Experiments with Online Reinforcement Learning in Real-Time Strategy Games

Kresten Toftgaard Andersen, Yifeng Zeng*,
Dennis Dahl Christensen and Dung Tran
Dept. of Computer Science, Aalborg University
DK-9220 Aalborg, Denmark

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 firstly examine the suitability of applying the online RL in various computer games. RL application depends much on both RL complexity and the game features. We then propose a multi-layer framework for implementing online RL in an RTS game. The framework significantly reduces RL computational complexity by decomposing the state space in a hierarchical manner. We implement an RTS game - *Tank General*, and perform a thorough test on the proposed framework. We consider three typical profiles of RTS game players and compare two basic RL techniques applied in the game. The results show the effectiveness of our proposed framework and shed light on relevant issues on using online RL in RTS games.

1 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. It demands players to monitor opponents' behaviors and to be adaptive on taking counter strategies. As the name RTS indicates, the players shall make real-time decisions while interacting with the opponents

*Corresponding Author E-mail: yfzeng@cs.aau.dk

over time. 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 as the game is moving on. Thus, an RTS application may frustrate conventional reinforcement learning (RL) (Sutton and Barto 1998) techniques due to the curse of dimensionality on the state space and insufficient data.

Most of game developers use RL or other learning algorithms in the test phase of game programming or around this phase (Laird and van Lent 2005). The learning algorithm is well tested before a game product is shipped out to the target customers. When the learning converges toward one or more usable and optimal strategies the strategies are then hard-coded into a script that a computer player would read later on. By following the scripted strategies, computer players exhibit monotonic behaviors and become dumb over time when the strategies are detected by their opponents. To the best of our knowledge, most of current computer games do not implement online RL when players start the game. This may be firstly due to RL's requirement on a large amount of computation which usually is not allowed online. Secondly, 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 current game development.

We firstly investigate the suitability of using online RL in most genres of computer games and conclude some game candidates. The suitability depends a lot on the complexity of games, the number of different entities demanding RL, the room for RL computation, and other issues. For example, in a story heavy game like $Halflife$ (Halflife 2008), it would not make much sense for the same player to get smarter during the game course since it would not be clear how knowledge would be passed on between enemies.

We choose the RTS game as the platform for online RL experiment and propose several techniques for relieving specific constraints of 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 RL convergence in an early game stage.

We implement an 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 compare a basic RL technique, SARSA (Sutton and Barto 1998),

with a more sophisticated method, Q-learning (Watkins 1989), within the proposed framework in the game. As expected, the Q-learning performs better than the SARSA in the studied game. We show that the multi-layer RL framework could effectively speed up RL computation and generate strategic behaviors in the course of games.

The rest of this paper is organized as follows. In Section 2, we study some of the important game genres in the light of online RL suitability. In Section 3, we propose a multi-layer RL framework together with a $Profiler$ model. More importantly, in Section 4, we implement the *Tank General* game and illustrate the application of the proposed framework. We convey the experience on using online RL techniques in the test-bed. Finally, we summarize the relevant works on applying RL in computer games and provide remarks.

2 Game Genres: Online RL Suitability

We will discuss the currently most dominating genres in the computer game industry and leave out some of the simple games like sidescrollers, puzzle games and so on. We present relevant issues on using online RL and conclude with game candidates that fit with online RL.

2.1 FPS - First Person Shooter

In a general FPS game many adversaries must be shot down or neutralized by different means. This per default makes RL unattractive to be used since most enemies have a short lifespan and are unable to pass on knowledge to each other. A practical example would be the *Rainbow Six* series (Six 2008) which present serial choices to scripted computer players with no need for RL. The general FPS is also experienced as one long trek toward a rewarding goal, which results in few data for RL. This makes FPS a poor choice to use RL within due to the relative complexity of RL compared to the simple scripted computer players.

One exceptional example is the very successful *Counter-Strike* game (Counter-Strike 2008) which features human players fighting again bots or other human players controlled by computers. Since the game structure is server-based the bots would have a chance to learn and update the policy using RL after each death. A smart bot would be able to adapt to individual players and also learn general tactics. Hence in a FPS like *Counter-Strike*, RL fits nicely.

