Altruism in Fuzzy Reinforcement Learning

Rachel M. Haighton, Howard M. Schwartz, Sidney N. Givigi

***Abstract*— We propose using a genetic algorithm to select hyperparameters in multi agent reinforcement learning settings. In particular, we look at this in the context of cooperation and altruism. We show through the use of 3 continuous space games, that certain algorithmic hyperparameters are better suited to allow to agents learn altruistic behaviors. The agents learn using fuzzy actor critic learning algorithms in either a hierarchical structure or a single actor critic policy. The genetic algorithm selects the discount factors, the reward weights, and the standard deviation of noise applied to actor during learning. The genetic algorithm uses a fitness function based on the ratio of successful tests the group of agents can pass after training. This automated selection of these specific hyperparameters show that they are important for cooperation and also not trivial to select.**

***Index Terms*—multi-agent reinforcement learning, altruism, cooperation, fuzzy systems**

This paragraph of the first footnote will contain the date on which you submitted your paper for review, which is populated by IEEE. It is IEEE style to display support information, including sponsor and financial support acknowledgment, here and not in an acknowledgment section at the end of the article. For example, "This work was supported in part by the U.S. Department of Commerce under Grant 123456." The name of the corresponding author appears after the financial information, e.g. *(Corresponding author: Second B. Author).* Here you may also indicate if authors contributed equally or if there are co-first authors.

The next few paragraphs should contain the authors' current affiliations, including current address and e-mail. For example, Rachel M. Haighton is with Carleton University, Ottawa, Ontario, Canada (e-mail: rachelhaighton@cmail.carleton.ca).

Howard M. Schwartz. is with Carleton University, Ottawa, ON, Canada. He is now with the Department of Systems and Computing Engineering. (e-mail: schwartz@sce.carleton.ca).

Sidney N. Givigi is with the Electrical Engineering Department, Queen's University, Kingston, ON, Canada (e-mail: sidney.givigi@queensu.ca).

Mentions of supplemental materials and animal/human rights statements can be included here.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org

I. INTRODUCTION

Multi-agent reinforcement learning (MARL) is a field of machine learning where multiple agents use reinforcement learning algorithms at the same time, either to learn to cooperate to achieve some common goal, or competitively, where agents compete against each other. This paper focuses on the cooperation between agents, which can have applications in autonomous vehicles, traffic control, resource allocation, and even the stock market. However, how do we know that agents are truly cooperating and not just working in parallel without regard for one and other? One problem in some of this research is the lack of clear cooperation among agents. According to [1] "cooperation is conceptually and empirically linked to altruism, which is defined as behavior in which one person foregoes a benefit to help another." Another source [2] states that altruism is "a behavior which is costly to the actor and beneficial to the recipient". Altruism is an important aspect often missing in cooperative reinforcement learning as it demonstrates that agents are working together as a group and not individually. The goal of altruistic reinforcement learning is to show that one agent learns to forgo something for the benefit of the group.

Common reinforcement learning benchmarks such as the StarCraft II Learning Environment (SC2LE) [3] does not necessarily demonstrate altruism in the cooperative aspect. We focus on the learning of altruism in multiagent systems using reinforcement learning. We use this paper to focus on the learning of altruism in multiagent systems using reinforcement learning which aids us demonstrate cooperative tendencies among agents.

Difficulties within altruism in reinforcement learning include both the agents and the scenarios. How do we come up with scenarios where the agents have altruism as a choice? How do we measure altruism? What kind of parameters within an agents learning algorithm impact altruism?

To show altruistic cooperation among agents we implemented modified versions of several MARL games taken from existing literature. These games give the agents the option of altruistic cooperation or not cooperating at all. The agents are trained with fuzzy actor critic reinforcement learning (FACL) using

different learning structures such as hierarchical reinforcement learning. Some of the FACL hyperparameters are selected using a genetic algorithm to better learn altruistic cooperation in the games. Fuzzy reinforcement learning is used due to its simplicity, as well as its interpretability which is advantageous over deep neural networks. In addition, a major benefit of using fuzzy reinforcement learning, is that when the environment changes, the fuzzy rules can easily adapt. Fuzzy reinforcement learning has been successfully used in many applications such as [4], [5], [6].

Key contributions of this paper include:

- Using hyperparameter optimization on multi agent reinforcement learning systems. In each game, the agent system has key hyperparameters selected by a genetic algorithm. Similar research generally only has a single agent.
- The creation of continuous space altruistic games. While games that display altruistic characteristics exist already [7], these games are often in the discrete domain.
- Using fuzzy reinforcement learning in a MARL altruistic cooperation setting. Many papers that do look at altruistic reinforcement learning use neural networks and other deep learning tools which lack interpretability and explainability. By using fuzzy systems in the cooperative setting, the decisions the agents make are much clearer and related to specific fuzzy rules.

II. Related Work

As mentioned, many papers that study the learning of cooperation using reinforcement learning do not study the altruistic aspect of cooperation. However, there are a few existing papers that do.

Franzmeyer et al. study how a single agent can learn to be altruistic to help other agents succeed at their goals. By training an agent to maximize the number of states another agent will be able to reach in its future, the agent will learn altruistic behavior. The authors observe that certain hyperparameters such as discount factors and "look-ahead horizon" effect altruistic tendencies of the agent. [8]

Forester et al. [9] study a dilemma in which a beneficial action to an agent might be disadvantageous to a neighboring agent in a game called the "Coin Game". The authors create an algorithm called "Learning with Opponent-Learning Awareness" using deep neural networks to show how agents can behave altruistically in circumstances where one agent can lose points due to another agent's actions.

In [7], the authors create a couple of cooperative games to show how social dilemmas can affect cooperation. Using deep-Q networks, the authors modify parameters such as discount factors, reward ratios, and hidden network neurons to see how they can affect the behavior of the agents. Results were studied in a game theory lens using empirical payoff matrices. They show how simple parameters can affect the resulting matrix payoff from the learned policy.

What these papers have in common is that they've identified important hyperparameters that are used to learn altruistic behaviors among agents. All three papers utilize the discount factor as a hyperparameters to study. However, these papers do not use any optimization algorithms to determine what the best combination of hyperparameters might be which happens to be a major gap in the cooperation domain. Additionally, these papers use Q-based learning algorithms; our paper utilizes an actor-critic learning model for the continuous domain.

The use of genetic algorithms has been used to optimize hyperparameters within reinforcement learning algorithms in the past. The following papers [10] [11] [12] have shown success using genetic algorithms for optimization. However, an existing problem with reinforcement algorithms is repeatability. It is possible to get different results despite using the same hyperparameters. [13] [14] Many hyperparameter optimization papers lack analysis regarding the repeatability of this process. How repeatable are the hyperparameters that were selected via genetic algorithm? This paper does an analysis on this aspect by running the reinforcement learning algorithm 200 times to see how the resultant policies may deviate from each other. Additionally, these papers use some form of neural network (CNN, Deep-Q, etc) whereas we have opted to use a fuzzy system as our non-linear approximator which makes the explainability of hyperparameters clearer.

Finally, the use of using a hierarchical reinforcement learning algorithm has been used before in multi agent cooperative agent systems in [15] [16] [17]. More specifically, the use of a hierarchical reinforcement learning algorithm has been used within [18] [19]. In [18] hierarchical reinforcement learning using fuzzy systems is used to train an agent to learn deception in pursuer-evader games. The lower levels carry out primitive tasks and the higher level decides when to change tasks. In [19] a hierarchical structure is used to learn both the policy and the reward function simultaneously in the context of an adversarial game. The authors adjust the fuzzy actor critic algorithm to handle multiple reward functions with different time horizons in the hierarchical structure. In this paper we bridge the gap by applying the fuzzy hierarchical reinforcement learning algorithm to a cooperative scenario.

III. Problem Statements

It's important to design games in a way that altruism occurs in the cooperating agents. According to West et al. "cooperation is conceptually and empirically linked to altruism, which is defined as behavior in which one person foregoes a benefit to help another" [2]. This isn't always displayed in classic cooperative games in the reinforcement learning field, which makes it difficult to determine if agents are truly acting cooperatively. Three problems/games are described here that force the agents to cooperate.

A. Continuous Hallway

Taking inspiration from the Hallway game in [20], a similar game is proposed with some key differences. The goal of this game is for 2 agents to race each other to the end of the hallway, which is 15m long, but agents get a higher terminal reward if they get to the end of the hall at the same time. Instead of a discrete domain, the agents are in a continuous space to better mimic real life. The altruistic aspect here is that while the agent could get a terminal reward by going to the end of the hall first, it should learn to slow down and allow the partner to catch up so that they both get rewarded.

