Survey and Analysis

Name: Junjin Chen

Supervisor: Eleni Vasilaki

Module: COM3610

Date:

This report is submitted in partial fulfilment of the requirement for the degree of BSc by Junjin Chen

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Junjin Chen

Signature:

Date:

Abstract

Inspired by Deep reinforcement learning of Atari games done by Deep Mind which in many games outperforms than human player, this project will demonstrate the learning algorithm on a Real Time Strategy (RTS) game Starcraft2. Applying learning method in RTS games are generally considered challenging since there are usually large action space and large state space in the game as well as large amount of possible strategies for future states, especially in this case where Starcraft2 is a typical RTS game. However, open source from the cooperation between deep mind company and Blizzard entertainment has provided useful and convenient platform for researchers. It includes convolutional layering from the raw data, simplified maps and some example base-agents. Learning methods and the policy-reward functions are only needed to be concern from the researchers. Other researches about other games online also provide the theory of how to apply the learning algorithm into the game. In this article, important terminologies and relevant literature reviews are presented.

Contents

[1. Introduction (700) 6](#_Toc500083774)

[1.1 History 6](#_Toc500083775)

[1.2 Recent work 6](#_Toc500083776)

[1.3 DeepMind with Starcraft2 6](#_Toc500083777)

[1.4 Report Overview 7](#_Toc500083778)

[2. Literature Survey (2300) 8](#_Toc500083779)

[2.2 Starcraft2 8](#_Toc500083780)

[2.2.1 game structure 8](#_Toc500083781)

[2.2.2 Common strategies 9](#_Toc500083782)

[2.1 Theory of RL 9](#_Toc500083783)

[2.1.1 Action value methods 9](#_Toc500083784)

[2.1.2 Instruction 10](#_Toc500083785)

[2.1.2 Reinforcement comparison 11](#_Toc500083786)

[2.1.3 Returns 11](#_Toc500083787)

[2.1.1 Markov Decision Process 11](#_Toc500083788)

[2.1. Bellman Equation 12](#_Toc500083789)

[2.1.2 Dynamic Programming 12](#_Toc500083790)

[2.1.3 Monte Carlo Method 13](#_Toc500083791)

[2.1.4 Temporal Difference Method 13](#_Toc500083792)

[2.1.5 Eligibility Trace 14](#_Toc500083793)

[2.1.6 Function approximation 14](#_Toc500083794)

[2.3 RL application 14](#_Toc500083795)

[2.3.1Previous work of DeepMind on SC2 14](#_Toc500083796)

[2.3.2 Bubble Shooter 15](#_Toc500083797)

[2.3.3 Racing Game 16](#_Toc500083798)

[2.3.4 Flappy bird 16](#_Toc500083799)

[2.3.5 First Person Shooting game Doom 16](#_Toc500083800)

[2.3.6 Atari game 16](#_Toc500083801)

[2.3.7 Simple Board game 16](#_Toc500083802)

[3. Requirements and Analysis (1000) 18](#_Toc500083803)

[3.1 Objectives 18](#_Toc500083804)

[3.2 Tools 18](#_Toc500083805)

[3.3 Problem Analysis 18](#_Toc500083806)

[4. Progress (500) 20](#_Toc500083807)

[4.1 Architecture 20](#_Toc500083808)

[4.2 Parameters setting 21](#_Toc500083809)

[5. Conclusions and Plan (500) 23](#_Toc500083810)

[Reference 26](#_Toc500083811)

[Appendices 27](#_Toc500083812)

# Introduction (700)

Reinforcement learning (RL) is an Artificial Intelligence (AI) method that enables the computer to solve problems and find optimal solution in a fast way. The problems could be computer games and also could be real world problems such as city planning. Some problems are so complex that may require ages using brute force computing to find the best solution. RL method enables the computer (here we call Agent) using the past experience and the reward from a result to improves itself gradually. In some problem, RL method can outperform the human.