2.2 RTS - Real Time Strategy

The RTS genre has been popular since the dawn of the early PC strategy games, spawning ground-breaking titles like *Command and Conquer* (Command and Conquer 2008) and *Warcraft* (Warcraft 2008), and continues to thrill the game audiences by fast paced actions combined with tactical decisions being made in a split second. Computer players in these games have varied from plainly stupid, cheating and very predictable to clever, responsive and challenging over the years.

The RTS game is often split into a campaign part and a skirmish part. In the campaign part human players are playing a battle against computer players with a lower or greater amount of units/buildings. In a skirmish game both sides start on an equal size, and therefore the strategies to beat the computer players depends much on players' skills. It differs from the campaign part where the strategies rely on a level setup in the game.

Using RL in the campaign part of an RTS game seems over-skilled and is essentially not needed because computer players can be easily hard-coded without being dull for human players to defeat. The skirmish part on the other hand demands to have RL since the policy could be primed against certain types of behaviors while still being improved after a single skirmish game is ended. Some part of the policy table might also be updated during the course of games. RL implementation seems quite difficult, but the potential is certainly present.

2.3 TBS - Turn Based Strategy

Turn based games are traditionally divided into turns where only one player/opponent is allowed to make decisions regarding his/her play. The turn offers RL to have sufficient time to update policies and execute them. In the typical game *Civilization* (Civilization 1991) RL residing in the top level could decide to put a city near a certain resource, but the follow-up actions, such as creating units, moving them to the spot and starting up cities, might challenge RL. Some TBS games adopt simultaneous turns to make opponents act at the same time as computer players, which turns the TBS almost into real time movement. This structure allows little time to learn the policy, but would not be impossible for RL use.

The TBS game structure gives RL opportunity to have reasonable time to make up the policy; however, this must still be fast since players will not be pleased with waiting for too long before they can take a new turn. In general, RL is quite suitable in the TBS genre.

2.4 Sport

Two major genres are often featured in the sport game: simulation and manager. Both genres would benefit greatly from RL awarding them with skilled characters (players in the match). For the simulation game, RL could give experience to individual characters in the form of improved policies which tell characters how to put their skills to be the best use in the match. The *FIFA* series (FIFA 2008) model behaviors from different characters without having to hard-code them. The characters may improve their performance during or after the match, which depends on RL implementation.

For the manager game, RL would learn the behavior of other managers on owning teams like in the *Football Manager* series (Manager 2008), to help players buy/sell characters and place them in correct positions in the match and so on. This use of RL would occur in a high level for manager instead of characters in the game. The manager game shares similar structures with the TBS game by providing sufficient time to RL. Definitely, RL would bring much interest to two sport game genres.

2.5 Adventure

Computer players in a conventional adventure game do not exist and hereby RL is useless from the beginning. The example game *Myst* (Myst 2008) is nothing more than a connected line of pretty pictures without any form of computer intelligence. In the action adventure game featured by the highly regarded game *Diablo* (Diablo2 2008) the small monsters and bosses may own specific policies. Unfortunately, the policies may become completely predictable since the limited behaviors of monsters and bosses refrain RL to present interesting strategies. Here RL could be put in, but the benefit is still very questionable.

2.6 Arcade

The arcade game industry is a growing branch recently due to the fact that mobile phones and other hand-held devices have become more powerful than ever. The general arcade game is fairly simple in nature and is quite successful by being entertainment to player continuously. Perfect examples would be *Snake* (Snake 2008), *Tetris* (Tetris 2008) and *Pong* (Pong 2008) all of which are easy to start playing, but hard to be mastered. The high repeatability of games is a suitable place to accommodate RL which may train computer players to be adaptive. However, RL complexity and expensive computation would be unacceptable by game developers for mobile devices.

2.7 RL Suitability

We study the suitability of online RL applications in different genres of computer games and show that RTS, TBS and Sport games are prime candidates for using RL. The other game genres do not accommodate the online RL quite well due to the restriction of players' behavior and computational requirements of RL. As our interest we challenge online RL in RTS games.

3 Multi-layer RL Framework

We have seen that an RTS game is a sensible choice of game genre to make online RL within. Similar to the normal use, RL can take place before shipping off the game to the customers. However, it would be more interesting and useful to have 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 online RL implementation in an RTS game. Our solutions will result in a multi-layer RL framework together with a $Profiler$ model.

