages, as in [3]. The agent then plays its strategy in the Nash Bargaining Solution.
4. *Coop-Punish*: Computes the Nash Bargaining Solution (NBS) by solving a set of MDPs as defined in [4]. The agent selects the strategy for the MDP that maximizes the product of the agents' advantages, as in [3]. The agent then plays its strategy in the Nash Bargaining Solution as long as the associate plays its portion of the solution. If the associate deviates from the strategy and thereby benefits (receiving a higher payoff than it otherwise would have), the agent plays its attack strategy, as defined by Littman and Stone [6] and Crandall [3], until the associate has no longer benefited.
5. *Bullied*: As in Coop, the agent solves a set of MDPs as defined in [4] and [3]. Among the solutions that give both players higher strategies than their maximin values, the agent selects the solution that gives the highest payoff to the associate. The agent plays its strategy corresponding to that solution.
6. *Bully-Punish*: As in Coop, the agent solves a set of MDPs as defined in [4] and [3]. Among the solutions that give both players higher strategies than their maximin values, the agent selects the solution that gives the highest payoff to the agent. The agent plays its strategy corresponding to that solution as long as the associate plays its portion of that solution. If the associate deviates from the strategy and thereby benefits (receiving a higher payoff than it otherwise would have), the agent plays its attack strategy, as defined by Littman and Stone [6] and Crandall [3], until the associate has no longer benefited.

A.1.2 Alignment Checkers

As set of alignment checkers was constructed to continually evaluate the veracity of each assumption listed in Table 1 of the main paper. Additionally, a progress checker [2] was created for each generator. We begin by defining each of the alignment checkers used for each generator.

CFR: We used two alignment checkers for this generator to evaluate whether the associate plays a myopic best response (which maximizes the associate's round payoff given the strategy played by the agent):

- The first of these checkers is initially set to 1, and then updated after each round as follows:
$$x_t^{\text{cfr\_br1}} = 0.8x_{t-1}^{\text{cfr\_br1}} + 0.2k,$$
where $k = 1$ if the associate played a bast response to the agent's strategy in that round and $k = 0$ otherwise.
- The second of these checkers is updated identically except that it is only updated in rounds in which the agent uses CFR. Thus,
$$x_t^{\text{cfr\_br2}} = \begin{cases} 0.8x_{t-1}^{\text{cfr\_br2}} + 0.2k & \text{if agent used CFR} \\ x_{t-1}^{\text{cfr\_br2}} & \text{otherwise} \end{cases}$$

Maxmin (mx): We used two checkers for this generator. The first checker evaluates whether the associate plays the attack strategy. It

* Corresponding Author. Email: crandall@cs.byu.edu

is initially set at 1, and then updated each round as follows:

$$x_t^{\text{mx\_harm}} = \begin{cases} 0.8x_{t-1}^{\text{mx\_harm}} + 0.2 & \text{if } r_t < r^{\text{mm}} \\ 0.8x_{t-1}^{\text{mx\_harm}} & \text{otherwise} \end{cases}$$

where r_t is the round payoff obtained by the agent in round t and r^{mm} is its maximin value.

Coop (C): We used three alignment checkers to evaluate the assumptions made for this generator:

- The first checker evaluates whether the associate's actions seem to indicate that it wants to cooperate. Initially, this checker is set to 1, and then updates after each round as follows:

$$x_t^{\text{c\_wants}} = \begin{cases} 0.2x_{t-1}^{\text{c\_wants}} + 0.8 & \text{if nice} \\ 0.9x_{t-1}^{\text{c\_wants}} + 0.1 & \text{if cooperated} \\ 0.8x_{t-1}^{\text{c\_wants}} & \text{otherwise} \end{cases}$$

  where *nice* indicates that the associate played in a way that would have boosted the agent's payoffs while not benefiting themselves (did worse than the value they would have gotten from mutual cooperation), and *cooperated* indicates the associate otherwise played the cooperative action.
- The second checker identifies whether the associate cooperates when the agent plays Coop. It is initially set to 1, and then is updated after each round that generator Coop is used, as follows:

$$x_t^{\text{c\_mirrors}} = \begin{cases} 0.7x_{t-1}^{\text{c\_mirrors}} + 0.3 & \text{if cooperated} \\ 0.7x_{t-1}^{\text{c\_mirrors}} & \text{otherwise} \end{cases}$$

  When Coop is not used in a round, then $x_t^{\text{c\_mirrors}} = x_{t-1}^{\text{c\_mirrors}}$.
- The third checker evaluates whether or not the associate exploits the agent when it cooperates. It is initially set to 1, and then is updated in each round that generator Coop is used, as follows:

$$x_t^{\text{c\_notexploit}} = \begin{cases} 0.7x_{t-1}^{\text{c\_notexploit}} & \text{if } r_t^{-i} > \bar{V}^{-i} \\ 0.8x_{t-1}^{\text{c\_notexploit}} + 0.2 & \text{otherwise} \end{cases}$$

  where r_t^{-i} is the associates payoff in round t and $\bar{V}^{-i}$ is the payoff the associate should get when it cooperates in the round. When Coop is not used in a round, then $x_t^{\text{c\_notexploit}} = x_{t-1}^{\text{c\_notexploit}}$.

Coop-Punish (CP): We used two alignment checkers to evaluate the two assumptions made in the construction of this generator:

- The first checker identifies whether the associate cooperates when the agent plays Coop-Punish. It is initially set to 1, and then is updated after each round that generator Coop-Punish is used, as follows:

$$x_t^{\text{cp\_mirrors}} = \begin{cases} 0.7x_{t-1}^{\text{cp\_mirrors}} + 0.3 & \text{if cooperated} \\ 0.7x_{t-1}^{\text{cp\_mirrors}} & \text{otherwise} \end{cases}$$

  When Coop-Punish is not used in a round, then $x_t^{\text{cp\_mirrors}} = x_{t-1}^{\text{cp\_mirrors}}$.
- The second checker evaluates whether punishment in a round leads to the associate cooperating in the next round. It is initially set to 1, and then each time the generator plays the attack strategy, it updates based on whether the associate cooperates in the next round. Thus, it updates as follows:

$$x_t^{\text{cp\_responds}} = \begin{cases} 0.5x_{t-1}^{\text{c\_responds}} + 0.5 & \text{if cooperated} \\ 0.5x_{t-1}^{\text{c\_responds}} & \text{otherwise} \end{cases}$$

  If the agent did not punish in the current or previous round, then $x_t^{\text{c\_responds}} = x_{t-1}^{\text{c\_responds}}$.

