Correlated Q-Learning

Scott Merrill
smerrill7@gatech.edu
git hash: 41b50c9dfec35d99961ae798177c55b0da706c6f

I. Introduction

Reinforcement Learning (RL) is typically modeled as a Markov Decision Processes (MDP) and is designed to enable a single agent to identify the series of actions that lead to the maximum cumulative reward. While this framework is sufficient for many problems, it ignores actions taken by friends, adversaries and other agents who themselves are altering the environment. This paper explores RL environments involving multiple agents with shared or competing goals.

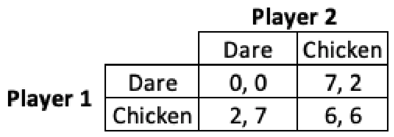
While many algorithms have been proposed to identify optimal equilibrium policies in multiagent environments, current solutions either have limited applicability or require strict criteria to converge. Littman's Friend-or-Foe-Q (FF-Q) [3] can identify optimal policies specifically in two player zero-sum games. Hu and Wellman's Nash-Q [2] is applicable to general sum games, however it converges only when a single unique Nash Equilibrium exists. Thus, a solution is needed to generalize the larger class of general sum multiagent environments with multiple equilibria. Greenwald and Hall's Correlated-Q (CE-Q) [1] generalized both Nash-Q and FF-Q and demonstrates empirical evidence of convergence to the optimal equilibrium points in several Markov Games. In this paper, we replicate Greenwald's Soccer Game experiment to analyze the convergence properties of CE-Q in zero-sum games and comment on its appropriateness in general-sum games.

II. Reinforcement Learning for Game Thoery

A. Nash Equilibrium

Game theory studies problems concerning non-cooperative competing agents whereby each agent acts as to maximize their own utility. All finite games have a Nash Equilibrium (NE), or a solution such that no player can increase their utility by taken an alternate action. Thus, each agent attempts to maximize their reward with respect to the action probability distribution of the other players. Such equilibrium points are interesting as they define the logical actions of players that need not provide optimal payoffs.

Consider the two-player game of Chicken in Figure 1 where both the row and column players can dare the other player for a riskier payoff of 7 or 0 or be chicken for risk averse payoffs of 2 and 6. The game has two pure-strategy NE at states (Dare, Chicken) and (Chicken, Dare); in these states, neither player can increase their reward by changing their action. In a single iteration of the game, the average expected reward to each player is 4.5 which is suboptimal; if both players coordinate and Chicken, they each would receive a reward of 6.



		Player 2	
		Dare	Chicken
Player 1	Dare	0, 0	7, 2
	Chicken	2, 7	6, 6

Figure 1: Chicken Payoff Table

In multiple iterations of Chicken, however, their also exist a mixed-strategy NE. Rather than taking a deterministic action, each agent can choose to Chicken or Dare based on some probability distribution. In this particular game, the optimal decision of each player can be shown to dare 1/3 of the time and play chicken 2/3 of the time. Such results in an average reward to each player of 4 2/3. Moreover, while this mixed strategy NE provides a higher average reward than the pure strategy NE, it is still less than the expected value to both players if they cooperated.

B. Correlated Equilibrium

A correlated equilibrium (CE) occurs when each agent chooses their action based on a public signal. These signals enable inference allowing players to optimize with respect to the other player's probability distribution conditioned on their own; thus, CE generalizes Nash.

In the Chicken example, a CE can be created by introducing a central planner that advises each agent on the action they should select. This central planner is designed to prevent both agents from choosing to dare and receiving a reward of zero. Thus, the planner chooses amongst 3 combinations of advice to provide; he can tell both players to chicken or tell one player to dare and the other to chicken. Both players know of the planner's intentions and if told to chicken, they can induce a 50% chance the planner told the other player to chicken and a 50% chance he told the other player to dare. Thus, the player's expected payoff if they chicken is 4 $\left(\frac{1}{2} * 6 + \frac{1}{2} * 2\right)$ and their expected payoff if they dare is 3.5 $\left(\frac{1}{2} * 7 + \frac{1}{2} * 0\right)$. Since, a rational agent will always chicken if told to do so, it follows they will also dare if told to dare; as when told to dare there is a 100% chance the other player was told to chicken, a reward of 7 would be received. Moreover, if each agent acts according to the planner, their expected payoff would be 5 $\left(\frac{2}{3} * 4 + \frac{1}{3} * 7\right)$. Thus, the CE created by introducing the planner is an improvement over both pure and mixed-strategy NE.