3.1 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, computer players would learn counter-policies accordingly in a quick time. Another solution is to let computer players take a general or decent policy at the early stage of games and then gradually learn more specific behaviors.

3.2 Unit and Commander Level Policies

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 game play in the most profound way.

RL could be placed at the unit level so that an individual unit learns from its surroundings. The learning would be wasted when the unit dies and there is no knowledge sharing among units. 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.

A logical idea would be to make the units learn from each other by combining policies or by other means sharing their knowledge. It could be applied so that the units would pass on the accumulated knowledge to the new units when they die themselves. This would make new units start with prior knowledge about the game. However, the issue of data starving agent might occur and there is a potential danger that the units will never live long enough to learn anything useful.

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. RL would be asked almost instantly after each order. The constant queries put much pressure on RL and the amount of data that RL would collect to generate policies may grow to be huge. Consequently, the commander RL may lose a big picture on the game.

3.3 Our Solution

We consider policy learning at both the unit level and the commander level, 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 $Profiler$

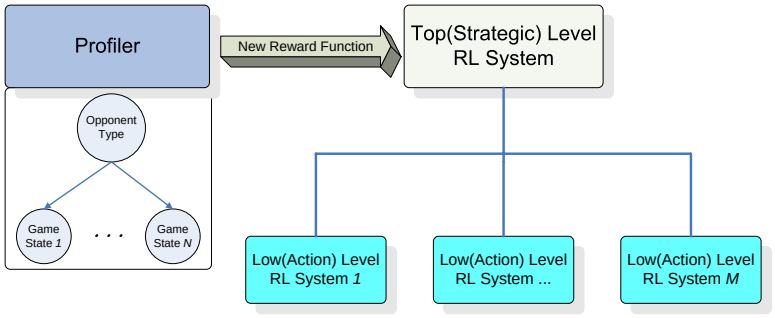

Figure 1: The multi-layer RL framework contains both the $Profiler$ model and RL in each level. The $Profiler$ using Naive Bayesian model identifies the player type and then provides an appropriate reward function that becomes an input of RL in the top level. RL at the top level learns a general strategy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

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 (Geiger, Goldszmidt, Provan, Langley, and Smyth 1997) to classify opponents' type over time. The utility of using Naive Bayesian model to identify players' types is well studied in (Zeng and Doshi 2008). The parameters of Naive Bayesian model are learned from data which are collected through 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 an 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 RL convergence in the early stage of the game play. Notice that we do not specify any specific RL technique, like the SARSA or Q-learning, in RL system. The framework is a general architecture which can accommodate various RL methods and implement them in a different layer.

For a self-contained paper, we include both the SARSA and Q-learning algorithms in Figs. 2 and 3 respectively. The Q-learning differs from the SARSA by not using the action-value for the next state, but instead taking the maximum value of actions. The theory study shows that the Q-learning converges faster than the SARSA since it utilizes a greedy method for determining the action-value rather than selecting a random action with a certain

probability.

Initialize for all states $s \in S$ and actions $a \in A$ the value $Q(s, a)$
to a constant value
Repeat for each episode:
Initialize s
Choose a for s using the policy π
Repeat for each step in the episode, until s is terminal:
Execute action a and save reward r and next state s'
Choose a' for s' using π
$Q(s, a) = Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
$s = s'$ and $a = a'$

Figure 2: The SARSA calculates the Q-value for each non-terminal state where α is the step-size parameter and γ the reward discount factor.

Initialize for all $s \in S$ and $a \in A$ the value $Q(s, a)$ to a constant value
Repeat for each episode:
Initialize s
Repeat for each step in the episode, until s is terminal:
Choose a for s using π
Execute action a and save reward r and next state s'
$Q(s, a) = Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)]$
$s = s'$

Figure 3: The Q-learning updates the Q-value whenever action a is executed in state s leading to new state s'.