## 1.1 History

The theory of RL was developed since late 1950s. The problem was to find a general stochastic optimal control. Then many theories such as “Dynamic Programming” and “Markov Decision Process” were developed and applied in more complex applications.(p16)

## 1.2 Recent work

Google Deep Mind has applied RL in Alpha Go, an agent that plays the board game Go. It became famous after defeating the human professional Go player with 9 dan (highest rank in Go) Lee Sedol with the score of 4-1. In fact, AlphaGo was initially given 100,000 games to train and mimic human player, after that it was ordered to play the games itself 30,000,000 times. In each training session, the RL agent managed to improve itself little by little. This resulted in that the Alpha Go outperform the professional human player.

Before that, Deep Mind company has applied deep reinforcement learning in playing Atari games. The RL agent only received raw data as the input. It then learns to play games by training. The agent was able to play across wide range of games, which made it the world’s first ‘General purpose machine’. The result came out that the RL agent played better than human player in some games. An example game was the space invader. After training for one night, the agent was able to make the prediction shot at the end. This performance is difficult for human to achieve, since the invaders move faster towards the end and the bullet moves slowly. But in some games it performs poorly since that game requires advance knowledge. Montezuma's Revenge is an example of the game in which the agent did not perform well, because it was a Role Play game and it requires the knowledge of English Language.

Overall, RL is quite strong when solving some problem that do not require advance knowledge.

## 1.3 DeepMind with Starcraft2

[10] Deep Mind company now cooperate with Blizzard entertainment and provides the open source of Starcraft2 (SC2) learning environment and agent for researchers. Starcraft2 from Blizzard is a Real Time Strategy (RTS) game with rich culture and colourful background story. There are also many elements in the game such as different species (Protoss, Terran and Zerg) with different units and different abilities. In a game, players need to collect resources from over the map, build different structures and produce different units, while doing some micro-control and planning the long-term strategies to defeat enemies. It has complex rules but it is also fast paced game which usually last for 10mins. It sometimes can last for an hr. The challenge for applying RL in SC2 that Deep Mind was facing was that: firstly, in multi-player game (2v2 or 3v3, 4v4) agents’ actions influence the overall situation, meaning that a single agent may not know why it was getting this reward. In other words, an agent might think it had contributions to the game when it was choosing the wrong action. Secondly, the map is covered by fog until the agent explores the area, such that the agent will only have partial clue of the map. This restricts what the agent will learn. Finally, the number of possible action is large since in a map, there are many clickable points, as well as the non-spatial actions such as a special ability of a unit. This leads to large state space, as different position of army, different amount of resources counts different state. Deep Mind has done some work for SC2. The company developed three architecture of the agent: Atari agent, full convolutional agent and fullconv LSTM agent. The detail of these architecture will be explained in later chapter. It came out that Atari agent outperforms in combat game, while only full convolutional network agent learned to produce the worker to increase the income in game. When playing the full game, the best agent only learned to avoid losses by moving the building out of the attack range from enemies. (Terran’s special ability)

## 1.4 Report Overview

In the next big chapter, Literature Review, a detailed description of SC2 will be revealed initially, then theories and some terminologies of RL will be explained when the SC2 will be used as example, other research of RL will be presented. The chapter after that, Requirement and Analysis, will demonstrate the objectives of this project, some problems that may be in the way and some useful tools and technique to solve the problems as well as the result that we expected to have. The last chapter will be showing the things I have done so far, and the plan of implementation presented as Gantt chart.

# Literature Survey (2300)

## 2.2 Starcraft2

[2] Starcraft2 is a classic RTS game with many elements. In the game, players can control their units to construct base and build armies and fight for resources. Players needs to send his workers to collect the resources, using the resources to build some buildings, produce armies, develop technology to enable some special ability of the units, explore the map, produce different units and combine them to perform efficiently in compact, while making balance among those elements.

### 2.2.1 game structure

Collect resources: player controls the workers to collect mineral or gas, workers will store those resources in the base building. Thus worker needs to travel between mineral field or gas field and base building. And thus the base building should be built as close to the resource field as possible. Resources are the economy of the game, it can be transfer to other things in the game, hence effective collection of resources will have a good position in the game.

Buildings: some buildings produce units; some buildings enable the player to develop some abilities of the units or be able to produce different units; some buildings can be used in defense of intruders. Different buildings may cost different resources.

Units: works are used to build base, some units contribute the damages to enemy, some units can be used to transport other units, some units support the fighting units using their special abilities. Players will need to combine different units to perform an effective compact. There are loans of combinations and weights of the army which contributes to the large number of the strategies in the game.

