electronics

MDPI

Article

Spellcaster Control Agent in StarCraft II Using Deep Reinforcement Learning

Wooseok Song, Woong Hyun Suh and Chang Wook Ahn *

Artificial Intelligence Graduate School, Gwangju Institute of Science and Technology (GIST), Gwangju 61005, Korea; wooseok503@gm.gist.ac.kr (W.S.); woong.hyun.suh@gm.gist.ac.kr (W.H.S.)

* Correspondence: cwan@gist.ac.kr

Received: 11 May 2020; Accepted: 11 June 2020; Published: 14 June 2020

Abstract: This paper proposes a DRL -based training method for spellcaster units in StarCraft II, one of the most representative Real-Time Strategy (RTS) games. During combat situations in StarCraft II, micro-controlling various combat units is crucial in order to win the game. Among many other combat units, the spellcaster unit is one of the most significant components that greatly influences the combat results. Despite the importance of the spellcaster units in combat, training methods to carefully control spellcasters have not been thoroughly considered in related studies due to the complexity. Therefore, we suggest a training method for spellcaster units in StarCraft II by using the A3C algorithm. The main idea is to train two Protoss spellcaster units under three newly designed minigames, each representing a unique spell usage scenario, to use 'Force Field' and 'Psionic Storm' effectively. As a result, the trained agents show winning rates of more than 85% in each scenario. We present a new training method for spellcaster units that releases the limitation of StarCraft II AI research. We expect that our training method can be used for training other advanced and tactical units by applying transfer learning in more complex minigame scenarios or full game maps.

Keywords: deep reinforcement learning; A3C; game AI; StarCraft II; spellcaster; minigame

1. Introduction

Reinforcement Learning (RL), one of the nature-inspired computing algorithms, is a powerful method for building intelligent systems in various areas. In developing such systems, Artificial Intelligence (AI) researchers often examine their algorithms in complex environments for the sake of prospective research applying their models in challenging real-world settings. In doing so, solving complex game problems with Deep Reinforcement Learning (DRL) algorithms have become a common case. DeepMind unraveled Atari-breakout with DQN [1], for instance, and OpenAI showed significant performance in the Dota2 team play with PPO [2]. Particularly, Real-Time Strategy (RTS) games have emerged as a major challenge in developing real-time AI based on DRL algorithms.

StarCraft II, one of the representative games of the RTS genre, have become an important challenge in AI research. It is supported by substantial research community—AI competitions on StarCraft II; e-sport fans vitalizing the StarCraft II community; and open Application Program Interfaces (APIs) for researchers to manipulate actual game environments. Many related and advanced research have been published since DeepMind's work on creating AI players in StarCraft II using RL algorithms [3]. Especially, training an agent to control units in real-time combat situations is an active as well as a challenging area of RL research in StarCraft II due to its complexity. It is because the agent has to decide over thousands of options in a very short period of time. Unlike other turn-based games, such as Chess or Go, an agent playing StarCraft II is unable to obtain the entire information of the game's action space, and this feature hinders an agents' long-term planning of a game.

1.1. Related Works

Recent AI research in StarCraft II has diversified its extent from macroscopic approaches, such as training entire game sets to win a full game, to microscopic approaches, such as selecting game strategies or controlling units [4–6]. Several studies have partially implemented some detailed controls over units (often referred to as 'micro') in combat. Although spellcaster units are crucial components for winning the game especially in the mid-late phase of game combats, there is no research on controlling the spellcasters. When spellcaster units cast spells, they have large amounts of options to consider such as terrain, enemy, and ally information. For this reason, effectively casting spells during combat is generally thought of as a difficult task for AI players. As StarCraft II is an RTS game, however, it is challenging to apply the algorithms used in turn-based games directly into StarCraft II. Players of StarCraft II have to deal with various sub-problems such as micro-management of combat units, build order optimization of units and constructions, as well as efficient usage of mineral resources.

There have been various research works approaching to solve the difficulties of developing StarCraft II AI. First, there are methods that train Full-game Agents. Hierarchical policy training which uses macro-actions won against StarCraft II's built-in AI in a simple map setting [7]. Self-playing pre-planned Modular Architectures method showed a 60% winning rate against the most difficult built-in AI in ladder map settings [8]. Deepmind produced Grandmaster-level AI by using imitation learning through human's game replay data by self-playing within Alphastar League [9].

Some works have suggested training micro-control in specific situations. Micro-control means to control units during combat. For instance, several works have been conducted on training with DRL to control units and applying transfer learning to employ the training results in game situations [9–11]. The minigame method was first introduced by DeepMind. The method focuses on scenarios that are placed in small maps constructed for the purpose of testing combat scenarios with a clear reward structure [3]. The main purpose of minigame is to win a battle against the enemy via micro-control. DeepMind trained its agents using 'Atari-Net', 'FullyConv', and 'FullyConv LSTM' in the minigame method and showed higher scores than other AI players. By applying the Relational RL architectures, some agents of the minigame method scored higher than that of Grandmaster-ranked (the highest rank in StarCraft II) human users [10].

Some other studies have worked on predicting the combat results. Having knowledge of the winning probability of combat based on the current resource information, in advance, assists AI's decision makings whether one should engage the enemy or not. In StarCraft I, the prediction was done by a StarCraft battle simulator called SparCraft. The limit of this research is that units such as spellcasters are not involved due to numerous variables [12]. There is a battle simulator that takes one spellcaster unit into account [13]. However, considering only one spellcaster among many other spellcaster units is insufficient to give precise combat predictions.

