Reinforcement Learning for Real-Time Strategy games

Preprint · January 2019

DOI: 10.13140/RG.2.2.14346.11204/1

2 authors:

Maximilien Germain
20 PUBLICATIONS **244** CITATIONS
SEE PROFILE

Othmane Marfoq
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Reinforcement Learning for Real-Time Strategy games

Maximilien GERMAIN
MVA ENS Paris-Saclay
maximilien.germain@ens-paris-saclay.fr

Othmane MARFOQ
MVA ENS Paris-Saclay
othmane.marfoq@ens-paris-saclay.fr

Abstract

This project aims to offer an overview of the machine learning methods developed for Real-Time Strategy games (RTS). More precisely, it focuses on Reinforcement Learning (RL) techniques and presents the challenges faced when one wishes to provide successful AI algorithms for this type of games. A comparison with board games is conducted to show why it is not possible to directly transpose in this new setting the methods designed to solve them. The current state of the art models for RTS games and their performances are presented.

1 Introduction

The recent success of Reinforcement Learning (RL) in learning to play board games such as Go or chess asks the question of the extension of these techniques to more complex adversarial situations. Even though Go was considered as an highly difficult game to be mastered by a program, Deep RL approaches used in AlphaGo (combined with supervised learning) offered an efficient and promising solution for the training of powerful AIs. Indeed, it would be interesting to apply RL algorithms to real-world interactions and gain insight into the strategies to use in large scale economy challenges or diplomatic decision-making.

However, in order to tackle these infinitely more involved challenges, a first less ambitious but necessary step could consist in extending the RL methods to even more complex games. That's why Real-Time Strategy games offer an encouraging playground to progress towards this direction. We will review the difficulties faced when one wants to design a human-level bot in RTS games. One guideline will be the game StarCraft, an example of such games studied in several RL papers as a test environment for their developed algorithms. Then, we will present the state of the art methods used to perform tasks in StarCraft and compare there performances. Finally we will point out the remaining open questions to be investigated by future research in the area.

2 Challenges emerging from RTS games

2.1 Differences with board games

Real-time Strategy is a sub-genre of strategy video-games where the game does not progress incrementally in turns but rather online. The main purpose of the game is to gather resources, to research technologies and to train units in order to build a military power able to destroy the opponent's base. This type of games is quite different from board games (such as Go or chess) on many aspects and we should first notice that there is a gap between designing a human-level bot for traditional board games and one for Real-Time Strategy games. Indeed, to achieve this objective, one must face several additional challenges:

- The game is a real-time game, it means that players don't take turns to play, but may instead simultaneously perform actions in an approximately continuous flow of time (24 frames per second for StarCraft). Thus, players need to act fast and should also take into consideration the fact that actions may take a certain time to be realized. For instance, creating units is not instantaneous but may take a few seconds or minutes.
- In opposition with classic board games like Go and chess, RTS games are in general partially observable. This lack of observability is refereed to as the *fog-of-war*, which obscures areas of the map that are out of sight of the controlled units. Therefore, RTS games can be qualified as a game of imperfect information, like Poker. These games have been studied by several authors, in particular by *Moravcík et al.* who introduced in [2] DeepStack, an algorithm which deals with the imperfect information setting of poker.

  Another difficulty that one may face in most RTS games is that the actions are not necessarily deterministic: there is a stochastic parts in the state transitions. However, this wouldn't be a big challenge for methods based on reinforcement learning or MDPs which already take into account this uncertainty.
- RTS games are quite complex in comparison with other games. They have much bigger state spaces and more possible actions during each decision cycle. Let's take a look at the state space size for few usual board games. For chess it is estimated to 10^{50}, for Go to 10^{170} and to 10^{80} in the case of heads up no-limit Texas holdem poker. However, it is estimated to more than 10^{1685} for StarCraft, which asks for the design of methods adapted to this new scale.