Support: the number of units that can be produce is limited by support. Different units may require different amount of support. Additional support will be required when player wants to produce a unit that may cause exceed of the support. Depends on species, support can be increased by build some buildings or produce a certain kind of units (that unit will not cost any support). The maximum support is 200, no more units can be produced if 200 support reached even if you build more support structures.

Technology: technologies can be researched in certain buildings. They make the army stronger and efficient and substantial. Some technologies increase the units’ damage or armor; others enable the units’ special. They cost certain amount of resources and time to be researched. The player needs to balance the time to develop a technology. Too early researching may lead to bankrupt in the early game and less units can be produced to defend the enemy. Too late researching may cause the units in combat are weaker than enemy and loss fight quickly. One can stop a research of the opponent by destroying the building where the research is undertaking.

### 2.2.2 Common strategies

Explore the map: the map is covered by fog until the player explores it. The enemy can be anywhere in the map, player should regularly visit some place of the map in order to observe the enemy’s actions and predict its motivation.

Expand the base: resources field will be exhausted in the future, and player needs to find another resources field to collect the resources in order to keep good economy. When doing the expansion and migration of those workers, player should also consider about the position of the resources in the map, whether it is good in defense or attack based on the situation the player is in.

## 2.1 Theory of RL

[1] Reinforcement learning is defined as a method to make the agent learns from reward or punishment of past experience. In a problem such as playing games, the agent takes an action in a state of the game, then the game will update its state for the agent. In this process there will be reward signals which tells the agent whether it is good or bad in this situation. The agent will then upgrade its policy and try to maximize the reward signal. However, things can be complex, as defined in the problem, i.e. the rule of the game. In some cases, actions from the agent may affect future rewards. The current maximum reward may not be able to present the overall optimal situation. Thus that the agent should not only select a same action every time that has temporally maximum reward (exploitation), but also be able to search other actions to see the future result (exploration). Below will list several methods that can make the agent to explore the possible actions while exploiting the action space.

An RL agent has its policy, reward function, value function, and optionally, the model of the environment.

Policy tells the agent what action to choose in a particular situation, it maps the states to actions.

Reward function tells the agent whether current situation is good or bad.

Value function tells that whether the current situation is good for future states, it basically calculates the expectations of the future reward function.

Model tells what the environment will be in the agent’s sense. It is basically some knowledge prepared for the agent which sometimes could be wrong. A good model helps the agent to do the planning.

Below are some RL methods….

### 2.1.1 Action value methods

Action value methods determine how good is an action. A natural way to do that is to calculate the mean value of the rewards of the situation that an action that being chosen by many times. In this way, the agent will get a more accurate action value when it performs exploitation. Below is a simple action value function, where Q is the action value, r is the reward and k is the number of time that the agent choose action a:

For policy, a simple way to determine which action to take is using the Greedy function. Greedy function will always select the action that has the highest value calculated by action value methods. The disadvantages of this method is that if a value of an action is constantly maintaining the highest (local maximum), the agent will not explore other actions. One solution for that is to have a parameter in the greedy function that there will be a small chance when the agent completely ignores the action value methods and choose an action randomly. This enables the exploration for the agent. However, since it is random, the agent might make the worst decision which will affect the future state. This could lead to time wasting on those not reasonable decisions. Solution for that is applying the Softmax action selection rules. These methods will rank the actions with probabilities relative to the action values, such that the agent will be more likely to choose some reasonable actions. A general function of softmax method uses Gibbs, or Boltzman distribution:

where is the action value, is the temperature that determines the intense of difference among the actions.

For example, in the SC2, suppose the agent is choosing a coordination to build the base. The base should be built as close to the mineral field as possible for ease of collecting resources. In the map, there are several mineral fields. With the greedy function, the agent once knows a place to build the base is good, it will always build the base there, without exploring the places near other mineral field. And it will keep doing this even if the mineral field becomes poor in the late game, which is bad. However, if we only add a parameters to the agent, at some point it will choose a random place to build the base. If lucky, the agent will find another place that is beneficial to collecting mineral source. However, this chance is very small since number of the possible places to build the base is far more than the optimal places. This is also bad. With softmax methods, the agent will tend to choose some near-best actions to explore and skip those ‘bad actions’, which is more efficient than -greedy functions.