Lastly, StarCraft II AI bots do not consider spell use significantly. For instance, the built-in AI in StarCraft II does not utilize sophisticated spells, such as 'Force Field'. TStarBot avoided the complexity of micro-control by hard-coding the actions of units [14]. However, this approach is not applicable to spell use in complicated situations and good performance is not guaranteed. Also, SparCraft, a combat prediction simulator, does not include spellcasters in combat prediction because the prediction accuracy was poor when considering spellcasters [12]. One of the related studies integrated its neural network with UAlbertaBot, an open-source StarCraft Bot, through learning macro-management from StarCraft replay; however, advanced units such as spellcasters were not included [15]. In this respect, research on the control of spellcaster units is needed in order to build improved StarCraft II AI.

1.2. Contribution

Among the aforementioned studies, examinations on spell usage of spellcaster units are mainly excluded. Although spells are one of the most important components in combat, most StarCraft II AI do not consider spell use significantly. This is because spellcaster units are required to make demanding decisions in complex situations. It is difficult to precisely analyze situations based on

given information or to accurately predict the combat results in advance. Otherwise, inaccurate use of spells could lead to crucial defeat. Combat in StarCraft II is inevitable. To win the game, a player must efficiently control units with 'micro', appropriately utilize spellcaster units, and destroy all the enemy buildings. In the early phase of a game, micro-controls highly affect the combat results; during the mid–late phase of a game, however, effective use of spells, such as 'Force Field' and 'Psionic Storm', significantly influence the overall results of a game [16].

In this paper, we propose a novel method for training agents to control the spellcaster units via three experiment sets with the DRL algorithm. Our method constructs benchmark agents as well as newly designed minigames for training the micro-control of spellcasters. They are designed to train spellcaster units in specific minimap situations where spells must be used effectively in order to win a combat scenario. Our work can be applied to general combat scenarios in prospective research using trained parameters in order to improve the overall performance of StarCraft II AI.

2. Background

2.1. Deep Reinforcement Learning

Reinforcement Learning (RL) searches for the most optimal actions a learner should take in certain states of an environment. Unlike supervised or unsupervised learning, RL does not require training data but an agent continuously interacts with an environment to learn the best actions which return the largest amount of rewards. This feature is one of the main reasons that RL is frequently used to solve complex problems in games such as Backgammon [17], Chess [18], and Mario [19]. Deep Reinforcement Learning (DRL) is a sophisticated approach of RL employing deep network structures into its algorithm. DRL has solved problems that were once considered unsolvable by using RL only. Adopting deep network methods in RL, however, is often followed by a few drawbacks. DRL algorithms are thought to be unstable for non-stationary targets and the correlation between training samples during on-line RL updates is very strong.

Among others, Deep Q-Network (DQN) is one of the most famous DRL algorithms that solved such a problem. DQN used Convolutional Neural Network (CNN) as a function approximator and applied the 'experience replay' and 'target network' method. DQN showed meaningful achievements in Atari 2600 Games using raw pixel input data, and it surpassed humans in competition [1]. Before DQN, it was challenging for agents to deal directly with high-dimensional vision or language data in the field of RL. After the success of DQN, many DRL algorithm papers have been published, including double DQN [20], dueling DQN [21], and others [22,23] as well as policy-based DRL methods such as DDPG [24], A3C [25], TRPO [26], PPO [27]. These methods solved problems involving the challenge of continuous control in real-time games: StarCraft II [9], Dota2 [2], and Roboschool [27].

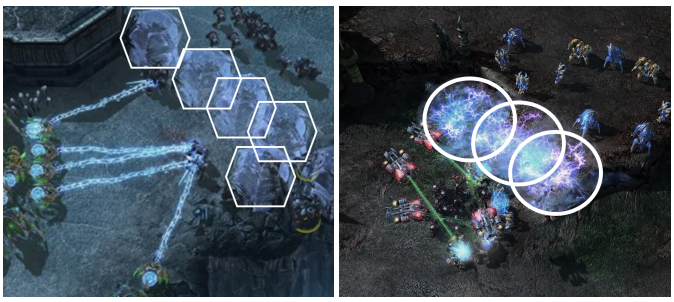
This paper employs the Asynchronous Advantage Actor-Critic (A3C) algorithm [25]. While DQN demonstrated meaningful performances, it has a few weaknesses. The computational load is heavy because it uses a large amount of memory. It requires off-policy algorithms that update from information that is generated by older policies [25]. A3C solves the strong correlation problem by using an asynchronous method that executes multiple agents in parallel. This feature also reduces the computational burden compared to DQN. For these reasons, DeepMind used the A3C algorithm in its StarCraft II AI research. Detailed descriptions of the algorithm applied to this paper are provided in the Method section.