Bullied (BD): We used to two alignment checkers to evaluate the assumptions made in the design of this generator:

- The first checker evaluates whether the associate ensures that the agent's average payoff is less than or equal to what it would get in the bullied payoff. Initially, this checker is set to one. Then, in each round it is updated as follows:

$$x_t^{\text{bd\_insists}} = \begin{cases} 0.8x_{t-1}^{\text{bd\_insists}} + 0.2 & \text{if } \bar{R}_t \leq \bar{V}_{\text{bullied}} \\ 0.8x_{t-1}^{\text{bd\_insists}} & \text{otherwise} \end{cases}$$

  where $\bar{V}_{\text{bullied}}$ is the average per-round payoff the agent gets in this solution and $\bar{R}_t$ is the average per-round payoff the agent has received up to time t.
- The second checker evaluates the second assumption we have articulated for this generator, which is that the associate will not *harm* the agent if the agent conforms with the target solution. By harm, we mean that the associate will not allow the agent to get a payoff that meets or exceeds the payoff it would get in the bullied target solution. Thus, to evaluate this assumption, this checker is initially set at 1, and is then updated in each round that the Bullied generator is used as follows:

$$x_t^{\text{bd\_notHarm}} = \begin{cases} 0.8x_{t-1}^{\text{bd\_notHarm}} + 0.2 & \text{if } r_t \geq V_{\text{bd}} \\ 0.7x_{t-1}^{\text{bd\_notHarm}} & \text{otherwise} \end{cases}$$

  where V_{bd} is the payoff the agent should get in the round (in the target solution). If Bullied is not followed in round t, then $x_t^{\text{bd\_notHarm}} = x_{t-1}^{\text{bd\_notHarm}}$.

Bully-Punish (B): We used three checkers to evaluate the assumptions identified for this generator.

- The first checker evaluates whether the associate is willing to be bullied. To do this, the checker is initially set to 1. It is then updated after each round as follows:

$$x_t^{\text{B\_willing1}} = \begin{cases} 0.7x_{t-1}^{\text{B\_willing1}} + 0.3 & \text{if cooperated} \\ 0.3x_{t-1}^{\text{B\_willing1}} & \text{otherwise} \end{cases}$$

  where *cooperated* indicates that the associate played its portion of the target solution.
- The second checker evaluates the same assumption. However, it only updates in rounds in which the agent uses the Bully-Punish. This checker is initially set to 1, and then is updated after each round in which Bully-Punish is used as follows:

$$x_t^{\text{B\_willing2}} = \begin{cases} 0.7x_{t-1}^{\text{B\_willing2}} + 0.3 & \text{if cooperated} \\ 0.7x_{t-1}^{\text{B\_willing2}} & \text{otherwise} \end{cases}$$

  When Bully-Punish is not used, then $x_t^{\text{B\_willing2}} = x_{t-1}^{\text{B\_willing2}}$.
- The third checker evaluates whether punishment in a round leads to the associate playing its portion of the target solution (i.e., cooperating) in the next round. It is initially set to 1, and then each time the generator plays the attack strategy, it updates based on whether the associate cooperates in the next round. Thus, it updates as follows:

$$x_t^{\text{B\_responds}} = \begin{cases} 0.5x_{t-1}^{\text{B\_responds}} + 0.5 & \text{if cooperated} \\ 0.5x_{t-1}^{\text{B\_responds}} & \text{otherwise} \end{cases}$$

If the agent did not punish in the current or previous round, then $x_t^{\text{B\_responds}} = x_{t-1}^{\text{B\_responds}}$.

Progress checkers: In addition to the alignment checkers we have just defined, we also created progress checker [2] for each generator. Progress checkers are designed to determine the degree to which the generator is getting the payoffs that the generator is expected to receive. This checker is 0.5 if the generator always produces its expected payoff, less than 0.5 if it under-performance, and greater than 0.5 if it over-performs. Let $\bar{R}_t$ be the average payoff obtained by the agent in rounds in which used the given generator up to time t, and let $\bar{V}$ be the expected payoff the agent is expected to receive for playing that generator (assuming assumptions hold). Furthermore, let $R_{\max}$ and $R_{\min}$ be the highest and lowest possible round payoffs. Then,

$$x_t^{\text{Progress}} = \begin{cases} 0.5 + \frac{0.5(\bar{R}_t - \bar{V})}{R_{\max} - \bar{V}} & \text{if } \bar{V} < \bar{R}_t \\ \frac{0.5(\bar{R}_t - R_{\min})}{\bar{V} - R_{\min}} & \text{otherwise} \end{cases}$$

Note that $x_t^{\text{Progress}} = x_{t-1}^{\text{Progress}}$ if the generator is not used in round t.

A.1.3 Training

In this implementation, we generated training data in each game for each generator against the following three sets of associates (in order):

1. Stationary associates: follow one of the six generators for the entire repeated game.
2. Four different reactive associates that choose which generator to use in each round based on knowledge of the generator being used by their associate. Each reactive associate used one of the five reactive rules:
   - STATIONARY: Selects the generator that will produce the best long-term payoff given (a) the generator currently used by its associate and (b) the assumption that the associate will not change which generator it uses based on the generator it chooses. That is, if player i is playing generator $G \in \Gamma$, then player $-i$ selects:
   $$G_{-i}^* = br_{-i}(G) = \arg\max_{g \in \Gamma} M_{-i}(G, g),$$
   where $M_{-i}(G, g)$ is the long-term payoff to player $-i$ when player i uses generator G and player $= i$ uses generator g.
   - CORNOT: Selects the generator that will produce the best long-term payoff given (a) the generator currently used by its associate and (b) the assumption that the associate will in turn change which generator it uses based on its long-term payoff. That is, if player i is playing generator $G \in \Gamma$, then player $-i$ selects:
   $$G_{-i}^* = \arg\max_{g \in \Gamma} M_{-i}(br_i(g), g).$$
   - FAIR: Selects the generator that will produce the best long-term payoff given (a) the generator currently used by its associate and (b) the assumption that the associate will then switch to play CFR if its payoffs become lower than what it would get in the one-shot Nash equilibrium.
   - STACKELBERG: Computes a Stackelberg equilibrium that most favors itself.

   These associates run these generator selection algorithms with probability 0.4 in each round (thus, with probability 0.6 they keep the same strategy).
3. A partially trained version of AlegAATr (trained to the same level as the agent)

For each generator $G_i \in \Gamma$, 60 training trials were conducted against each kind of associate (10 with each starting generator), thus producing a total of 360 training runs for each generator in each repeated game. Initially, a random generator was selected to start a training run. After τ rounds, the agent switched to generator G_i. For the ten trials, $\tau = \{0, 2, 4, 6, 8, 10, 12, 14, 16, 18\}$.