  One can also estimate the complexity of a game by means of its branching factor. It corresponds to the number of possible actions a player can choose to perform at any moment. If n actions can be chosen individually for 200 units, the branching factor is n^{200}. For Starcraft, the branching factor is estimated in [3] between 10^{50} and 10^{200}. In chess, the branching factor is near 35, and around 180 for Go.
- Both partial information setting and the need for long-term strategies makes it challenging for a model to directly understand the correlation between actions and rewards. Indeed, some of the rewards, associated with quick actions, are immediate, whereas the effects of global strategies can sometimes be seen only in the late game. These possibly delayed rewards in a huge state space are quite difficult to tackle.

The differences mentioned above make RTS games more difficult and more challenging to automate in comparison with other games like chess or Go. For instance, standard adversarial planning approaches, such as game tree search cannot be directly applied (see [3]). Nevertheless as we will see thereafter, papers like [22,23] use instead Monte Carlo Tree Search (MCTS) methods (as in AlphaGo) to play RTS .

2.2 Tasks classification

A natural way to divide RTS games, used by players relies on considering two different tasks:

- *micro-management* : It is the lowest level of abstraction, which includes controlling the units individually. It relates especially to battle tactics: moving units, firing, targeting, fleeing, kiting. Therefore, it often requires to make quick decisions at unit level. A typical example is the need to react to a local attack of your troops.
- *macro-management* : It is the highest level of abstraction, which corresponds to the ability of defining and applying a winning strategy. This strategy relies on choosing the proper units to build but also on collecting information and gathering resources.

Research papers sometimes distinguish between even more specific tasks. It refines the previous division in smaller parts: strategy, tactics, reactive control, terrain analysis and intelligence gathering. These tasks are almost all part of Macro-management except reactive control and a part of tactics which correspond to micro-management.

In their work [3] in 2013, *S. Ontañon et al.* presented a survey on RTS games. They pointed out the challenges faced when building a bot able to play such a game, gave a presentation of RTS games and described the state of the art bots for StarCraft at the time. They noticed that the performance of top ranked bots at the time depended on carefully handcrafted and non-adaptive behaviors, rather than on on-line decision making procedures. They also presented a list of problems unsolved at the time (Learning and adaptation, Planning, Integration and domain knowledge).

In the next sections, we will present state of the art methods that addresses some of those unsolved problems, and also methods able to address tasks coming from each level of abstraction (Strategy, Tactics, micro-management).

3 State of the art methods

Some usual baseline methods used in Reinforcement Learning papers are DQN and REINFORCE algorithms. Q-learning algorithms learn action-value functions thanks to ε-greedy exploration and by solving the Bellman equation associated with the MDP. When choosing the form of Q functions as deep neural networks we obtain DQN methods. This technique was introduced in [21] by the DeepMind team. With it, they were able to build an human-level AI for some Atari games. Once the action-value is approximated, one can then use it to determine an optimal policy.

REINFORCE algorithm is a policy-gradient algorithm which focuses on finding an optimal policy regardless of the value function. It is also possible to combine a method learning the value function with another method computing the optimal policy. It was used for AlphaGo [24]. We will now present Reinforcement Learning methods especially introduced for RTS games.

3.1 Monte Carlo Tree Search for tactics

The main idea behind Monte-Carlo Tree Search (MCTS) is to efficiently sample the state space for performing exploration-exploitation. It can even be applied to large state spaces after a first step of abstraction of the game state. This is done in [22] by regrouping actions into smaller categories, but also regrouping single units in squads, which significantly reduces the branching factor: from higher than 10^{100} to 10^{10}. The MCTS algorithms explore a tree made of game states with two complementary methods: a tree policy designed to select what next state to explore and a default policy whose goal is to simulate actions (according to some probability distribution) from the selected node to a terminal state. This allows to evaluate the value function on interesting states.