2.2. StarCraft II and Spellcasters

In StarCraft II, the most common game setting is a dual. Each player chooses one of the three races—Terran, Protoss, and Zerg—with their distinct units and buildings, requiring different strategies for each race. Each player starts with their small base. In order to win, one has to build constructions and armies, and destroy all of the opponent's buildings. Spellcasters refer to any unit that uses energy-based spells. Spells are, and can be, powerful skills depending on how often and effectively

a player uses them based on the amount of energy, which is passively generated at 0.7875 energy per second and capped at 200 per unit. Trading energy in the form of spells with enemy units is the most cost-efficient trade in the game since the energy is an infinite and free resource [16]. In short, spellcasters have powerful abilities compared to other combat units, but the spells must be used effectively and efficiently due to the limited energy resources.

Protoss has two significant spellcaster units: Sentry and High Templar. The screenshot images of two spellcaster units are provided in Figure 1. Sentries provide 'Force field', a barrier that lasts for 11 s and impedes the movement of ground units. Designed to seclude portions of the enemy forces, it helps realize the divide-and-conquer strategy, only if properly placed. 'Force field' is one of the spells that are difficult to use because the surrounding complex situations must be regarded before creating it in proper placement coordinates. Generally, it benefits greatly from (1) choke-blocking at a choke point or a ramp, (2) dividing the opponent's army in an open field, and (3) combining with other spells like 'Psionic Storm'. High Templar casts 'Psionic Storm' which is one of the most powerful spells in the game which deals 80 damage over 2.85 s in a target area. Considering the fact that the average hit point of Terran and Zerg's basic units is 40, 'Psionic Storm' is a powerful spell which can deal high amount of energy in a very short period of time. However, it also deals damages to the ally units, so careful 'micro' with it is required as well. 'Psionic Storm' is a versatile skill that can be used in almost every scenario including assault, harass, chokepoint defense, and static defense support.



(**a**) (**b**)

Figure 1. Screenshot images of (**a**) Sentry's 'Force Field' and (**b**) High Templar's 'Psionic Storm'.

3. Method

In Particular, we use the A3C algorithm that is constructed as an Atari Network (Atari-Net) to train the StarCraft II spellcaster units. The 'Atari-net' is DeepMind's first StarCraft II baseline network [28]. One of the reasons we chose the Atari-net and A3C algorithm is to improve the on-going research topic and learn under the same condition as the minigame method in StarCraft II combat research demonstrated by DeepMind [3]. This paper suggests that spellcaster units are well trained through the methodology that we demonstrate.

A3C, a policy-based DRL method, has two characteristics: (1) Asynchronous, (2) Actor-Critic. (1) The asynchronous feature utilizes multiple workers in order to train more efficiently. In this method, multiple worker agents interact with the environment at the same time. Therefore, it can acquire a more diverse experience than a single agent. (2) The actor-critic feature estimates both the value function $V(s)$ and the policy function $\pi(s)$. The agent uses $V(s)$ to update $\pi(s)$. Intuitively, this allows the algorithm to focus on where the network's predictions were performing rather poorly. The update of A3C algorithm is calculated as $\nabla_{\theta'} \log \pi\left(a_t|s_t;\theta'\right) A\left(s_t,a_t;\theta,\theta_v\right)$, where $A\left(s_t,a_t;\theta,\theta_v\right)$ is an estimate of advantage function given by $\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V\left(s_{t+k};\theta_v\right) - V\left(s_t;\theta_v\right)$, which k varies among states. A3C particularly use CNN which produces one softmax output for policies $\pi\left(a_t|s_t;\theta\right)$, and one linear output for value function $V\left(s_t;\theta_v\right)$. In short, the policy-based algorithm updates policy parameter θ asynchronously as below,

$$d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi_{\theta'}\left(a_i|s_i\right)\left(R - V_{w'}\left(s_i\right)\right) \quad (1)$$

The Atari-Net accepts current available behaviors and features from the minimap as well as from the screen. This allows the algorithm to acknowledge situations of both the entire space and specific units. The detailed network architecture is in Figure 2. A3C implements parallel training where multiple workers in parallel environments independently update a global value function. In this experiment, we used two asynchronous actors in the same environment simultaneously, and used epsilon-greedy methods for initial exploration to use spells at random coordinates [29]. We utilized eight workers initially but soon found that two workers make the training more time-efficient.

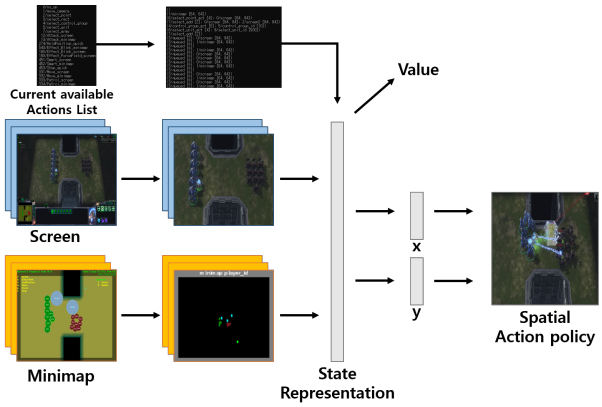


Figure 2. Atari network architecture in the StarCraft II minigame.

