Exploring Reinforcement Learning to Small Scale Combat in the Real-Time Strategy Game: StarCraft: Brood War

Ang Li[1], Xiang Sun[2], and Yunpeng Wu[3]

[*]Department of Electrical and Computer Engineer
[*]Colorado State University, Fort Collins, USA
[1]angli@rams.colostate.edu
[2]flysx1937@gmail.com
[3]yunpeng@rams.colostate.edu

***Abstract*—We are trying to imitate paper "Applying the Reinforcement Learning to Small Scale Combat in the Real-Time Strategy Game: Star Craft: Brood War" in this project. The BWAPI is used in this project to get connection between our program and StarCraft instance. One-Step Q-learning is used in this project to train agent unit in StarCraft. Java and BWAPI java are used to realize the learning process. The RL agent controlled unit is trained to fight with 6 enemy units in our test case, and we compare the reward gained for our result and paper's result. For the last part, we can compare the game won rate of One-Step Q-learning and random action pick up. This project/experiment can demonstrate how refinement learning can deal with the complex piratical real-time problems.**

***Index Terms*—One-Step Q-learning, Real-time strategy, Complex Environment, Unsupervised Learning**

I. Introduction: Why Choose StarCraft

A. Problem Statement

Basically, we need to apply the machine learning technique into the specific environment in Starcraft. According to the paper we choose, we decide to apply reinforcement learning algorithm to deal with a complex AI. In our project, we set up a small scale combat that consist of one RL agent unit and six enemy units. Our goal is make the RL agent controlled unit to learn the best strategy to attack the six enemy units completely. In this process, we need to develop a good algorithm, distinguish the several significant features and improve the efficiency of whole program.

B. Motivation for the Work

For the most primary motivation of this project, is our passion about StarCraft. We have been played this game for couple years, and we did many research into this game. To discover the relationship between each units, the trade-off in attack mode and developing mode. Besides, by our investigation and discussion, we always believe this game have the most complicate balance system with hundreds of feature and variables. Some other game like league of legends only have the small combat part of the Starcraft, and some other game like SimCity only have the construction part of the Starcraft. The combat for some games will not be efficient enough for us to compare the effect of the Q-learning.So, this game will provide more reasonable platform for the machine learning. On the other side, as the video game widespread[1], the AI of the game is taking more significant role in game playing. In a long term, the game developer always pursuit to develop AI to implement into the intricacy game.

C. Review of Previous Work

For the previous work, Berkeley have hold the Starcraft AI competition for couple years. In the competition, each side would set up the AI to finish the work from the mineral income, unit producing, all the combat until final winning. From the paper we chosen, they completed a small scale combat which is a RL agent controlled unit fight with 6 enemy units. The enemy units has less weapon range than the RL agent controlled unit. The enemy units has less speed than RL agent controlled unit. The RL agent controlled unit has a stronger power of weapon than the enemy units which means the RL agent controlled unit can win in a 1 vs 1 combat, the enemy units need to surround the RL agent unit and make more attack times to win the game. So, the RL agent controlled unit can finish a work as hit and run. Firstly, the RL agent controlled unit cannot beat all the enemy units, and always be beaten by the enemy units. After the hundreds of learning process, the RL agent controlled unit will formulate a intelligent strategy to beat

all the enemy units successfully and keep itself survival. And then do the comparison between several algorithm to know which algorithm can have a better performance. As we known, for the AI development work, the developer used to use the hard strategy like attack the most nearby enemy, It can only deal with the most easy problem in some conditions. For some single player game, this AI maybe enough for the game to play, it make the life of the game shorter. In this game, they author proposed several method to applied into the game, such as one-step Q learning, Watkins-Q, One step Sarsa and Sarsa. The reinforcement learning also do great work on the AI development. It will let the machine learning work more practical.[1][2]

D. Open Questions in the Domain

In the domain, different algorithm and learning method have different effect on the learning result, we need to develop the better algorithm to have a better performance. The specific algorithm may not suit for all the problem we met in the real combat. Such as, the unit on the land combat will highly influence by the landscape. But for the air force combat, the weather may be the most important feature, the reward may not the same. So, we need to have large amount algorithm to deal with all the works, it can make the AI smarter. With the development of the AI for computer games, the developer always use a specific code to let the units controlled by the computer to realize some easy AI like attack automatically, move automatically or patrol. But it cannot change by the time process. Some players often feel boring with the easy AI. The AI based on the very original game need to be better, and can learn the new strategy by themselves.