Paper [22] of Ontañón introduced as a tree policy the informed ε-greedy sampling. This method relies on a Multi-Armed Bandits (MAB) setting considering a prior distribution on the rewards (like in AlphaGo [24] which uses a UCB type algorithm). Another interesting aspect from his work is the use in [23] (Uriarte, Ontañón) of game replays from professional players to construct this bayesian prior distribution.

In [26], Ontañón uses Combinatorial Multi-armed Bandits (CMAB) for the same tasks. The CMAB and the associated naïve sampling technique are introduced in [8]. CMAB are a version of MAB with several variables. At each iteration, one has to select a "legal" collection of arms, each of them being related to one variable. Naïve sampling consists in the approximation of the reward by the sum of rewards functions only depending on one variable. Ontañón shows that for high branching factors, like in go and RTS games, naïve sampling perform better than other techniques.

3.2 Dealing with micromanagement

Micromanagement tasks were explored in 2016 by *Nicolas Usunier et al.*, who have proposed a reinforcement learning algorithm called zero-order gradient. In their paper [6] they have proposed several *micromanagarial* tasks of type **mXvY** or **wXvY**, which are tasks where we control **X** Marines (ranged ground units) or Wraiths (ranged flying units) and the opponent has **Y** of the same units. The difficulty of this tasks is the fact that the units should take actions that depend on each others.

To solve this problem, the authors used a greedy MDP, in order to be able to choose units one by one, knowing the commands that was chosen by other units. They have also proposed a raw state information and a list of 17 features used in order to describe the states of the greedy MDP (unit type, unit position, unit health-points, target position...). They have shown trough different experiences that the method they propose, Zero Order (ZO), gives better performances in comparison with other state-of-the-art methods like reinforce and policy gradient, that have proven their efficiency in solving other board games like Go and chess. They also shown that ZO outperforms some classical hard-codded strategies like concentrating fire on the closest enemy or on the weakest one. Moreover, this method is able to generalize, i.e. an agent trained to do a specific task on a specific map can also perform new tasks on new maps which it wasn't directly trained on.

Micromanagerial learning was also explored by *Kun Shao et al.*, in [28] they have proposed reinforcement learning and curriculum transfer learning based method for StarCraft Micromanagement. They have proposed a more concise state representation, their representation in opposition to the one proposed in [6] doesn't depend on the number of units, and takes into consideration the current and the last state and also the last action. Each state contains information about the distances, and the units hit-points, in total this state representation is embedded in 93 dimensions, while the state representation in [6] contains 17 dimensions per unit. *Kun Shao et al.* formulated SC micromanagement as multi-agent RL model, and used a parameter sharing multi-agent gradient descent Sarsa(λ) (MAGDS)(Algorithm 1.) to train the model. They have also proposed a reward function based on the damage and the hitpoints of every agent and its enemies, in order to face the problem of sparsity and delay in micromanagement. They have trained their model to perform tasks similar to the ones proposed by *Nicolas Usunier et al.*. And showed that their method outperforms rule-based and other DRL (Zero Order [6] and BicNet [29]) approaches on the task of Marines vs. Zerglings and Goliaths vs. Zealots.

3.3 State estimation in the partial information setting

Reinforcement Learning approaches require to be aware of the state of the studied system. In RTS games, it corresponds to the count and position of all units, together with their internal states (such as current action or energy...). Then, as far as RTS games are partially observable, one needs to perform an estimation of the enemy state in order to treat value functions and then compute the best actions to make. We will focus on the localization and count of the enemy forces, without trying to value internal states of enemies. Overcoming the *fog-of-war* can therefore be a worthy strategic task because it allows to modify one's strategy according to the opponents units movements. Several papers investigated this question by designing defogger systems.

