12-2021

Hierarchical control of multi-agent reinforcement learning team in real-time strategy (RTS) games

Weigui Jair ZHOU
Nanyang Technological University

Budhitama SUBAGDJA
Singapore Management University, budhitamas@smu.edu.sg

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

Darren Wee Sze ONG
DSO National Laboratories, Singapore

Hierarchical Control of Multi-Agent Reinforcement Learning Team in Real-Time Strategy (RTS) Games

Weigui Jair Zhou

School of Computer Science and Engineering, Nanyang Technological University, Singapore

Budhitama Subagdja, Ah-Hwee Tan

School of Computing and Information Systems, Singapore Management University

Darren Wee-Sze Ong

DSO National Laboratories, Singapore

Abstract

Coordinated control of multi-agent teams is an important task in many real-time strategy (RTS) games. In most prior work, micromanagement is the commonly used strategy whereby individual agents operate independently and make their own combat decisions. On the other extreme, some employ a macromanagement strategy whereby all agents are controlled by a single decision model. In this paper, we propose a hierarchical command and control architecture, consisting of a single high-level and multiple low-level reinforcement learning agents operating in a dynamic environment. This hierarchical model enables the low-level unit agents to make individual decisions while taking commands from the high-level commander agent. Compared with prior approaches, the proposed model provides the benefits of both flexibility and coordinated control. The performance of such hierarchical control model is demonstrated through empirical experiments in a real-time strategy game known as StarCraft: Brood War (SCBW).

Email addresses: jairzhou@ntu.ed.sg (Weigui Jair Zhou), budhitamas@smu.edu.sg (Budhitama Subagdja), ahtan@smu.edu.sg (Ah-Hwee Tan), oweesze@dso.org.sg (Darren Wee-Sze Ong)

Keywords: Hierarchical control, self-organizing neural networks, reinforcement learning, real-time strategy games

1. Introduction

Real-time strategy (RTS) games have been popularly used to study reinforcement learning in recent years. Unlike deterministic fully-observable board games like Go or Chess wherein Artificial Intelligence (AI) has been notably outperform humans (Hassabis, 2017), games such as StarCraft, Warcraft, and Unreal Tournament are still considered open challenges for reinforcement learning (Ontañón et al., 2013) as they may involve issues like dealing with very large state-action space, spatio-temporal reasoning, team coordination, opponent modeling, incomplete information, and decision making under uncertainties (Buro, 2003; Robertson & Watson, 2014). Leveraging reinforcement learning agents playing RTS games or battle simulation of some sort for analyzing and evaluating strategic behaviors may also be applicable in practice like discovering military doctrines from combat simulation or computer generated forces (CGF) Teng et al. (2013).

In a game that includes controlling several units to achieve the game objective like StarCraft, multiagent reinforcement learning (MARL) becomes an important field to investigate. Beyond the traditional reinforcement learning approach that a single agent learns by trial-and-error to improve its own performance, the effect of actions of each agent to one another and to the entire team objective must also be considered (Gronauer & Diepold, 2021). However, most of these learning approaches are still emphasized on micromanagement skills acquisition that each agent tries to improve its own low-level performance based on its individual input (like a step or movement, attacking towards a target, or running away) with less consideration on the macromanagement strategy to govern the activities of the entire team (Shantia et al., 2011; Wender & Watson, 2012; Gabriel et al., 2012). Each agent unit is made to learn to play the game from scratch that will end with the elimination

of either self or the opponent's unit. This approach of learning individually may end up in unstable performance as the agents face non-stationarity or moving target problem since every agent co-adapts to each other especially when there is a high dependency among the agents (Gronauer & Diepold, 2021). Some models tackle this non-stationarity issue by applying Centralized-Training-Decentralized-Execution (CTDE) method. In this method, learning is conducted in a centralized manner that all agents share special reward or value system during training so that every agent learns its policy interdependently to one another. However, the policy is still applied independently by each individual agent during testing or performing phases after training (e.g (Lowe et al., 2017; Rashid et al., 2018; Foerster et al., 2017)). A similar model of separating the phases of training and testing or execution while emphasizing micromanagement skills has also been applied even in a model that has notably reached the level of human grand-mastery in the StarCraft II RTS game through learning with self-play without the actual sparring opponents (Vinyals et al., 2019).