C. Markov Games

Stochastic games are multiagent repeated games with probabilistic transitions. Such can be formally expressed by a tuple $\langle I, S, A_i(s), P, R_i(s, \vec{a}) \rangle$, where I is the set of players, S is a set of states, $A_i(s)$ cooresponds to the action space for player i at state s, P is a probabilistic transition function and $R_i(s, \vec{a})$ corresponds to player i's reward for joint actions $\vec{a} \in (A_1(s), \ldots, A_n(s))$ taken by all agents at state s. Markovian games are a subclass of stochastic games in which the transition function is independent of past actions. That is, $P[s_{t+1}|s_t, \vec{a}_t, \ldots, s_0, \vec{a}_0] = P[s_{t+1}|s_t, \vec{a}_t]$.

MDP's are single-agent Markov Games whose optimal action-value function can be characterized by Bellman's Equations shown in (1), (2) and (3). $Q^*(s, a)$ describes the long-term reward from taking action a in state s, receiving reward $R(s, a)$ and following the optimal policy thereafter. $V^*(s)$ is an estimate of the value of state s and $\pi^*(s)$ is the policy which defines the action that maximizes the agent's value at state s.

$$Q^*(s, a) = (1-\gamma)R(s, a) + \gamma \sum_{s'} P[s'|s, a]V_i(s') \quad (1)$$

$$V^*(s) = \max_{a \in A(s)} Q(s, a) \quad (2)$$

$$\pi^*(s) \in argmax_{a \in A(s)}\, Q^*(s, a) \quad (3)$$

In Markov games, $Q^*(s, a)$ and $V^*(s)$ are defined over state and action-vector pairs.

$$Q_i(s, \vec{a}) = (1-\gamma)R_i(s, \vec{a}) + \gamma \sum_{s'} P[s'|s, \vec{a}]V_i(s') \quad (4)$$

$$V(s) = \max_{a \in A(s)} Q(s, \vec{a}) \quad (5)$$

However, the analogue of π^* where each player choses the action that optimizes reward with respect to others actions is insufficient as deterministic policies that satisfy these conditions need not exist.

III. Soccer Game

The soccer game environment defined originally by Littman [3] and later adopted by Greenwald [1] consists of two adversaries A and B. The field is a 2 X 4 grid with Player B given initial possession and a starting position of (0, 1) while Player A is defending at position (0, 2). These conditions define the starting state s. At the beginning of an episode, each player can choose to move North, South, East, West or stick and these actions are executed in random order. If both players try to move to the same location, only the first player moves. Additionally, if the player with the ball attempts to move to a location occupied by the other player, possession of the ball is transferred. Player B's goal, indicated by the column of B's in Figure 2, is at location at (0, 3) and (1, 3) and Player A's goal, marked by the column of A's, is at (0, 0) and (1, 0). When a goal is scored, the game ends and a +100 reward is given to the scoring player and a -100 reward is given to the other player. If an "own" goal is scored these rewards are flipped.

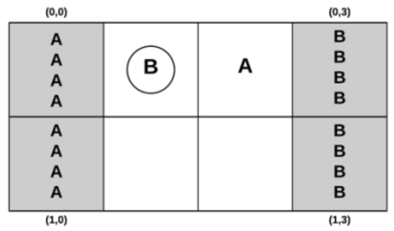


Figure 2: State s of Soccer Game Environment

Note that optimal policies in this game are not deterministic since any deterministic action selection strategy from Player B in state s can be blocked indefinitely by Player A. Moreover, the lack of a deterministic equilibrium policy makes the environment an interesting one to study. This paper uses Li Zeng's replication of this environment [5] to test the convergence properties of multiagent Q-learning, Friend-Q, Foe-Q and CE-Q.