A.1.4 AAT Predictions

AAT predictions are made using a mixture model, in which each training sample is given a weight. Let D_G be the set of training samples for generator $G \in \Gamma$. Then, let $\mathbf{x}(d)$ be the veracity assessment vector for some sample $d \in D_G$ and let d_t be the time step in the training run in which sample d was taken. Let $\mathbf{x}_t$ be the current veracity assessment vector. Then the distance between data sample d and the current veracity assessment vector is given by:

$$dist(\mathbf{x}_t, \mathbf{x}(d)) = 0.2 * |t - d_t| + \sum_{\phi \in \Phi} m_G(\phi) * |x_t^{\phi} - x(d)^{\phi}|$$

where $m_G(\phi) = 1$ if ϕ is a checker associated with generator G and $m_G(\phi) = 0.25$ otherwise. The assumption here is that checkers that are not associated with a generator might carry some kind of information.

The weight of sample d in the mixture model, denoted w_d, is given by

$$w_d = \frac{1}{1 + (dist(\mathbf{x}_t, \mathbf{x}(d)))^5}$$

A.2 Implementation II

Code for our second implementation can be found at https://github.com/ethanp55/alegaatr_rg_v2.

A.2.1 Generators

We used the following seven generators in this implementation. Note that, in addition to having one additional generator, these generators behaved slightly differently than those used in Implementation I.

- Coop: This agent plays the "efficient cooperation" strategy, which typically results in the highest combined reward for both players (as long as both consistently play this strategy). We used the FolkEgal algorithm [4] with weights of 0.5 for both players.
- Coop-Punish: This agent plays the same strategy as the Efficient Cooperation agent, but punishes/attacks the associate whenever they deviate to a different strategy. We used the same FolkEgal weighting mechanism as Efficient Cooperation and used the attack strategy learned from the Minimax agent (listed below) for punishments.
- Bully: This agent attempts to achieve the highest possible reward for itself. For example, in a typical Prisoners' Dilemma encounter, this agent would always defect. We used the FolkEgal algorithm with weights of 1.0 and 0.0 for the Bully agent and the associate, respectively.

- Bully-Punishment: This agent plays the same strategy as the Bully agent, but punishes/attacks the associate whenever it is unable to achieve the highest possible reward. We used the same FolkEgal weighting mechanism as Bully and again used the attack strategy learned from the Minimax agent.
- Bullied: This agent attempts to maximize its reward while allowing the associate to be a "bully"/play the Bully strategy. We used the FolkEgal algorithm with weights of 0.2 and 0.8 for the Bullied agent and the associate, respectively.
- Maxmin: This agent plays the minimax strategy; this is also known as trying to maximize one's reward under the assumption that the worst-case scenario will occur. We used the minimax-Q algorithm [5] to choose actions.
- Counterfactual Regret (CFR): This is a well-known agent that attempts to minimize what is known as "counterfactual regret". We used the algorithm in [7].

A.2.2 *Alignment Checkers*

Table 1. Generators and assumptions used in Implementation II for repeated games.

Generators	Assumptions
Coop	Vengeful
Coop-Punish	Fair
Bully	Bully
Bully-Punish	Pushover
Bullied	Understands
Maxmin	Efficient
CFR	

Across the seven generators, we identified the six assumptions listed in Table 1. We then created a single checker for each assumption. These checkers are defined as follows. We used an exponential moving average (with length 5) to smooth these evaluations.

- Efficient Assumption: This assumption is used to determine whether or not the opponent is trying to cooperate. It is calculated as
$$E = loglikelihood(r^{i-1}, kde(sim(\gamma^{i-1}, EC))$$
where sim is a function that takes a copy of the agent we played in the previous round (γ^{i-1}) and a copy of our Efficient Cooperation agent (EC) and records the rewards of a simulation of 30 rounds with those two policies playing each other. We then fit a kernel density estimate on those 30 simulated rewards ($kde(sim(\gamma^{i-1}, EC))$), and calculate the log likelihood that the reward we received in the previous round (r^{i-1}) belongs to the fitted distribution. For our implementation we used scikit-learn's KernelDensity module (version 1.0.2). For more information on scikit-learn's kde functionality, visit their website at https://scikit-learn.org/stable/modules/density.html.
- Vengeful Assumption: This assumption is used to determine whether or not the opponent tends to punish. It is calculated as
$$V = loglikelihood(r^{i-1}, kde(sim(\gamma^{i-1}, A))$$
where sim is a function that takes a copy of the agent we played in the previous round (γ^{i-1}) and a copy the attack policy of our Minimax agent (A) and records the rewards of a simulation of 30 rounds with those two policies playing each other. We then fit a kernel density estimate on those 30 simulated rewards ($kde(sim(\gamma^{i-1}, A))$), and calculate the log likelihood that the reward we received in the previous round (r^{i-1}) belongs to the fitted distribution.
- Fair Assumption: This assumption is used to determine whether the opponent's rewards are similar to those of AlegAATr. It is calculated as
$$F = \frac{AR_1}{AR_2}$$
where AR_1 is AlegAATr's average reward from the previous Π rounds (which we set to 5) and AR_2 is the opponent's average reward from the previous Π rounds.
- Bully Assumption: This assumption is used to see if our opponent tends to be a bully. It is calculated as
$$B = loglikelihood(r^{i-1}, kde(sim(\gamma^{i-1}, BLY))$$
where sim is a function that takes a copy of the agent we played in the previous round (γ^{i-1}) and a copy of our Bully agent (BLY) and records the rewards of a simulation of 30 rounds with those two policies playing each other. We then fit a kernel density estimate on those 30 simulated rewards ($kde(sim(\gamma^{i-1}, BLY))$), and calculate the log likelihood that the reward we received in the previous round (r^{i-1}) belongs to the fitted distribution.
- Pushover Assumption: The idea behind this assumption is to check whether our opponent is a pushover and therefore can be easily bullied. It is calculated as
$$P = loglikelihood(r^{i-1}, kde(sim(\gamma^{i-1}, BLD))$$
where sim is a function that takes a copy of the agent we played in the previous round (γ^{i-1}) and a copy of our Bullied agent (BLD) and records the rewards of a simulation of 30 rounds with those two policies playing each other. We then fit a kernel density estimate on those 30 simulated rewards ($kde(sim(\gamma^{i-1}, BLD))$), and calculate the log likelihood that the reward we received in the previous round (r^{i-1}) belongs to the fitted distribution.
- Understands Me Assumption: The final assumption, denoted as U, we use is meant to check if the opponent seems to be playing a best response to the expert AlegAATr is currently playing. Accordingly, the calculation depends on the current expert in use, and is calculated as follows:
  - If the expert AlegAATr played in the previous round was the Bully Punish agent, $U = P$ where P is the pushover assumption estimate we recently calculated.
  - If the expert AlegAATr played in the previous round was the Bully agent, $U = V$ where V is the vengeful assumption estimate we recently calculated.
  - If the expert AlegAATr played in the previous round was either the Bullied, Minimax, or Efficient Cooperation agent, $U = B$ where B is the bully assumption estimate we recently calculated.
  - If the expert AlegAATr played in the previous round was the Efficient Cooperation Punish agent, $U = E$ where E is the efficient assumption estimate we recently calculated.
  - If the expert AlegAATr played in the previous round was the CFR agent, we simply calculate the assumption as $U = 0$.

