# Soccer Game with CE-Q

Shide Qiu
*dept. of Computer Science*
*Georgia Institute of Technology*
*sqiu74@gatech.edu*

***Abstract*— This project report replicated the result of the Soccer Game experiment found in Greenwald's paper [1]. Q-learning, Friend-Q, Foe-Q, and Correlated-Q (CE-Q) were studied and discussed in this report.**

## I. Introduction

CE-Q, a multiagent Q-learning algorithm based on the correlated equilibrium solution concepts, was introduced by Greenwald [1]. In this project, four agents including Q-learning, Friend-Q, For-Q, and CE-Q were trained to in the two-person zero-sum Soccer game environment. The Figure 3 of Greenwald's paper was reproduced and discussed.

## II. The Soccer Game

The Soccer game implemented in this report was recreated from section 5 of Greenwald's paper. The game is a zero-sum game for which there does not exist deterministic equilibrium policies. The game is played on a 4×2 grid as shown in Figure 1. The circle in the figure represents the ball. There are two players, A and B, occupy distinct squares of the grid and can choose one of the 5 valid actions on each turn: N, S, E, W, and stick. The players' actions are executed in random order once both players have selected their actions. The position will not be changed if a player attempts to move to a square that is not on the board. When a player executes an action that would take it to the square occupied by the other player, possession of the ball goes to the stationary player and the move does not take place. The objective of this game is to carry the ball into the appropriate goal (left for A, right for B) to scores +100. At the same time, the other player scores -100. However, if the player with the ball moves into the other player's goal, the player scores -100 while the other one scores +100. The game ends when there is a goal scored from either player.

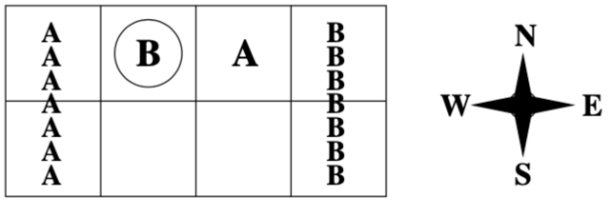

Fig. 1. Soccer Game. State $s$.

The state depicted in Figure 1 does not exist deterministic equilibria. Any deterministic policy for player B is subject to indefinite blocking by player A. We assumed that state $s$ is the initial state for every new game after the end of previous one. The square index is defined as a sequence of number from 0 to 7. The top-left square is indexed as 0; the top-right square is 3; the bottom-left square is 4; and the bottom-right square is 7. As the initial state, Player A is in square 2 and Player B is in square 1. The ball possession is defined by 0 for A while 1 for B. The state is defined as a list of Player A's position, Player B's position, and the ball possession. For example , the state $s$ can be represented as [2,1,1]. Five valid actions are represented as 0 for N, 1 for S, 2 for E, 3 for W, and 4 for Stick. If Player A moves to square 0 or 4, A scores +100 while B scores -100. On the other hand, if Player A steps into square 3 or 7, A scores -100 and B scores +100. The same rule applies to B.

**Algorithm 1: Soccer Game Environment**

```
Input: actions from Player A and B
Output: [Player A position, Player B position, ball possession], scores, termination
Init scores: [0, 0]
Init termination: False
Randomly select action order
First player moves
If collision then
    If the first player has the ball then
        Switch ball possession
Else
    Update first player position
    If the first player scores then
        Update scores, termination =True
        Return [Player A position, Player B position, ball possession], scores, termination
Second player moves
If collision then
    If the second player has the ball then
        Switch ball possession
Else
    Update second player position
    If the second player scores then
        Update scores, termination =True
        Return [Player A position, Player B position, ball possession], scores, termination
Return [Player A position, Player B position, ball possession], scores, termination
```

The Soccer environment was implemented as shown in Algorithm 1. The game was reset whenever a goal was scored by either player. The initial state is [2,1,1] for each new game.

## III. Markov Games

### A. Nash Equilibrium

The Nash equilibrium is a decision-making theorem within game theory that states each agent's choice $(\pi_1, \dots, \pi_n)$ is the best response to the other agents' choices. An agent's choice is defined as a vector of independent probability distributions over actions. Thus, no agent can gain by unilateral deviation [2]:

$R_i(\pi_1, \dots, \pi_n) \geq R_i(\pi_1, \dots, \pi_{i-1}, \pi_i{}', \pi_{i+1}, \dots, \pi_n)$

For all policies $\pi_i{}'$ and $1 \leq i \leq n$. A game can have more than one Nash equilibrium and the expected payoff to player $i$ can vary depending on the equilibrium considered.