IV. MultiAgent Q-Learning Implementation

Generalizing Q-learning to multiagent environments is intuitive. For each agent, $Q_i(s, \vec{a})$ and $V_i(s)$ are initialized arbitrarily. For each episode, agents select an action, observe a reward and a new state. The value of this resulting state is estimated using some objective function. These rewards and values are then used to update $Q_i(s, \vec{a})$. The learning rate, α, is slowly annealed and the process terminates after a designated number of iterations. Full pseudocode provided by Greenwald is shown in Algorithm 1.

```
Algorithm 1: Multiagent Q-learning

for t = 1 to T:
    1. Simulate actions $a_1, \ldots, a_n$
    2. Observe rewards $R_1, \ldots, R_n$
    3. for i = 1 to n:
        a. $V_i(s') = f_i(Q_1(s'), \ldots, Q_n(s'))$
        b. $Q_i(s, a) = (1-\alpha)Q_i(s, a) + \alpha[(1-\gamma)R_i +$
           $\gamma V_i(s')$
    4. Agents choose action $a_1', \ldots, a_n'$
    5. $s = s', a_1 = a_1', a_n = a_n'$
    6. Decay $\alpha$ according to decay schedule
```

A. *Q-learning*

The standard Q-learning algorithm was implemented according to Algorithm 1. Two Q-tables—one for Player A and one for Player B—with a dimension of (8, 8, 2, 5) were used. These dimensions correspond to the 8 available locations for Player A, the same 8 locations for Player B, a binary flag indicating if Player A has the ball, and the 5 available actions an agent can take. The value function in step 3(a) of Algorithm 1 is consistent with the single agent case and shown in (6). The Q-function is updated using this value according to 3(b).

$$V_i(s') = \max_a Q(s_{t+1}, a) \quad (6)$$

B. *Friend – Q*

In Friend-Q both agents assume other agents are friendly and will act in their best interest. Moreover, each player's strategy relies on the actions of others and thus, the Q-tables need an additional dimension. Each players Q-function is a table of size (8,8,2,5,5) with the first 3 states defined similarly as Q-learning and 4th and 5th dimensions corresponding to Player A's and Player B's action respectively. In addition, to account for the assumed collusion, the value function in 3(a) is updated according to (7).

$$V_i(s') = \max_{a_i \in A_1, a_2 \in A_2} Q_i[s, a_i, a_2] \tag{7}$$

C. *Foe – Q*

Foe-Q treats Player A and B as adversaries whereby each attempt to minimize the reward received by the other player. Von Neuman's minimax is used in place of the value function in 3(a). The minimax value update used for the Foe-Q algorithm is shown in (8) where $\sum_i(s)$ denotes the probabilistic action space of player *i* at state *s*.

$$V_1(s') = \max_{\sigma_1 \in \Sigma_1(s)} \min_{a_2 \in A_2(s)} Q_1(s, \sigma_1, a_2) = -V_2(s) \tag{8}$$

Because σ_1 is a probability distribution over actions, $Q_1(s, \sigma_1, a_2)$ is different from $Q_1(s, a_1, a_2)$ seen previously. The two are related however by (9).

$$Q(s, \sigma_1, a_2) = \sum_{a_1 \in A_1} \sigma_1(a_1) Q(s, a_1, a_2) \tag{9}$$

In other words, Player A assumes Player B will follow the probabilistic policy that minimizes Player A's expected reward. Since the Soccer Environment is a zero-sum game, (9) can also be interpreted as each player attempting to maximize their reward with respect to the other players actions and thus, the minimax operator is a generalization of Hu and Wellman's Nash-Q in zero-sum games [2].