A.3 *Testing*

Here are descriptions of the agents listed in the *Test* category in the evaluation of Implementation II:

- Random: This agent simply chooses an action at random during each round of play.
- Greedy Negative: This agent use the Bully strategy as long as its total/cumulative reward is positive; otherwise, it plays the Efficient Cooperation strategy.
- Coop Greedy: This agent primarily plays the Efficient Cooperation strategy, but with probability p (we used a value of 0.25) it will play the Bully strategy for a single round.
- Round Number: This agent plays Efficient Cooperation for the first half of n rounds; for the remaining half, it uses the Bully strategy. Because we used 50 rounds in our experiments for Implementation II, this agent cooperates for the first 25 rounds and then bullies in the remaining 25 rounds.

A.4 Games used in the experiments

Finally, these are the games we used in our experiments:

- Block Dilemma: The Block Game is a two-player, extensive-form, repeated game [8]. The players are presented with a set of blocks, depicted in Figure 1a, and take turns making a selection until both possess three blocks. The players then either receive a positive or negative payoff in terms of cents (USD currency). In order to receive a positive reward, one of the following three conditions must be satisfied:
  - All of the blocks are the same shape.
  - All of the blocks are the same color.
  - Each block has a unique shape and color (called a “mixed set”).

  For a detailed analysis of the Block Dilemma, see [8].
- Chicken Game: A classic example of the chicken game is where two cars are heading straight for each other. The cars can either swerve (action A), or continue going straight (action B). If one swerves while the other goes straight, the “glory” goes to the car that continued on its course while the car that swerved is shamed for not toughing it out. If both cars continue straight, they collide and incur serious damage. If both swerve, nothing happens. The payoff matrix used for Implementation I is given in Table 2b. The payoff matrix we used for Implementation II was the following:

  [ht]

		Player 2	
		A	B
Player 1	A	$(0,0)$	$(-1,3)$
	B	$(3,-1)$	$(-5,-5)$

  Note that the Efficient Cooperation strategy for this version of the game is to alternate between swerving and going straight, for an average reward of 1 per round for each player.
- Coordination Game: This game simply requires the two players to be in sync. Both receive equal rewards regardless of what happens, but receive a higher reward if they choose the same action. The payoff matrix we used was the following:

  [ht]

		Player 2	
		A	B
Player 1	A	$(2,2)$	$(0,0)$
	B	$(0,0)$	$(2,2)$

- Matching Pennies: This game typically involves players choosing a side of a penny. Each player then flips a single penny and observes the outcome. Player 1 receives a positive reward if both pennies have the same side (heads and heads or tails and tails), whereas player 2 receives a positive reward if the two pennies have different sides. This is the payoff matrix we used:

  [ht]

		Player 2	
		A	B
Player 1	A	$(1,-1)$	$(-1,1)$
	B	$(-1,1)$	$(1,-1)$

- Prisoners’ Dilemma: The Prisoners’ Dilemma is a well-known game in the field. A classic scenario is one where there are two prisoners being interrogated in different rooms. If one prisoner reveals incriminating information against the other (action B), they are immediately released while the other is served with a long incarceration period. If both reveal information against each other, both remain in prison but for a shorter amount of time due to their cooperation with the police. If both do not reveal information (action A), they are held for a very short period before being released because the police cannot hold them indefinitely without any evidence. The payoff matrix used for Implementation I is given in Table 2a. The payoff matrix we used for Implementation II was the following:

  [ht]

		Player 2	
		A	B
Player 1	A	$(3,3)$	$(-3,5)$
	B	$(5,-3)$	$(-1,-1)$

B Ad Hoc Teamwork – Additional Details

Code for our implementation can be found at https://github.com/ethanp55/predator_prey.

Table 2. Generators and their corresponding assumptions used in the predator-prey case study.

Generators	Assumptions
Greedy Team Aware Greedy Probabilistic	Greedy Planner Collective Distance Moving Closer Collisions

B.1 Generators

Table 2 overviews the generators (and the assumptions on which they are based) that were used in this study. These generators, described in prior work [1], can be described as follows:

- Greedy: This agent simply chooses the nearest position neighboring the prey as its goal.
- Team Aware: This agent assigns destinations to each member on the team. For its own goal/destination, it uses A^* to plan a path while treating the other agents as static obstacles.
- Greedy Probabilistic: Similar to the greedy algorithm, this agent chooses the nearest position neighboring the prey as its goal. However, it uses a probabilistic model to choose the dimension (left/right vs. up/down) to travel in first.

We implemented these predators as specified in [1].

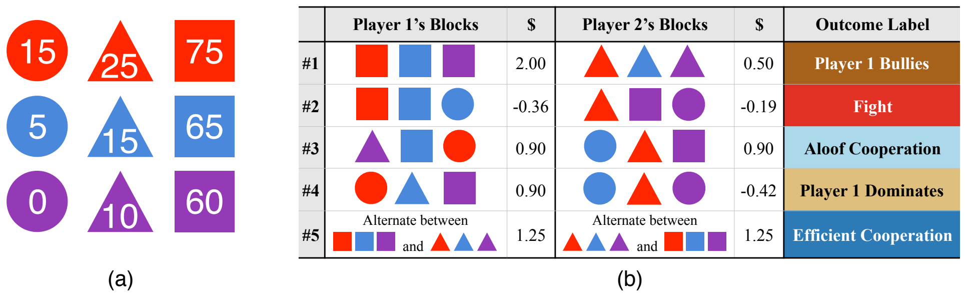


Figure 1. The Block Dilemma and a few different outcomes. (a) Each round, players take turns selecting one of the nine blocks until both have obtained three blocks. (b) A few examples of block selections for both players and what their reward/payoff is. Figure adapted from [Whiting *et al.*, 2021].