Figure 3 describes the overall structure of the training method used in this paper. The agent in each worker executes minigames by interacting with the StarCraft II environment directly through the PySC2 python package. This paper had 8 workers in parallel. The (x, y) coordinates for spells in minigames are based on three input values—minimap, screen, and available actions. With those input values in consideration, the agent decides the spell coordinates taking the policy parameter given by the global network into account. One episode is repeated until when the agent is defeated in combat or when it has reached its time limit. After the training of each episode is over, the agent returns the scores based on the combat result and the coordinates where spells are used. Then, each worker trains through the network to update policy parameters using the coordinate and score values. Lastly, the global network receives each worker's policy loss and value loss, accumulates those values in its network, and then sends updated policy values to each agent. Algorithm 1 below provides a detailed description of the A3C minigame training method.

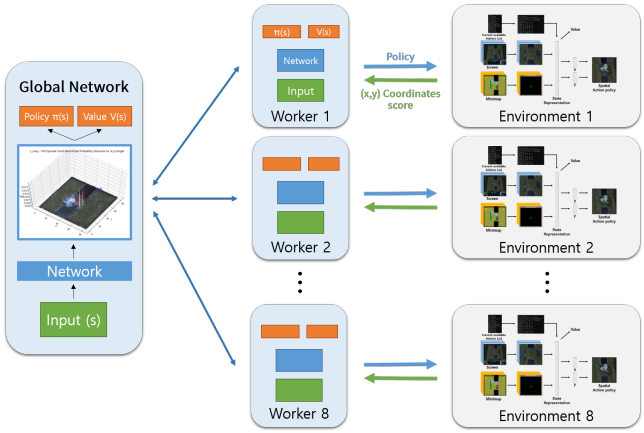


Figure 3. Overall diagram of A3C architecture in the StarCraft II training method.

Algorithm 1 A3C Minigame Training Method

Require: Num of Workers N_w, Num of Episodes N_e, Epsilon ϵ, Minigame size M, Replay Buffer $\boldsymbol{B}$

Initialize global policy parameters θ shared among agents
Global variable $Counter \leftarrow 0$
Replay Buffer $\boldsymbol{B} \leftarrow []$
for $worker = 1, 2, \ldots, N_w$ **do**
    for $Counter$ = 1 to N_e **do** ▷ Episode
        while *True* **do** ▷ Step
            if $random(0, 1) < \epsilon$ **then**
                (x, y) = randint(0,M)
            else
                $s_t \leftarrow$ Minigame's screen, minimap, available action
                Set (x, y) according to policy $\pi\left(a_t | s_t; \theta'\right)$
            end if
            Use spell at (x, y) cordinates
            if $Defeat$ or $TimeOver$ **then**
                Break
            else
                append game's observation to $\boldsymbol{B}$
            end if
        end while
        Update Global Policy π adopting replay Buffer $\boldsymbol{B}$ ▷ Training
    end for
end for

4. Experiment

4.1. Minigame Setup

The minigames are a key approach to train the spellcasters in this experiment. Each minigame is a special environment created in the StarCraft II Map Editor. Unlike normal games, this map provides a combat situation where the agent repeatedly fights its enemies. These are specifically designed scenarios on small maps that are constructed with a clear reward structure [1]. It can avoid the full game's complexity by defining a certain aspect and can be used as a training environment to apply to the full game.

We trained the agent in three typical circumstances. There are three different minigame scenarios: (a) choke blocking at a choke point or a ramp, (b) dividing the opponent's army in an open field, and (c) combination with other spells such as 'Psionic Storm'. The player's units are located on the left side of the minigame. The enemy armies are placed on the right side of the minigame. The images of the minigame scenarios are provided in Figure 4. In each minigame scenario, Sentry's 'Force Field' and High Templar's 'Psionic Storm' are trained for effective spell use. Zergling is a fast unit capable of melee attacks of fast attack speed. Roach is a ranged unit capable of ranged attacks. Detailed descriptions of the properties of units used in training are provided in Table 1 below.

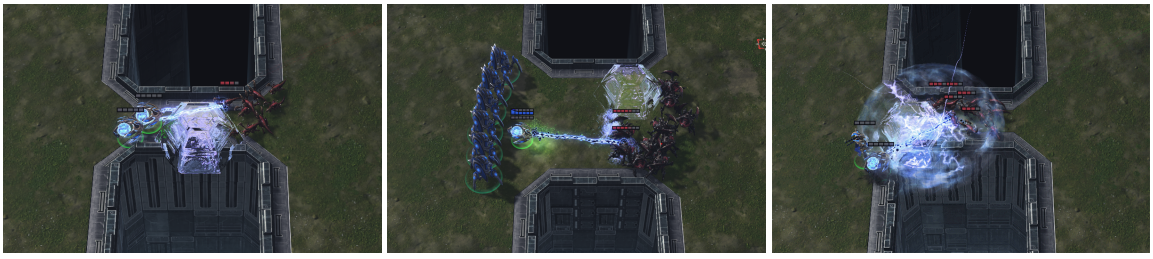
(**a**) Choke Playing (**b**) Divide Enemy (**c**) Combination with Spell

Figure 4. Initial minigame settings before training.

Table 1. The comparative attributes of different units in out minigame scenarios.