Unlike Q-learning and Friend-Q, the Foe-Q value update rule requires finding an optimal probability distribution at each time step. To find this probability distribution a set of linear equations needs be solved. We solve these equations using a linear programming library called cvxopt which requires modeling the problem in conical form shown in (10).

$$\begin{array}{ll} \textit{Minimize}: & c^T x \\ \textit{subject to} & Gx \le h \\ \textit{and} & Ax = b \end{array} \tag{10}$$

To re-write (8) in linear programming form, we define $\sigma_{1,s'} = (\sigma_1(s', a_1), \dots, \sigma_1(s', a_5))$ as a probability vector representing the chance of selecting some action a_i in state s'. Additionally, we define the action-value function $Q_1(s', a_1, a_2)$ as a (8,8,2,5,5) table with the same interpretation as in the case of Friend-Q. Lastly, we'll define a value x_0 as the action taken by the second player that minimizes (9) and $\boldsymbol{x_0}$ as a (1 X 5) vector| with every value being x_0. Thus, $x_0 = \min_{a2} Q^T_{1,s'} \sigma_{1,s'}$ and $\boldsymbol{x_0} = (x_0, \dots, x_0)$.

From this formulation it follows that $\boldsymbol{x_0} - Q^T_{1,s'} \sigma_{1,s'} \le 0$, leading to our first constraint. We re-write this constraint to be consistent with the form shown in (10):

$$G_1 x \le 0, where\ G_1 = (\mathbf{1},\ -Q^T_{1,s'}) \text{ and } x = \begin{pmatrix} x_0 \\ \sigma_{1,s'} \end{pmatrix}$$

In the constraint above, **1** denotes a 1 X 5 vector of ones. Thus, $G_1$is a 5 X 6 vector, where the first element of the vector is a one and the remaining is the action-value function for Player 1. x is a 6 X 1 vector with the first element x_0 and the remaing elements being $\sigma_{1,s'}$. Moreover, $G_1 x$ is a set of 5 linear equations shown in (12).

$$y = x_0 - \sum_{a_1 \in A_1} \sigma_1(a_1) Q(s, a_1, a_2), a_2 = 1, \dots, 5 \tag{12}$$

Constraints 2 and 3 follow from the fact that $\sigma_{1,s'}$ is a probability distribution over actions and thus each action must be non-negative and sum to 1. Constraint 2 and 3 are presented below.

$$G_2 x \le 0, \quad where\ G_2 = \begin{pmatrix} 0 & 0 \\ 0 & -\boldsymbol{I} \end{pmatrix} and - \boldsymbol{I}\ is\ the\ 5\ X\ 5\ identity\ matrix$$

$$a^T x = 1, \quad where\ a = (0, 1, \dots, 1)^T$$

We vertically stack G_1 and G_2 and define $G = \begin{pmatrix} G_1 \\ G_2 \end{pmatrix}$. This formulation summarizes the minimization actions taken by Player 2. Additionally, Player 1 attempts to maximize his own reward, however the standard form used by cvxopt looks for minimum values. Moreover, the value $-x_0$ needs to be minimized. To do this we set $c = (-1, \dots, 0)$ resulting in Player 1 attempting to maximize x_0, subject to the adversarial actions of Player 2 trying to minimize this value.

D. *Correlated - Q*

The potential existence of multiple equilibria in Markov Games makes learning optimal policies challenging. Greenwald proposes that the use of different objective functions that identify a single equilibrium can solve this problem. He specifically introduces four CE-Q algorithms; utilitarian (*u*CE-Q), egalitarian (*e*CE-Q), republican (*r*CE-Q) and libertarian (*l*CE-Q). These policies help ensure the existence of a single equilibrium policy and thus the convergence of the CE-Q. Consistent with the charts shown in his paper, we implement *u*CE-Q which maximizes the sum of both players' rewards. The *u*CE-Q value update in 3(a) of Algorithm 1 is given by (13). That is, we are looking for the action-probability distribution of each player that maximizes the total reward expectation.

$$\sigma \in \mathrm{a}\max_{\sigma \in CE} \Sigma_{i\in I} \Sigma_{\vec{a}\in A} \sigma(\vec{a}) Q_i(s,\vec{a}) \qquad (13)$$

To cast this as a linear programming problem, we first note a rationality constraint for Player 1 shown below:

$$Q_{1,s}(i,:)\sigma^T(i,:) \geq Q_{1,s}(k,:)\sigma^T(i,:), \forall k = 1,\ldots,5$$