Papers [4] of (Synnaeve et al) and [13] from (Kahng et al) both construct encoder-decoder architectures to achieve this task. When [13] only uses Convolutional Neural Networks (CNN), which were proven efficient for computer vision tasks, [4] compares the performances of CNN and Convolutional-LSTM (CL). Their study shows that CL is more efficient overall. Indeed, the LSTM networks takes into account the temporal structure of the observations thus provide results which turn out to be more adapted to the units movements through time. When providing intelligence from this defogger to an existing bot, it appears that it helps the building actions module but can affect the tactics unit. In fact, errors in the position estimation of the enemies can push the tactics system into moving troops to a wrong area of the map. This behavior takes a lot of time to correct if one wishes to make the troops come back to their previous location. However, overall, the defogger model can quite accurately predict how many units are present on the battlefield, which represents valued intelligence data for strategy and tactics.

3.4 Predicting opponents actions and productions

One way to be able to choose strategies and actions in an adaptive way, is by predicting the opponents actions, several papers explored this area and have worked on uncertainty and opponents behaviour. Weber and Mateas in [17] proposed a data mining approach to opponent modeling in RTS, and applied machine learning learning algorithms to the task of detecting an opponent's strategy before it is executed and predicting when an opponent will perform strategic actions. In [18] Ethan Dereszynski et al. explored probabilistic behavior on high-level strategic models, they encoded the game states as a Hidden Markov Model, and learned this model using collections of game logs, the advantage of their work in comparison with [17], is the fact that they take in account the knowledge of the timing of observed events.

Marius Stanescu and Michal Certicky [15] used Answer Set Programming to predict the number and type of units a StarCraft or WarCraft III player trains in a given amount of time. And showed that their method had good performance in 10th to 20th minute of game, for predictions of length 1 and 3 minutes, which is the most useful case for prediction, *Marius Stanescu et al.* proposed in [16] a Bayesian model to predict the outcome of battles, and what units are needed to win a battle given the opponents army composition. However, they have worked with a simulated data set, and assumed that units have independent contributions to the battle outcome.

3.5 Learning from observations

S. Ontañon et al. mentioned in [3] that the problem of learning by observing game play of other players was an unsolved problem. Several works explored this field, for example *Niels Justesen* and *Sebastian Risi* used deep neural networks to learn macro-management tasks directly from replays [9]. They have trained a 4 layers fully connected neural network taking as input a 210 dimension feature vector and outputs a 58 dimensional vector representing the probability of producing each of the Protoss buildings. The input vector gives an idea about the games' state and contains information about the number of each type of units in building or already belt, The number of each upgrade and technology type researched in the game by the player, The number of supply used by the player...

To train their network *Niels Justesen* and *Sebastian Risi* used a data set of $789,571$ state-action pairs extracted from $2,005$ replays of Protoss versus Terran games. Using this architecture they reached a top-1 error, a top-3 error and a top-10 error of 54.6%, 22.92% and 4.03%. They have also integrated this trained neural network in the UAlbertaBot, which is one of the best StarCraft bots, by replacing its hard-coded strategy that decides the units to build and the technologies to upgrade, by the trained neural network which predicts the next action (or strategy) given the state of the game. By doing that the obtained bot is able achieve a win rate of 68% against the games built-in Terran bot. while the original version of UAlbertaBot achieves a in 100% win rate.

For the same task, deciding what is the proper build order to express, the authors of [1] build a LSTM network whereas the paper [9] doesn't take into account the sequentiality of the data. Indeed, it only considers state-action pairs. Paper [9] doesn't achieve good results during games as far as it doesn't make use of the RL setting but only consists in supervised learning. Nevertheless, it could be applied to pretrain the RL models in early game. In comparison, [1] directly learns the value of the Q-function by off-policy training and then improves its accuracy online. We will talk more about RL models in the next section.

3.6 Reinforcement Learning for macromanagement

As mentioned in several papers ([3],[19]), the current top performing bots to play RTS games such as StarCraft still rely mainly on hard-coded strategies. Thus, Even the most advanced StarCraft bots (ranked between D and D+) are outperformed by professional human players(ranked between A- and A+) , or even amateur players (ranked between C+ and B). The main cause behind this gap, is the fact that humans are very adept at recognizing scripted behaviors and exploiting them to the fullest. To solve this problem, bots need to both recognize the strategy and intent of an opponent's actions, and how to effectively adapt its own strategy to overcome them.

