Tank War Using Online Reinforcement Learning

Kresten Toftgaard Andersen, Yifeng Zeng, Dennis Dahl Christensen and Dung Tran
Dept. of Computer Science, Aalborg University
DK-9220 Aalborg, Denmark
line 3: City, Country
$\{$*krest,yfzeng*$\}$*@cs.aau.dk*

***Abstract*—Real-Time Strategy(RTS) games provide a challenging platform to implement online reinforcement learning(RL) techniques in a real application. Computer as one player monitors opponents'(human or other computers) strategies and then updates its own policy using RL methods. In this paper, we propose a multi-layer framework for implementing the online RL in a RTS game. The framework significantly reduces the RL computational complexity by decomposing the state space in a hierarchical manner. We implement the RTS game - *Tank General*, and perform a thorough test on the proposed framework. The results show the effectiveness of our proposed framework and shed light on relevant issues on using the RL in RTS games.**

***Keywords*-Reinforcement Learning; Real-Time Strategy Game**

I. Introduction

A Real-Time Strategy (RTS) game is a computer game in which players intend to defeat opponents by gathering resources, building bases, and exploring game fields. Generally, the RTS setting contains a large number of game states and complex relations among them. The data about both players' behaviors and game fields accumulates only when the game is moving on. Thus, a RTS application may frustrate conventional reinforcement learning (RL) [3] techniques due to the curse of dimensionality on the state space and insufficient data.

Most of game developers use the RL or other learning algorithms in the test phase of game programming or around this phase [2]. To the best of our knowledge, most of current computer game do not implement the online RL when players start the game. This may be firstly due to the RL's requirement on a large amount of computation which usually is not allowed online. Secondly, the online RL utilization puts computer players out of the control of game developers since unexpected paths toward the game goal may occur in the course of games. Hence, implementing various online RL techniques is more desirable in computer games, but challenges the current game development.

We choose the RTS game as the platform for the online RL experiment and propose several techniques for relieving specific constraints of the RL implementation. We present a multi-layer(e.g. hierarchical levels) RL framework that could offer tractable computation online. The top layer provides a general tactic through a RL system while the low layer learns specific actions in the game. In addition, we introduce a $Profiler$ model into the multi-layer RL framework. The $Profiler$ identifies a type of opponents and then activates the top level RL system to be configured with a new reward function. The model is necessary since it would further optimize the policy and speed up the RL convergence in an early game stage.

We implement a RTS game called *Tank General*, where two armies(blue and red) are fighting against each other in a middle-size map full of uncertainty. We show that the multi-layer RL framework could effectively speed up the RL computation and generate strategic behaviors in the course of games.

II. Multi-layer RL Framework

The RTS game is a sensible choice of game genre to make the online RL within. Similar to the normal use, the RL can take place before shipping off the game to the customers. However, it would be more interesting and useful to have the RL online and continue the learning even after the game starts. The learning would allow computer players to be more adaptive to the specific strategies of human players. We will discuss main issues regarding the online RL implementation in the RTS game. Our solutions will result in a multi-layer RL framework together with a $Profiler$ model.

A. Fast Learning

The learning must be fast, meaning that computer players must not seem dumb during the first several games, and must quickly pick up some clues of what human players are doing while playing the game. It may not consume a large amount of processing power during or after a specific game turn since this would annoy the players very much. Learning fast means learning from a small amount of data, which is always problematic due to the huge margin of variation in the limited data. The learning shall be able to recognize relevant data and only use them to update the policy. The recognition is general difficult especially in the early stage of games. One solution is to have computer players create various profiles of opponents and provide a method to identify a true type of the opponents . By knowing the opponent's type, the computer players would learn counter-policies accordingly in a quick time.

B. Unit and Commander Level Policies

The RL is able to generate and update policies which will adapt behaviors of computer players to real-time actions of human players. The policies could be placed into different parts of the order hierarchy. Their placement must be considered carefully since the consequence can affect the gameplay in the most profound way.

The RL could be placed at the unit level so that each individual unit learns from its surroundings. The learning would be wasted when the units die. A number of problems also arise if units do not synchronize their actions with others. If one unit is going to explore a certain area, it makes little sense that all other units are performing the same task. On the other hand, the synchronization would result in the deadlock problem since units are locked in decisions waiting for others to take their decisions. A possible solution could combine the synchronization with a central coordination mechanism.

The RL placement at the commander level, being able to give orders to the units and coordinate their actions, seems another alternative. The consequence is that the commander RL would have to collect data from all units and provide a single policy to units. The orders of a pure commander level RL are very precise and the number of orders becomes extremely large. The RL would be asked almost instantly after each order. The constant queries put much pressure on the RL and the amount of data that the RL would collect to generate policies may grow to be huge. Consequently, the commander RL may lose a big picture on the game.