Despite their remarkable achievements (like beating human players in humans vs AI StarCraft tournament (Vinyals et al., 2019)), this kind of learning does not really capture the strategic control model among the agents and for a certain training approach like self-play, it is only effective when the opponents have symmetrical capabilities with the same characteristics of game objectives. No organizational structure of the team nor actual macromanagement strategy are considered in this case as relations among the agents are still assumed to be flat. This also implies that to reach an adequate level of performance, this kind of multi-agent reinforcement learning requires very extensive training iterations conducted separately from its main performance (testing) phase.

In this case, leveraging the structure of the organization, command hierarchy, and control of a team of agents in a real-time strategic game requires a distinct architecture from the flat organization structure mostly employed in existing multiagent reinforcement learning. The architecture should also ensure that the learning can be conducted effectively to capture strategic and micromanagement levels of knowledge. On the other hand, evaluating the behavior of

the agents and the effectiveness of the reinforcement learning model in acquiring both strategic and micromanagement knowledge requires a method of interpreting the learned knowledge explicitly. This enables more thorough comprehension of the overall performance of the agents after the learning which is still a great challenge in most existing multiagent reinforcement learning methods (Gronauer & Diepold, 2021). Another significant issue in most current multiagent reinforcement learning methods is the capacity to insert prior knowledge to the agents based on knowledge reuse from past learning (Gronauer & Diepold, 2021) or some hand-crafted doctrines (Teng et al., 2015) to initiate and speed-up the exploration in learning. Most reinforcement learning models mentioned above are based on deep learning or neural networks of some sort that requires extensive learning iterations from scratch with learned policies in the form of distributed representation which is too difficult, if not impossible, to interpret or to specify in advance.

To address the above issues and challenges in multiagent reinforcement learning, this paper presents a model of hierarchical control and learning of a multiagent team to play an RTS game. In contrast to other approaches of multiagent reinforcement learning that focus on micromanagement aspects of behavior with an initial flat organization structure, the hierarchical control and learning architecture enable a complex multi-agent learning problem to be decomposed into two subproblems, one at the centralized strategic manner and the other at the micromanagement unit level.

Besides the hierarchical structure of control and learning, the model allows the reinforcement learning to be investigated with the option of initially starting out from initial strategies to direct the agents. In this case, rather than having distinct phases training and testing, it is also considered that learning and performing stages are integrated and conducted continually without separation.

In order to realize the capabilities as mentioned, a class of self-organizing neural networks known as Fusion Architecture for Learning and Cognition (FALCON) is selected as the model of the learning agents (Tan, 2004; Tan et al., 2008). FALCON has been successfully developed and benchmarked in partially

observable learning tasks like reinforcement learning in first-person shooting game environment known as Unreal Tournament (Di & Tan, 2015), cooperative reinforcement learning for network routing problem (Xiao & Tan, 2013), and self-regulating reinforcement learning in pursuit-evasion domain (Teng et al., 2015). In this paper, FALCON is applied for multiagent reinforcement learning in the RTS game.

FALCON supports continual cycle of pattern retrieval (readout) and learning (Tan, 2004; Tan et al., 2019) so that no separation of training and execution is necessary allowing the reinforcement learning to be conducted online continuously. Moreover, the incremental clustering on explicit properties of state, action, and reward in its reinforcement learning algorithm allows interpretable knowledge to be learned or to be directly inserted to the neural network (Tan, 2004). Therefore, the hierarchical control for multiagent reinforcement learning model enables the learned knowledge to be interpretable as rules so that the learned behaviours and strategies can be analysed. The learning performance can also be enhanced with pre-inserted rules as the initial knowledge to follow both at primitive and strategic level of control.

For evaluation, empirical experiments are conducted in StarCraft RTS game environment. Recently, StarCraft has become a standard testbed for evaluating AI techniques in RTS games (Ontanon et al., 2015) and used by various large communities in many international AI RTS game tournaments (Ontañón et al., 2013). With the existing programming libraries and APIs, StarCraft can also be customized to simulate certain real-world tasks and missions, like battlefields or military operations. In this case, an asymmetric warfare scenario is applied to the game wherein the main objective of the agents (as battle tanks) is to advance from a starting position in the environment to a designated goal location whereas some adversary units (as battle tanks controlled internally by the game) may deter the advancing objective. This kind of scenario is suitable for evaluating the use of prior knowledge, the relevance of the learned knowledge, and the learning outcome. It is also more relevance to practical applications like analyzing strategic knowledge in computer generated forces or CGF.