B. Markov Stag Hunt

Three agents are tasked to either hunt a stag or collect berries. It takes at least 2 agents to take down a stag. While it is simpler to get berries than take a stag, the stag is much more rewarding for the group. However, since the collection of food is split between all agents, if all 3 agents hunt the stag, there is less food to go around than if one of the three agents chose to pick the berries.

The dilemma is as follows: if all agents hunt the stag, all agents receive a lower terminal reward than if one agent defects to berries. But the agent who defects to berries will get a comparatively lower reward than either of the agents who had chosen the stag. The rewards are on a weighted sum basis: the team reward is the sum of everyone's rewards, and an individual reward is the value of its own catch.

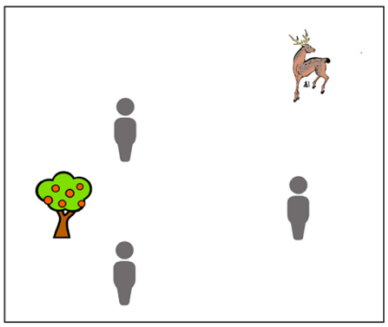

Fig. 1. Visualization of the Stag Hunt game where 3 agents must maximize their rewards.

C. Leaving A Room

The concept of having robots learn to leave a room has been seen in [21]. This game has 5 agents (robots) inside a 10x10 continuous room who have to learn to leave the room without bumping into each other. The actions the agents can choose from is to either stop or to go. The agents get rewards based on the order of those leaving the room. Letting an agent pass in front of it, means a potential lower terminal reward for the agent that waited.

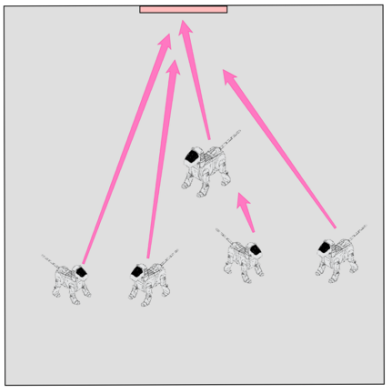

Fig. 2. Visualization of the game Leaving A Room. Five robots must exit a room without crashing into each other.

IV. Algorithms

This section describes the algorithms used to solve the games. While each game has slight differences in the algorithm (rewards etc), the basic Fuzzy Actor Critic Learning algorithm (FACL) is described here, along with the genetic algorithm used for hyperparameter optimization.

The fuzzy actor critic algorithm is used to train the agents. There are two structures that are discussed here: a simple FACL and a hierarchical FACL, which is similar to the structure presented in [22].

For each game we implement a genetic algorithm. We define an initial population where each member is a set of hyperparameters. For each set of hyperparameters, the fuzzy reinforcement learning algorithm starts learning. When the learning is complete, the agents then play a few dozen games using their static policies. The ratio of the number of successful games is the fitness for that specific set of hyperparameters. Once the total population of the hyperparameters has been assigned a fitness, the child population is created using selection, crossover, and mutation operators. The next iteration is done using the child population. Many iterations of this process are done many times to ensure convergence. All members of the final population should have the same values for each hyperparameter. Once we have these hyperparameters, we replicate the training 200 times purely for analysis purposes. We'd like to see how the results of using the hyperparameters deviate from each other. The results are show in histograms in Fig. 5, 8, and 10.

A. Fuzzy Actor Critic Learning

The Fuzzy Actor Critic Algorithm uses a fuzzy inference system as the critic and a fuzzy logic controller as the actor. The actor updates its weights based on the critic. The output of the actor at time t is calculated as follows:

$$u_t = \Sigma_{l=1}^{L} \phi^l \omega_t^l \quad (1)$$

where ω_t^l is the actor's output parameter of rule l, L is the total number of rules, and ϕ^l is the firing strength of the rule l. In this scheme we use triangular membership functions per input for their computational speed. The number of rules that fire at

any time instant is simply the number of membership functions to the power of the number of inputs. In order to calculate the firing strength of rule l, computed as

$$\phi^l = \frac{\partial u}{\partial \omega^l} = \frac{\Pi_{i=1}^{n} \mu^{F_i^l}(\bar{x}_i)}{\Sigma_{l=1}^{L}\left(\Pi_{i=1}^{n} \mu^{F_i^l}(\bar{x}_i)\right)} \quad (2)$$

$\mu^{F_i^l}(\bar{x}_i)$ calculates the membership degree of the input $\bar{x}_i$ with n being the number of inputs. During training, noise is added to the output of the actor to encourage exploration. The noise is from a normal distribution with a mean of 0, and a standard deviation σ that is specified by the user. Thus, the actual action used by the agent during learning is:

$$u_t' = \Sigma_{l=1}^{L} \phi^l \omega_t^l + N(0, \sigma) \quad (3)$$

The value functions at t and $t+1$ must be calculated in order to eventually update the output parameters of the fuzzy rules. The value function is simply the expected sum of discounted rewards and is approximated by the fuzzy inference system as:

$$V_t = \Sigma_{l=1}^{L} \phi_t^l \zeta_t^l \quad (4)$$

$$V_{t+1} = \Sigma_{l=1}^{L} \phi_{t+1}^l \zeta_t^l \quad (5)$$

Using the value function, we can estimate the prediction error/temporal difference.

$$\Delta_t = R_{t+1} + \gamma V_{t+1} - V_t \quad (6)$$

The discount factor, γ, is between 0 and 1, in this paper we propose to use a hierarchical genetic algorithm to tune the discount hyperparameter γ. The term R_{t+1} is the reward received. The critic output weights in the fuzzy inference system can then be updated using the temporal difference at t and learning rate, α.

$$\zeta_{t+1}^l = \zeta_t^l + \alpha \Delta_t \phi_t^l \quad (7)$$

Note how only the rules that fired are updated, this way if there are environmental changes, the system can easily adjust.
The output weights of the actor fuzzy inference system are updated as:

$$\omega_{t+1}^l = \omega_t^l + \beta \Delta_t \phi_t^l (u_t' - u_t) \quad (8)$$

Where β is the learning rate of the actor, and Δ_t is the temporal difference at time t, and R_{t+1} is the reward at a given timestep.

In the hierarchical structure, the lower-level policies are independently trained to achieve a particular task. Once completed, the higher-level policy is then trained while using the completed lower-level policies.

B. Genetic Algorithm for Hyperparameter Optimization

In order to select ideal parameters in the FACL algorithm, a genetic algorithm can be used for the hyperparameter optimization. The main advantage of using a genetic algorithm to aid in the selection of hyperparameters is that many processors can be used in parallel which greatly speeds up the process. Genetic algorithms are generally good for complex search spaces.

Algorithm steps:

1. Create a randomly generated population of size n and initialize parameters.
2. Train the policy(s) and assign fitness values for each set of hyperparameters in the population
3. Start loop to create new child population
4. Binary tournament to select parents to create offspring members of the next population
5. Generate a random number to check for crossover. If above crossover rate, use parents in the child population, if below, perform the crossover operation
6. Generate another random number, if below mutation rate, replace a parameter in the offspring with appropriate randomly generated value.
7. Add offspring to new child population
8. End loop once child population is full

In step 1, the population is randomly chosen, which means creating sets of the hyperparameters to test. Here we call a set of hyperparameters a member of the population, where the population is many sets of hyperparameters. A couple members are planted into the initial population that have known hyperparameters that work satisfactorily. This is called seeding the initial population and is done to speed up the search. In step 2, one implements reinforcement learning. The fitness function chosen is up to the user, for example in [10] the authors use the inverse of number of epochs it takes for the agent to reach a success rate of 0.85 for the very first time. We use a fitness function that corresponds to the number of successful tests after training. After the reinforcement learning algorithm has reached a terminal state for training, the policies of the agents are put through a number of test scenarios for evaluation. The fitness thus corresponds to how many scenarios were successfully passed by all agents. Once all the members of the current population have been trained and assigned a fitness value, the next population can be created. We use the selection, mutation, crossover, and elitism operators to create the next population to test, also called the child population.