In order to face this problem, *Sijia Xua et al.* proposed in [19] a deep reinforcement learning method for macro action selection, using an approach based on a combination of Ape-X DQN with LSTM. They used a Partially Observable Markov decision process (POMDP) to model the macro-management decision process. The state space is similar to the one used in [9], the set of possible actions contains 54 macro actions (building, upgrading, different types of attack). They also employed the score provided by the game engine to define the reward function. Because of the sparsity of the rewards, classical Deep Q-networks (DQN) can't be useful for learning. Thus, the authors of [19] used *Ape-X DQN*[20] which is a distributed architecture for Deep Reinforcement Learning at scale. Ape-X is composed from two parts: an *acting* part that explore the environment, evaluate a policy implemented as a deep neural network, and store the observed data in a replay memory. and a *learning* part responsible of updating the policy parameters. Using *Ape-X DQN* consists in training a neural networks representing the q function, and taking as input the current state by minimizing the loss function $\frac{1}{2}\left(G_t - q(S_t, A_t, \theta)\right)^2$ with

$$G_t = \lambda^n q\left(S_{t+n}, \operatorname{argmax}_a q(S_{t+n}, a, \theta), \theta^-\right)$$

Sijia Xua et al. have trained there bot via making it play against AIIDE 2017 StarCraft Competition bots. And have achieved a 83% win-rate, outperforming 26 bots in total 28 entrants.

Although those results are good, there are several improvement perspectives, we can for example train the bot by making it play against it self, or against more bots. We can maybe use curriculum learning to help the learning of the bot, by designing enemy bots with increasing level of difficulty

4 Most advanced Starcraft bots

New machine learning and RL methods for RTS games are yearly tested during Starcraft competitions. The report [27] presents the results of Starcraft AIIDE 2018 Competition. It also describes the teams and the bot they developed. During the event, 34 694 games were performed to confront the algorithms. For the first time, amateurs have been outperformed by teams from two major tech companies. On the one hand, the bot ranked second, CherryPi, is designed by Facebook team leaded by Gabriel Synnaeve. On the other hand, the winner bot, called SAIDA, is Samsung's AI. Other bots are designed by amateurs or university teams.

SAIDA is based on UAlberta bot and also uses a CNN Encoder-Decoder algorithm to decide when to fight. In the previous section, we spoke about the efficiency of this architecture type for the defogging problem. It appears to be also efficient with other tasks. Furthermore, "The secret of our success was in our AI bot. It plays as if it is human by actively responding to the counterpart's strategies, then striking them at the right time while allowing detailed unit control. We applied machine learning technologies to 110,000 sets of pro-gamers' replay data in order to further enhance competitiveness," said SAIDA team leader, Chang-hyeon Bae. This shows the effectiveness of learning from observations and how it could able StarCraft bots to have better performance. CherryPi has also an interesting ability which is the selection of its strategy over a dozen of them, in connection with the previous games it played against its current opponent.

Bot	Games	Win	Loss	Win	Affiliation
SAIDA	2390	2298	92	96.15	Samsung
CherryPi	2392	2173	219	90.84	Facebook AI Research
CSE	2391	2113	278	88.37	Independent
Locutus	2386	2000	386	83.82	Independent
DaQin	2390	1830	560	76.57	Independent

Table 1: Starcraft AIIDE 2018 Competition top 5 results

5 Open questions

As illustrated in this review project, several problems in RTS were explored and solved in the last few years thanks to advances in the field of reinforcement learning. For instance, several papers proposed methods able to efficiently deal with micromanagement tasks in StarCraft, and achieve results superior to hard-coded strategies. However, there are still many other problems which require improvements :