### B. Correlate Equilibrium

Correlated equilibrium is also a probability distribution over actions, and it can be perceived as a generalization of Nash equilibrium. However, the main difference is that correlated equilibrium considers the joint space of actions rather than independent actions in which all agents optimize with respect to one another's probabilities, conditioned on their own. In other words, it permits dependencies among the agents' probability distributions, while maintaining the property that agents are optimizing.

Moreover, correlated equilibrium is a convex polytope which means it can be computed easily via linear programming. In this report, correlated equilibrium was formulated as a set of constraints and an objective function for a linear programming solver (ECOS) to solve.

### C. Markov Games

A Markov game is a generalization of the one-stage game to multiple stages [3,4]. A stochastic game is defined as a tuple $\langle I, S, \left(A_i(s)\right)_{s\in S, 1\leq i\leq n}, P, (R_i)_{1\leq i\leq n}\rangle$, where $I$ is a set of n players, $S$ is a set of states, $A_i(s)$ is the $i$th player's set of actions at state $s$, $P$ is the probability transition function that describes state transitions, conditioned on past states and joint actions, and $R_i(s, \vec{a})$ is the $i$th player's reward for state $s \in S$ and joint actions $\vec{a}_t \in A(s)$. Markov game is stochastic games for which the probability transitions satisfy the Markov property.

In Markov games, The Q value is defined over states and action-vectors $\vec{a} = (a_1, \dots, a_n)$, rather than state-action pairs in MDP:

$$Q_i(s, \vec{a}) \leftarrow R_i(s, \vec{a}) + \gamma \sum_{s'} P[s'|s, \vec{a}]\, V_i(s')$$

Here, we used the Q function from Littman's paper [4] without the normalization constant $1-\gamma$ described in Greenwald's paper [1]. It does not affect any result discussed in this report.

Several alternative definitions of the value function have been discussed in detailed in the next section.

## IV. Algorithm

### A. Q-Learning

Q-learning, an off-policy learning algorithm, is one of the early breakthroughs in reinforcement learning. It learns the action-value function Q and directly approximates the optimal action-value $q^*$. For Q-learning, it does not consider its opponent's action which means Player A and B have their own Q table and update independently. The update rule is defined as:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a)]$$

where $s_t$ is the current state; $s_{t+1}$ is the next state; $a_t$ is the player's action; $\alpha$ is the learning rate; $R_{t+1}$ is the reward of action; $\gamma$ is the discount factor.

### B. Friend-Q

Coordination equilibrium is a special kind of Nash equilibrium. In a coordination equilibrium ,all players achieve their highest possible values:

$$R_i(\pi_1, \dots, \pi_n) = \max_{a_1\in A_1, \dots, a_2 \in A_2} R_i(a_1, \dots, a_n)$$

Friend-Q works on the principle of coordination equilibrium which all players' reward functions are equivalent, and the idea is simply that $i$'s friends are assumed to work together to maximize $i$'s value. The main difference between Q-learning and Friend-Q is that the latter one's actions are formed in joint pair. In other words, it takes the opponent's action into account. Therefore, only one Q table is needed for Friend-Q. The update rule is defined as:

$$Q_i[s, a_1, \dots, a_n] \coloneqq (1-\alpha_t)Q_i[s, a_1, \dots, a_n] + \alpha_t(r_i + \gamma Nash_i(s', Q_1, \dots, Q_n)$$

$$Nash_1(s, Q_1, Q_2) = \max_{a_1\in A_1, a_2\in A_2} Q_1[s, a_1, a_2]$$

It shows that the optimal value is from the joint action that maximize the Q value since we consider the two players are friends.