	Sentry	**High Templar**	**Stalker**	**Zergling**	**Roaches**
Race	Protoss	Protoss	Protoss	Zerg	Zerg
Unit Type	SpellCaster	SpellCaster	Combat Unit	Combat Unit	Combat Unit
Hit Point (Shield/HP)	40/40	40/40	80/80	35	145
Damage Factor (DPS)	6 (6)	4 (2.285)	18 (9.326)	5 (7.246)	16 (8)
Fire Range	5	6	6	1	4

The '(a) Choke Playing' minigame is designed to test whether training the agent under a rather simplified combat environment with a spell is feasible. The minigame consists of two Sentry ally units, and eight Zergling enemy units. It also tests whether the training agent can use the 'Force field' to utilize the benefit of the choke point. The '(b) Divide Enemy' minigame is designed to test whether training the agent under an environment that resembles a real combat with spells is feasible. The minigame consists of six Stalker and one Sentry ally units, and ten Roach enemy units. It also tests whether a training agent can use the 'Force field' to divide the enemy army. This technique can also force the enemy to engage if it manages to trap a portion of its army. The '(c) Combination with Spell' minigame is designed to test whether the agent can learn two spells at the same time under an environment similar to Choke Playing. The minigame consists of one Sentry and one High Templar ally units, and ten Zergling enemy units. It tests whether a training agent can use a combination of spells ('Force Field' and 'Psionic Storm'). Figure 5 shows the images of minigame combats after training.



(**a**) Choke Playing (**b**) Divide Enemy (**c**) Combination with Spell

Figure 5. Screenshot images of Minigame combat after training.

The objective of these minigames is to defeat the enemy armies on the right side of the minigame in every episode of training, by effectively casting spells around the choke point (narrow passage between the walls). Each episode takes two minutes. Each time all the enemy armies are defeated, the environment is reset. When the player's units are all defeated in battle, each episode terminates as well. The score of the minigame increases by ten points when an enemy unit dies and decreases by one when a player's unit dies. Since the enemy units rush forward to kill the player's units, the agent should strive to kill as many enemy units as possible without losing too many of its own. Consequently, careful 'micro' on the Sentries with regards to efficient use of 'Force field' is required because the scenario is disadvantageous to the player; the Sentries can use 'Force field' only once. In this experiment, during the process of learning to win, the agent learns to use the appropriate spells in needed situations.

4.2. Result

The graphs of the experiment results are presented in the figures below. Figure 6 shows an integrated score graph of the three minigame methods. Figure 7 provides score graphs of each training method. In Figure 7, the orange line depicts the scores of trained agents and the blue line depicts the scores of random agents that cast spells at random coordinates.

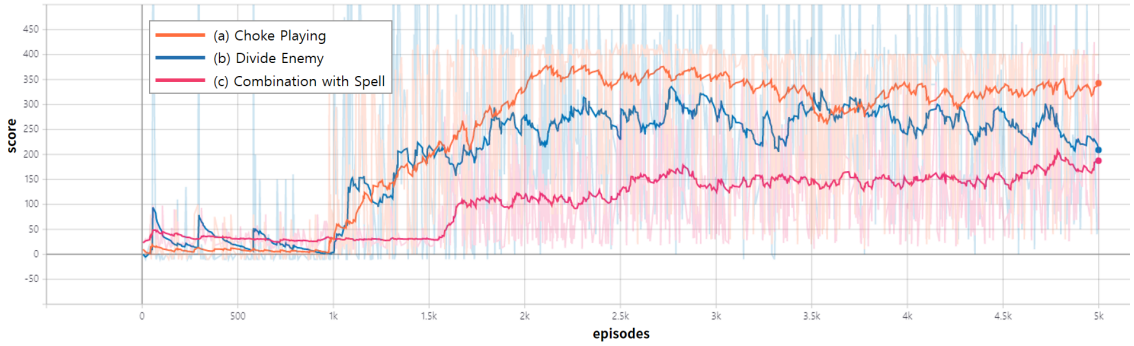


Figure 6. Integrated score graph of three minigame training methods.

(**a**) Choke Playing (**b**) Divide Enemy (**c**) Combination with Spell

Figure 7. Score graphs over 5000 episodes of training.

Table 2 provides the statistics of the training results. Win rate corresponds to the winning rate on each episode. Mean corresponds to the average score of the agent performances. Max corresponds to the maximum observed individual episode score.

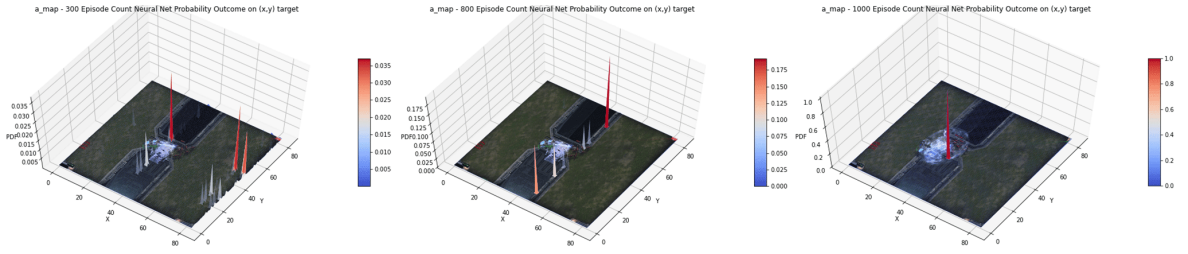
Table 2. Aggregated results for trained agent and random agent on minigames.