### 2.1.2 Instruction

Instruction tells the agent what the right action would have been no matter what action the agent selects. Few of them would be useful to direct agent’s action in selection rule. This could be helpful when we know some actions must be taken to get the optimal state. Without instruction, the agent with 0 knowledge could waste time on figuring out the necessary action to be take.

### 2.1.2 Reinforcement comparison

Another way to construct a policy is using Reinforcement comparison method, which uses a reference reward to indicate whether the reward gain by an action is good or bad. These kinds of method are sometimes more efficient than action value method.

### 2.1.3 Returns

The way that the agent learns from past experience and evaluates in a similar situation is to calculates the Returns. Returns sums the rewards from current state to the future final state, where the ‘future rewards’ come from experience. After having been through a play, the agent will record the information of rewards and when play the game next time, it will be able to estimate the returns.

However, sometimes it takes too long to play a full game, we can then break the game into subsequences in a natural way; they are called episodes. Episodic tasks are easier to compute than continuous task because in episodic task, the actions affects shorter number of rewards, and hence the returns of each episode are shorter to computer.

For nonstationary problems, the current state is more relevant to its near neighbour state. Thus, it makes sense to weight recent rewards heavier than long-past ones or further future ones. This can be done using a discounting factor . For instance, the discounted return will be:

An example of using return could be in a combat game of SC2, where players need to produce some specific units to counter the attack from enemy. This is an episodic task of the whole game where the agent in this task does not need to consider the future late game states. Assume the agent loss the previous game because it produced wrong units for defense. In the next game, the agent using the return value will know that produce the same units as previous game will get bad rewards in the future. Thus it will produce some other units in next game. In conclusion, the agent learns from experience by calculating the returns.

### 2.1.1 Markov Decision Process

To break the problem into sequence of sub-problem naturally, we need to consider the Markov Property. Markov Property is presented when a state summarizes the past and be able to predict future, in other words, when it succeeds to include all relevant information for the agent to make a decision. This sounds impossible in practice since there is too much information that is relevant in practice, but we can divide the problem into episodes such that the states in each episode are close to have Markov Property.

Markov Decision Processes (MDP) builds on Markov property and use a function to determine which action to pick. MDP has two quantities, one calculates the probability of possible next states as:

where is the state at time , is the action and is the next possible state.

Another quantity calculates the expectation of rewards given current state and action along with next state:

where is the reward.

MDP is important in theory of RL.

### 2.1. Bellman Equation

Recall that Action Value function calculates the average rewards of an action, a more general function is the Value function which calculates the expected return for state (denote V) or action value (denote Q) in a policy. Value function tells how good is the state that agent wants to get, it evaluates states.

Solving a problem may require calculating lots of rewards, we can have an optimal value functions that calculate expected return for optimal state-action pair. Recall that policy maps from states to actions. There must be one or several optimal policies among other policies that can achieve the highest state-action values. Optimal value functions represent the state value or action value for optimal policies.

Bellman equation uses the value function to expresses the relationship between current state and its successor states in turns of their values. The bellman equation for state value of a policy can be written as:

where and are the quantities of MDP, is the discounting factor.

Similarly, Bellman optimality equation is the bellman equation for optimal value functions, denoted , and can be expressed as:

### 2.1.2 Dynamic Programming

Dynamic programming are some algorithms that can find the optimal policy provided a model that has Markov property. Since the Markov property is impossible in real world, there will be some limits when implement DP in RL. Key idea of DP is to pick the good policies from all policies ranked by the value function.

The process of DP can be divided in to policy evaluation and policy improvement. Policy evaluation can simply be using the value function to evaluate the state-action pairs in a problem. Policy improvement finds a policy that has higher value. These two tasks may not find the optimal policy in one run, thus, we need to iterate them. However, for policy evaluation, it could sometimes cost too much time in a complex problem, while the improvement has to wait until it is done. In this case we can use the Asynchronous Dynamic Programming. ADP algorithms back up the values when performing the policy evaluation and make use of them in policy improvement, such that the policy may not need to wait for years to be improved.