### C. Foe Q

Adversarial equilibrium is another special case of Nash equilibria. It has the property that no player $i$ is hurt by any change of the other players:

$$R_i(\pi_1, \dots, \pi_n) \leq R_i(\pi_1', \dots, \pi_{i-1}{}', \pi_i{}', \pi_{i+1}{}', \dots, \pi_n{}')$$

Foe-Q is to find the adversarial equilibrium which $i$'s foe is working together to minimize $i$'s value. Just like Friend-Q, Foe-Q also considers opponent's action and only need one Q table. However, the update rule is different since the opponent is a foe not friend here:

$$Q_i[s, a_1, \dots, a_n] \coloneqq (1-\alpha_t)Q_i[s, a_1, \dots, a_n] + \alpha_t(r_i + \gamma Nash_i(s', Q_1, \dots, Q_n)$$

$$Nash_1(s, Q_1, Q_2) = \max_{\pi\in\prod A_1} \min_{a_2\in A_2} \sum_{a_1\in A_1} \pi(a_1) Q_1[s, a_1, a_2]$$

Foe-Q is based on the minimax-Q algorithm provided by Littman [4]. The objective is to maximize the worst case over all actions that the opponent can execute which means we consider the opponent is fully adversarial. The Foe-Q can be solved with linear programming.

### D. CE-Q

CE-Q, based on correlated equilibrium solution concept, generalizes both Friend-Q and Foe-Q. The set of correlated equilibria contains the set of Nash equilibria in general-sum games. In constant-sum games, where Nash and minimax equilibria coincide, the set of correlated equilibria contains the set of minimax equilibria [1]. The main difficulty in learning equilibrium policies in Markov games is the equilibrium selection problem due to the existence of multiple equilibria. Greenwald introduced four variants of CE-Q, based on four different equilibrium selection functions, to solve this issue. The four CE-Qs are described below:

1. Utilitarian CE-Q (uCE-Q): Maximize the sum of the players' rewards

$$\sigma \in arg \max_{\sigma \in CE} \sum_{i \in I} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

2. Egalitarian CE-Q (eCE-Q): Maximize the minimum of the players' rewards

$$\sigma \in arg \max_{\sigma \in CE} \min_{i \in I} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

3. Republican CE-Q (rCE-Q): Maximize the maximum of the players' rewards

$$\sigma \in arg \max_{\sigma \in CE} \max_{i \in I} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

4. Libertarian CE-Q (lCE-Q)Maximize the maximum of each individual player I's rewards: let $\sigma = \prod_i \sigma^i$, where

$$\sigma^i \in arg \max_{\sigma \in CE} \sum_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})$$

The value function is

$$V_i(s) \in CE_i(Q_1(s), \dots, Q_n(s))$$

$$CE_i\left(\vec{Q}(s)\right) = \{\sum\nolimits_{\vec{a} \in A} \sigma(\vec{a}) Q_i(s, \vec{a})\}$$

where $\sigma$ satisfies the four CE-Q defined above.

All of the four CE-Q can be solved via linear programming with the probability and rationality constraints. The rationality constraints means that the expected rewards to play the recommended policy is no worse than playing any other policy.

## V. Experiment and Results

In this section, the Figure 3 from Greenwald's paper was replicated, and the four Q-learning algorithm were discussed in detail.

As mentioned in Section II, the Soccer game environment was implemented as shown in Algorithm 1. The game will terminate and return scores of [0,0] if no one scores a goal over 1000 steps. Every time after the end of a game, the new game will be reset to the initial state which is represented as [2,1,1] (Player A's position, Player B's position, ball possession). The environment keeps the state information and takes in new actions selected from Player A and B. The sequence of the action execution is randomly selected. The first mover executes its action and the environment checks if this action will cause collision and ball possession switch. If there is no collision, it updates the player's position and check the termination status (the player scores for himself or the opponent). If not terminated, do the same process for the second mover. At the end, the state information (Player's position, Player B's position, ball possession), scores, and termination status will be returned.

The four different Q-learning algorithms were implemented in the Soccer game environment and the convergence of a state-action pair were discussed in the following sections. The error (Q-value difference in Fig. 2-5) at time $t$ for agent $i$ is the difference between $Q(s, \vec{a})$ at time $t$ and $Q(s, \vec{a})$ at time $t-1$:

$$ERR_i^t = |Q_i^t(s, \vec{a}) - Q_i^{t-1}(s, \vec{a})|$$

Here, the error values shown in Fig. 2-5 reflect Player A's Q-values corresponding to state $s$ (Fig. 1), with Player A taking action S and Player B sticking as described in Greenwald's paper.