Agent	Metric	(a) Choke Playing	(b) Divide Enemy	(c) Combination with Spell
Trained	WIN RATE	94%	85%	87%
	MEAN	250.08	111	207.59
	MAX	430	459	1515
Random	WIN RATE	5.36%	30.96%	5.88%
	MEAN	6.67	58.39	6.918
	MAX	380	265	900

In order to win and get a high score, spells must be used properly at relevant coordinates. Because the coordinates of the enemy units' location are random in every situation, a high score indicates that spells were used effectively in randomly different situations. In that respect, a high max score implies that the agent is well trained to use spells at optimal coordinates. In short, trained agents achieved high mean and max scores compared to random agents. This aspect is especially noticeable in scenario (a) and (c). In scenario (b), due to the characteristic of the minigame, the score difference between the trained agent and the random agent was not significant as much as the score difference in the other two scenarios. Compared to the other two scenarios, minigame (b) took a comparatively long time to converge to a certain score while training.

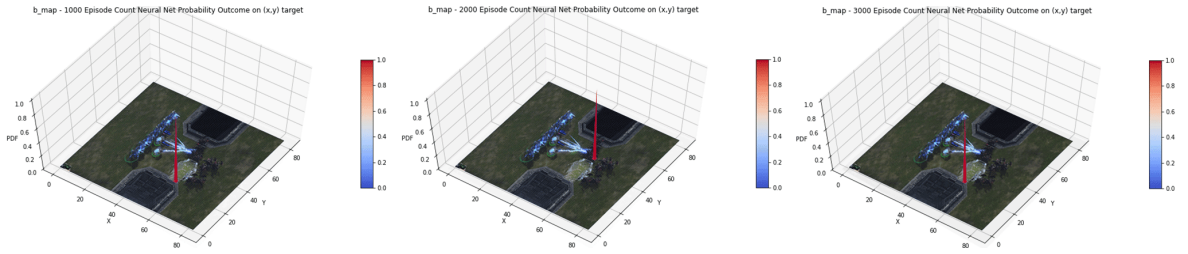
In the beginning, agents trained with the '(a) Choke Playing' used 'Force Field' to divide the enemy army to delay defeat. Over 1000 episodes, however, the agent converged to using 'Force Field' at a narrow choke-point to block the area and attacking the enemy with a ranged attack without losing health points. The agent took an especially long time to use 'Force Field' properly with the '(b) Divide Enemy' minigame. Because the score was obtained without using 'Force Field', the presence or absence of 'Force Field' was comparatively insignificant. Over 1600 episodes, the agent converged

to using 'Force Field' to divide the enemy army efficiently and taking advantage of the enemy's poor ability to focus fire. Agent trained with the '(c) Combination with Spell' could not use 'Psionic Storm' in combination with 'Force Field', but instead had its units killed along with enemy Zerglings under 'Psionic Storm'. Over 1000 episodes, the agent converged to using both spells simultaneously. They solved the scenario by blocking the choke point first, and then properly casting 'Psionic Storm'. Probability Density Functions (PDF) of the linear output from the global network structure are shown in Figures 8–10. The graphs depict the probability of the (x, y) coordinates converging into a specific location as the training episodes increase.



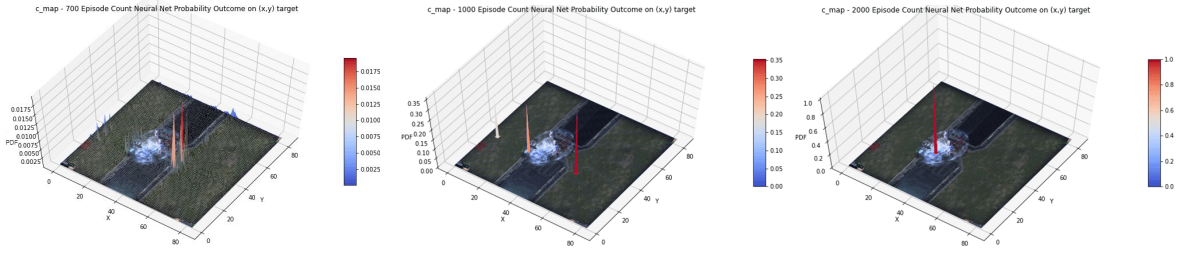
(**a**) (**b**) (**c**)

Figure 8. Probability Density Functions (PDF) of global network for '(**a**) Choke Playing' Scenario each describing (**a**) 300 episodes, (**b**) 800 episodes, and (**c**) 1000 episodes, respectively.



(**a**) (**b**) (**c**)

Figure 9. Probability Density Functions (PDF) of global network for '(**b**) Divide Enemy' Scenario each describing (**a**) 1000 episodes, (**b**) 2000 episodes, and (**c**) 3000 episodes, respectively.



(**a**) (**b**) (**c**)

Figure 10. Probability Density Functions (PDF) of global network for '(**c**) Combination with Spell' Scenario each describing (**a**) 700 episodes, (**b**) 1000 episodes, and (**c**) 2000 episodes, respectively.

For this experiment result, we applied the following hyperparameters. We used an 84×84 feature screen map and 64×64 minimap size. The learning rate was 5×10^{-4} and its discount was 0.99. For exploration, we used the epsilon greedy method. The initial epsilon (ϵ) value was at 1.0 and the epsilon decay rate was 5×10^{-4}. We trained the agent on a single machine with 1 GPU and 16 CPU threads. Agents achieved a winning rate of more than 85% against the enemy army in respective minigame situations.