To create the child population 4 random members of the parent (current) population are chosen and paired off. All members in the population have an equal probability of getting selected. A binary tournament is used as the selection operator. Whichever member in the tournament pair has a better fitness is used, so now there are two parent members. A crossover operation ensues, a random number between 0 and 1 is generated if the value is above the crossover rate (which is set to 0.75) then the parents are kept, otherwise crossover occurs. Up to two parameters are randomly chosen for crossover. The new parameters are created by crossing the parent parameters p_1 & p_2 using

$$p_{new\,1} = (1-\beta)p_1 + \beta p_2 \quad (9)$$

$$p_{new\,2} = \beta p_1 + (1-\beta)p_2 \quad (10)$$

β is a randomly generated value used for scaling. Note that this method does not allow for introduction of values beyond the extremes already represented in the population. For example, if the highest parameter value in the initial population is 0.8, the crossover method will not get any value greater than 0.8 using this method. The next step is mutation, where a new random value is generated to decide if these offspring will mutate. If the random number is above the mutation rate then the offspring members are not mutated, and if it is under the mutation rate then one of the corresponding hyperparameters in the offspring members is replaced with a random value within range. These offspring are then added to the child population, and the loop

restarts until the child population is replenished; then the process restarts. Through iterations, the values of the population will converge based on maximizing the fitness.

Fig. 3 illustrates a hierarchical learning structure with the hyperparameter selection. Note that only the higher level is tuned.

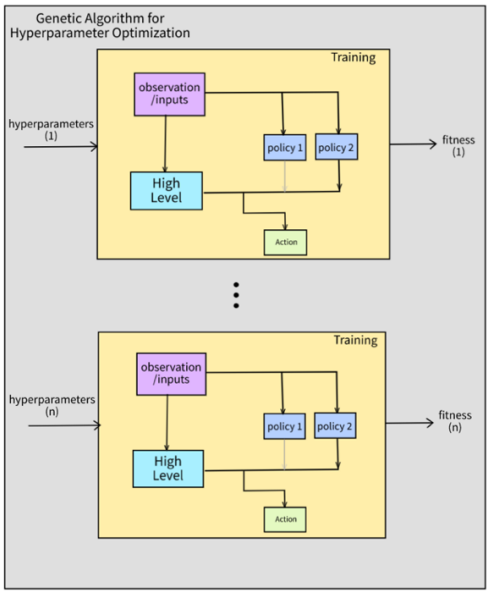

Fig. 3. A Diagram of the hierarchical structure for FACL with the genetic algorithm applied. A yellow box denotes a single FACL training session for a given set of hyperparameters in the population. The grey box shows the genetic algorithm over top.

V. Results

We now present our findings on how the genetic algorithm converged, and how well the resultant policy performs with these chosen hyperparameters.

Fig. 4, 7, and 9 show how the genetic algorithm converged to a solution. The two lines represent the average and the maximum fitness in each iteration's population. After the genetic algorithm has converged, the learning program is run 200 times to show how it performs with these hyperparameters.

In the spirit of reporting data, it is important to show metrics for reproducibility and variability using robust statistics like the interquartile range [13]. The interquartile range helps determine the variability across training runs; ideally each run does not significantly differ from other runs. The interquartile range is the difference between the 75th and 25th percentiles, it's a robust statistic that is appropriate for asymmetric distributions. The authors in [14] recommend to also report the interquartile mean to measure metrics to summarize performance across tasks since it is more robust than the simple median. They also recommend showing the tails of the distribution. Once the genetic algorithm has chosen the hyperparameters, we show these statistics for the resultant learned policies.

The hyperparameters that we are interested in are discount factor, the terminal reward weight, and the magnitude of the standard deviation of noise used during training. In [7] the authors showed that different discount factors can change the cooperative behavior of the agents. The standard deviation for noise is of interest since the amount of noise during training can greatly affect results. The authors in [23] mention how runs can be different depending on the random number seed used. By using the standard deviation of noise as a hyperparameter, the effects of using different seeds can also be minimized. And finally finding the reward weight (w) of

$$R_{terminal} = w * R_{individual} + (1 - w) * R_{team} \quad (11)$$

is of great importance, this weight shows how important individual reward is in comparison to team reward and thus defines the level of cooperation and altruism.

A. Continuous Hallway

Before looking into the optimization of hyperparameters, some extra details about the algorithm used specifically for the continuous hallway game must be discussed such as reward design, dynamics, and inputs into the system. Table 1 shows information used in the continuous hallway game.

TABLE 1
INFORMATION USED IN CONTINUOUS HALLWAY GAME

Description	
Number of Membership Functions	7
Number of Inputs	4
Maximum Training Epochs	50k
Time per training episode	80s episode -> 80 000 iterations
Inputs	Position of agent, velocity of agent, difference between position of agents, velocity of other agent
Dynamics	$a = \left(\frac{1}{m}\right) * F - bv$
Shaping Reward	$r_{t+1} = w * \Delta(distance\ to\ finish\ line) + (1 - w) * 0.01e^{-\left(\frac{distance\ away\ from\ partner}{0.1}\right)^2}$
Terminal Reward	+15 for ending with partner +3 for ending alone

The agents have been named Diana and Sharon and they must get to the end of the hall at the same time (within a tolerance of 0.3m), to do this their actor outputs a force into the dynamics of the system. The shaping reward is a weighted equation between how close the agent is to the end of the hall, and how far the agent is away from the other agent.

The hyperparameters we are interested in finding are the discount factor of agent Sharon γ_{sharon}, the discount factor of agent Diana γ_{diana}, the reward weight w, and the standard deviation of noise applied during training σ. The mutation rate was set to 0.05, crossover rate 0.75, and scaling rate β were randomly generated values between 0 and 1. The population used in the genetic algorithm was set to 20. Since the population is quite small, it was seeded to ensure the space was being properly searched.

In the FACL algorithm we have chosen the following system parameters arbitrarily: critic learning rate $\alpha = 0.1$, actor learning rate $\beta = 0.05$, mass of agent $m = 1$kg, friction coefficient $b = 0.1$, and Fuzzy Inference System (FIS) limits of the actor are ± 3. Note that these parameters are selected and remain constant.

The fitness function used is a ratio of how many successful test games there are post-learning. Once the reinforcement learning has been completed the agents play many games, the fitness is the ratio of how many games agents succeeded at getting to the end of the hallway together. These tests place agents at various starting positions along the hallway.

Fig. 4 shows the genetic solution converged quickly at approximately iteration 10 of the genetic algorithm. Many fitness values in the population are 0.9997 at convergence which tells us that 99.97% of test cases are passed after training. After 42 iterations of the genetic algorithm, the hyperparameters for all 30 members of the population were: $\gamma_{sharon,} = 0.995$, $\gamma_{diana} = 0.995$, $w = 0.9819$, $\sigma = 0.9$.

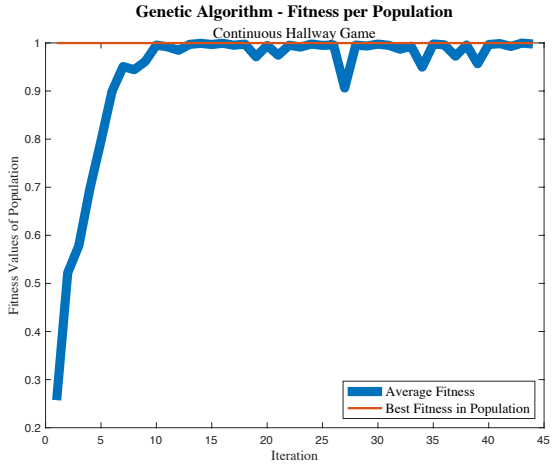

Fig. 4. Plot showing the fitness of the population per iteration for the hallway game.

Once the genetic algorithm converged to a set of hyperparameters as seen in Fig. 4, we ran only the learning simulation an additional 200 times to ensure repeatability with these parameters. Examining the performance of the hyperparameters selected for the training, we see it is quite robust. Fig. 5 shows a histogram of the performance after training with the hyperparameters that were learned from the genetic algorithm. The majority of the 200 training sessions using the learned hyperparameters go quite well, with only 7/200 runs achieving scores below 80% after training.

With the interquartile mean of 0.9997, and interquartile range of 0.9997-0.9997 (the same value) the results show that the policy can succeed at 99.97% of tests post training. The mean is 0.9739 since some runs did not perform as well.

Fig. 5. Plot of the two hundred runs using the hyperparameters chosen by the genetic algorithm

Since this game is quite simple, there are many other hyperparameters that would be successful as well, implying that there are many local maximas.

B. Stag Hunt

To solve this problem, each agent uses its own hierarchical learning structure, where both the lower levels and the higher level is trained with the FACL algorithm. The lower-level policies output the required heading needed to get the agents to their goal (stag or berries). The higher-level policy decides which lower-level policy to use; it decides if the agent should hunt the stag or collect the berries based on the positions of the other agents.