E. Proposing to Address Them

Firstly, we choose the right method to deal with the problem, we learn from several papers to know more about the Monte Carlo method, 1 step Q-learning combine with the dynamic programming. For this method we can realize the reinforcement learning. We decide to deal with this problem from the very small combat, we can let the AI more smarter from a very small combat. In this battle, we can easily find the learning process of the AI. For the future work, we can make the AI from the very beginning such as the mineral income, we can make the worker to work for every single mineral. The system will decide the worker to do the most right thing. And we will combine the different combat, the different units on land or air force can fight with each other, and learned automatically.[3] At last, the method can deal with almost all the real time strategy game. And then we can apply the strategy to all the games we have.

II. Method of Learning

We are trying to use reinforcement learning to our agent unit, which is kind of popular in training AI in complex environment. The paper we are reading introduced three different algorithms such as one-step Q-learning, One-Step Sarsa and Eligibility Traces but we found that those algorithms are similar to each other so we choose to perform only One-Step Q-learning into our environment since it took a lot of time to build out the starcraft training environment.

A. One-Step Q-learning

One-Step Q-learning algorithm is a refinement learning algorithm which composed of Monte Carlo method and Dynamic Programming. This method can update the rewards matrix at each step and look one step forward to get right strategies. It will finally have a policy matrix to determine which state transaction is the best path to maximum reward at the end of the training. As the

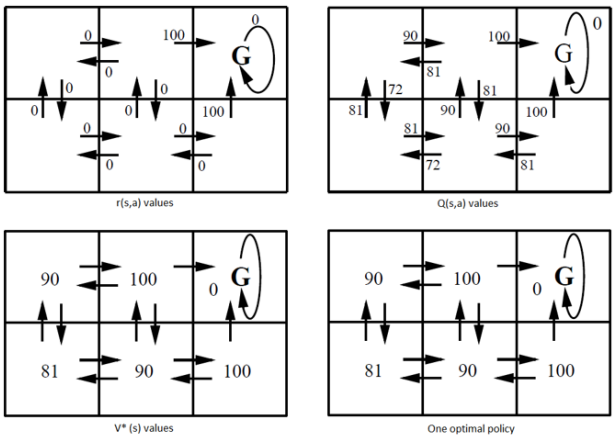

Fig. 1. QleaningExample

picture showed above, the initial reward matrix only has reward points at the final goal state (actually, the goal were defined as the place that can earn a lot of rewards), all other middle states only have 0 points. The program will start from a random state and randomly choose possible path to next state and calculate the potential reward. The potential reward will choose the maximum Q for next state and update the reward points of current state in reward matrix. After enough time of iteration times, the agent can find the best state transaction path to get maximum rewards, showed as upper left of the 1

When compare with the previous example, the Q-learning in dynamic environment first have no matrix to look up, so if any non-existing state come out, the max reward function will return 0 in this case. At the beginning, the reward matrix is an empty set and the size will grow along with the training going. If the

one step looking forward cannot find state in Reward Matrix, it will use 0 as max Q for future potential.

```
    double maxReward(state s) {
      actions actionsFromState;
      if(action_list.containsKey(s)){
        actionsFromState = action_list.get(s);
      }
      else{
        return 0;
      }
        double maxValue = Double.MIN_VALUE;
        for (int i = 0; i < actionsFromState.
    size(); i++) {
            double value = actionsFromState.q.
    get(i);

            if (value > maxValue)
                maxValue = value;
        }
        return maxValue;
    }
```

B. Action Set Establish

The possible action at each state were composed by Fight: If the weapon cooldown is 0, adding all enemy units that in weapon range to attack list.
Retreat : The length and height of the agent unit can be simplified to square, which has 25*25 units dimensions. So the retreat action will add/minus x direction and y direction 4 times of unit dimensions, respectively. So total 8*8 = 64 move actions will be added to action sets