4 Experimental Results

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

4.1 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

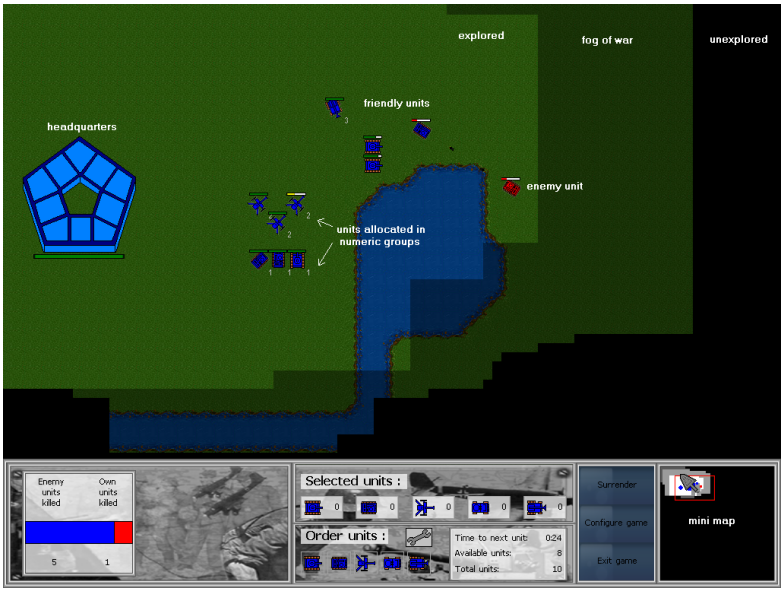

Figure 4: The main frame shows units' composition and locations of the blue army in a small area of battlefield where one enemy unit from the red army is intruding on the border of the fog of war. The bottom-left corner records the unit statistic in both armies and the bottom-right corner presents a 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.

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 shots of the *Tank General* game and final achievement screen are shown in Figs. 4 and 5 respectively.

We analyzed the *Tank General* game and got 20 unique attributes in the game. Most of the attributes are around 2-3 values but up to 4 values, 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

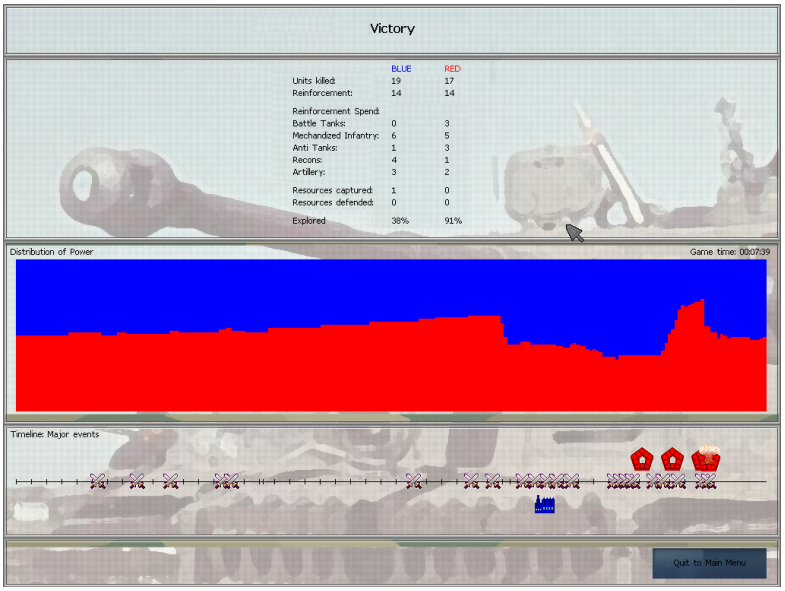

Figure 5: The achievement screen summarizes various information and statistic when the game is ended. The middle frame shows the power distribution of both armies when the game is moving on. The bottom displays a timeline on which major battles are plotted as a symbol of two crossed-swords. A small headquarter symbol indicates being attacked while being destroyed if there is a mushroom cloud on the top. A small symbol of blue (or red) war factory shows the captured factory at the time.

into 5 components and built the multi-layer RL framework in Fig. 6.

Similar to opponent modeling in an RTS game, we consider three types of opponents, *Aggressive*, $Defensive$ and *Resource*, in the $Profiler$ model. Using Naive Bayesian model, the profile evaluation depends on relevant attributes. For instance, the *Aggressive* profile would be evaluated regarding the elements of $NumberOfEnemyUnits$, $TotalShot$, $TimeWindow$, and $AverageRateOfFire$. When a certain opponent type is identified we change the reward function in the top level RL.