- **Adaptation to opponent strategy**: Predicting opponents actions and strategies was explored by different papers ([15], [16], [17], [18]) but none of them was able to predict accurately the strategy of the enemy in real time. Furthermore, no one explored a method to adapt the current strategy in order to counter the opponents strategy after its prediction.
- **Integration**: RTS bots use specialized units relatively independent from each other. Therefore it is a tremendous challenge to create a global algorithm exploiting the whole capacity of each and every one of the different functions. Papers like [4] pointed out that providing new intelligence to a bot can help some of its modules but harm the performances of others. It thus remains to understand how to design more reliable and general integration techniques.
- **Strategies**: Even in state of art architectures, the admissible strategies are hard-coded and most Reinforcement Learning macro or micro management methods learn which one to select. However, we don't have yet a method producing new strategies by itself. Indeed, in

computer vision, research made the handcrafted features used to describe images evolve to deep learning automatically learned features. This could call for future work to rely less on "human" strategies and behaviors.

- **Reward function delay and sparsity:** The main purpose in StarCraft is to destroy opponent's units, but if the reward is only based on the final result, it would be sparse and will suffer from a large delay. To tackle this problems, several papers have designed some new simple reward functions. However, methods for solving delayed and sparse reward like Hierarchical RL weren't explored enough in the field of RTS games.
- **State representation :** As mentioned in [28], state representation in RTS is still an open problem. Different hand coded state representations were proposed either in [28] or [6], thus learning a state representation can be a subject for future research.

6 Conclusion

In this review project, we have explored Real Time Strategy Games, with a focus on WarCraft. We have seen that RTS games are more complex and more difficult to solve in comparison with board games, like chess or AlphaGo (larger state-action space, real-time and partially observed setting...). In order to design bots able to play RTS games, the gameplay is divided into partial tasks : *micro-management* and *macro-management*, map exploration and opponent prediction... We have listed and compared state of the arts methods dealing with those partial tasks: Monte Carlo Tree Search, Zero Order, Parameter Sharing Multi-Agent Gradient-Descent Sarsa(λ), State estimation using defogger architecture, Ape-X DQN + LSTM. We have also explored some supervised learning techniques taking advantage of previous observations, especially with replays from professional players. We have finally compared state-of-the-art StarCraft bots by presenting their performance on StarCraft AIIDE 2018 Competition.

Despite the advances in the field of Reinforcement Learning, several problems in RTS games remain unsolved (Adaptation to opponent strategy, State representation...). Thus, human professional players still outperforms the most advanced StarCraft bots. Finally, RTS games represent a challenging problem, and solving it would be a first step towards the objective of solving more complex problems like road traffic, finance, or weather forecast.

References

[1] J. Gehring, D. Ju, V. Mella, D. Gant, N. Usunier, and G. Synnaeve. High-level strategy selection under partial observability in starcraft: Brood war. CoRR,abs/1811.08568, 2018

[2] M. Moravcíkk, M. Schmid, N. Burch, V.Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. H. Bowling. Deepstack: Expert-level artificial intelligence in no-limit poker. CoRR, abs/1701.01724,2017.

[3] S. Ontañon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. IEEE Transactions on Computational Intelligence and AI in games, 5(4):1–19, 2013.

[4] G. Synnaeve, Z. Lin, J. Gehring, D. Gant, V. Mella, V. Khalidov, N. Carion, and N. Usunier. Forward modeling for partial observation strategy games - a starcraft defogger. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 10761–10771. Curran Associates, Inc., 2018.

[5] G. Synnaeve, N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier. TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games.working paper or preprint, Dec. 2018.

[6] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. CoRR, abs/1609.02993, 2016.

[7] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. P. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing. StarCraft II: A new challenge for reinforcement learning. CoRR, abs/1708.04782, 2017.

[8] Ontañón, Santiago. "The Combinatorial Multi-Armed Bandit Problem and Its Application to Real-Time Strategy Games." AIIDE (2013).