C. Our Solution

We consider RL at both the unit level and the commander level in the RTS game, and provide a multi-layer RL framework in Fig. 1. The right side of the framework represents a hierarchical RL system which moves from the learning of a general strategy downward to the learning of specific behaviors. The top level RL learns a general strategy which commands main activities of all units in the current game stage. Under a general strategy, a specific RL system at the low level is used for learning optimal actions that all units can execute directly to counter opponents. Note that the framework could contain more layers.

The left side of the framework, $Profiler$, offers a mechanism to identify the opponent type given current game states. Subsequently, the $Profiler$ system proposes a corresponding reward function(predefined for each type of opponents) to the top level RL on the right. We use Naive Bayesian model [1] to classify opponents' type over time. The utility of using Naive Bayesian model to identify players' types is well studied in [4]. The parameters of Naive Bayesian model are learned from data which are collected through

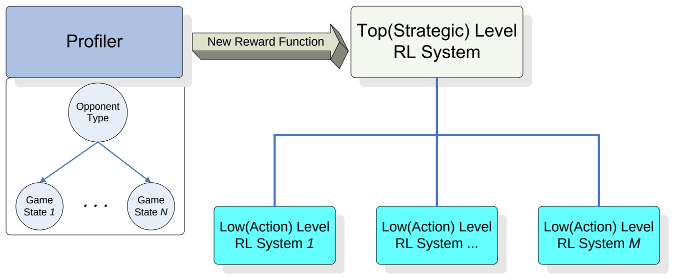
Profiler
New Reward Function
Top(Strategic) Level RL System
Opponent Type
Game State 1
. . .
Game State N
Low(Action) Level RL System 1
Low(Action) Level RL System ...
Low(Action) Level RL System M

Figure 1. The multi-layer RL framework contains both the $Profiler$ model and the RL in each level. The $Profiler$ using Naive Bayesian model identifies the player type and then provides an appropriate reward function that becomes an input of the RL in the top level. The RL at the top level learns a general strategy that triggers one of the low level RLs for learning more specific actions.

the game and characterize the state of the game as well as opponents' strategies in the history. Given a new observation on the current game state, the opponent is classified into one of three conventional character types, such as $Aggressive$, $Defensive$, or $Resource$ player, in the RTS game.

The framework is a conversion of a single layer RL system by decomposing a large state space of games and separating players' general strategy from specific behaviors. The net results generate a tractable state space for an individual RL system, which avoids heavy computation online and speeds up the RL convergence in the early stage of the gameplay. Notice that we do not specify any specific RL technique, like the SARSA or Q-learning, in the RL system. The framework is a general architecture which can accommodate various RL methods and implement them in a different layer.

III. Experimental Results

We implemented the RTS game *Tank General* and thoroughly tested the proposed framework.

A. Game Description and Analysis

The game setting is a battlefield where two teams, referred as the blue army and the red army respectively, are fighting against each other and aiming to destroy the headquarters of the other. Each team consists of five different types of units which differ in both exploring ability and shooting power. During the game each army may receive reinforcement points which would be used to build new units or to repair the headquarter. The resources called *War Factory* are randomly located in the battlefield. The occupied resource will give its owner more reinforcement points. In addition, to increase the strategical elements in the game, we initialize the battlefield covered by the fog of war. Both armies need to make strategic manoeuvres by sending out scouting units to explore the unknown areas. Screen shot of the *Tank General* game is shown in Figs. 2.

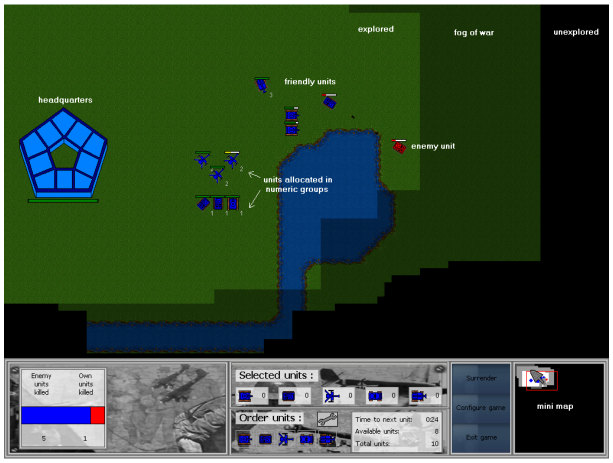

Figure 2. The main frame shows units' composition and locations of the blue army in a small area of battlefield where one enemy unit from the red army is intruding on the border of the fog of war. The bottom-left corner records the unit statistic in both armies and the bottom-right corner presents a mini-map. The mini-map zooms out the whole battlefield and exhibits the location of the currently explored area(with the red rectangular frame). Users can zoom in any area in the mini-map.