In this scenario, a genetic algorithm is used to find near optimal hyperparameters for all 3 discount factors for the agents, the terminal reward weight, and the standard deviation of noise in the higher-level decision policy.

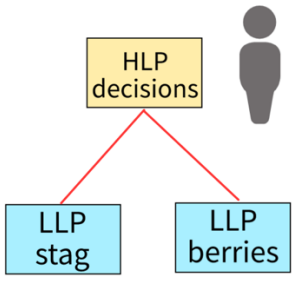

Fig. 6. Visualization of the higher and lower levels that each agent has when earning the stag hunt game

Information about the Lower Level

A policy is trained for the agent to reach the stag, and another to reach the berries. The input into the FIS is the x and y co-ordinates of the agent, and the output is the required heading for the agent to get the goal.

TABLE 2
PARAMETERS USED TO TRAIN LOWER LEVEL

Parameter (LLP)	Symbol	Value
Discount factor	γ	0.9
Standard Dev. Of noise	σ	1
Critic Learning Rate	α	0.1
Actor Learning Rate	β	0.05
Number of membership functions per input	N/A	9
Total Number of Rules	N/A	9^2
Epochs	N/A	81

The shaping reward was structured as distance from goal $r_{t+1} = d(t) - d(t+1)$. The terminal reward is +10 for getting to the goal.

Information about the Higher Level

The goal of the higher-level policy (HLP) is to decide when to choose to hunt for the stag, and when to forage for berries. Each agent trains its own higher-level policy. Every 3 seconds in the game each agent reassesses whether to capture the stag or forage for berries. The HLP for each agent will measure the x and y position of each agent including itself. These x and y positions are the inputs into each agent's HLP FIS. The output for each agent's HLP FIS is either a positive or negative number. If it is a positive number, then the actor selects the lower-level policy for berries, and a negative output selects the lower-level policy for stag. There is no formal communication between the agents, their decisions are solely made through x and y position data that updates every 3s.

The actor learning rate is 0.05, and the critic learning rate is 0.1. These values were chosen arbitrarily. The higher-level policy is trained for 100k episodes, with each episode lasting up to 30s. There are 6 inputs, which are the x and y position for each of the three agents and the range of each input is covered by 4 membership functions resulting in $4^6 = 4096$ rules. The terminal rewards are on a weighted sum basis: the team reward is the sum of everyone's rewards, and individual reward portion is the value of the catch. In this scenario, the stag is worth 10 points, the berries are worth 4 points. For demonstration purposes we show an example calculation with these values using the reward weight set to 0.5 using (11).

Scenario 1: All agents chose to go hunt the stag with a reward Weight=0.5

$$R_{ind_1} = R_{ind_2} = R_{ind_3} = \frac{10}{3}$$
$$R_{team} = 10$$
$$R_1 = R_2 = R_3 = 0.5 * 3.33 + 0.5 * 10 = 6.665$$

Scenario 2: Agents 1 and 2 hunt the stag, agent 3 collects berries with a reward weight of 0.5

$$R_{ind_1} = R_{ind_2} = \frac{10}{2}, R_{ind_3} = 4$$
$$R_{team} = 10 + 4$$
$$R_1 = R_{ind_1} * 0.5 + R_{team} * 0.5 = 5 * 0.5 + 14 * 0.5 = 9.5 = R_2$$
$$R_3 = R_{ind_3} * 0.5 + R_{team} * 0.5 = 4 * 0.5 + 14 * 0.5 = 9$$

Hyperparameter Optimization

A parallel genetic algorithm implementation was done to help speed up the process. Using the genetic algorithm described in section IV B, 200 iterations of the genetic hyperparameter selection were done using a population of 30. The mutation rate was kept at 0.01, the crossover rate was kept at 0.75, the scaling rate was a randomly generated number between 0 and 1 for every crossover. The fitness function evaluates the post-training policy on 30 different scenario tests. Each test scenario has the agents placed into the game field at locations deemed 'difficult' and require altruism. Since the population of the genetic algorithm is 30 and therefore, relatively small, some members were planted in the initial population. These members had the extreme values (0 or 1) for the parameters, this was to ensure that all values in the space had a chance during crossover, and we did not need to depend on mutation as much. Fig. 7 shows the fitness plot, after about 30 iterations, the values converged.

The hyperparameters the genetic algorithm converged to were: $\gamma_{nora} = 0.9791$, $\gamma_{jean} = 0.9823$, $\gamma_{ivan} = 0.5517$, $w = 0.018$, $\sigma = 0.9985$.

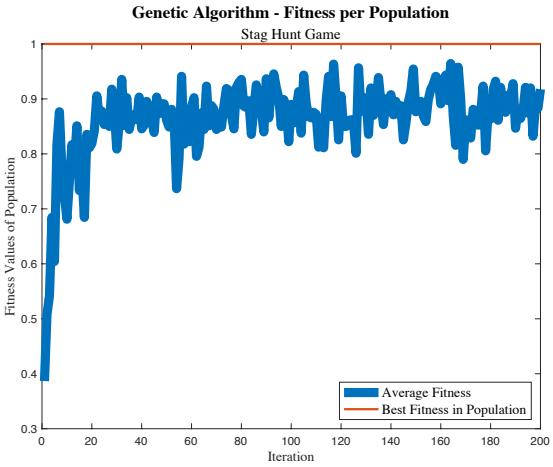

Fig. 7. Average fitness of the population of 30 for each iteration of the Stag Hunt game.

Once the genetic algorithm converged to the same set of hyperparameters, the tuning was stopped but the higher-level reinforcement learning program was run 200 times with these same hyperparameters using different random generator seeds to ensure consistency between training results. The results are shown in Fig. 8.

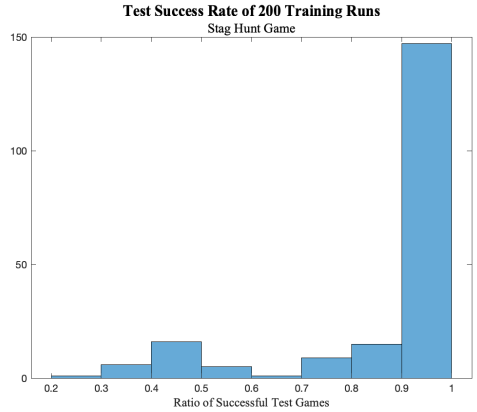

Fig. 8. Histogram of 200 policy results from training with the hyperparameters tuned with the genetic algorithm.

Looking solely at the consistency of the reinforcement learning results with the selected hyperparameters, the histogram in Fig. 8 shows 147 of the policies trained were 90%+ successful in the test scenarios, and the fitness plot shows that the best member in the population always scored 100%. The test scenario is marked as successful if two agents hunt the stag and one agent collects the berries from wherever they were placed in the space. Otherwise, the test was not successfully passed. The fixed test score had an interquartile range of 0.1 (10%). This stipulates that from the 25th percentile to the 75% percentile, there was a difference of 10% success rate among the tested policies. Since there were 30 different test scenarios, 10% difference in the IQR infers that only 3 tests were not passed at the extremes of this range: not a large difference. The mean of the 200 policy results from Fig. 8 was 0.8909 and the interquartile mean was 0.8785. Since the interquartile mean is lower than the regular mean, it implies that there are more highly successful runs than failed runs.

C. Leaving A Room

In this game, there are 5 robots placed in a 10x10 room with a door placed in the middle of a wall. The goal is for the robots to all leave the room without bumping into each other (if robots are within 0.3 m of each other, it counts as a crash). The 5 robots are placed randomly at least 5m away from the door. All the robots must train a single common policy; the goal is to have the robots cooperatively learn when it is beneficial to forgo a greater future reward and allow another robot to pass in front of it.

The lower-level policies either output a heading or nothing at all; the higher-level policy will choose from two options. This means the robots can either stop or go. The robots receive 5 inputs, which are all distances away from the other robots and the door. The distance input is positive if the object is in front of it, but negative if the object is behind in the y direction.

The higher-level policy will call a lower-level policy and then execute the lower-level policy for 3 seconds before returning to the higher-level policy. If the higher-level policy chooses to move, then the lower-level policy that gets executed will input the x and y coordinates and the robot will move according to the heading being output from the actor. If the high-level policy chooses to stop, then the robot stays in place for those 3 seconds.