4.3. Discussion

This paper presents the method of minigame production to train the spellcasters and make a combat module. First of all, we showed that spellcaster units can be trained to use spells properly and correspondingly with the minigame situations. Designing unique maps that provide various situations for spell usages showed valid effects in training spellcasters. For example, both the (a) and (c) minigames were more adaptive to the training method, because it is difficult to defeat the enemy units without training the spells in their environments.

This approach breaks the limit from using only 'attack, move, stop, and hold' commands in training for combat by including the spell commands of the spellcasters. This training method trains the agents to use spells effectively for a certain situation in both simple and complex battles. In addition, this minigame method can be used not only for training spellcaster unit control but also in training advanced unit control. This can be done with additional minigame designs particularly aimed for training control of other units in StarCraft II. Those units may include advanced combat units, such as Carrier or Battlecruiser, which does not appear frequently in melee games due to expensive costs and trade-offs in managing resources, or tactical units like Reaper or Oracle which are used for hit and run, scouting, and harassment strategies. Through this training method, the extent of employing Transfer Learning in AI research for StarCraft II will be enlarged.

In order to use this agent in real combat, one needs to train minigames in a wider variety of situations and apply transfer learning. A related study that trained existing minigames using the A3C algorithm proposed transfer learning method by applying pre-trained weights into other minigames and showed significant performance [30]. The paper also suggests applying transfer learning from minigames to Simple64 map or other full game maps. Because DRL models have low sample efficiency, a large number of samples are required [10]. Our work can be utilized in generating various and many samples. Additionally, it might be effective if one adds conditions such as region and enemy information in further studies. Therefore, we recommend extending the training of spellcaster units to the same degree as training other combat units.

We deem this an unprecedented approach since most other studies have focused on training their agents merely based on the units' combat specifications without considering the effects of exploiting complex spells. The strengths of our method are that minigame situations can reduce the complexity of training micro-control, and they can train challenging combat situations through various minigame scenarios. The limitations of our approach are that we used two spellcasters among many others, and the proposed minigames are designed to be applicable for 'Force Field' and 'Psionic Storm' spells. While we used one of the most complicated utility spells, the 'Force Field' spell, to provide a baseline for training spellcaster units, our training method can be referred to train other advanced units as well. Through the training, we have succeeded in inducing our agent to deliberately activate the 'Force Field' in critical moments. We have focused on 'Force Field', the most used and complex-to-use spell, and we plan to make use of the spellcaster agents in AI bots.

Our study is a part of the contribution to building more advanced StarCraft II AI players, especially in the aspect of maximizing the benefit of spells that can potentially affect combat situations to a great degree. We expect that developing more versatile agents capable of employing appropriate spells and applying their performances to realistic combat situations will provide another step to an ultimate StarCraft II AI. In the future, we will put effort into transfer learning techniques to use this method.

5. Conclusions

We proposed a method for training spellcaster units to use spells effectively in certain situations. The method is to train an agent in a series of minigames that are designed for specific spell usage scenarios. Hence, we made three minigames, called '(a) Choke Playing', '(b) Divide Enemy', and '(c) Combination with Spell', each representing a unique spell usage situation. By using these minigames, we trained two spellcasters to use 'Force Field' and 'Psionic Storm' in minigame

scenarios. The experiment showed that training spellcasters within the minigames improved the overall combat performance. We suggested a unique method to use minigames for training the spellcasters. This minigame method can be applied to train other advanced units besides spellcaster units. In the future, we look forward to finding a way to utilize this method in real matches. By adopting transfer learning, we expect other complex sub-problems in developing powerful StarCraft II AI can be solved. While this paper employed a DRL algorithm called A3C to minigame method, there is a lot of potential for other nature-inspired methods to be applied to solve the problem. Some of the most representative nature-inspired algorithms such as the Monarch Butterfly Algorithm (MBO) [31], the Earthworm Optimization Algorithm (EWA) [32], the Elephant Herding Optimization (EHO) [33], and the Moth Search (MS) [34] algorithm should be considered in future research. We believe that such an attempt will be another step towards the construction of an ultimate StarCraft II AI.