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. All state spaces, actions and reward values for each RL system are detailed

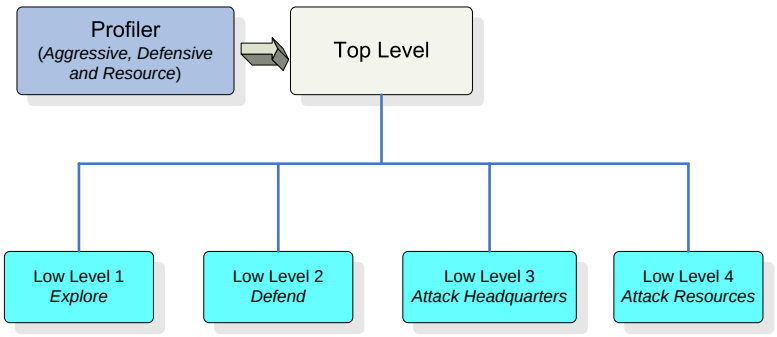

Figure 6: The multi-layer RL framework is provided in the *Tank General* game.

in (Andersen, Christensen, Tra, and Buch 2008).

We implemented both the SARSA and the Q-learning in the RL system. The parameter step-size is set to 1 since the *Tank General* is a non-stationary game world where a high and constant value is appropriate to use. We chose 0.1 for a balance between exploitation and exploration actions. The discount factor is set as 0.5 in the running.

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. 7,9 and 11 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.

4.2 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. 7) 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 top-left chart in Fig. 7). 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) as shown in Fig. 8. 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.

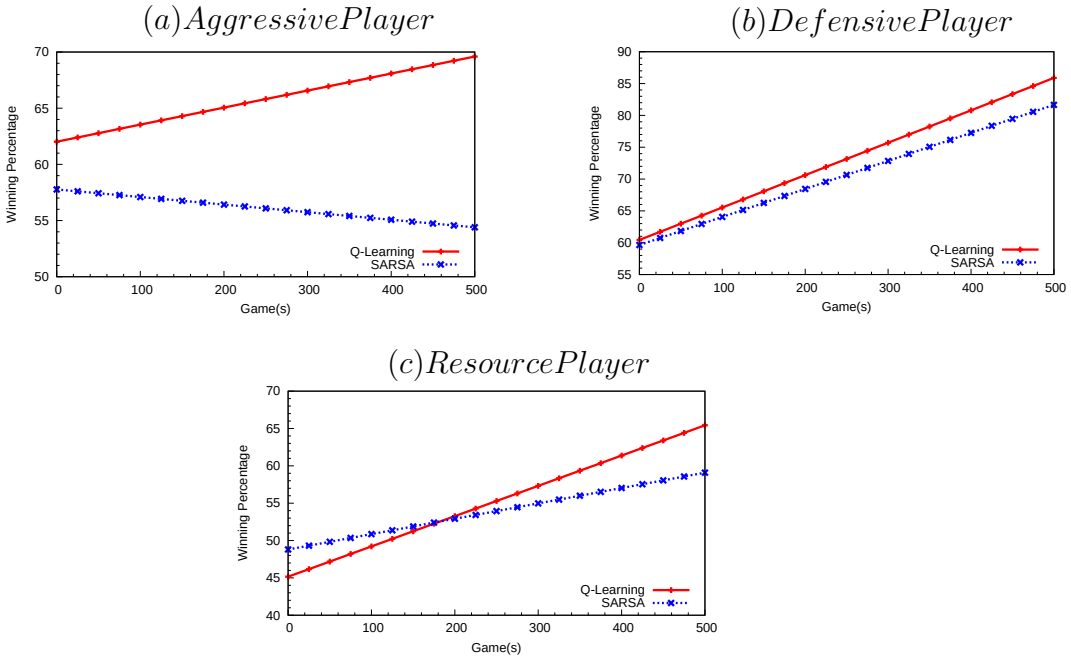

Figure 7: The Q-learning performs slightly better than the SARSA where the computer player using the multi-layer RL framework competes with a scripted player of three types.

4.3 Test 2: Profiler versus Non-Profiler

The purpose of the second test is to demonstrate the *Profiler* would improve the performance of 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. 9.