The distance used as an input is calculated below. If a neighboring robot is closer to the exit door in the y direction, the distance value is positive, otherwise it is negative.

$$d = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2} \qquad (15)$$
$$if\ y_a > y_b, d \leftarrow -d$$

All 5 robots train the same higher-level policy, the same actor and critic. The goal is for the robots to find one single policy that dictates when the robots should let another one pass in front of it to avoid a collision, and when it should be the one to do the passing. To do this, each robot uses (7) and (8) to update the same actor and critic, based on its current observations from the rules fired during a training iteration. This means that every game iteration will have an update to the actor and critic made by each of the 5 robots. The robots get a reward of -0.1 for every higher-level policy iteration, and their terminal reward is dependent on the order of those that exit. For example, the first robot that exits receives a reward of 8, the second robot that exits receives a reward of 7 etc. Their terminal reward is described by (11). It is once again a weighted function of individual reward based on the order they exited, and the sum of the rewards received for the team reward. If robots collide, then each robot receives -2 as their $R_{individual}$.

TABLE 3
INFORMATION USED TO TRAIN LEAVING A ROOM GAME

Parameter	Symbol	Value - LLP	Value - HLP
Discount factor	γ	0.9	Hyperparameter
Standard Dev. Of noise	σ	0.5	Hyperparameter
Critic Learning Rate	α	0.1	0.3
Actor Learning Rate	β	0.05	0.15
Number of membership functions per input	N/A	6	6
Total Number of Rules	N/A	2^6	6^5
Epochs	N/A	2000	12k

The parameters presented in Table 3 were arbitrarily chosen.

The genetic algorithm used had a population of 50, each population member contains 3 hyperparameters (discount factor, terminal reward weight, standard deviation of noise used for exploration), with 100 iterations of hyperparameter tuning completed. The crossover rate was 0.75, with a mutation rate of 0.01.

A population of 50 was randomly generated with the extreme values of the search [0,1] seeded into the initial population. The higher-level policy is then trained with each set of hyperparameters in the population. In the training, each agent takes the distance inputs and outputs a value. The sign of that value indicated whether to stop or go. Then the lower-level policies execute accordingly for 3 seconds before going back to the higher level. This is repeated in the game until either all agents make it out the door or 80s passed. Once the 12,000 epochs of training have completed for a member of the population, there are 55 games played that require altruism from at least one agent. Each training epoch corresponds to a

game where the agents are randomly placed in the room and are required to exit without bumping into each other. The fitness value is assigned based on the ratio of successful games where all 5 robots exit the door. Once all 50 members of the population have been assigned a fitness value, the genetic operators are used to create a new population and repeat 200 times. The genetic algorithm had 200 iterations done to ensure convergence, but convergence appeared around iteration 15. To test the converged hyperparameters, the learning session was run 200 times with the fitness results of each session plotted in Fig 10. A fitness result from a learning session is the ratio of success from the 55 test games.

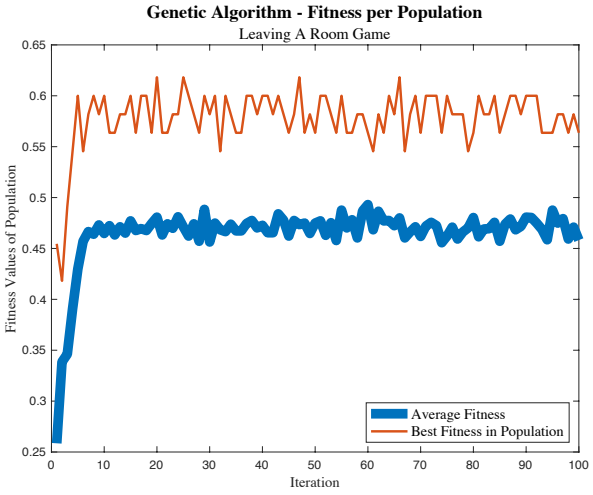

Fig. 9. The fitness per population of the Leaving A Room game. A convergence happens around 7 iterations

The hyperparameters converged to $\gamma = 0.0325$, $\sigma = 0.8843$, $w = 0.8619$. A very low discount factor of 0.0325 implies that the agents optimize for the immediate reward.
Fig 10. shows the test results from the 200 individual training sessions. The training session is all 5 agents learning to exit the room in 12,000 epochs of training with the test results being the ratio of the 55 successful games played after training. These 200 training sessions were completed with these hyperparameters to look at the variation in policies and results. The policies were then tested using 55 test cases. The interquartile range was 0.0727, with an interquartile mean of 0.4731 (or 26 altruistic test cases passed).

Fig. 10. A Histogram of the success rate of 200 training sessions

While not perfect, the hyperparameters the genetic algorithm chose showed that a low discount factor was beneficial, high variation of noise was needed, and that there is a large individual reward component in the terminal rewards.

VI. Discussion

This section is divided into three discussions: discount factors, reward weights, and the standard deviation of noise used. The purpose is to individually discuss how these hyperparameters can contribute to altruism among agents with respect to the games.

A. Discount Factors

As previously mentioned, the discount factor has been found to change cooperative behavior [7]. One popular perspective of the discount factor is that it controls the time horizon, and thus the priority of short-term rewards. Another perspective is that it helps with the stability of an algorithm. The authors in [23] claim that "some common approximation error bounds may be tightened with a lower discount factor." However, finding an appropriate discount factor for performance is non-trivial. While high values of discount factors allow for a greater time horizon, it also means that the stability of the algorithm worsens due to function approximation errors that are not getting discounted. The goal of using the discount factor as a hyperparameter is to find a balance between these aspects and see how it affects the cooperation among agents. The results from the genetic algorithm are then compared to cases using other values that were chosen using user intuition.

Here we analyze two games in reference to the stag hunt game: one where all the agents start at the same position in the field, and another where the agents start at different positions. Four HLPs are analyzed with respect to the different discount factors seen in Table 4. For the Stag Hunt game, we ran the hyperparameter optimization again using a population of 50, with very little difference in results. Hyperparameter set 1 corresponds to the results selected by the genetic algorithm with a population of 30, and hyperparameter set 2 is with the

population of 50. Notice that the largest difference is with discount factor 3 (γ_3). With a discount factor of 0.1768 the agent barely sees the next 2 steps, and with 0.5517 the agent sees the next 5 steps which isn't a large difference in a continuous domain. Hyperparameter set 3 keeps the optimized values for w & σ but replaces the discount factors with 0.9 for all agents. Finally, hyperparameter set 4 is used to demonstrate what user intuition for the hyperparameters might give.

TABLE 4
DESCRIPTION OF POLICY SETS FOR ANALYSIS

Policy set #	Description	γ_1	γ_2	γ_3	w	σ
1	Selected with a population of 30	0.9791	0.9823	0.5517	0.0184	0.9986
2	Selected with a population of 50	0.9887	0.9785	0.1768	0	0.9974
3	Replaced with new discount factors	0.9	0.9	0.9	0.0184	0.9986
4	Selected with user intuition	0.9	0.9	0.9	0.5	0.5

In Fig. 11, the two cases are plotted. The top plot shows the agents paths, and the bottom part of the plot shows the actor FIS against time for each agent. In case 1 where all agents start at (25,15) the agents know where to go right from the first step. Ivan, the low discount factor agent simply goes to the goal closest, and the high discount agents choose the stag. In case 2 the agents are spread out positioned at (0,40), (10,50), (15,0), it takes the agents 5 decision steps to reach a similar consensus. Hyperparameter set 2 plots are not shown due to the similarity in results of hyperparameter set 1.

In the case where all agents have the same discount factors (as seen in Fig. 12), the agents unanimously choose the berries. If a single agent selects the stag, then this agent receives a negative reward, whereas the berries do not have this effect. The berries do not have any such restriction. It does not come as a surprise that no agent behaves altruistically when all agents see the same steps ahead. Hyperparameter set 4 was excluded from the plots since it resembled that of hyperparameter set 3. Since the results mimic hyperparameter set 3, it's clear that the discount factors have the greatest impact.

An analysis of the actual policies shows that the decision of whether to hunt the stag or collect the berries generally is chosen correctly within the first few decisions. Based on the hyperparameters, it appears beneficial to have agents learn with different discount factors. The discount factor appears to give agents a role in the system. For example, we see that the agent with a low discount factor will choose the goal that is closest, whether that is a stag or berries. When the berries are the closest, the agent takes on a more altruistic role.

In case 2 with the hyperparameters optimized, the agents with high discount factors initially choose the nearest goal as well, this can be seen in the Agent FIS diagrams. However, within a couple time iterations, the agents correct course.

Another interesting aspect is that the lower discount factor agent, Ivan, will also always choose stag if he is much closer to it. In this case one of the other agents will compensate and collect the berries. Since Ivan has a low discount factor, he is more interested in rewards that can be achieved sooner. Fig. 11 shows that at first Ivan's HLP output is very small, and then after outputs larger and larger negative values corresponding to the stag.