The remainder of the paper is organized as follows. Section 2 provides a discussion on related work. Section 3 provides the summary of the StarCraft RTS game environment as the domain problem discussed in this paper. Section 4 provides a summary of the architecture and algorithm of FALCON. Section 5 presents the proposed hierarchical architecture model and the collaborative mechanism. Section 6 reports the experiments and discusses the simulation results. Section 7 concludes and provides a brief discussion of the future work.

2. Related Work

Reinforcement learning has been a widely adopted approach to playing real-time strategy or RTS games. Considered as a recent milestone in the field, AlphaStar has been applied to play StarCraft II against human players (Vinyals et al., 2019). It has reached the grand-master level of the game by making use of multi-agent deep reinforcement learning and exhaustive training from scratch against itself through self-play without sparring any opponent. Previous attempts have trained a team of unit players or agents to engage in battle situations in RTS games like StarCraft. A reinforcement learning method of State-Action-Reward-Action (Sarsa) was applied to control units in StarCraft small battles (Shantia et al., 2011). Artificial neural networks were used here to learn the expected reward of the actions performed and select the best action based on the expected reward. Various learning algorithms that include Temporal-Difference (TD) Q-Learning and Sarsa in the unit control have also been studied focusing on their abilities to learn the kiting strategy by unit agents in small-scale combat scenarios (Wender & Watson, 2012). Gabriel et al. (2012) used a neuro-evolutionary algorithm to create and train an artificial neural network. The created network rtNEAT evolves both the topology and connection weights of the neural networks for individual units. One of the issues in this kind of micromanagement model of multi-agent reinforcement learning is the non-stationary behaviors of different agents that simultaneously learn and adapt in decentralized manner.

Beyond independently learning the skills of individual units, others have investigated learning coordination and strategies at the team level while tackling the non-stationary issue. Adopting the centralized training with decentralized execution mechanism, some recent approaches augment additional information about the policies of all the agents to update the value function during training but apply the learned policy for individual agent independently in decentralized fashion during testing. Multi-Agent Deep Deterministic Policy Gradient (MADDPG) is one approach that takes all the states and actions of the agents into consideration during training by a critic module but applies individual policies to be executed by individual actors independently (Lowe et al., 2017). Another recent model has applied Deep Q-Network for each individual unit but using shared experience replay buffer augmented with the probability of joint-action, the rate of exploration, and the number of training iteration to handle the non-stationary issue in multi-agent reinforcement learning (Foerster et al., 2017). Another example is QMIX that also uses the shared experience buffer during training, but also learn to associate a global state with a joint action-value function in a dedicated mixing network at the same time (Rashid et al., 2018). In this way, QMIX can learn complex monotonic value function for the entire team to be used to update the policy of each individual agent. for the testing stage. These centralized training and decentralized execution models require separate phases of training and testing. The additional information about the team value function applies only during training but the execution or testing phase is conducted independently by each agent without considering the value to change the policy.

The separation of learning and testing phase suggests that the policy learned through this centralized training does not really reflect the strategic knowledge to win the game. The individual policy directs the agent only to act reactively. Unless the multi-agent reinforcement learning is model-based that explicitly extracts a model of the environment and the team interaction while conducting reinforcement learning (Zambaldi et al., 2018), it is impractical to interpret the learned policy to understand the strategy and thus, to ensure the continuity of

the learning.