C. Features Choosing

In our implementation, the definition of the state is important since it determines what kind of features were considered in the learning process. At the first we defined the reward function as: We have a lot of candidate features when we first play with the training since StarCraft: Brood War is a complex game and it has many features even in a small scale combat. I.e Weapon cool down of each unit, healthy point's, distance from closest enemy, distance to all enemies and even position of the agent unit. However, we first throw all features into the training process and we just got some random moving since the agent can never find a same state to follow with, too many features cause some over fitting in the training process, the total training process is totally random and no training efforts at all.
After many attempts of the feature choosing, we determins features as following:

```
public boolean equals(Object obj){
    if (obj == null) {
        return false;
    }
    if (!state.class.isAssignableFrom(obj.
    getClass())) {
        return false;
    }
    final state other = (state) obj;
  if(this.cooldown==other.cooldown&&this.
    eneCount==other.eneCount)
    if(abs(this.minihealth-other.minihealth)/
    this.minihealth<=0.25)
      if(abs(this.minidistance-other.
    minidistance)/this.minidistance<=0.25){
        System.out.println("Hit!");
        return true;
      }
  System.out.println("false");
  return false;
}
```

The features we chooses from the environment has been showed above, the weapon cool down time for agent unit, distance to closest enemy, healthy of the closest enemy and number of enemies. For some numbers that are hardly exact match such like closest enemy healthy point and distance of the closest enemy, we use percentage to compare them. It's works like if the differential of two different state is smaller than 10 percents, the leaning process will treat them as same state.
The Q updated rule has been defined (We also play with the alpha and gamma in our training to get max performance, our final alpha =0.1 and gamma =0.9):[1]

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{1}$$

And the One-Step looking forward has been implemented as following:

```
    public int update(){
        state s_next = new state(game);
        reward  = -(s_next.eneHitPoints -
    eneHit - (selfHit - s_next.selfHitPoints));
      double previous_r = Actions.q.get(index);
        double maxQ = maxReward(s_next);
        Actions.q.set(index,reward+alpha * (
    previous_r + gamma * maxQ - reward));
        action_list.put(s, Actions);
        return reward;
    }
```

As the code showing, the reward function will be calculated by:

$$reward_{t+1} = \sum_{i=1}^{m} enemy\_unit\_health_{i_t} - enemy\_unit\_health_{i_{t+1}} - (agent\_unit\_health_t - agent\_unit\_health_{t+1})$$

Fig. 2. Reward Function

D. Environment Set Up

For the training environment set up, the starcraft game test case, one agent unit against six enemy units are attached as follows:

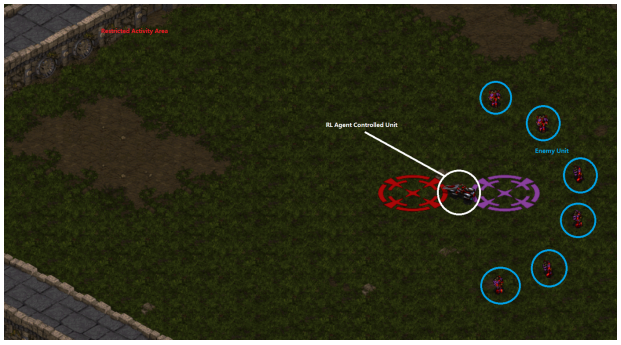

Fig. 3. Game Environment

Not exactly, but similar to the paper's test environment, which enable use to compare our training performance and paper's performance. And we used same sychroized method as paper described which were realized by check status of the agent unit to make sure the reward is being calculated by each action instead of each frame:

```
if(game.enemy().getUnits().size()<=1||self.getUnits().size()!=2){
    if(flag.action_list.Move_List.size()!=0&&y<=100){
        if(flag.index<flag.action_list.Attack_List.size()){
            if(flag.action_list.Attack_List.get(flag.index).getHitPoints()!=flag.previous);
            else return;
        }
        else if(obj.s.V.getPosition().equals
            (flag.action_list.Move_List.get(flag.index-flag.action_list.Attack_List.size())));
        else return;
    }
```

The total process flow chart looks like follows (The same with the paper we are trying to reproduced):

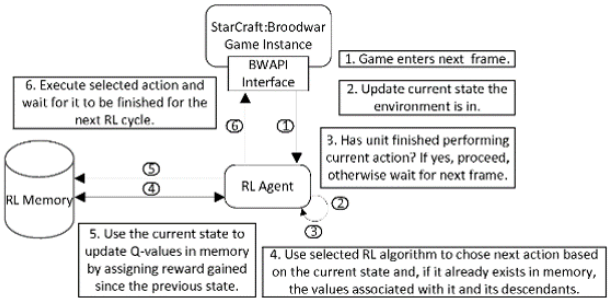

Fig. 4. FlowChart

III. Result and Analysis

Demo videos also attached in for this part.