Thus, there exists some action *i* for Player 1 whose expected reward dominates all other actions given Player 2 acts rationally following the optimal stochastic policy. To encode this in conical form, we rewrite this constraint as $G_1\sigma^* \leq 0$, where σ^* is the row-wise vectorization of σ and G_1 as a block diagonal matrix where each block is equivalent to $G_{1,i} = \begin{pmatrix} \vdots \\ Q_{1,s}(k,:) - Q_{1,s}(i,:) \\ \vdots \end{pmatrix} \forall\ k \neq i$. Thus, G_1 is a (20 X 25) matrix and σ^* is a (25 X 1) probability vector.

Player 2 is bounded by the same rationality constraint but selects actions based on the columns of Q_2 and σ as shown:

$$Q_{2,s}(:,k)\sigma^T(,j) \geq Q_{1,s}(:,k)\sigma^T(:,j), \forall k = 1,\ldots,5$$

Similar to Player 1, we re-write this constraint as $G_2\sigma^* \leq 0$ and define G_2 as a block diagonal matrix with each block $G_{2,i} = \begin{pmatrix} \vdots \\ Q_{2,s}(:,k) - Q(:,i) \\ \vdots \end{pmatrix}, \forall\ k \neq i$. However, since σ^* is the (25 X 1) vectorization of the (5 X 5) joint probability matrix σ, the columns in σ corresponds to every 5 row elements in σ^*. For example, the first column c_1 of σ is related to σ^* by $c_1 = (\sigma^*(0), \sigma^*(5), \sigma^*(10), \sigma^*(15), \sigma^*(20))$. Moreover, 4 columns of zeros are required after each $G_{2,\mathrm{i}}$ to relate σ to σ^* and ensure each $G_{2,i}$ is multiplied with the correct probabilities. An illustration of G_2 is shown in Figure 3.

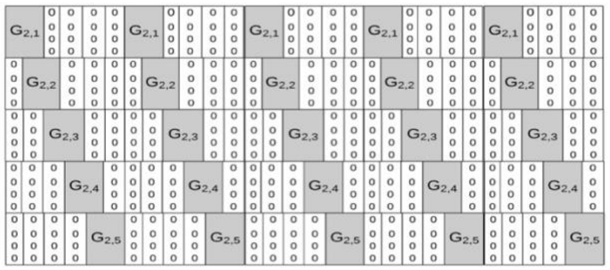


Figure 3: Matrix G_2

Finally, since σ^* is a joint probability vector, two constraints are needed to ensure probabilities are non-negative and sum to 1. Similar to Foe-Q, we ensure non-negativity of probabilities with the constraint $G_3\sigma^* \leq\ 0$ where G_3 is the $(25\ X\ 25)$ negative identity matrix. We combine constraints G_1, G_2 and G_3 by vertically stacking them letting $G = \begin{pmatrix} G_1 \\ G_2 \\ G_3 \end{pmatrix}$. To ensure the joint probability vector sums to one, we define our final constraint as $a^T\sigma^* = 1$, where a^T is a vector of 25 ones.

*u*CE-Q, given by (13), maximizes the sum of total reward, thus, we define our target c accordingly as the sum of each player's vectorized Q-function; thus $c = (Q_{1,s} + Q_{2,s})^T$. Finally, we can express the problem in linear programming standard form summarized below:

$$\begin{array}{ll} Minimize: & c^T\sigma^* \\ subject\ to & G\sigma^* \leq 0 \\ and & a^T\sigma^* = 1 \end{array}$$

V. Experiments & Results

Figure 3 in Greenwald's paper graphs the changes in Player A's Q-values at state *s* when Player A takes action S and Player B chooses to stick using traditional Q-learning, Friend-Q, Foe-Q and *u*CE-Q. Q-value changes are defined over consecutive iterations by $ERR_i^t = |Q_i^t(s,\vec{a}) - Q_i^{t-1}(s,\vec{a})|$. Greenwald leaves out specific details on the hyperparameters used in his experiments only mentioning that $\gamma = 0.9$ and ϵ and α were decayed to a minimum value of 0.001. Moreover, initial values of ϵ and α and their respective rates of decay weren't specified.