Author Contributions: Conceptualization, W.S.; methodology, W.S.; software, W.S. and W.H.S.; validation W.S., W.H.S. and C.W.A.; formal analysis, W.S. and W.H.S.; investigation, W.S. and W.H.S.; data curation, W.S. and W.H.S.; writing—original draft preparation, W.S. and W.H.S.; writing—review and editing, W.S., W.H.S. and C.W.A.; visualization, W.S. and W.H.S.; supervision, C.W.A.; project administration, W.S.; funding acquisition, C.W.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2019R1I1A2A01057603), and IITP grant funded by the Korea government (MSIT) (No. 2019-0-01842, Artificial Intelligence Gradate School Program (GIST)).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
2. OpenAI. OpenAI Five. 2018. Available online: https://openai.com/blog/openai-five/ (accessed on 7 June 2020).
3. Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A.S.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J.; et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv* **2017**, arXiv:1708.04782.
4. Hsieh, J.L.; Sun, C.T. Building a player strategy model by analyzing replays of real-time strategy games. In Proceedings of the 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, China, 1–8 June 2008; pp. 3106–3111.
5. Synnaeve, G.; Bessiere, P.A. Bayesian model for plan recognition in RTS games applied to StarCraft. In Proceedings of the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, CA, USA, 10–14 October 2011.
6. Usunier, N.; Synnaeve, G.; Lin, Z.; Chintala, S. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv* **2016**, arXiv:1609.02993.
7. Pang, Z.; Liu, R.; Meng, Z.; Zhang, Y.; Yu, Y.; Lu, T. On Reinforcement Learning for Full-length Game of StarCraft. In Proceedings of the Thirty-First AAAI Conference on Innovative Applications of Artificial Intelligence (AAAI'19), Hilton Hawaiian Village, Honolulu, HI, USA, 27 January–1 February 2019.
8. Lee, D.; Tang, H.; Zhang, J.O.; Xu, H.; Darrell, T.; Abbeel, P. Modular Architecture for StarCraft II with Deep Reinforcement Learning. In Proceedings of the Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference, Edmonton, AB, Canada, 13–17 November 2018.
9. Vinyals, O.; Babuschkin, I.; Czarnecki, W.M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **2019**, *575*, 350–354. [CrossRef] [PubMed]
10. Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; et al. Relational deep reinforcement learning. *arXiv* **2018**, arXiv:1806.01830.
11. Shao, K.; Zhu, Y.; Zhao, D. Starcraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *3*, 73–84. [CrossRef]

12. Stanescu, M.; Hernandez, S.P.; Erickson, G.; Greiner, R.; Buro, M. Predicting army combat outcomes in StarCraft. In Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Boston, MA, USA, 14–18 October 2013.
13. Alburg, H.; Brynfors, F.; Minges, F.; Persson Mattsson, B.; Svensson, J. Making and Acting on Predictions in Starcraft: Brood War. Bachelor's Thesis, University of Gothenburg, Gothenburg, Sweden, 2014.
14. Sun, P.; Sun, X.; Han, L.; Xiong, J.; Wang, Q.; Li, B.; Zheng, Y.; Liu, J.; Liu, Y.; Liu, H.; et al. Tstarbots: Defeating the cheating level builtin ai in starcraft ii in the full game. *arXiv* **2018**, arXiv:1809.07193.
15. Justesen, N.; Risi, S. Learning macromanagement in starcraft from replays using deep learning. In Proceedings of the 2017 IEEE Conference on Computational Intelligence and Games (CIG), New York, NY, USA, 22–25 August 2017; pp. 162–169.
16. Games Today. Spellcasters in Starcraft 2. 2019. Available online: https://gamestoday.info/pc/starcraft/spellcasters-in-starcraft-2/ (accessed on 7 June 2020).
17. Tesauro, G. Temporal difference learning and TD-Gammon. *Commun. ACM* **1995**, *38*, 58–68. [CrossRef]
18. Baxter, J.; Tridgell, A.; Weaver, L. Knightcap: A chess program that learns by combining td (lambda) with game-tree search. *arXiv* **1999**, arXiv:cs/9901002.
19. Tanley, K. O.; Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evol. Comput.* **2002**, *10*, 99–127. [CrossRef] [PubMed]
20. Van Hasselt, H.V.; Guez, A.; Silver, D. Deep reinforcement learning with double Q-Learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16), Phoenix, AZ, USA, 12–17 February 2016.
21. Wang, Z.; Schaul, T.; Hessel, M.; Van Hasselt, H.; Lanctot, M.; De Freitas, N. Dueling network architectures for deep reinforcement learning. *arXiv* **2015**, arXiv:1511.06581.
22. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *arXiv* **2015**, arXiv:1511.05952.
23. Nair, A.; Srinivasan, P.; Blackwell, S.; Alcicek, C.; Fearon, R.; De Maria, A.; Panneershelvam, V.; Suleyman, M.; Beattie, C.; Petersen, S.; et al. Massively parallel methods for deep reinforcement learning. *arXiv* **2015**, arXiv:1507.04296.
24. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
25. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Lillicrap, T.P.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1928–1937.
26. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 1889–1897.
27. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
28. Bellemare, M.G.; Naddaf, Y.; Veness, J.; Bowling, M. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.* **2013**, *47*, 253–279. [CrossRef]
29. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; MIT Press: Boston, MA, USA, 1998; ISBN 978-02-6203-924-6.
30. Alghanem, B. Asynchronous Advantage Actor-Critic Agent for Starcraft II. *arXiv* **2018**, arXiv:1807.08217.
31. Wang, G.G.; Deb, S.; Cui, Z. Monarch butterfly optimization. *Neural Comput. Appl.* **2019**, *31*, 1995–2014. [CrossRef]
32. Wang, G.G.; Deb, S.; dos Santos Coelho, L. Earthworm optimisation algorithm: A bio-inspired metaheuristic algorithm for global optimisation problems. *IJBIC* **2018**, *12*, 1–22. [CrossRef]
33. Wang, G.G.; Deb, S.; Coelho, L.D.S. Elephant herding optimization. In Proceedings of the 3rd International Symposium on Computational and Business Intelligence (ISCBI 2015), Bali, Indonesia, 7–9 December 2015; pp. 1–5. [CrossRef]
34. Wang, G.G. Moth search algorithm: A bio-inspired metaheuristic algorithm for global optimization problems. *Memetic Comput.* **2018**, *10*, 151–164. [CrossRef]