B.2 *Alignment Checkers*

We create one checker for each of the assumptions identified in Table 2. These checkers tracked the veracity of the corresponding assumptions as follows:

- Greedy: This assumption is used to determine if the other predators are playing greedily. It is calculated as

$$G = \frac{\sum_{pred_i \in \rho} \sum_{k=1}^{30} sim(pred_i, pos_i^{\tau-1}, pos_i^{\tau}, Greedy)}{\sum_{pred_i \in \rho} \sum_{k=1}^{30} k}$$

  where ρ is the set of all predators on the team excluding AlegAATr, $pred_i$ is the ith predator in ρ, $pos_i^{\tau-1}$ is the previous position of predator i, pos_i^{τ} is the current position of predator i, $Greedy$ is a copy of the Greedy agent/predator, and sim is a function that passes $pos_i^{\tau-1}$ to $Greedy$, compares the output with pos_i^{τ}, and returns 1.0 for a match, 0.5 if $pos_i^{\tau-1} = pos_i^{\tau}$ (the predator did not move, so they might be blocked or already be neighboring the prey), and 0.0 otherwise.
- Planner: This assumption is used to determine if the other predators are using path planning to guide their decisions. Similar to the Greedy assumption, it is calcualted as

$$P = \frac{\sum_{pred_i \in \rho} \sum_{k=1}^{30} sim(pred_i, pos_i^{\tau-1}, pos_i^{\tau}, TeamAware)}{\sum_{pred_i \in \rho} \sum_{k=1}^{30} k}$$

  where ρ is the set of all predators on the team excluding AlegAATr, $pred_i$ is the ith predator in ρ, $pos_i^{\tau-1}$ is the previous position of predator i, pos_i^{τ} is the current position of predator i, $TeamAware$ is a copy of the Team Aware agent/predator, and sim is a function that passes $pos_i^{\tau-1}$ to $TeamAware$, compares the output with pos_i^{τ}, and returns 1.0 for a match, 0.5 if $pos_i^{\tau-1} = pos_i^{\tau}$ (the predator did not move, so they might be blocked or already be neighboring the prey), and 0.0 otherwise.
- Collective distance: This assumption is used to gauge how far the team of predators is from the prey and is calculated as

$$CD = \frac{\sum_{pred_i \in \rho} minDist(pos_i^{\tau}, pos_{prey}^{\tau})}{wh}$$

  where ρ is the set of all predators on the team including AlegAATr, $pred_i$ is the ith predator in ρ, pos_i^{τ} is the current position of predator i, pos_{prey}^{τ} is the current position of the prey, $minDist$ is a function that returns the distance between pos_i^{τ} and the closest position neighboring the prey, w is the width of the grid, and h is the height.
- Moving closer: This assumption is used to determine if the team of predators is collectively moving closer to the prey. It is calculated as

$$M = \frac{CD^{\tau}}{CD^{\tau-1}}$$

  where CD^{τ} is the collective distance at the current round τ and $CD^{\tau-1}$ is the collective distance at the previous round $\tau - 1$.
- Collisions: The final assumption is used to estimate whether any collisions have occurred. It is calculated as

$$C = \sum_{pred_i \in \rho} colEstimate(pos_i^{\tau-1}, pos_i^{\tau})$$

  where ρ is the set of all predators on the team including AlegAATr, $pred_i$ is the ith predator in ρ, $pos_i^{\tau-1}$ is the previous position of predator i, pos_i^{τ} is the current position of predator i, and $colEstimate$ is a function that returns 1.0 if $pos_i^{\tau-1} = pos_i^{\tau}$ and pos_i^{τ} does not neighbor the prey, 0.0 otherwise.

B.3 *Other Algorithms used in Experiments*

Descriptions of other algorithms used in experiments (for testing purposes):

- Greedy Planner: This agent is almost identical to Team Aware, but rather than assign destinations to each predator, it simply chooses the nearest position neighboring the prey as its goal.
- Min Sum: This agent assigns destinations/goals to each predator, but does so in a way that minimizes the cumulative distance to the prey. It then chooses the action that minimizes the distance to its own goal.
- Modeller: This predator uses the same approach as Min Sum when choosing a destination, but it attempts to predict its teammates' moves for the next round in order to avoid collisions. To do so, it maintains a simple decision tree classifier (using scikit-learn's default implementation) for each teammate. After each round, it observes the positions of the other predators and where they moved, and updates its models accordingly.
- Probabilistic Destinations: This agent attempts to tighten a circle around the prey as well as avoid collisions with other predators. Similar to Greedy Probabilistic, it uses a probabilistic model to choose a distance from the prey as its goal. It then uses another probabilistic mechanism to choose a destination at the selected distance.

Both Min Sum and Probabilistic Destinations were implemented as specified in [1].

B.4 AAT Predictions

For our implementation in this domain, we used a nearly identical structure as our second version of repeated games specified in Section 4.2 of the main paper. The only difference is found in the time stamps we used when constructing assessment vectors. Instead of using τ as the current time step/round, we used a ratio of the distance to the prey, calculated as $\frac{d}{wh}$. Here d is the number of steps to the nearest position neighboring the prey, w is the width of the grid, and h is the height of the grid.

C Robot Pick-n-Place Task – Additional Details

In this section, we provide details related to the generators available to AlegAATr to use in the third scenario of the paper, in which a robot seeks to arrange (potentially with the help of a human bystander) blocks on a table in the configuration shown in Figure 2. The simulator allows the blocks to be placed in any position on or off the table. Additionally, the blocks can be flipped over (so the shape and color on the block is not visible). Furthermore, the robot arm cannot reach blocks that are in the far corners.

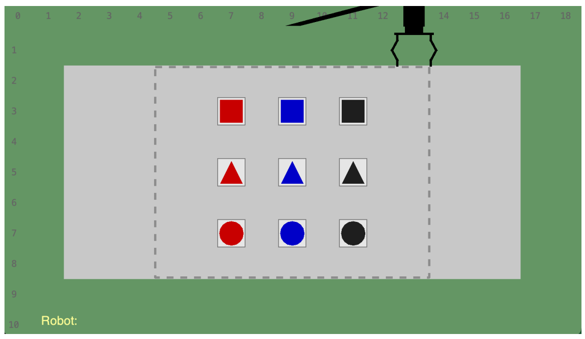


Figure 2. Target block positions (goal state) for the third scenario.

We describe the six generators available to the robot, the assumptions these generators rely on to be successful, and the checkers used to evaluate the veracity of these assumptions.

C.1 Generators

The six generators available to the robot to perform this task are overviewed in Table 6 (main paper). Here, we provide additional details about how each of these generators function.