In practice, DP may not be suitable for those complex problems, since it requires the Markov Property for the environment. However, DP methods are somehow efficient than other methods, with the worst case time as polynomial to the number of states and actions.

### 2.1.3 Monte Carlo Method

Different from DP methods, Monte Carlo methods do not require Markov Property in environment. The agent with Monte Carlo methods learns value functions and optimal policies from only experience in the form of sample episodes. Technically speaking, it calculates the average state-action values over random samples. Recall that the state-action values calculates the expected return of a state, and the return is the sum of rewards from current state to the future final state. Monto Carlo methods observe the returns from sample experience, then average those returns and converge to an expected value. This is the policy evaluation process of Monte Carlo and it is different to the DP methods, but the policy improvement can still be the same.

A problem of this method is that if the policy explores every action just once, there will be no average of the state-action values, in other words, the agent will not be experienced enough to improve the policy. Thus the actions for agent to decide should be ranked or weighted using stochastics method.

The policy iteration of evaluation and improvement of Monte Carlo methods can be applied with on-policy and off-policy. On policy means that the algorithm is able to evaluate and improve the policy which determines the action, in other words, the policy that determines the action is the same as the policy to be evaluated and improved. Off policy means that the policy that determines the action may not be evaluated or improved, but a hidden policy can be evaluated and improved by observing. Advantage of off policy is that episodes can be generated and sampled continuously with the same policy that determines the action.

The advantages of Monte Carlo methods over DP methods are firstly, they do not require a model of environment; secondly, they can be applied in simulation and some sample models. The disadvantage is however, if the episode is too long, it may need to wait for a long time in one iteration.

### 2.1.4 Temporal Difference Method

Temporal-Difference (TD) methods combines the ideas of Monte Carlo methods and DP methods. What it takes from Monte Carlo methods is to learn from experience, such that it will not require a model of environment; and what it takes from DP methods is to back up the values in the run and use them to improve the policy without waiting for a complete run of a game.

Same as Monte Carlo methods, TD agent uses experience to predict the action value in the current play. The different is that it does not use the whole experience of the complete game for the prediction. Instead it uses partial experience that can represent the near future states in prediction. This is called a bootstrapping method, which is used in DP methods.

Sarsa is an on-policy TD method that use five parameters to evaluate the current action value using TD method prediction of future action value. The algorithm is generally written as:

where is the step size parameter and the arrow means the update of the value.

Q-Learning methods in other way is an off-policy TD method. The formula is written as:

The difference between Q-learning and Sarsa is that in the prediction part, Q-learning formula uses the discounted maximum action value among all the actions, while Sarsa is using the discounted action value determined by the policy.

Actor-Critic methods are on-policy TD methods that structure the agent as an actor and a critic, where the actor is the policy and the critic has the value function. Different from other methods, the policy in Actor-Critic methods is independent to the value function. The critic also calculates the error of whether an action has made a better position or a worse one. This error will then direct how policy is going to be improved.

## 2.3 RL application

### 2.3.1Previous work of DeepMind on SC2

[10] The previous work done by DeepMind in Starcraft2 game was that using a method called Asynchronous Advantage Actor Critic (A3C) to learn the policy parameters. A3C looks like this:

Where is the return that the agent aims to maximize.

There are 3 architectures that DeepMind used in the experiment. One is the Atari-Net architecture, which process the non-spatial features to a linear layer with a tanh non-linearity. The screen and mini-map are processed to convolutional layers, then combine with the result from non-spatial features. Then they are processed into a linear layer with ReLU activation before inputting to the agent. The second one is the Full-Convolutional Network architecture, where it concatenates the result of convolutional layered screen and mini-map, with the non-spatial features as state representation. By this means the agent predicts spatial action directly through the state representation.

The result was that, overall, the full-convolutional network performed better than Atari-Net. In particular, Atari-Net performed strongly in combat task, where full-convolutional network was able to learn to produce more workers in the task of CollectMineralsAndGas. Compare with human player, both networks in some task only able to outperforms a casual human player, but seems very weak comparing to Grand Master player.

[3] There is some other research of RTS game in Glest. This game also has many different elements such as Technology and Magic. Players in this game need to collect resources, build different buildings and armies and fight the other team. The problem of current AI in this game is that they are not intelligent and once human player finds it weakness, it is easy to be defeated.