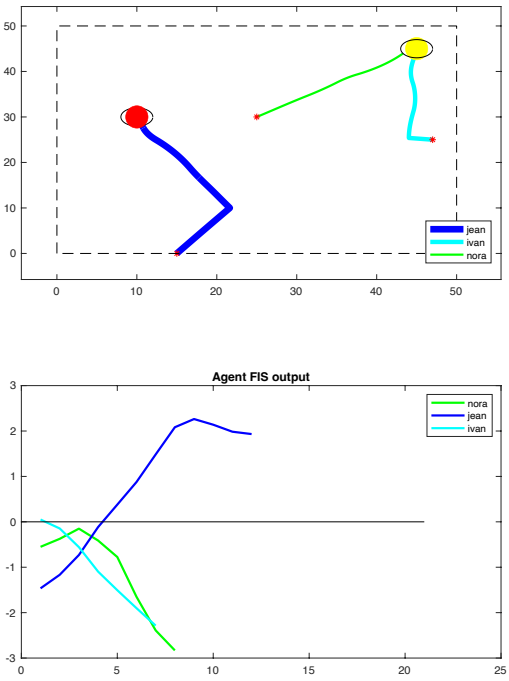

Fig. 11. These plots show the Agent's paths and FIS outputs over time. We can see that when Ivan is much closer to the stag, he will choose it instead of the berries. The top plot shows the path of each agent in the x-y plane. The bottom plot shows the HLP FIS output over time.

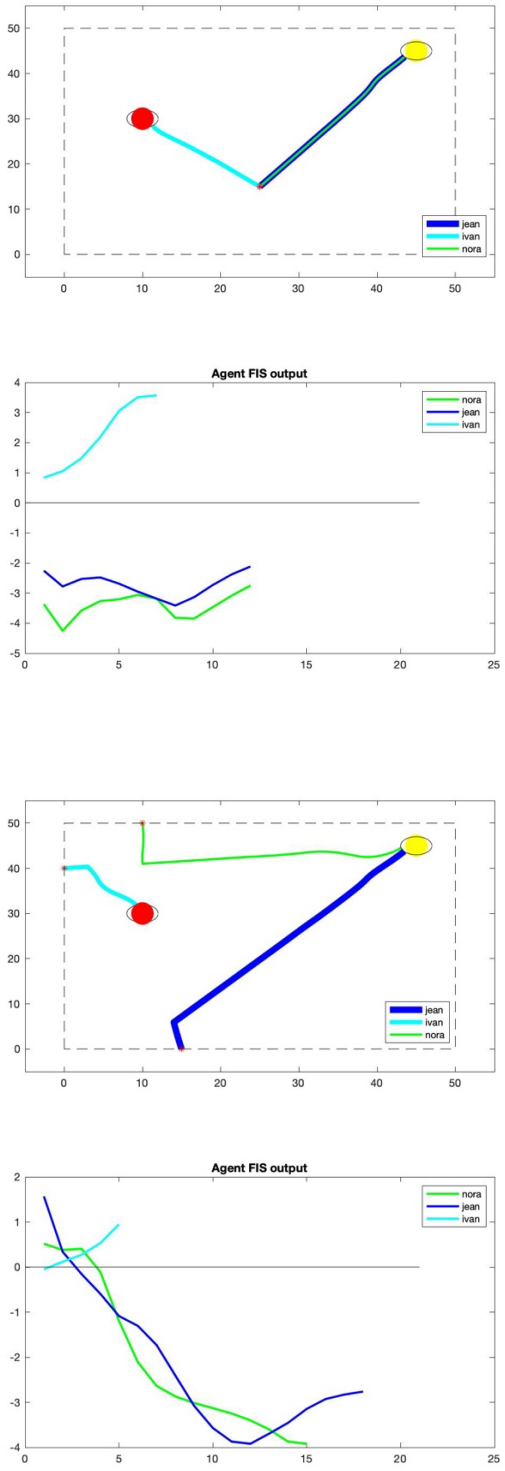

Fig. 12. Using hyperparameter set #1, the top plots show case 1 where all agents start from the same place, and the bottom plots shows game 2.

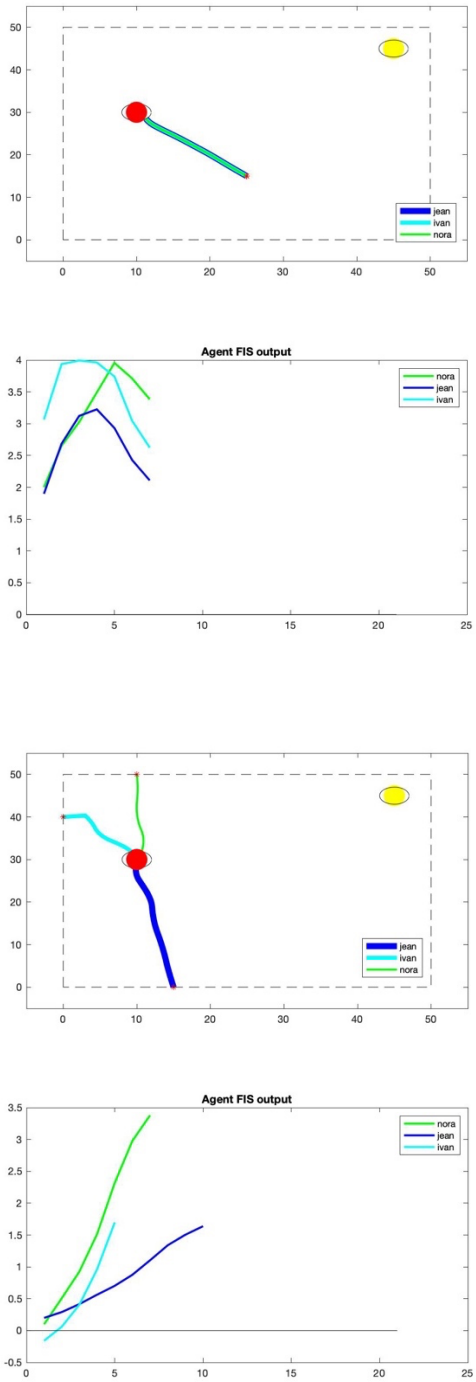

Fig. 13. Hyperparameter set 3 case 1 (top) and case 2 (bottom)

The continuous hallway game is also affected by the discount factors. Fig. 14 and Fig. 15 show the position versus time plots with the velocity versus time plots underneath.

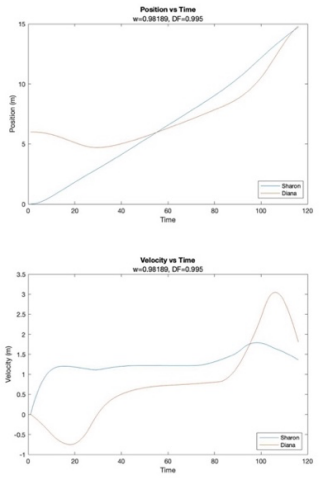

Fig. 14. The top plot shows position versus time and the bottom plot shows velocity versus time from a policy that was trained using the hyperparameters that were selected by the genetic algorithm: $\gamma_{sharon} = 0.995, \gamma_{diana}$ =0.995, w=0.9819, σ=0.9

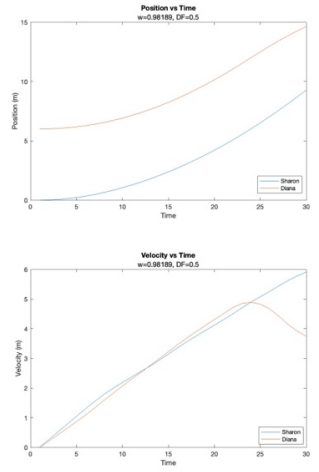

Fig. 15. The top plot shows position versus time and the bottom plot shows velocity versus time from a policy that was trained with $\gamma_{sharon} = 0.5, \gamma_{diana}$ =0.5, w=0.9819, σ=0.9

In Fig. 14 and Fig. 15 we can see how the discount factor impacts the learned output weights for each rule. The policy learned by Diana in Fig. 15 shows the rules that get fired near the end of the hallway have highly negative weights, which is to decrease speed (negative force applied). (For example, rule 1895 corresponds to a -1.7 weight) The effect of this is seen in the velocity plot. However, with a larger time horizon as seen in Fig. 14 we do not see this. (Rule 1895 is just -0.0117) It's clear that a higher discount factor for both agents is better since their terminal reward varies largely depending on the result of the game. A further time horizon was necessary to better learn the weights of the rules.