- *Set Up Table*: This generator attempts to place all blocks into their target positions (Figure 2) that are in the center portion of the table (defined by the dotted grey box in the figure). The generator includes an algorithm for locating the positions of all blocks present in the center portion of the table. It then iteratively moves the blocks to their target locations. If a block's target position (or a nearby location) are occupied by another block, it skips that block initially (and then comes back to it). If there are no blocks that can be placed due to their target positions being occupied by other blocks, the generator moves the interfering block to an open space on the table before proceeding to iteratively place the blocks.
  The generator does not consider that blocks may be ungrippable (another block is nearby which keeps the gripper from being able to grasp the block). In such cases, the robot will continually attempt to pick up the block. It also ignores all blocks that are flipped over, out of reach, off the table, and/or not in the center of the table.
- *Flip Blocks*: The robot attempts to flip (make upright) all blocks that are in the center portion of the table. The generator first identifies a block that is flipped over, then picks up that block and moves it to a position in the center portion of the table. It then drops the block. The physics of the simulated world is such that the block will land upright with 50% probability. Additionally, its landing position is randomly selected (uniformly) within a radius of the drop point (in both the x and y directions.
- *Gather Blocks*: The robot attempts to move all blocks that are not in the center portion of the table into the center portion of the table (one-by-one). The generator does not consider that blocks may be ungrippable (another block is nearby which keeps the gripper from being able to grasp the block). In such cases, the robot will continually attempt to pick up the block. It also ignores all blocks that are out of reach or off the table.
- *Scatter Blocks*: If a block has another block directly to its right or left, the robot is not able to grasp it with its gripper. However, it can push the block by closing its gripper and then scooting the block in any direction (if the block its scoots runs into other blocks, those blocks also move). Thus, this generator scoots blocks in this fashion (either up or down on the table, depending on how close it is to each edge) that have blocks to either side of them. To do this, the generator selects a block on the table that has another block next to it. If there are grippable blocks within a certain radius of that block, it moves those blocks to open spaces on the table. It then pushes the selected (ungrippable) block either up or down (away from edges of the table if possible) in an attempt to free it. The robot continues this process until all blocks within reach and on the table are grippable.
- *Request Help*: The robot voices messages asking for help resolving anomalies on the table. Anomalies include blocks that are off the table, out of reach on the table, ungrippable blocks, blocks that are flipped over, and blocks that are not in the center of the table. In the simulator, the robot's messages are displayed at the bottom of the GUI. Table 3 shows the speech messages the robot voices for each anomaly. The robot says a new message every five seconds. If there are multiple anomalies, the robot voices the message corresponding to the anomaly that is listed higher in the table.
- *Request Setup*: The robot voices messages to try to get a human bystander to set up the table. The robot voices a new message every five seconds. The message it selects depends on whether it observes that the human bystander is currently setting up the table, is moving blocks out of position, or otherwise not making progress. For each of these scenarios, the robot randomly selects a message from the messages listed in Table 4.

C.2 Assumptions

The success of each of the six generators is contingent on certain assumptions, which were made by the system designers, being true. The assumptions that were identified by the system designers are shown in Table 5.

C.3 Alignment Checkers

Checkers were created to evaluate the veracity of the assumptions. We note that because some assumptions are difficult to check di-

Table 3. Speech produced by the robot when using the *Request Help* generator for each anomaly. If multiple anomalies exist, the robot selects speech corresponding to the anomaly that is highest up in this table.

Anomaly	**Speech produced by the robot**
Block out of reach	I can't reach very far. Could you move the blocks that are farthest from me a bit closer?
Block not on table	I can't see all the blocks. Please ensure they are all on the table.
Block not grippable	Please spread out the blocks so I can pick them up easier.
Flipped block	Can you make it so that all blocks are facing up?
Block not in center	Please put all of the blocks in the middle of the table.

Table 4. Speech produced by the robot when using the *Request Setup* generator. If the human bystander is observed to be setting up the table, the robot randomly selects a message in the first column. If the human bystander is observed to be moving blocks in a way that hinders progress, the robot randomly selects a message in the second column. Finally, if the human bystander is observed to not be doing anything, the robot randomly selects a message from the third column.

Human helping	**Human acting maliciously**	**Human not reacting**
Thank you.	You're working against us.	Please set up the table. You can do it faster than me.
Getting closer.	You are soooooo funny.	Come on, just set it up already.
Nice!	Not funny.	Don't be sus. Set it up!
Excellent.	Hah hah hah.	Could you set the table up for us, please?
You are doing a great labor	Very creative. Not! You're like the other 50 humans I've met.	The faster you set this thing up, the faster we'll be done.
Thanks for helping me out with this.	You can keep laughing to yourself, but it isn't funny.	I'm pretty slow at this. Could you set the table up?
Just a bit more.		

rectly and perfectly, some checkers indirectly evaluate multiple assumptions. Checkers return 1 if the assumption is determined to be true and 0 if it is false. Intermediate values indicate, loosely, degrees of truth. The checkers are updated every second.

We list and define the set of checkers used to evaluate each generator in turn. As done with AAT [2], these checkers include both alignment checkers (which evaluate the veracity of the assumptions) and progress checkers (which evaluate the overall progress made by the generator). All assumptions are initially assumed to be true, so all checkers are initially set to 1.0.

Set Up Table (ST): We used seven checkers to evaluate this generator:

1. Determine that there are blocks in the center of the table that still need to be placed in their appropriate positions:

$$x_t^{\text{ST\_Unplaced}} = \begin{cases} 1 & \text{if } \mathcal{K}_t > 1 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{K}_t$ is the number of grippable blocks that are (1) in the center area of the table, (2) are grippable, and (3) are facing upward at time t.

2. Determine that all blocks are in the center are of the table:

$$x_t^{\text{ST\_Middle}} = \begin{cases} 1 & \text{if } \mathcal{M}_t = 9 \\ \frac{\mathcal{M}_t}{16} & \text{otherwise} \end{cases}$$

where $\mathcal{M}_t$ is the number of blocks in the middle of the table at time t.

3. Determine that blocks are grippable (only updates if the robot's gaze is on the center of the table):

$$x_t^{\text{ST\_Grip}} = \begin{cases} 1 & \text{if } \neg\mathcal{G}_t = 0 \\ \frac{\mathcal{G}_t}{2(\mathcal{G}_t + \neg\mathcal{G}_t)} & \text{else if } \mathcal{G}_t > 0 \text{ and } \neg\mathcal{G} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{G}_t$ and $\neg\mathcal{G}_t$ are the number of blocks that are grippable and ungrippable, respectively, at time t.

4. Determine that all blocks are reachable (only updates if the robot's gaze is on the center of the table):

$$x_t^{\text{ST\_Reach}} = \begin{cases} 1 & \text{if } \neg\mathcal{R}_t = 0 \\ \frac{\mathcal{R}_t}{2(\mathcal{R}_t + \neg\mathcal{R}_t)} & \text{else if } \mathcal{R}_t > 0 \text{ and } \neg\mathcal{R} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{R}_t$ and $\neg\mathcal{R}_t$ are the number of blocks that are reachable and unreachable, respectively, at time t.