In their experiment, LSPI and SARSA was used as their RL methods. Firstly, they create their own score scheme, then they developed formulas for both SARSA and LSPI, they implement those methods in C++ as that was the language the game using. Next they test both methods on every 10 games training. And play games for 20 tests, in other words, totally 200 games are played. Since RTS games are usually required a long-time to play, 200 is a number enough in the experiment. They finally get a conclusion that using LSPI with SARSA sample performs best among using SARSA method or LSPI with random samples. They did not test the game with human player.

### 2.3.2 Bubble Shooter

[4] Bubble shooter is a puzzle game consist bubbles and a shooter. In the board there are different colors of bubbles. Play needs to control the shooter below those bubbles and find a proper angle to shoot a bubble of a color. Objective is to clean the bubbles in the board and the way to do that is to trigger the bubbles with a number of same colors to explode.

The RL agent play this game will need to figure out which angle maps the result that can eliminate the bubbles. This is similar in SC2 where the agent needs to choose a unit and do a mass production according to the enemy units.

A research applied the Atari Network in the agent. The board is also simplified to a 2D array of integers. A customer reward scheme was also implemented in the RL agent. The result was that the agent wins 95% of the game, which was better than a random agent. However, it does not outperform human player. The researcher believed that the time of training the agent is not enough.

### 2.3.3 Racing Game

[5] Racing game is a simple balancing game where player only needs to control the direction and the speed of his car in the racing track.

A research applied the deep RL in the agent where it only received raw screen image as input. The input was feed through the convolutional neuron network. Then the Q-learning method was applied in the agent. The result came out that the agent performed excellent actions in early game as it knows to turn when the far scenery changed, while in the late game as entering the desert, where the color of the road and the desert was too similar, the agent began to make mistakes and often crashed by hitting the side. The agent was programmed as to have human-level experience. However, the paper did not show any comparison with the human score.

One thing needs to mention is that in SC2, there will be some units that has clock-ability. A casual player will notice the dangerous as he sees the clock enemy when they are moving. However, the image information that has been through convolutional network could ignore the clock enemy as the color of the background is very similar to when it is occupied by a clock unit.

### 2.3.4 Flappy bird

[6] Flappy bird is a simple game where player only needs to click to make the bird flap in order to pass random generated barriers (pipe). The player need to predict the altitude when the bird passes the pipe. Too high or too low of the prediction will results in loss of the game

This game in SC2 sense could be that the agent decides the time to develop a technology, which should not be too early or too late, as discussed previously.

A research used two layer convolutional networks with the ReLu methods to process the image. The Q-learning off-policy method was applied in their RL agent. Replay memory was used to store the experience to enable the prediction in next game. However, the result turned out that with both loss functions the agent does not outperform than human player.

### 2.3.5 First Person Shooting Game: Doom

[7] FPS game such as Doom usually test players about their attention of the environment in the game. In FPS games, player has 3D experience as the game allows player to rotate the camera and point to an object and shoot. This makes the process of handling images for agent quite difficult, since the screen changes very often and it is hard for agent to recognize the patterns of the screen.

In SC2, there are similar situations, such as using the mini-map in the game for quick observation of an area in the map. When using the mini-map, the screen changes rapidly, professional player usually jumps between one screen to another to control the units and observing the maps.

In a research, 2 convolutional network layers were used in the architecture to process the image through multiple layers, the signal will then go through the recurrent neural network such as LSTM. Frame skip technique was used for the agent where the skip step was 4. Again, Q-learning method was used in the RL agent. The result turned out that the agent outperforms human players in both single-player and multiplayer situations.

### 2.3.6 Atari games

[8] Atari games such as Space Invader and Breakout usually have simple rules. Deep mind has developed an agent that can learn and play across those Atari games. However, this is very difficult to apply to SC2 since it has much more complex rules.

The research used 2 convolutional networks process the input then feed through the deep RL. TD-Gammon architecture was used in the learning process to update the parameters by estimating value function. Both deep Q learning method and Sarsa method are used in the agent and the results are compared between these methods. The agent was tested across 7 Atari games without any scripted program in any games. The result shows that in some games the agent outperforms human but other games the agent does not, however, DQN agent across all games performs better than SARSA agent.