We find 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 RL framework without the *Profiler* achieves better performance in the third case (the bottom chart in Fig. 9). The *Resource* player focuses on capturing all resources and defending for

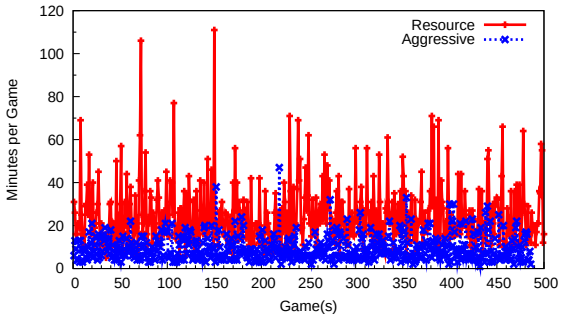

Figure 8: Comparison of game time when the player using the multi-layer RL framework plays against either the *Aggressive* or the *Resource* player.

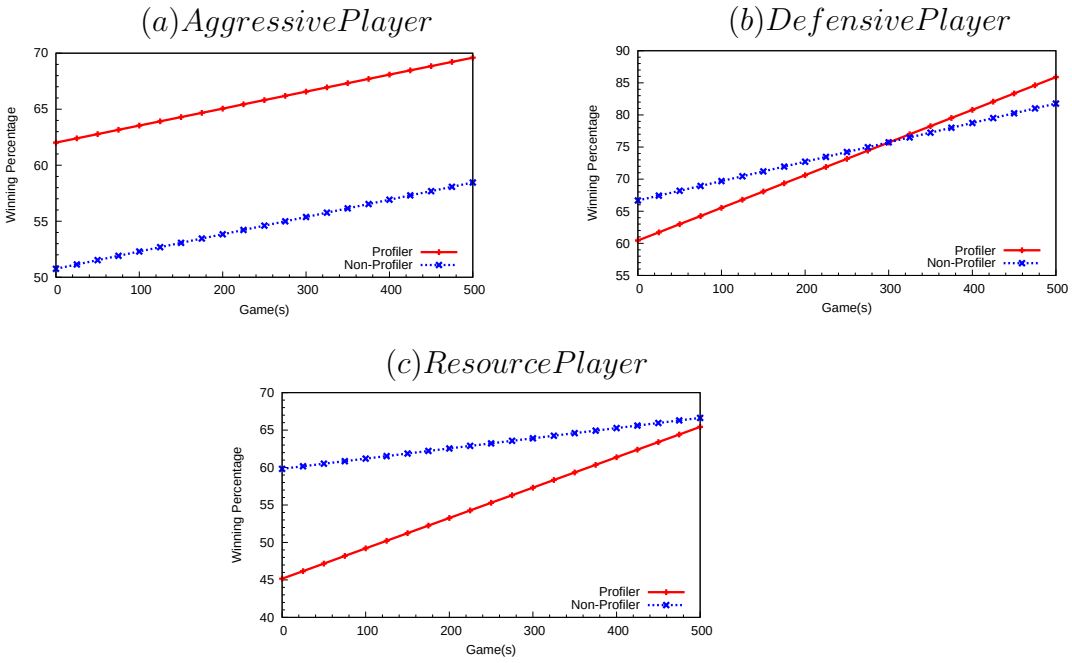

Figure 9: The *Profiler* model improves RL performance in the multi-layer RL framework. Notice that the *Profiler* shows the benefit of modeling opponents.

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 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.

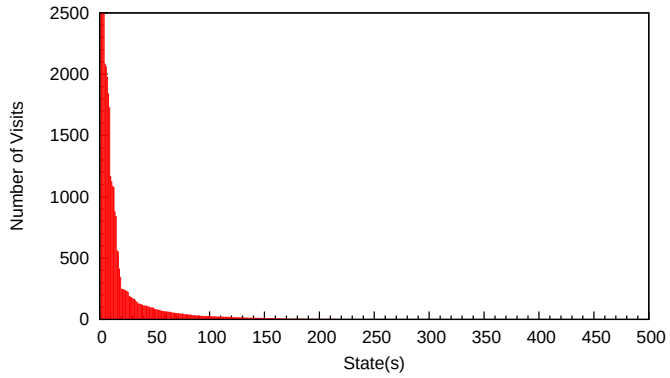

Figure 10: Some states are visited frequently while other states are never visited.

4.4 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 into two levels in Fig. 6. Ideally the implementation of the single-layer 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 that has been seldom approached in the game.