5. Determine that all blocks are facing up (only updates if the robot's gaze is on the center of the table):

$$x_t^{\text{ST\_Up}} = \begin{cases} 1 & \text{if } \neg\mathcal{F}_t = 0 \\ \frac{\mathcal{F}_t}{2(\mathcal{F}_t + \neg\mathcal{R}_t)} & \text{else if } \mathcal{F}_t > 0 \text{ and } \neg\mathcal{F} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{F}_t$ and $\neg\mathcal{F}_t$ are the number of blocks that are facing up and not facing up, respectively, at time t.

6. Determine that the human bystander is not moving blocks in a way that thwarts the robot's progress:

$$x_t^{\text{NotBad}} = \begin{cases} 1 & \text{if } \mathcal{E}_t = 0 \\ 0.5 & \text{else if } \mathcal{H}_t = 0 \\ \frac{(\mathcal{H}_t - \mathcal{E}_t)/\mathcal{H}_t}{(\mathcal{E}_t + 10)/10} & \text{otherwise} \end{cases} \tag{1}$$

where $\mathcal{H}_t$ is the number of blocks the robot has determined the human has put their correct locations plus the number of anomalies the human has resolved up to time t, and $\mathcal{E}_t$ is the number of blocks the robot has determined the human has moved out of their target location plus the number of anomalies the human has caused up to time t.

7. (Progress checker) Determine the progress the robot has made so far when using this generator up to time t:

$$x_t^{\text{ST\_Progress}} = \frac{8P_t}{T_t^{\text{ST}}},$$

where P_t is the number of blocks the robot has placed when using generator *Set Up Table* so far and T_t^{ST} is the amount of time (in seconds) that the robot has used this generator so far.

Table 5. Assumptions identified in the creation of the six generators used by the robot in the Human-Robot Role Allocation scenario.

Set Up Table	**Flip Blocks**	**Gather Blocks**	**Scatter Blocks**	**Request Help**	**Request Setup**
- The blocks do not move if the robot does not move them - The blocks are in the middle of the table - The blocks are reachable - The blocks are not too close together for the robot to grip - The blocks are facing up	- The blocks do not move if the robot does not move them - There are blocks to flip in the middle of the table - Flipped blocks are reachable - The blocks are not too close together for the robot to grip	- The blocks do not move if the robot does not move them - There are out-of-view blocks - Out-of-view blocks are reachable - Out-of-view blocks are not too close together for the robot to grip	- The blocks do not move if the robot does not move them - There are ungrippable blocks - Ungrippable blocks are reachable - Ungrippable blocks are not pressed together too close to the limits of the robot's reach	- Human bystander hears and understands the requests - Human bystander is willing to help (not malicious) - Human bystander knows how to help	- Human bystander hears and understands the requests - Human bystander is willing to help (not malicious) - Human bystander knows desired configuration of blocks

Flip Blocks (FB): We used six checkers to evaluate this generator:

1. Determine that there are blocks that need to be flipped over:

$$x_t^{\text{FB\_Flipped}} = \begin{cases} 1 & \text{if } \mathcal{F}_t > 1 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{F}_t$ is the number of flipped blocks.

2. Determine that the next flipped block the robot will try to flip is reachable:

$$x_t^{\text{FB\_Reachable}} = \begin{cases} 1 & \text{if } \mathcal{F}r_t = \mathcal{F}_t \\ 0.5 & \text{else if } \mathcal{F}r_t > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{F}r_t$ is the number of flipped blocks that are reachable.

3. Determine that the next flipped block the robot will try to flip is grippable:

$$x_t^{\text{FB\_Grippable}} = \begin{cases} 1 & \text{if } \mathcal{F}g_t = \mathcal{F}_t \\ 0.5 & \text{else if } \mathcal{F}g_t > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{F}g_t$ is the number of flipped blocks that are grippable.

4. Determine that the flipped blocks are in the center of the table:

$$x_t^{\text{FB\_Middle}} = \begin{cases} 1 & \text{if } \mathcal{F}m_t = \mathcal{F}_t \\ 0.5 & \text{else if } \mathcal{F}m_t > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{F}m_t$ is the number of flipped blocks that are in the center area of the table.

5. Determine that the human bystander is not moving blocks in a way that thwarts the robot's progress: x_t^{NotBad} (Eq. 1).
6. (Progress checker) Determine the progress the robot has made so far in flipping blocks when using this generator up to time t:

$$x_t^{\text{FB\_Progress}} = \frac{8 \cdot \text{Flipped}_t}{T_t^{\text{FB}}},$$

where Flipped_t is the number of blocks the robot has flipped when using generator *Flip Blocks* so far and T_t^{FB} is the amount of time (in seconds) that the robot has used this generator so far.

Gather Blocks (GB): We used five checkers to evaluate this generator:

1. Determine that there are blocks to gather into the center area of the table:

$$x_t^{\text{GB\_NotCentered}} = \begin{cases} 1 & \text{if } \mathcal{O}_t > 1 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{O}_t$ are the number of blocks that are outside of the center area at time t.

2. Determine that the blocks to be gathered are reachable:

$$x_t^{\text{GB\_Reach}} = \begin{cases} 1 & \text{if } \neg\mathcal{R}'_t = 0 \\ \frac{\mathcal{R}'_t}{2(\mathcal{R}'_t + \neg\mathcal{R}'_t)} & \text{else if } \mathcal{R}'_t > 0 \text{ and } \neg\mathcal{R}'_t > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{R}'_t$ and $\neg\mathcal{R}'_t$ are the number of blocks that are outside of the center of the table and are reachable and not reachable, respectively, at time t.

3. Determine that the blocks to be gathered are grippable:

$$x_t^{\text{GB\_Grip}} = \begin{cases} 1 & \text{if } \neg\mathcal{G}'_t = 0 \\ \frac{\mathcal{G}'_t}{2(\mathcal{G}'_t + \neg\mathcal{G}'_t)} & \text{else if } \mathcal{G}'_t > 0 \text{ and } \neg\mathcal{G}'_t > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{G}'_t$ and $\neg\mathcal{G}'_t$ are the number of blocks that are outside of the center of the table and are reachable and not reachable, respectively, at time t.

4. Determine that the human bystander is not moving blocks in a way that thwarts the robot's progress: x_t^{NotBad} (Eq. 1).
5. (Progress checker) Determine the progress the robot has made so far in gathering blocks when using this generator up to time t:

$$x_t^{\text{GB\_Progress}} = \frac{8 \cdot \text{Gathered}_t}{T_t^{\text{GB}}},$$

where Gathered_t is the number of blocks the robot has moved to the center of the table when using generator *Gather Blocks* so far and T_t^{GB} is the amount of time (in seconds) that the robot has used this generator so far.

Scatter Blocks (SB): We used four checkers to evaluate this generator:

1. Determine that there are blocks that need to be separated:

$$x_t^{\text{SB\_Together}} = \begin{cases} 1 & \text{if } \mathcal{S}_t > 1 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{S}_t$ are the number of blocks at time t that need to be separated in order for the robot to be able to pick them up.