A. Comparison With Original Paper

In our setting environment, there are 6 enemy units, and the health score for each unit is 50. And for the RL agent controlled units, the health score is 60 and the total health score of the enemy is 300. So, based on the rewards function, we can calculate the final accumulated rewards should between 240 to 300. The output graph

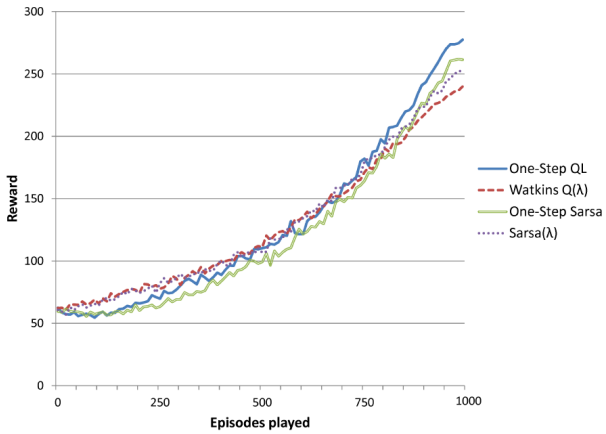

Fig. 5. Result From Paper

from the paper «Applying the Reinforcement Learning to Small Scale Combat in the Real-Time Strategy Game: StarCraft: Brood War», we only focus on the blue line and we can found that as increasing of episode played, the rewards are also increased correspondingly. From the graph, we can find that the initial reward is approximately 60, which is the health points of our agent unit. And after nearly 1000 times of learning, the final rewards value is approaching to 300. At the end of learning, as descripted in paper, the agent unit are able to win the game at most of the times, which we also achieved at the end, although there's a slightly different. Our results graph shown the rewards have big fluctuations as increasing of episode played and the final rewards approaching to zero. However, our RL agent controlled units beat all other 6 enemy units at end.

For our result, if the enemies have reached out of the agent unit's view, the program are not able to get health points from those enemies since the StarCraft API wants AI to has the same information as the human player. So we cannot get a perfect graph like above showed, our results shows that the cumulative rewards will increase with more episodes played but not continue increase since the reward is not really describe the healthy of the enemies. However, this bug even made positive effect to the final learning result since the more distance kept from agent unit and enemy, the more attack can made by agent unit. The reason is because the agent unit has bigger weapon range than enemy units. Therefore, our agent unit have better performance than the paper stated,

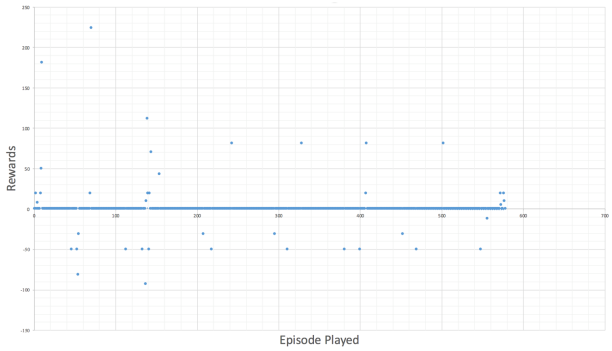

Fig. 6. Our Q-learning Result

i.e 600 episodes to get all enemies eliminated in most of the time.

B. Compare With the Random Action Pick up

To get better insight of One-Step Q-Learning, we also compare our agent's performance with the random action set pick up on game winning rate:

It's easy to observe that our agent AI definitively have

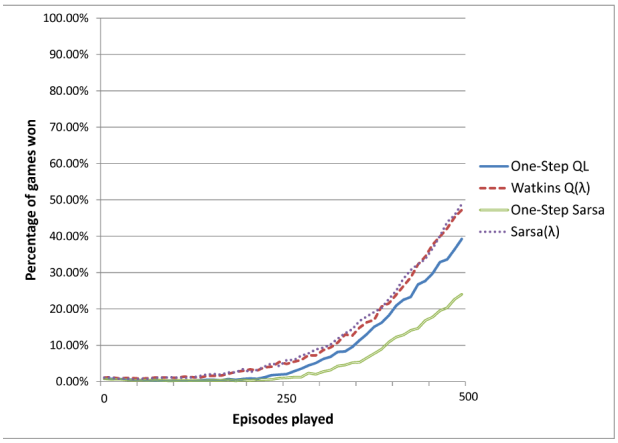

Fig. 7. Paper:Compare with Won Rate

better performance than random action set pick up and it is even better than the paper stated, i.e only 500 episodes to reach 70 percent won rate. (Videos attached)