### A. *Q-Learning*

Figure 2 shows the Q-value difference corresponding to state $s$ and action S of Player A for Q-learning algorithm over 1 million training steps. Q-learner computes Q-values for each of their own possible actions, ignoring their opponents' actions. The Q table was constructed for Player A and B independently with dimension of (8,8,2,5) representing 8 possible positions for Player A, 8 possible positions for Player B, 2 ball possessions, and 5 possible valid actions for the player. In Greenwald's paper, the Q-learner used $\varepsilon$-greedy algorithm to select actions with decaying of $\varepsilon$ to 0.001. The discount factor was 0.9 and the learning rate $\alpha$ was decayed to 0.001. However, the initial $\alpha$, initial $\varepsilon$, and decay rate were not provided in the paper. Thus, the tuning of hyperparameters were completed and the hyperparameters used in this project were shown in Table 1. $\alpha$ decay and $\varepsilon$ decay were defined as:

$$\alpha = \max\left(\alpha * \alpha_{decay}, \alpha_{final}\right)$$

$$\varepsilon = \max\left(\varepsilon * \varepsilon_{decay}, \varepsilon_{final}\right)$$

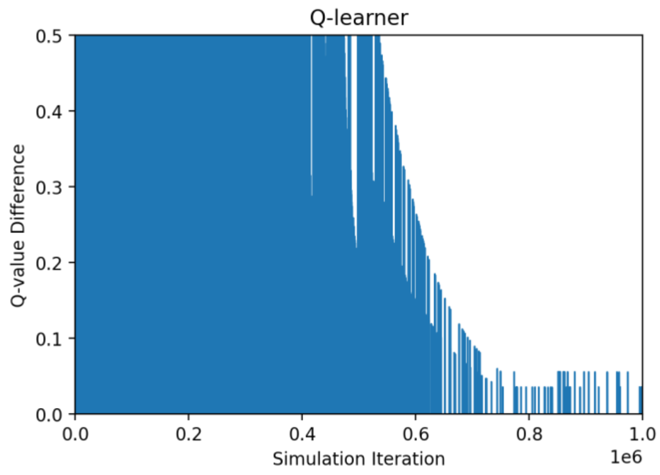

Fig. 2. Convergence in the soccer game. Q-learning

With the parameters shown in Table 1, Fig. 2 almost replicated the Q-learning figure in Greenwald's paper. It showed a decreasing Q-value difference over iterations. However, it does not converge due to the way we updated the Q table. We only considered Player A's action without Player's B's action. Without the consideration of the opponent's action, it learned from a noisy dynamic environment and the convergence cannot be guaranteed. For example, with the same action for Player A, Player B could have 5 different actions and the action execution sequence is random. As a result, there could be a huge difference in Q-value. In other words, Q-learner looks for deterministic optimal policy while Soccer game does not exist such policy. The decreasing is solely due to the learning rate $\alpha \rightarrow 0.001$.

It is impossible to match the figure in Greenwald's paper perfectly. In Greenwald's paper, the oscillations in Q-value difference over 0.6 million iterations were higher than which in Figure 2. The difference is mainly due to the different hyperparameters which are unknown from the paper. Also, the randomness of the actions executed in the games could be another possible factor causing such difference.

**Table 1. Hyperparameters for Q-learning**

| |
|---|
| Discount factor: $\gamma = 0.9$ |
| learning rate $\alpha_{initial} = 1.0$ |
| final learning rate $\alpha_{final} = 0.001$ |
| $\alpha$ decay rate = 0.99999 |
| Initial $\varepsilon$: $\varepsilon_{initial} = 1.0$ |
| Final $\varepsilon$: $\varepsilon_{final} = 0.01$ |
| $\varepsilon$ decay rate: 0.99999 |
| Max iterations: 1000000 |
| Max steps: 1000 |

### B. *Friend-Q*

Figure 3 shows the Q-value difference corresponding to state $s$ with action S of Player A and action sticking of Player B for Friend-Q algorithm over 1 million training steps. Friend-Q considers the opponent as a friend and will maximize its reward. Only one Q table was constructed with dimension of (8,8,2,5,5) representing 8 possible positions for Player A, 8 possible positions for Player B, 2 ball possessions, 5 possible valid actions for Player A, and 5 possible valid actions for Player B. The only hyperparameters provided in Greenwald's paper was $\gamma = 0.9$ and $\alpha \rightarrow 0.001$. Table 2 shows the hyperparameters used in the project. The Friend-Q learner selected actions for Player A and B randomly. The $\alpha$ decay policy was same with Q-learning algorithm.