In the game Leaving a Room the five robots are tasked to learn to exit a room without bumping into each other. All 5 robots train one single shared policy. The higher level uses 5 inputs which are 4 distances away from the other robots and 1 distance away from the door. The discount factor found by the genetic algorithm was 0.03. A discount factor of near 0 implies that the current reward is the most important. Here we look at a test scenario where the agents are similarly positioned in the room. The learning algorithm is run once with $\gamma = 0$ and another time with $\gamma = 1$ (keeping the other hyperparameters the same) to compare.

A set of videos showing some example tests for Leaving A Room are available at [24]. In these videos we see test cases that show scenarios where if some agents simply went towards the goal without stopping, there would be a crash and hence negative rewards allocated. This game has the agents decide between whether to simply stay in the same spot for the entire game (not to risk a costly crash) or go forward towards a large reward. We see that when comparing discount factors, the time horizon plays a major role in altruistic behaviors. A smaller discount factor allows agents to decide in the moment, without the consideration of a large terminal reward. Since the shaping reward is simply -0.1 regardless of if an agent chooses to stop or go, there is more incentive to be safe and let other agents go ahead. We see this in the video, agents who are further from the door tend to wait until the other agents get much closer to the goal. These further agents forgo a higher terminal reward instead of risking a crash. Fig. 15 shows two frames from the video showing these agents waiting while others go, thus acting altruistically to ensure that there are no crashes where the lines intersect.

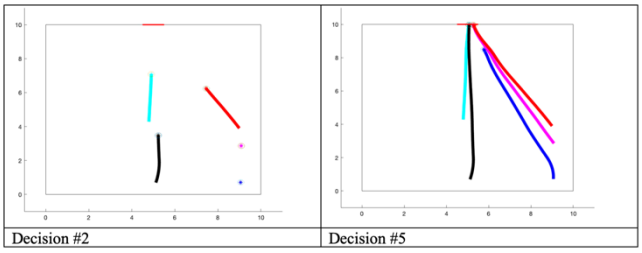

Fig. 16. Side by side plots of the agents early in the game, vs later in the game. Decision 2 corresponds to the second output of the higher-level FIS policy etc. Blue and Magenta initially chose to wait earlier on in the game.

In Fig. 17 we compare the genetic algorithm's selected discount factor of 0.018 with a discount factor of 0.99, while keeping the other parameters constant. In a high discount factor scenario, the red and the magenta agents will crash if one or the other does not act altruistically. With the discount factor of 0.018, the red agent who is closer to the door goes first and the magenta agent waits until it is safe to pass. When looking at the policy trained with a discount factor of 0.99, magenta and red crash into each other and the other agents who had started moving toward the door stop, to not risk another crash. Lower discount factors will make agents favors immediate rewards, which minimizes the amount of crashes.

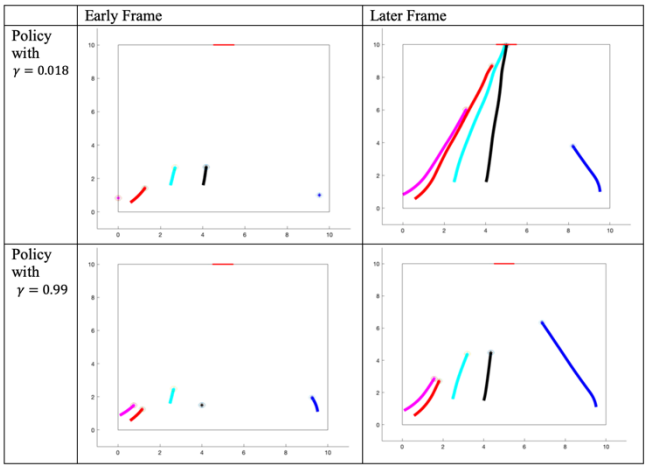

Fig. 17. A comparison with policies. The top shows the progression of the game with a low discount factor. The bottom shows the same with a high discount factor.

To get a general idea of the policies that developed we can take an average of all the weights associated to rules that were trained. In this game a positive value corresponds to stop, and a negative value corresponds to go. Taking the average of all the weights in the HLP policy we can see that the policy with $\gamma = 0.99$ has an average of 0.3110 whereas the policy trained with $\gamma = 0.018$ has an average of -0.4267. So, an HLP policy with more positive weights is more likely to have the agent stop than go.

Since not all test cases are successful, future work would look at how using a genetic algorithm to select more of the hyperparameters such as learning rate and the time between higher level policy calls (decisions) would improve the success rate.

Looking at the games, we can see that it can be beneficial for agents to all have their own discount factor. We also see with the Leaving a Room game that immediate danger of crashing requires a lower discount factor. A higher discount was more likely to be chosen for agents whose reward for success was determined by the terminal reward. The discount factor can clearly impact the altruistic tendencies of learning agents. The discount factors are also chosen alongside the reward weight which also impacts the altruistic tendencies.

B. Reward Weight

One of the major parameters that helps the agents learn to behave in a certain way are the terminal rewards given at the end of a game. In the stag hunt game, the terminal reward is given based on the individual reward contribution and the overall reward value achieved by the group (team reward). The genetic algorithm shows us that the team reward is a very important component (w=0.018). Comparing the agent's behavior with a team reward that favors individual contributions (larger values of w) shows that all agents defect to the berries, the safest option which still gets a positive reward for all agents. If the terminal reward weight were set to 1 (which implies that all agents get only their individual reward and no team reward), there would be high risk for choosing the stag alone, since one agent choosing the stag corresponds to a -2 reward. In that scenario, the agent would be given a negative reward; however, by always choosing the berries, there is no risk of a negative reward. The same pattern of berry defection happened with the discount factors mentioned in section VI A, where all agents defect to berries.

To get an idea of the importance of having the terminal reward weight favor the team reward aspect, we can take an average value of all the fuzzy rule weights in the policy learned in each agent. During training, each rule has a corresponding weight that is updated. Negative values correspond to the stag, whereas positive values correspond to the berries. We can see that by retraining the agents with the reward weight of 0.9 which favors individual contribution over team reward weight, the average weights learned are positive and lean heavily toward favoring the berries. The optimized policy parameters ($\gamma_{nora} = 0.9791$, $\gamma_{jean} = 0.9823$, $\gamma_{ivan} = 0.5517$, $w = 0.018$, $\sigma = 0.9985$) show that two agents on average prefer the stag and one agent prefers the berries, which makes much more sense for the game.

TABLE 5
AVERAGE VALUE OF ACTOR OUTPUT WEIGHTS WITH RESPECT TO REWARD WEIGHT USED IN TRAINING

	Nora ($\gamma = 0.9791$)	Jean ($\gamma = 0.9823$)	Ivan ($\gamma = 0.5517$)
$w = 0.018$	-0.2186	-0.0935	0.0456
$w = 0.9$	0.3650	0.2931	0.1330

With respect to the hallway game, we can see that the reward weight used in the shaping reward effects the speed of the game played and the exactness. By looking at the plots in Fig. 18 showing position versus time and velocity versus time, we can see how the weight of the shaping reward impacts the system. In these plots Diana starts at the 6m mark, and Sharon starts at the 0m mark. A higher reward weight favors getting closer to the end of the hall, and a smaller reward weight favors staying closer to the other agent. Fig. 18 shows a game trained with w=0.2, and Fig. 19 shows the same game trained with w=0.8. It's clear that having the weight favor 'stay close to other agent' results in a longer and poorer strategy, however the agents are within 0.0032m of each other at the end of the hallway. When the weight favors getting closer to the end of the hallway, the agents are much faster, but they are within 0.1782m of each other instead. Having a genetic algorithm chosen reward weight of 0.9819 implies that it's very important for agents to be rewarded for getting to the end of the hallway, but still slightly important that they stay within a reasonable distance of their partner agent.