IV. Conclusion

By this experiment, the one-step Q-learning is an unsupervised reinforcement learning can be that apply into the complex system such as the real-time strategy game. Without the applying hard code, the controlled unit can synthesize the different trick and tradeoff to develop the best strategy. And by compare with the random action pick up, the one-step Q-learning can help to optimize the performance controlled unit. And the reason we are using only one-step Q-learning is because

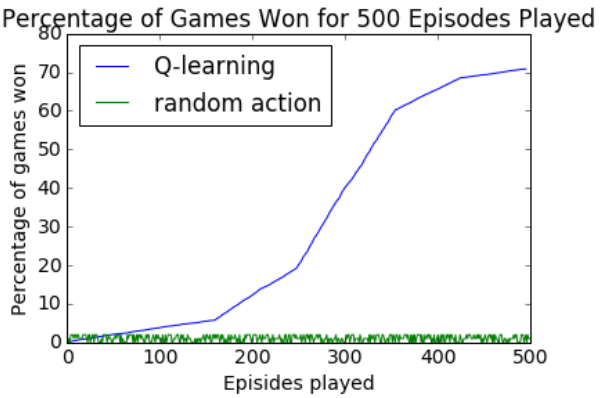

Fig. 8. Compare with Won Rate

the real-time strategy game needs AI to response in a hard-limited time which means deep learning cannot be used widely in real-time environment. Therefore, we discovered that reinforcement learning is useful in partial complex environment.[2]

V. Future Work

In this project, the learning and AI part only depends on the small combat, in the future, we can do the AI and learning part extend to the whole combat, it can including the mineral income, the unit producing and all the combat the game included. From the small combat to the big combat. In this game, we also have the air force, the air force will ignore the landscape. Obviously, the air force would be easier than the army. And with the army on land, we need to decide which landscape the army on land can move to. In the large combat, we can try to make the system learned to use different units to take the hit one by one, it will be a better strategy than take all the hit by one unit, since, the unit will be vanished, and our damage per second will decrease. Also, we can let the system learned to know some units with more health or heavier armor can take more damage. It will make the AI smarter and smarter. As the whole game, the construction part is also important, with the control of the feature, we can let the system to run the worker to have the most efficient way to get the resource, and build the building in a scientific way. With the building construction, the AI can take advantage with the position it captured. It could be a completely AI, and cannot be defeated easily, and could be stronger and stronger.[1][3] It could be a better AI and take the market and improve more games.

With the learning part, we can add more features with the small combat and use different features in the different features in the different circumstance. Also, we can find some better algorithm to have the better performance. We can concrete different features and

algorithms on different combat or production. It can be tested in the AI competition.

We can also apply the concept for some other games. We can use different state to make the performance better. This method in the real time strategy game is really pervasive, We will create multiple layers to control all the processors of the game. We can use this complete AI to participate in the Starcraft AI competition held by Berkeley. In the future, with using this method provided in this project, we can do some more work with the development of the AI

VI. Work Distribution

Xiang looks through the paper and finds out how the algorithm works, and writes state.java action.java ,determines how the state sets and action sets looks like.

Yunpeng setup the environment of the Starcraft and doing the synchronize between the java and the Starcraft.

Ang writes Qlearning code and combined project and qlearning togerther.

All: play with training algorithm and make brainstorm with which feature should be taken in and which feature should be taken out, and make out the conclusion.

Acknowledgment

References

[1] Stefan Wender, Ian Watson *Applying Reinforcement Learning to Small Scale Combat in the Real-Time Stratey Game StarCraft:BroodWar*. Brazilian Symposium on Computer Games and Digital Entertainment (SBGAMES),2014 .

[2] Ben G.Weber, Michael Mateas, Arnav Jhala *Applying Goal Driven Autonomy to StarCraft*. Expressive Intelligence Studio, UC Santa Cruz, 2010.

[3] Aleksandar Micic, Davíð Arnarsson, Vignir Jónsson *DEVELOPING GAME AI FOR THE REAL-TIME STRATEGY GAME STARCRAFT Research Report*. Computer Science B.Sc., Spring 2011.