Another type of multi-agent reinforcement learning incorporate communications and interactions among the agents as parts of the input to be learned either during the training or testing phase. A deep neural network controller using heuristic reinforcement learning algorithm (Usunier et al., 2017) has been proposed that selects the unit agent with different methods of target selection including random static target, built-in AI control, nearest target, weakest target and no overkill with static target. Peng et al. (2017) formulated multi-agent learning for StarCraft combat task as a zero-sum Stochastic Game, wherein agents can communicate through their proposed bidirectional-coordinated net as an interaction protocol for the agent to perform and learn. These interactive, models however did not provide a clear description of the unit coordination strategy. This lack of unit coordination during battles may thus lose the element of cohesiveness and weaken the overall fire power. Moreover, their concerns are mainly on federated controls in reinforcement learning wherein the learning can be conducted across multiple agents in decentralized manner to make it efficient and scalable. The federated control can be realized practically through a negotiation framework for reaching agreements upon joint actions among the agents which may be moderated by a meta-controller (Kumar et al., 2017). It can also be applied as a communication protocol devised for a unit agent to exchange its internal state and policy with others within a spatio-temporal proximity (Chu et al., 2020).

Although the federated control models can be more practical and scalable, in this paper, we emphasize more on the aspect of organizational structure and generalization in learning to influence the resulting performance and learned strategies. In our approach, the learned policy and values are still shared among the agents though each may have different observation and action to take. Unlike the popular centralized training with decentralized execution model for multi-agent reinforcement learning, the proposed model in this paper considers indivisible phases of training and execution for each agent. Similar to the federated model, the central commander can also be considered as a special type of

unit agent that run independently but always direct the actions of other units.

Besides the hierarchical organization structure and directed communication, the proposed model can also maintain the learned policy as explicit and interpretable knowledge or strategy. However, unlike the model-based approach (Zambaldi et al., 2018), the reinforcement learning algorithm can still be considered model-free as it does not construct any explicit model about the environment or the game being played besides the policy.

3. Problem Domain: StarCraft Brood War

StarCraft is a science fiction real-time strategy video game developed by Blizzard Entertainment in early 1998. The game was further expanded and released in late 1998 with Saffire company as an expansion pack known as StarCraft: Brood War (SCBW) (Wikipedia, 2019). The game play revolves around players collecting resources to build structures and units in order to defeat other players. Each player can choose one out of the three distinct interstellar species: Protoss, Terran, or Zerg.

The current application programming interface (API), known as Brood War API (BWAPI) (Doxygen, 2017), is well supported in the SCBW community. BWAPI was developed primarily for supporting the development of artificial intelligence agents to play the game autonomously. The complexity of real time strategy games is much higher than constrained board games and it can better represent a simplified military simulation (Robertson & Watson, 2014). BWAPI allows the community to build artificial agents using a variety of approaches, including hard-coded, planned-based, as well as machine-learning approaches. A survey by Ontañón et al. (2013) has documented different approaches and some of the popular competitions across the years, including AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), IEEE Conference on Computational Intelligence and Games (CIG), and Student StarCraft AI (SSCAI) Tournament.

As shown in Figure 1, both the environment and unit attributes are gathered

Figure 1: Communication bridge between StarCraft game and AI module with Brood War API.

from the callable methods in the BWAPI that provides the communication between the game and the developed AI Module. BWAPI also provides a terrain analyser module that can generate the shortest path between two points and identify potential choke points. Readers are encouraged to refer to (Jurenka.sk, 2014) for a complete listing of the available methods provided by BWAPI.

Figure 2: Concurrent views of units at different terrain height illustrating the effect of Fog of War.

In this StarCraft game, different terrain levels may provide different advantages for players to gain more scores. Figure 2(a) and (b) show the concurrent views of two different players with units at the low terrain and high terrain, respectively. Units at the higher terrain have the advantage of not being affected by the fog of war. In contrast, units at the lower terrain will not be able to detect the enemies on the higher terrain unless the enemies make an attack and expose their location. In addition, units attacking from a lower terrain to the

higher terrain will have a higher chance of missing the target. This gives an advantage to units that are on the higher terrain.