Several experiments were run to identify the hyperparameters that produce results most similar to Greenwald's. For each learning algorithm, ten initial values for alpha (1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1), ten initial values for epsilon (1.0, 0.975, 0.95, 0.925, 0.9, 0.875, 0.85, 0.825, 0.8, 0.775) and ten decay factors (0.99999, 0.999991, 0.999992, 0.999993, 0.999994, 0.999995, 0.999996, 0.999997, 0.999998 and 0.999999) were tested. All possible combinations of these hyperparameters were modeled and the set of parameters resulting in the charts most similar to Greenwald's were maintained.

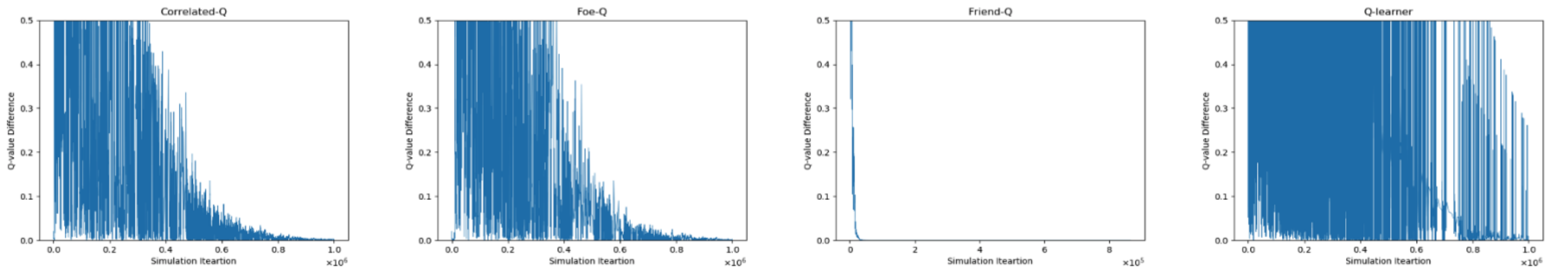
A. *Q-learning*

Figure 4(d) shows ERR_i^t for the Q-learning algorithm with $\alpha = 0.3, \epsilon = 0.8$ and their respective decay factors set to 0. 999995 and 0. 999994. The chart is somewhat similar to Greenwald 's with differences likely explained by inital values and decay speeds chosen for α; the algorithm proved to be very sensitive to these parameters. Regardless, the important conclusion in the lack of convergence is maintained. While ERR_i^t is decreasing, this can be explained by the decay of α as the size of the oscillations and α decline proportionally.

This lack of convergence demonstrates the instability of Q-learning in Markov Games. Since Q-learning doesn't consider the motives of others when determining an action, the Q-function is learnt from a noisy, dynamic environment and convergence can't be guaranteed. In other words, the Q-learner is attempting to find a deterministic policy, but since another agent is also acting in the environment, the world isn't stationary and such a policy doesn't exist.

B. *Friend – Q*

Figure 4(c) shows ERR_i^t for *Friend*-Q with $\alpha = 0.1, \epsilon = 1.0$ and both decay factors set to 0.999995. The chart is very



(a) uCE-Q *(b) Foe-Q* *(c) Friend-Q* *(d) Q-learner*

Figure 4: Reproducing Greenwald Graphs

similar to Greenwald's and demonstrates extremely quick convergence due to each agent learning a deterministic policy.

Since each agent in Friend-Q, assumes their opponent will cooperate, Player B erroneously assumes Player A will always take action E. Thus, Player B concludes if he/she also always takes action E, a goal will be scored in two moves. The first move in this sequence can be seen by Player B's Q-function in state *s* shown in Figure 5; the maximum value of 95.16, and the actions assumed by Friend-Q, occurs when both agents move East on their first turn.