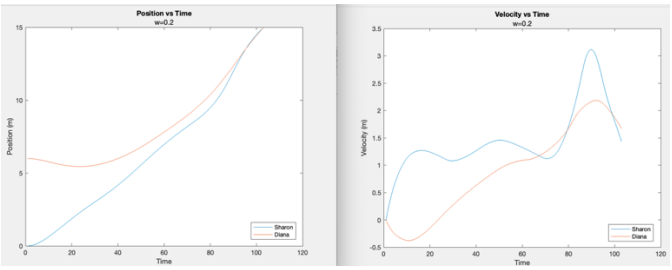

Fig. 18. Plot of the agents path and velocity vs time with a shaping reward weight of 0.2

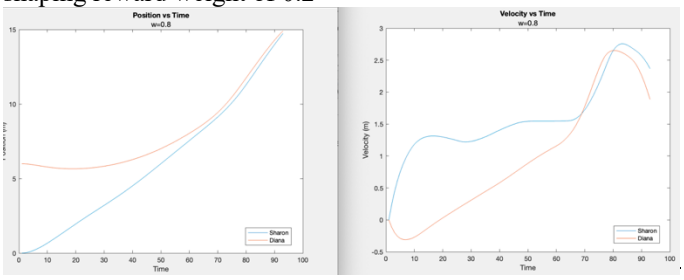

Fig. 19. Plot of the position and velocity of the agents vs time with a shaping reward weight of 0.8

With the reward weight set to 0.8629 in the Leaving A Room game, the individual reward is favored more than the team reward. Using the same example as in Fig. 16 the red and magenta agents crash into each other if one does not act altruistically. Analyzing what rewards these agents would get in the scenario where they crash would be:

$$R_{team} = 8 + 7 + 6 - 2 - 2 = 17$$

The first agent gets an individual terminal reward of 8, the second-place agent gets 7, and third place receives a reward of 6. The two agents that crashed both receive -2.
Using a weight of 0.8629 for the agents that crashed will give them a reward of

$$R_{terminal} = w * R_{ind} + (1 - w) * R_{team} \quad (11)$$
$$R = 0.8629 * -2 + (1 - 0.8629) * 17 = 0.6049$$

Using a reward weight of 0.2, the agents that crashed would get a terminal reward of 13.2, a much higher reward for failure which does not make sense to use. The reward weight of 0.8629 shows us that it is better to at least try to make it to the goal and risk a crash than it is to just sit in place. If a crash does happen, the reward is much less than if the agents made it to the finish line.

C. Noise

The last parameter to discuss is the standard deviation of noise applied to the actor outputs during training. In the FACL algorithm, noise must be added to the actor output in order to simulate exploration during training. The noise comes from a normal distribution, and the parameter of interest is the standard deviation of that normal distribution. The genetic algorithm had to select a standard deviation between 0 and 1. While this parameter does not have a direct impact on altruism, it is crucial for the learning process.

The three games studied all had high values chosen for standard deviations: 0.9, 0.99857, and 0.8843. This implies that a larger variety of noise is required for learning altruism in this setting.

VII. Conclusion

This study shows the importance of training agents for altruistic behaviors in cooperative games, without altruism it is difficult to determine if true cooperation is happening in multi agent settings. Three new continuous space games were designed to help identify when altruism was happening among agents. The agents were trained using fuzzy actor critic learning, and in two of the games a hierarchical structure was used in learning. While many hyperparameters in learning algorithms are often chosen arbitrarily, this paper shows that certain behaviors can emerge through the use of automated selection of hyperparameters in multi agent systems. By using a genetic algorithm to select hyperparameters, we can customize the fitness function to better select hyperparameters that pass altruistic tests. The hyperparameters that were used to help the agents learn altruistic behaviors were the discount factor, a reward weight used in either the shaping reward or terminal reward, and the standard deviation of noise applied to the actor during learning.

VIII. References

[1] A. A. M. Salem, M. Abdelsattar, A.-D. A. Mosaad, A. H. Al-Hwailah, E. Derar, N. A. Al-Hamdan and S. T. Tilwani, "Altruistic behaviors and cooperation among gifted adolescents," *Frontiers in Psychology,* 2022.

[2] S. A. West, A. S. Griffin and A. Gardner, "Social semantics: altruism, cooperation, mutualism, strong reciprocity and group selection," *Journal of Evolutionary Biology,* vol. 20, pp. 415-432, 2007.

[3] O. e. a. Vinyals, "StarCraft II: A New Challenge for Reinforcement Learning," *arXiv,* vol. arXiv:1708.0478, 2017.

[4] C. Fan, S. Bao, Y. Tao, B. Li and C. Zhao, "Fuzzy Reinforcement Learning for Robust Spectrum Access in Dynamic Shared Networks," *IEEE Access,* vol. 7, pp. 125827-125839, 2019.

[5] W. Wang, C. Du, W. Wang and Z. Du, "A PSO-Optimized Fuzzy Reinforcement Learning Method for Making the Minimally Invasive Surgical Arm Cleverer," *IEEE Access,* vol. 7, 2019.

[6] C.-F. Juang and C.-M. Lu, "Ant Colony Optimization Incorporated With Fuzzy Q-Learning for Reinforcement Fuzzy Control," *EEE Transactions on Systems, Man, and Cybernetics ,* vol. 39, no. 3, pp. 597-608, 2009.

[7] J. Z. Leibo, Z. Vinicius, M. Lanctot and J. Marecki, "Multi-agent Reinforcement Learning in Sequential Social Dilemmas," in *Sixteenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Sao Paulo, 2017.

[8] T. Franzmeyer, M. Malinowski and J. F. Henriques, "Learning Altruistic Behaviours in reinforcement Learning withut External Rewards," in *International Conference on Learning Representations*, 2022.

[9] J. Foerster and R. Y. Chen, "Learning with Opponent-Learning Awareness," in *International Conference on Autonomous Agents and Multiagent Systems*, Stockholm, 2018.

[10] A. Sehgal and H. Nguyen, "Deep Reinforcement Learning using Genetic Algorithm for Parameter Optimization," in *International Conference on Robotic Computing (IRC)*, Naples, 2019.

[11] C. G. J. M. Hossain Delowar, "Optimizing Deep Learning Parameters Using Genetic Algorithm for Object Recognition and Robot Grasping," *Journal of Electric Science and Technology,* vol. 1, no. 1, 2018.

[12] T. Hinz, Navarro-Guerrero, S. Magg and S. Wermter, "Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks," *International Journal of Computational Intelligence and Applications,* vol. 17, no. 2, p. 1850008, 2018.

[13] S. C. Chan, S. Fishman, J. Canny, A. Korattikara and S. Guadarrama, "Measuring the Reliability of Reinforcement Learning Algorithms," in *Internation Confrence on Learning Representations*, Addis Ababa, 2020.

[14] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville and M. G. Bellemare, "Deep Reinforcement Learning at the Edge of the Statistical Precipice," in *Neural Information Processing Systems*, 2021.

[15] J. Ma and F. Wu, "Feudal Multi-Agent Deep Reinforcement Learning for Traffic Signal Control," in *International Conference on Autonomous Agents and Multiagent Systems*, Auckland, 2020.

[16] J. Yang, I. Borovikov and H. Zha, "Hierachical Cooperative Multi-Agent Reinforcement Learning with Skill Discovery," in *International Confrence on Autonomous Agents and Multiagent Systems*, Auckland, 2020.

[17] A. Levy, G. Konidaris, R. Platt and K. Saenko, "Learning Multi-Level Hierarchies with Hindsight," in *International Conference on Learning Representations*, New Orleans, 2019.

[18] A. Asgharnia, H. Schwartz and M. Atia, "Learning multi-objective deception in a two-player differential game using reinforcement learning and multi-objective genetic algorithm," *International Journal of Innovative Computing, Information, and Control,* vol. 18, no. 6, pp. 1667-1688, 2022.

[19] A. Asgharnia, H. Schwartz and M. Atia, "Hierarchical Reinforcement Learning With Multi Discount Factors In A Differential Game," in *IEEE Symposium Series on Computational Intelligence*, Singapore, 2022.

[20] Wang, Tonghan; Wang, Jianho;, "Learning Nearly Decomposible Value Functions via Communication Minimization," in *International Conference on Learning Representations (ICLR)*, Addis Ababa, 2020.

[21] S. Givigi and H. Schwartz, "Swarm Robot Systems Based on the Evolution of Personality Traits," *Turkish Journal of Electrical Engineering and Computer Sciences,* vol. 15, no. 2, pp. 257-282, 2007.

[22] A. Asgharnia, H. M. Schwartz and M. Atia, "Deception in A Multi-Agent Adversial Game: The Game of Guarding Several Territories," in *IEEE Symposium Series on Computational Intelligence*, Canberra, 2020.

[23] M. Petrik and B. Scherrer, "Biasing Approximate Dynamic Programming with a Lower Discount Factor," *Advances in Neural Information Processing Systems,* pp. 1265-1272, 2009.

[24] R. Haighton, "Leaving A Room Videos Folder," 2023. [Online]. Available: https://drive.google.com/drive/folders/1Bg25stFGrfLa4z_oy382VTQbc4F8WoTa?usp=sharing.