We played 500 games using the multi-layer RL framework and estimated the frequency for which each state has been visited. It shows that in average each state has been visited for 5.16 times. However, some states are much more frequently approached than others. We show the distribution of visits for a part of state space in Fig. 10. We removed the states that have a low visit and kept only 16384 states. Consequently, we had a manageable state space in the single-layer RL framework.

As the results (in Fig. 11) 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. To make a further check, we found that the single layer RL system does converge, but ends in non-optimal solutions due to insufficient game information.

Instead of using the scripted players as the straw-man, we designed two players which are equipped with the multi-layer RL framework and the single

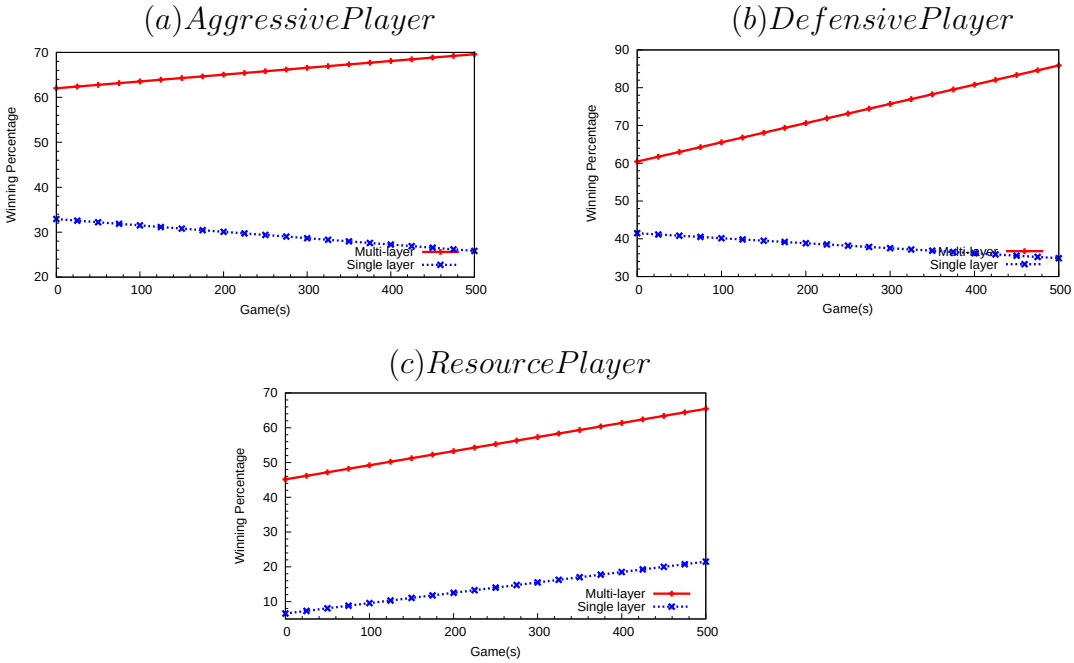

Figure 11: The multi-layer RL framework achieves better performance than the single layer one when both of them are used to compete against the scripted players.

layer one respectively, and let them compete with each other. We arranged 20 sets of games each of which contains 25 individual games, and computed the winning percentage for both players. In Fig. 12, the results are not surprising that the player using the multi-layer RL won 355 out of 500 games (71% winning rate).

5 Relevant Works

RL methods have been well studied in machine learning area where much of work is invested into the theoretical aspect on either the performance improvement or the extension from a single agent case to a multi-agent case. Good survey papers could be found in (Kaelbling, Littman, and Moore 1996; Woergoetter and Porr 2008). Although RL techniques have been demonstrated successfully in a classical board game (Tesauro 1994), computer game applications are recently caught into the right track (Manslow 2004). The major obstacle is the curse of dimensionality which makes RL be lacking of efficiency. Recently, much effort appears to address the difficulty and shows

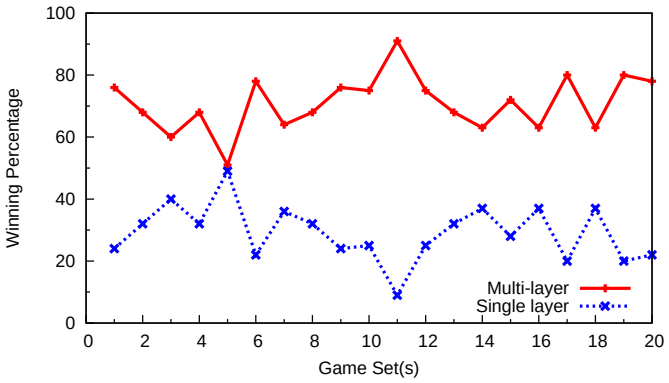

Figure 12: A direct war between the multi-layer RL framework and the single layer RL. The player using the multi-layer RL framework wins most of games.

interesting results.