[9] Justesen, Niels & Risi, Sebastian. Learning macromanagement in starcraft from replays using deep learning. IEEE Symposium on Computational Intelligence and Games (CIG), 2017.

[10] Vineet Padmanabhan, Pranay Goud, Arun K Pujari, Harshit Sethy. Learning in Real-time Strategy Games. 14th International Conference on Information Technology, 2015

[11] Justesen, Niels & Bontrager, Philip & Togelius, Julian & Risi, Sebastian. Deep Learning for Video Game Playing. 2017

[12] Stanescu, Marius & Barriga, Nicolas A. & Hess, Andy & Buro, Michael. Evaluating Real-Time Strategy Game States Using Convolutional Neural Networks. IEEE Symposium on Computational Intelligence and Games (CIG), 2016.

[13] Kahng, Hyungu & Jung, Yonghyun & Sang Cho, Yoon & Ahn, Gonie & Park, Young Joon & Jo, Uk & Lee, Hankyu & Do, Hyungrok & Lee, Junseung & Choi, Hyunjin & Yoon, Iljoo & Lee, Hyunjae & Jun, Daehun & Bae, Changhyeon & Bum Kim, Seoung. Clear the Fog: Combat Value Assessment in Incomplete Information Games with Convolutional Encoder-Decoders. 2018

[14] Alexey Dosovitskiy and Vladlen Koltun. Learning to Act by Predicting the Future, ICLR 2017

[15] Marius Stanescu and Michal Certicky. Predicting Opponent's Production in Real-Time Strategy Games with Answer Set Programming. IEEE Transactions on Computational Intelligence and AI in Games · October 2014

[16] Marius Stanescu, Sergio Poo Hernandez, Graham Erickson, Russel Greiner, and Michael Buro. Predicting army combat outcomes in StarCraft. In Ninth Artificial Intelligence and Interactive Digital Entertainment Conference, 2013.

[17] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in IEEE Symposium on Computational Intelligence and Games (CIG), 2009.

[18] E. Dereszynski, J. Hostetler, A. Fern, T. D. T.-T. Hoang, and M. Udarbe, "Learning probabilistic behavior models in real-time strategy games," in Artificial Intelligence and Interactive Digital Entertainment (AIIDE), AAAI, Ed., 2011.

[19] Sijia Xua, Hongyu Kuangb, Zhi Zhuanga, Renjie Hua, Yang Liua, Huyang Sun. *Macro action selection with deep reinforcement learning in StarCraft*. 2018

[20] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. arXiv preprint arXiv:1803.00933, 2018.

[21] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518, 529–533.

[22] Ontañón, Santiago. (2016). Informed Monte Carlo Tree Search for Real-Time Strategy games. 1-8. 10.1109/CIG.2016.7860394.

[23] Alberto Uriarte and Santiago Ontañón. "Improving Monte Carlo Tree Search Policies in StarCraft via Probabilistic Models Learned from Replay Data." AIIDE (2016).

[24] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel & Demis Hassabis. Mastering the game of Go without human knowledge. Nature volume 550, pages 354–359 (19 October 2017)

[25] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, Rodney Tsing. StarCraft II: A New Challenge for Reinforcement Learning. arXiv:1708.04782v1

[26] Ontañón, Santiago. Combinatorial Multi-armed Bandits for Real-Time Strategy Games. JAIR (58) 2017

[27] David Churchill, Richard Kelly. 2018 Starcraft AI Competition Report and Results, Memorial University of Newfoundland

[28] Kun Shao, Yuanheng Zhu, Member, IEEE and Dongbin Zhao, Senior Member, IEEE, StarCraft Micromanagement with Reinforcement Learning and Curriculum Transfer Learning, arxiv 1804.00810 2018

[29] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets for learning to play StarCraft combat games," arXiv preprint arXiv:1703.10069, 2017.