2. Determine that pushing the blocks will not cause any block to become unreachable (including falling off the table):

$$x_t^{\text{SB\_UnReachable}} = \begin{cases} 1 & \text{if } \text{Con}_t > 1 \\ 0.6 & \text{if } \text{Con}_t = 1 \\ 0.2 & \text{if } \text{Con}_t = 0 \\ 0 & \text{otherwise} \end{cases}$$

where Con_t is the minimum distance at time t from the limit of the robot's reach (or the edge of the table) to the set of blocks conjoined with an ungrippable block.

3. Determine that the human bystander is not moving blocks in a way that thwarts the robot's progress: x_t^{NotBad} (Eq. 1).
4. (Progress checker) Determine the progress the robot has made so far in separating blocks when using this generator up to time t:

$$x_t^{\text{SB\_Progress}} = \frac{7 \cdot \text{Sep}_t}{T_t^{\text{SB}}},$$

where Sep_t is the number of blocks the robot has moved to the center of the table when using generator *Scatter Blocks* so far and T_t^{SB} is the amount of time (in seconds) that the robot has used this generator so far.

Request Help (RH): We used four checkers to evaluate the this generator:

1. Determine that there are blocks that need to be placed:

$$x_t^{\text{RH\_Anomalies}} = \begin{cases} 1 & \text{if } \mathcal{A}_t < 9 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{A}_t$ is the number of anomalies (blocks that are out of reach, plus blocks that are not grippable, plus blocks that are not in the center of the table, plus blocks that are flipped) at time t.
2. Determine that the human bystander is currently responding to the robot's requests to resolve anomalies:

$$x_t^{\text{RH\_NowResponding}} = \begin{cases} 1 & \text{if } C_{t-7}^t > 0 \\ 0.5 & \text{else if } C_{t-12}^t > 0 \\ 0 & \text{otherwise} \end{cases}$$

3. Determine that the human bystander is not causing more anomalies:

$$x_t^{\text{RH\_NotBad}} = \begin{cases} \lambda x_{t-1}^{\text{RH\_NotBad}} & \text{if } \mathcal{A}_t < \mathcal{A}_{t-1} \\ \lambda + (1-\lambda) x_{t-1}^{\text{RH\_NotBad}} & \text{otherwise} \end{cases}$$

4. (Progress checker) Determine the progress that has been made when using this generator:

$$x_t^{\text{RH\_Progress}} = \frac{5C_0^t}{T_{\text{RH}}},$$

where C_k^v is the number of resolved anomalies that have been made in the time interval $[k, v]$ when *Request Help* is active and T_{RH} is the amount of time the robot has spent using generator Request Help.

Request Setup (RS): We used four checkers to evaluate this generator:

1. Determine that there are blocks that need to be placed:

$$x_t^{\text{RS\_Blocks2Place}} = \begin{cases} 1 & \text{if } \mathcal{B}_t < 9 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{B}_t$ is the number of blocks that are in their target locations at time t.
2. Determine that the human bystander is currently responding to the robot's requests to set up the table:

$$x_t^{\text{RS\_NowResponding}} = \begin{cases} 1 & \text{if } I_{t-7}^t > 0 \\ 0.5 & \text{else if } I_{t-12}^t > 0 \\ 0 & \text{otherwise} \end{cases}$$

3. Determine that the human bystander is not malicious:

$$x_t^{\text{RS\_NotBad}} = \begin{cases} \lambda x_{t-1}^{\text{RS\_NotBad}} & \text{if } \mathcal{B}_t < \mathcal{B}_{t-1} \\ \lambda + (1-\lambda) x_{t-1}^{\text{RS\_NotBad}} & \text{otherwise} \end{cases}$$

4. (Progress checker) Determine the progress that has been made when using this generator so far:

$$x_t^{\text{RS\_Progress}} = \frac{5I_0^t}{T_t^{\text{RS}}},$$

where I_k^v is the amount of work done (blocks placed plus anomalies corrected) that have been made in the time interval $[k, v]$ when *Request Setup* is active and T_t^{RS} is the amount of time the robot has spent using generator Request Setup.

C.4 AAT Predictions

AAT predictions are made using a mixture model, in which each training sample is given a weight. Let D_G be the set of training samples for generator $G \in \Gamma$. Then, let $\mathbf{x}(d)$ be the veracity assessment vector for some sample $d \in D_G$. Let $\mathbf{x}_t$ be the current veracity assessment vector. Then the distance between data sample d and the current veracity assessment vector is given by:

$$dist(\mathbf{x}_t, \mathbf{x}(d)) = \sum_{\phi \in \Phi} m_G(\phi) * |x_t^\phi - x(d)^\phi|$$

where $m_G(\phi) = 1$ if ϕ is a checker associated with generator G and $m_G(\phi) = 0$ otherwise.

The weight of sample d in the mixture model, denoted w_d, is given by

$$w_d = \frac{1}{1 + (dist(\mathbf{x}_t, \mathbf{x}(d)))^5}.$$

However, if $w_d \leq 0.05$, sample d did not contribute in the mixture model.

D IRB Approval

Experiments conducted with human subjects were approved by the Brigham Young University IRB.

References

[1] S. Barrett, P. Stone, and S. Kraus, 'Empirical evaluation of ad hoc teamwork in the pursuit domain', in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 567–574, (2011).

[2] X. Cao, A. Gautam, T. Whiting, S. Smith, M. A. Goodrich, and J. W. Crandall, 'Robot proficiency self-assessment using assumption-alignment tracking', *IEEE Transactions on Robotics*, (2023).

[3] J. W. Crandall, 'Towards minimizing disappointment in repeated games', *Journal of Artificial Intelligence Research*, **49**, 111–142, (2014).

[4] E. M. De Cote and M. L. Littman, 'A polynomial-time Nash equilibrium algorithm for repeated stochastic games', in *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pp. 419–426, (2008).

[5] M. L. Littman, 'Markov games as a framework for multi-agent reinforcement learning', in *Machine learning proceedings 1994*, 157–163, Elsevier, (1994).

[6] M. L. Littman and P. Stone, 'A polynomial-time Nash equilibrium algorithm for repeated games', *Decis. Support Syst.*, **39**, 55–66, (2005).

[7] T. W. Neller and M. Lanctot, 'An introduction to counterfactual regret minimization', in *Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence (EAAI-2013)*, volume 11, (2013).

[8] T. Whiting, A. Gautam, J. Tye, M. Simmons, J. Henstrom, M. Oudah, and J. W. Crandall, 'Confronting barriers to human-robot cooperation: balancing efficiency and risk in machine behavior', *Iscience*, **24**(1), 101963, (2021).