We analyzed the *Tank General* game and got 20 unique attributes by filtering out irrelevant features in the game. Most of the attributes are around 2-3 states but up to 4 states, and the combination results in an extremely large space(=181,395,528 states) which is intractable in a single RL system. We divided the whole state space into 5 components and built the multi-layer RL framework.

Similar to opponent modeling in a RTS game, we consider three types of opponents, $Aggressive$, $Defensive$ and $Resource$, in the $Profiler$ model. When a certain opponent type is identified we change the reward function in the top level RL. We have different reward sets, $Anti-Aggressive$, $Anti-Defensive$ and $Anti-Resource$, in the *Tank General*game.

The top level RL learns a general strategy among four: *Explore*, *Defend*, *Attack Headquarters*, and *Attack Resources*. When the strategy is output from the top level the corresponding low level RL is executed and learns specific actions for units. For example, under the general strategy *Explore*, the first RL system in the low level would be used to learn actions such as *BuildExploreUnit*,*ExploreRandomPosition*, *ExploreInfluenceMap* and so on.

To have a thorough test, we design two computer players: one uses the multi-layer (or single-layer) RL framework, and the other adopts a script and plays the game in a consistent manner. The scripted player could be $Aggressive$ type which attacks the enemy a lot, $Defensive$ which is very cautious and protective to headquarters, $Resource$ which focuses on capturing the resources. We have two computer players compete for 500 games and compute the winning percentage for each 25 games following the chronological order. All curves in Figs. 3, 4 and 5 are the trend-lines of all winning percentages(for 500 games) through a linear regression. A normal game takes 9-25 minutes depending on players.

B. Test 1: SARSA versus Q-learning

The first test aims for a comparison between two RL techniques: the SARSA and the Q-learning. As we expect (in Fig. 3) the computer game using the RL does really better than the scripted player and the Q-learning technique outperforms the SARSA, which verifies the online RL benefit and fast convergence in the Q-learning. We will therefore use the Q-learning as the RL technique in the subsequent tests. Notice that the computer player using the SARSA has a surprisingly decreasing tendency when it plays against the $Aggressive$ scripted player (the left chart in Fig. 3). This may be due to much randomness involved in the playing against the $Aggressive$ scripted player. In general, the game against the $Aggressive$ player(average 9 minutes) is much shorter than that against $Resource$(average 25 minutes). The $Aggressive$ type attacks its enemy as soon as it finds the enemy headquarter and wins most of time; however, the headquarter location is decided randomly when initializing the game.

C. Test 2: Profiler versus Non-Profiler

The purpose of the second test is to demonstrate the $Profiler$ would improve the performance of the RL in the *Tank General* game. Without the $Profiler$, the top level RL uses a fixed reward function which is the same as the one in a single layer RL framework. We show the results in Fig. 4.

We found that the multi-layer RL framework using the $Profiler$ performs clearly better against the $Aggressive$ scripted player and slightly better against $Defensive$. It seems that the RL framework without the $Profiler$ achieves better performance in the third case (the right chart in Fig. 4). The $Resource$ player focuses on capturing all resources and defending for them while occasionally attacking its opponents. This demands the $Profiler$ to take some time to identify the correct player type in the early game stage. We notice that the RL using the $Profiler$ steadily increases the winning chance and performs equally well after 500 games. Overall we conclude that the multi-layer RL framework gains benefit from the using of $Profiler$ model.

D. Test 3: Multi-layer RL versus Single layer RL

The third test demonstrates the performance of the multi-layer RL framework comparing with the traditional RL framework (single layer RL). The single layer RL framework considers the whole state space together with all actions in one RL system while the multi-layer RL decomposes the state space and organizes individual RL systems in two levels. Ideally the implementation of the single-layer

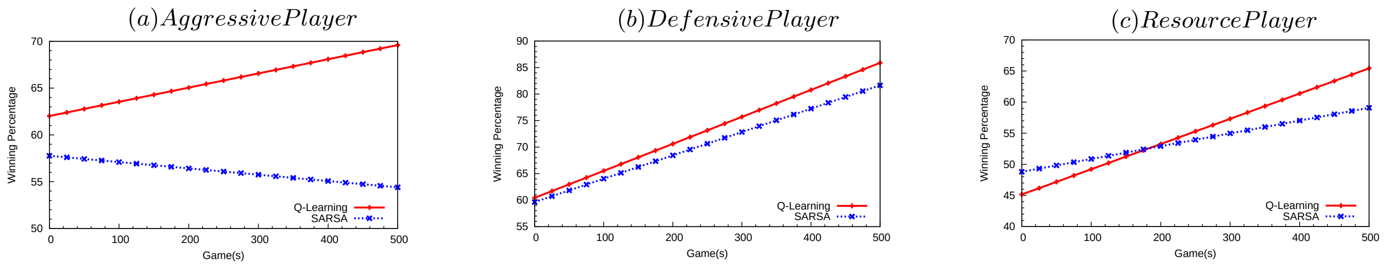

Figure 3. The Q-learning performs slightly better than the SARSA where the computer player using the multi-layer RL framework competes with a scripted player of three types.