# 3. Requirements and Analysis (1000)

## 3.1 Objectives

The objectives of this project are firstly, to apply the Reinforcement Learning to Starcraft2, and secondly to make the agent to be able to defeat human players.

The first task requires to design the architecture of the reinforcement learning agent in the game, then determine the learning methods to use in the reinforcement learning, design the code to perform the learning process. Analyze those methods and predict its outcome (e.g. what will be the score the agent will get after training for one day). Train the agent using the replay data and let it play the games itself. The result of the game play will be recorded to analysis and evaluate the agent. There could be some debugging sessions in case the code runs wrongly, which will be noticed if the result is not as expected.

The second task requires to evaluate the agent. Apart from judging the agent from the score, another way is to let the agent compare with human player and it is a more convincing. One way to do that is to compare the score the agent gets with the one that human can ever achieve, this way is cheap, we could simply search the internet for the data. Another way to evaluate the agent is to let it play with human player. This way is much expensive since it requires finding human volunteers and it has risk that the volunteers found may not be what we want or even worse, there is no volunteers.

## 3.2 Tools

Fortunately, the open source PYSC2 has provided some very useful tools for the reinforcement learning environment. It is written in Python and it uses the SC2 API and allows the control of the actions, observation of the maps and review of the replays in the game. It is written to be environment friendly to apply the RL methods for SC2. It also provides a base agent to be programmed and some mini-games to represents some tutorial functions in the game. Therefore, we only need to consider the parts of programming RL in the agent.

## 3.3 Problem Analysis

To design the architecture of the RL agent, we need to prepare the knowledge of the theory of RL. We need to understand the methods that are available as well as the rules of the SC2 in order to determine the methods for the agent. In this case, a full game of SC2 usually takes 10-20 min, and the break point of each episode is ambiguous, and the rules are very complex. Therefore, a combination of DP methods and Monte Carlo methods, which is the TD method should be used in the architecture.

To program the agent, we should understand the structure of the open source code. In this case, the environment of the reinforcement learning in StarCraft2 which has already built the convolutional layers and transformed the game state into spatial features (e.g. position of the mineral) and non-spatial features (e.g. the amount of mineral in the bank). There are also scripted actions for the agent to pick (e.g. select\_rect() to select an area of units or buildings.). Then we use the learning methods we picked to coding the agent.

We can initially design a sudo-code of the RL and determine its run-time, then we can write the code in python.

To train the agent, firstly we can test the agent in the small game such as DefeatRoaches and see if there is any glitch in the program. After the program is debugged, we could then train the agent in a super computer with both mini-game and full game.

To analyze results for those methods, we need to do some calculation of the values that we expect, which requires the knowledge of RL theory. We can also predict the results from other researches that use the same technique. After that we compare the results with expected one and see its performance. From the research that Deepmind previous did in SC2, we know that this game could be very complex for the agent to learn, and their agent achieved the level of a casual player. Refer to this, the result of our agent may achieve the same level.

We have discussed some methods to evaluate the agent after it completes the training session. When finding volunteer to test the agent, we should consider the master players that have been through some E-sport of SC2. We can watch some videos of E-sport competition of SC2 (e.g. ESL SC2) and do some research about the culture of Starcraft2. Then we will invite some of them as volunteer. Other people that has hobby of playing SC2 can also be considered if they are enthusiastic. Then we will let them play the game with our agent and ask them to send some feedback. The questionnaire to determine how good is the player will include some questions such as the number of minerals that the players are able to collect in some amount of time and etc.

# 4. Progress (500)

The project is now currently on the theory part, where all the theories of reinforcement learning and the rules of Starcraft2 are prepared. Other similar researches of RL are observed and analyzed. Below shows the architecture of the agent in the Starcraft2 and the plan of future working represented in Gantt chart in the next chapter.

## 4.1 Architecture

Figure 1. Architecture of PYSC2

Figure 1 shows the architecture of the whole game.

Screen

Mini-map

Non spatial features

State, Available actions

agent

game

reward

The game consists a screen, mini-map and some non-spatial features.

Screen: the current screen of the game that player stays on.

Mini-map: a summary of the whole map, indicates resources position, enemy position.