A concurrent hierarchical RL technique was used in the resource gathering game - *Stratagus* where a group of units are working together for achieving a common goal (Marthi, Russell, and Latham 2005). The proposed learning method adopts the multi-threaded partial program to effectively control units' actions and avoid the collision. In the *GO* game (Silver, Sutton, and Müller 2007), David *et al.* proposed to reduce the state space by considering the combination of pieces instead of the entire board, which definitely enhances RL performance. Furthermore, they discover location independent states and boost RL convergence. For a more commercial application, Gustavo *et al.* presented a dynamic game balancing method(on RL basis) to activate computer players for adapting the skills of human players thereby achieving more entertainment in games (Andrade, Ramalho, Santana, and Corruble 2005). Our proposed multi-layer RL framework shows successful experience in the RTS game and contributes to the growing lines of work on using RL in computer games.

6 Conclusion

Computer games are a useful arena for RL application. We have provided a broad survey on various game genres from the viewpoint of online RL suitability. The study discovers difficulty on using online RL in computer games. We choose RTS games for experimenting RL since the RTS involves more computer-human interactions and demands more strategic behaviors.

One of the main difficulty on using 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 *Profiler* enhances RL ability by providing a suitable reward function according to the identification of opponents' type in time. We implement the *Tank General* game and experiment the proposed framework across a large number of game competitions. The results verify the strength of RL utilization in computer games, and empirically demonstrate the capability of our proposed framework.

Future avenue of the *Tank General* game looks bright. We expect to apply sophisticated opponent modeling approaches for developing a more solid learning engine in an RTS game.

Acknowledgment

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

References

Andersen, K. T., D. D. Christensen, D. Tran, and A. Buch (2008). *Reinforcement Learning in RTS games*. Master Thesis, Department of Computer Science, Aalborg University, Denmark.

Andrade, G., G. Ramalho, H. Santana, and V. Corruble (2005). Automatic computer game balancing: a reinforcement learning approach. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1111–1112.

Civilization (1991). http://www.links.net/.

Command and Conquer (2008). http://www.commandandconquer.com/.

Counter-Strike (2008). http://store.steampowered.com/app/240/.

Diablo2 (2008). http://www.blizzard.com/us/diablo2/.

FIFA (2008). http://www.gamespot.com/.

Geiger, D., M. Goldszmidt, G. Provan, P. Langley, and P. Smyth (1997). Bayesian network classifiers. In *Machine Learning*, pp. 131–163.

Halflife (2008). http://www.gamespot.com/.

Kaelbling, L. P., M. L. Littman, and A. W. Moore (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research 4*, 237–285.

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

Manager, F. (2008). http://www.footballmanager.com/.

Manslow (2004). Using reinforcement learning to solve ai control problems. *AI Game Programming Wisdom 2*, 591–601.

Marthi, B., S. Russell, and D. Latham (2005). Concurrent hierarchical reinforcement learning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 779–785.

Myst (2008). http://www.mystworlds.com/us/.

Pong (2008). http://www.pong-story.com/.

Silver, D., R. Sutton, and M. Müller (2007). Reinforcement learning of local shape in the game of Go. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Six, R. (2008). http://www.redstorm.com/.

Snake (2008). http://www.snakegame.net/.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning - An Introduction*. The MIT Press.

Tesauro, G. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput. 6*(2), 215–219.

Tetris (2008). http://www.tetris.com/.

Warcraft (2008). http://www.blizzard.com/us/war3/.

Watkins (1989). *Learning from Delayed Rewards*. Cambridge University, Cambridge, England.

Woergoetter, F. and B. Porr (2008). Reinforcement learning. *Scholarpedia 3*, 1448.

Zeng, Y. and P. Doshi (2008). An information-theoretic approach to model identification in interactive influence diagrams. In *Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT)*, pp. 224–230.