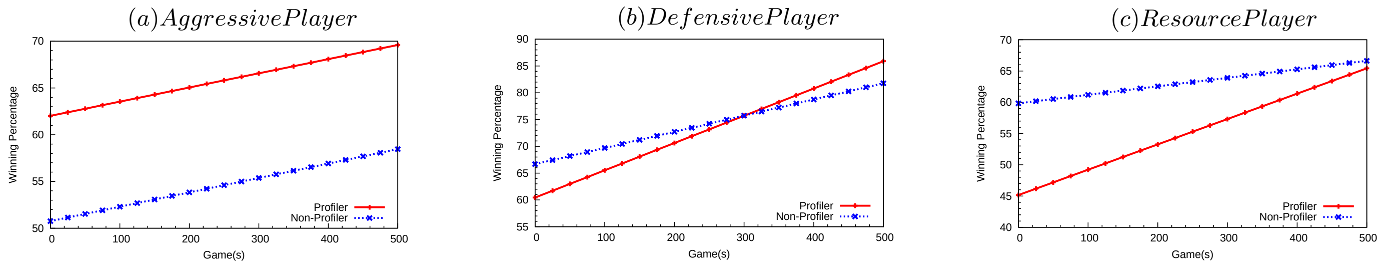

Figure 4. The *Profiler* model improves the RL performance in the multi-layer RL framework. Notice that the *Profiler* shows the benefit of modeling opponents.

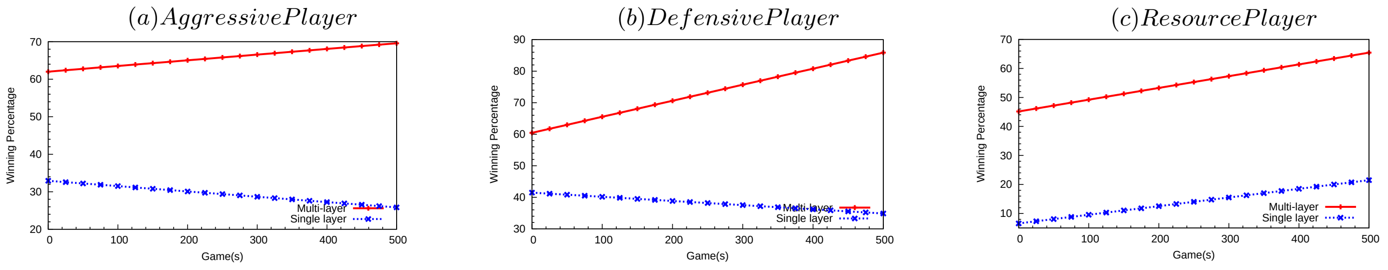

Figure 5. The multi-layer RL framework achieves better performance than the single layer one when both of them are used to compete against the scripted players.

RL system shall consist of all 181,395,528 states in the *Tank General* game. However, the space is so huge that the learning is extremely slow online. We therefore reduce the state space by filtering out less relevant states.

As the results(in Fig. 5) show that the single layer RL framework performs quite poor compared with the multi-layer RL framework. It results from the reduced state space even though we have a careful selection on the relevant states.

IV. Conclusion

Computer games are a useful arena for the RL application. One of the main difficulty on using the RL is due to an intractable state space in games. We propose a multi-layer RL framework by decomposing the large state space and then building several small RL systems which reside in different layers. The decomposition is not subjective, but distinguishes different granularities of strategies which offer an appropriate mechanism for controlling units' behaviors online. The results on the *Tank General* game verify the strength of the RL utilization in computer games, and empirically demonstrate the capability of our proposed framework.

Acknowledgment

We would like to thank Anders Buch for participating in this project and useful discussions on implementing the *Tank General* game.

References

[1] D. Geiger, M. Goldszmidt, G. Provan, P. Langley, and P. Smyth. Bayesian network classifiers. In *Machine Learning*, pages 131–163, 1997.

[2] J. E. Laird and M. van Lent. Machine learning for computer games. In *Game Developers Conference*, 2005.

[3] R. S. Sutton and A. G. Barto. *Reinforcement Learning - An Introduction*. The MIT Press, 1998.

[4] Y. Zeng and P. Doshi. An information-theoretic approach to model identification in interactive influence diagrams. In *IAT*, pages 224–230, 2008.