**Table 2. Hyperparameters for Friend-Q**

| |
|---|
| Discount factor: $\gamma = 0.9$ |
| learning rate $\alpha_{initial} = 0.1$ |
| final learning rate $\alpha_{final} = 0.001$ |
| $\alpha$ decay rate = 0.99999 |
| Max iterations: 1000000 |
| Max steps: 1000 |

Figure 3 shows a very fast convergence of the Q-value difference which is similar with the figure in Greenwald's paper. At state $s$ with Friend-Q, each player anticipates the other player will score for itself. Thus, Player A will pass the ball to Player B to let Player B score for Player A while Player B will pass the ball to Player A. Therefore, Friend-Q converges but its policies are irrational.

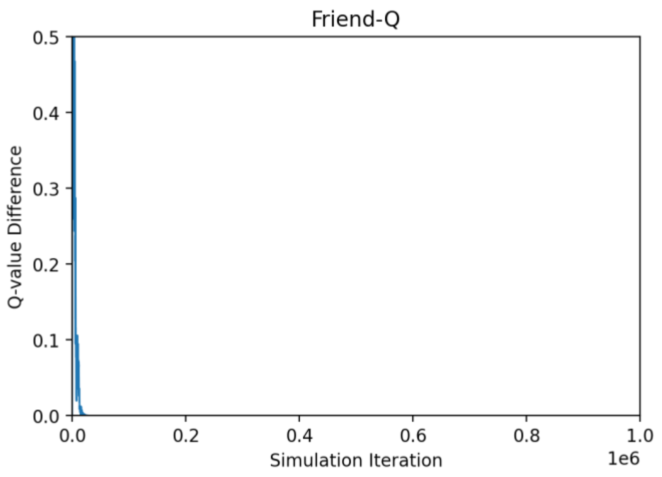

Fig. 3. Convergence in the soccer game. Friend-Q

The main difference with Greenwald's graph is that her Q-value difference converged around 50k iterations while Fig. 3 converged around 20k iterations. It could be the different hyperparameters and the way $\alpha$ decayed. The different exponential $\alpha$ decay rate were studied but showed no significant difference. Therefore, other decay method like linear decay might be helpful to replicate the graph.

### C. *Foe-Q*

Figure 4 shows the Q-value difference corresponding to state $s$ with action S of Player A and action sticking of Player B for Foe-Q algorithm over 1 million training steps. Foe-Q considers the opponent as a foe and will minimize its reward. Only one Q table was needed, and it was constructed the same as Friend-Q's Q table. No hyperparameters were provided in Greenwald's paper. Thus, the same hyperparameters and $\alpha$ decay policy of Friend-Q were used in Foe-Q algorithm (except $\alpha_{initial} = 1.0$ in Foe-Q). The actions of Player A and B were selected randomly. The minimax algorithm was solved by linear programming (ECOS solver).

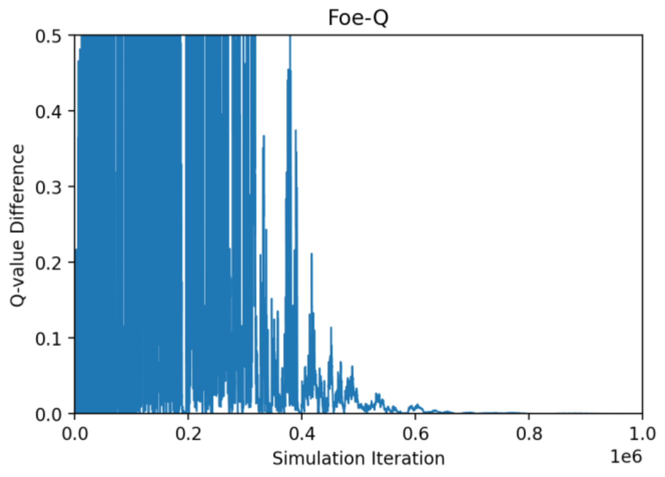

Fig. 4. Convergence in the soccer game. Foe-Q

The Figure 4 replicates the original graph in Greenwald's paper perfectly. Both converged around 700k iterations. Foe-Q converged to nondeterministic policies for both players where each one randomizes between sticking and heading south. The variation of Q-value difference in original graph was slightly lower than which in Fig. 4. This could be the slightly different hyperparameter selections. Also, different linear programming solver might cause a slightly different probability distribution.

## D. *CE-Q*

In the Soccer game (zero-sum finite-space game), all variants of CE-Q yield identical outputs because all equilibria at all states have equivalent values. The uCE-Q was used in this project according to Greenwald's paper.