Player A similarly makes irrational assumptions about Player B's motives and assumes that in state *s*, Player B will always move West, resulting in an immediate reward to Player A. Thus, Player A erroneously concludes his/her action is irrelevant. Moreover, while Friend-Q converges, both agents learn irrational policies.

Player	B					
	Action	E	W	N	S	Stick
A	E	95.16	-100.00	81.00	81.00	81.00
	W	87.82	-100.00	85.46	81.00	85.54
	N	90.00	-100.00	85.51	81.00	85.42
	S	90.00	-100.00	81.00	85.25	81.00
	Stick	90.00	-100.00	85.51	81.00	85.48

Figure 5: **Friend-Q action-value function in state s**

C. *Foe - Q*

Figure 4(b) shows ERR_i^t for Foe-Q with $\alpha = 0.2, \epsilon = 1.0$ and both decay factors set to 0.999995. The Figure closely resembles Greenwald's charts with both converging in around 800,000 iterations. Similar to Greenwald's experiment, the Foe-Q algorithm converges to a non-deterministic policy whereby each player randomly choses between sticking and moving south. The policy for Player A in state *s* shown in Figure 6. Note that Player A assigns no probability to actions E, W or N which makes sense as these moves intuitively don't help Player A play defense in state *s*. Moreover, we find that Foe-Q converges to a rational mixed-policy.

D. *Correlated - Q*

Figure 4(a) shows ERR_i^t for *u*CE-Q using the same parameters as Foe-Q. Similar to Greenwald's paper, the charts for Foe-Q and *u*CE-Q are nearly identical highlighting the fact that in zero-sum games, *u*CE-Q generalizes Foe-Q. We also find the *u*CE-Q algorithm converges to a similar non-deterministic policy as Foe-Q. As shown in Figure 6, both algorithms identify the joint actions taken in state *s* are approximately uniformly distributed over 4 possibilities (S, S), (Stick, Stick), (S, Stick) and (Stick, S). This confirms Greenwald's conclusion that in zero-sum games CE-Q learns the same policy as minimax.

Player	B					
	Action	E	W	N	S	Stick
A	E	0%	0%	0%	0%	0%
	W	0%	0%	0%	0%	0%
	N	0%	0%	0%	0%	0%
	S	0%	0%	0%	25%	22%
	Stick	0%	0%	0%	28%	25%

Action	Probability
E	0%
W	0%
N	0%
S	50%
Stick	50%

Figure 6: *u*CE-Q (left table) and Foe-Q (right table) policies

VI. CONCLUSION

Chris Watkins proved with repeated application of the Q-learning rule and adequate exploration of the environment, Q-learning convergences to the optimal action-value function with probability 1 [4]. The technique, however, doesn't scale to multiagent environments. Much recent research has studied algorithms that converge to equilibrium points in multiagent environments; Hu and Wellman demonstrate that Nash-Q converges under strict conditions and Greenwald's CE-Q generalizes Nash-Q but fails to address its weaknesses.

The difficulties in identifying converging algorithms in multiagent system (MAS) stem from the existence of multiple equilibria. In these environments, traditional Q-learning is non-ergodic attempting to converge to multiple equilibrium policies. Designing systems compatible with this multiagent framework is a balancing act between allowing agents to specify their own behaviors and creating a central planner to coordinate agent actions. The former can lead to miscoordination while the latter rarely creates rational polices. MAS designs relying on CE concepts can theoretically lead to agent coordination resulting in larger rewards. Moreover, Greenwald's paper provides an important foundation for future studies to identify more generalized, adaptive and decentralized procedures that can be applied to sophisticated environments. This paper confirms Greenwald's conclusions on the convergence properties of CE-Q and encourages future research into CE-Q as a solution in MAS design.

REFERENCES

[1] Greenwald, A., & Hall, K. (2003). Correlated-Q learning.

[2] Hu, J and Wellman, M. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm.

[3] Littman, M. L. (2001). Friend-or-Foe Q-learning in General-Sum Games.

[4] Watkins and Dayan (1992), Q-Learning.

[5] Zeng, L. (2019, July 14). ZengliX/SoccerGame. Retrieved July 18, 2020, from https://github.com/zengliX/SoccerGame