Non spatial features: such as the amount of the mineral and gas, total support, etc. which might limit the available actions.

Then the agent using RL technique to choose an action in the game.

The game then produces the reward according to the action taken by the agent.

Here is the diagram that shows the detail structure of the agent.

Figure 2 Detail architecture of the agent

Critic

Evaluation

Error

Actor

policy

reward

State

Game

Action

## 4.2 Parameters setting

In PYSC2 environment, all the work including the input and connection to the game as well as a base agent has been provided. Thus we only need to program the RL technique for the agent.

The architecture of the agent uses an Actor Critic method. Below is the detail of each part.

Policy: -greedy function can be used initially in the policy with softmax method. It will be improved through the experience after playing the game.

Evaluation: we can use the same function as in Sarsa that updates the action values, which will then produce an error to indicate the positive or negative effect of the current policy. This error will then go into the Actor and directs how the policy will be update.

Model: action space is large because it consists spatial features, previous solution was to sample some typical space in the action, which reduce the action space to a size of 300. Another solution can be that using the relations between the buildings and units, thus it stores the relative position, and the target position. This model can reduce the action space and enables the fast computing.

# Conclusions and Plan (500)

We have research what is reinforcement learning and how can it be apply to Starcraft2. We know RL is a powerful tool in artificial intelligence. It learns from rewards and improve itself. After doing loans of training, RL may be able to outperform human level. We have walked through the rules of the Starcraft2 and consider the problem that we might face in the implementation stage. We know Starcraft2 is a typical RTS game which has lots of elements in the game. The challenge for applying RL in SC2 is that SC2 has lots of states and available actions as well as possible strategies. It might require a super computer large amount of time to train the agent. We have also designed a structure of the agent and the methods that we can used in RL. We filled our agent with e-greedy policy, SARSA method to evaluate the policy. We have analyzed the pros and cons among DP, MC and TD method. We have also designed a possible model that can be used to reduce the action space.

Below is the Gantt chart of the implementation part of this project.





# Reference

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning An Introduction.* Cambridge London, England: The MIT Press, 1998.

[2] Blizzard Entertainment. “Starcraft2 Official Game Site” in *Game Overview,* 2017. [Online]. Available: https://starcraft2.com/en-gb/game. Accessed on: November 11, 2017.

[3] V. K. Dimitriadis, *Reinforcement Learning in Real Time Strategy Games Case Study on the Free Software Game Glest.* Chania: V. K. Dimitriadis, September 2009. [online] Available: <http://www.intelligence.tuc.gr/lib/downloadfile.php?id=347>

[4] L. Samson, *Deep Reinforcement Learning applied to the game Bubble Shooter.* Amsterdam: Science Park, June 24, 2016. [online] Available: <https://staff.fnwi.uva.nl/b.bredeweg/pdf/BSc/20152016/Samson.pdf>

[5] P. Lopes, "Deep Reinforcement Learning: Playing a Racing Game - Byte Tank", *Byte Tank*, 2017. [Online]. Available: <https://lopespm.github.io/machine_learning/2016/10/06/deep-reinforcement-learning-racing-game.html>.

[6] N. Appiah and S. Vare. *Playing FlappyBird with Deep Reinforcement Learning,* n.d. [online] Available: <http://cs231n.stanford.edu/reports/2016/pdfs/111_Report.pdf>.

[7] G. Lample and D. S. Chaplot, *Playing FPS Games with Deep Reinforcement Learning,* n.d. [online] Available: <http://www.cs.cmu.edu/~dchaplot/papers/aaai17_fps_games.pdf>.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, *Playing Atari with Deep Reinforcement Learning,* December 19, 2013. [online] Available: <https://arxiv.org/pdf/1312.5602v1.pdf>

[9] J. Landy, "Deep reinforcement learning, battleship – EFavDB", *Efavdb.com*, 2017. [Online]. Available: http://efavdb.com/battleship/.

[[10]](file:///U:/Dissertation%20Project/sc2le.pdf) O. Vinyals, T. Ewalds, K. Calderone, P. Keet, et al. *StarCraftII: A New Challenge for Reinforcement Learning,* n.d. [online] Available: <https://deepmind.com/documents/110/sc2le.pdf>.

# Appendices