Figure 5 shows the Q-value difference corresponding to state $s$ with action S of Player A and action sticking of Player B for CE-Q algorithm over 1 million training steps. The same hyperparameters and $\alpha$ decay policy of Foe-Q were used in CE-Q algorithm. The actions of Player A and B were selected randomly. The probability distribution over 25 different joint actions were calculated by linear programming (ECOS solver).

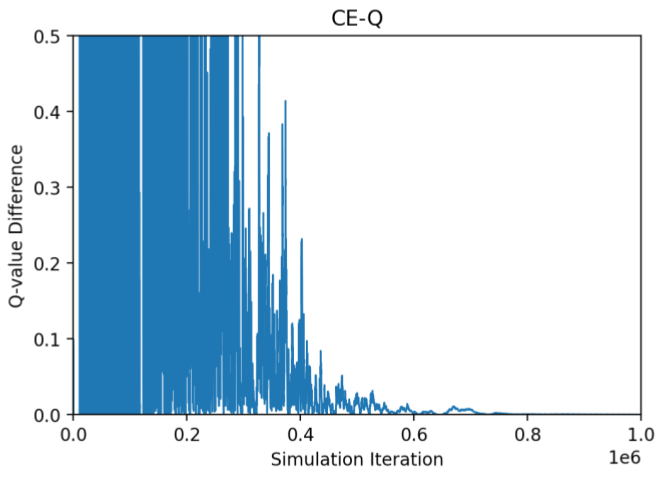

Fig. 5. Convergence in the soccer game. CE-Q

CE-Q learned Q-values that coincide exactly with those of Foe-Q. In other words, CE-Q learned minimax equilibrium policies in the Soccer game. Therefore, Figure 5 is identical with Figure 4 because the two algorithms learned the same Q-values (the tiny difference is due to the randomness of the game). Moreover, the reason for difference between Fig 5 and the original graph is exactly the same with the reason for Foe-Q learning discussed above.

## VI. Conclusions

In this project, the Soccer game environment was implemented in detail. Nash equilibrium and correlated equilibrium were introduced. Four different learning algorithms were covered in detail and were used in Soccer game to replicate the graph in Greenwald's paper. The difference between the replicated figures with original graphs were discussed. Overall, the solid understandings of Q-learning, Friend-Q, Foe-Q, CE-Q, and linear programming were gained by the implementation of these algorithms.

## A. *Pitfalls*

One of the most difficult parts in this project is the unclear parameters. Since Greenwald only provided very limited hyperparameters without any details, it is almost impossible to replicate the same result.

The computation power is also a limit for linear programming. Different linear programming solver could affect the training efficiency. In this project, the default linear programming solver ECOS was used. At the beginning, it took me 4 hours to train the Foe-Q algorithm and 10 hours to train the CE-Q algorithm (the code is CE-Q_old.ipynb in the jupyter notebook folder ). With the help from Piazza, I tried to vectorize my constraints for linear programming and it reduced the training time of CE-Q algorithm to 2.75 hours.

Another pitfall is that the linear programming solver sometimes returns super small negative probability distribution value due to precision errors. Therefore, a normalization function was implemented to reset all negative values to zero and then normalize (sum to one).

The Soccer game environment is complicated and requires long time for me to understand if my linear programming algorithm works or not. A good solution is to use the same algorithm on the easier environment like the "Chicken" game in Greenwald's paper which helps me debug my code easier. Apply the algorithm to our project environment after confirming it works with the easier environment.

## B. *Future Work*

Overall, even if the project is successfully solved in this report, there are still some interesting studies could work on in the future. The vectorization of Foe-Q could be done to improve its efficiency. Different learning programming solvers could be studied and discussed.

## References

[1] Greenwald, Amy, Keith Hall, and Roberto Serrano. "Correlated Q-learning." *ICML*. Vol. 3. 2003.

[2] Hu, Junling, and Michael P. Wellman. "Multiagent reinforcement learning: theoretical framework and an algorithm." *ICML*. Vol. 98. 1998.

[3] Littman, Michael L. "Friend-or-foe Q-learning in general-sum games." *ICML*. Vol. 1. 2001.

[4] Littman, Michael L. "Markov games as a framework for multi-agent reinforcement learning." *Machine learning proceedings 1994*. Morgan Kaufmann, 1994. 157-163.