Table 1: Descriptions of Symbols and Notations presented

Symbol	Description
F_1^k, F_2	Neural fields of fusion ART/FALCON neural network: respectively kth F_1 input fields and F_2 category field
$\mathbf{I}^k$, $\bar{\mathbf{I}}^k$	Respectively input vector and its complemented vector to present to the kth input field in Fusion ART/FALCON neural network
$\mathbf{x}^k$, $\mathbf{y}$, $\mathbf{w}_j^k$	Respectively the activity vector of kth input field in F_1, the activity vector of F_2 category field, and the weight vector associating jth node in F_2 with the input pattern in F_1^k field in fusion ART/FALCON neural network.
α^k, β^k, γ^k, ρ^k	Respectively choice, learning rate, contribution, and vigilance parameter of kth input field in fusion ART/FALCON neural network
T_j, m_J^k	Respectively bottom-up choice function for node j in F_2 field and top-down matching function of node J in F_2 to the kth F_1 field in fusion ART/FALCON neural network
$\wedge$, $\|\mathbf{p}\|$, $\|\|\mathbf{p}\|\|$	Respectively fuzzy AND operator defined by $(\mathbf{p} \wedge \mathbf{q})_i \equiv min(p_i, q_i)$, L1 norm operator defined by $\|\mathbf{p}\| \equiv \sum_i p_i$, and L2 norm operator defined by $\|\|\mathbf{p}\|\| \equiv \sqrt{\sum_i p_i^2}$
$\mathbf{S}$, $\mathbf{A}$, $\mathbf{Q}$, $\mathbf{Q}^*$	Respectively the state vector, the action vector, the (Q) value vector, and the maximum (Q) value vector
r, $Q(s,a)$, $TDErr$	Respectively the reward value, the expected accumulated (Q) value of taking action a from state s, and the temporal error term of the (Q) value in Temporal Difference learning
α, γ, $max_{a'}Q(s',a')$	Respectively the learning rate parameter, the discount parameter, and the maximum expected accumulated (Q) value of taking an action from state s'

4. Fusion Architecture for Learning and Cognition (FALCON)

Fusion Architecture for Learning and Cognition (FALCON) is a model for reinforcement learning based on the Fusion Adaptive Resonance Theory (Fusion ART) neural network architecture (Tan et al., 2007, 2019). Specifically, the FALCON model is a three-channel fusion ART model comprising a category field F_2 and three input fields, namely a sensory field F_1^1 for representing the

current state, an action field F_1^2 for representing the action to take, and Q (reward) field F_1^3 for representing the reinforcement value.

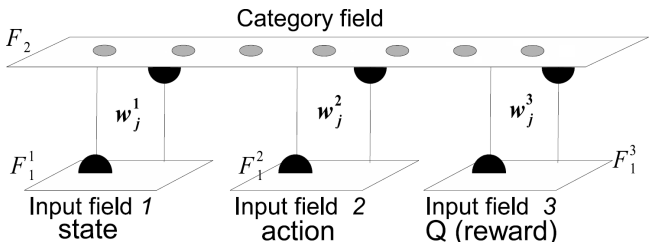


Figure 3: Fusion ART for Reinforcement Learning.

The state field F_1^1 represents the currect state of affair in the environment or the game. In a game like StarCraft, this can be the sensory information as received by the agent like the indication if an enemy agent is nearby, if the enemy is attacking, the current health status of the agent, and so on. On the other hand, the action field F_1^2 represents the action taken by the agent, like attacking or advance move in the StarCraft game, as the reaction to the current state as represented in F_1^1. After the action is taken and executed, the agent may receive feedback in terms of reward signal indicating the value of taking the action in the given state. In this case, the reward value is represented in F_1^3 field.

When the reinforcement value is obtained by the agent after the action is taken, the reinforcement learning is taking place in FALCON by associating the state-action pair with an evaluative reward value. This association is categorized and clustered in F_2 category field that can be explicitly interpreted as a code of triple or rule of behavior.

In what follows, the neural network representation and the dynamics of FALCON are described and explained in more details. Starting from this section, some symbols and notations for various concepts are defined and explained. All the symbols can be referred to Table 1.

4.1. FALCON dynamics

The FALCON neural network, as a three-channel fusion ART, can be defined as follows:

Input vectors: Let $\mathbf{I}^k = (I_i^1, I_i^2, I_i^3)$ be the input vector, where $I_i^k \in [0, 1]$ indicates the input i to channel k. In this case, I_i^1, I_i^2, and I_i^3 corresponds to the input attribute in the state, action, and Q (reward) field respectively. With *complement coding*, the input vector $\mathbf{I}^k$ is augmented with a complement vector $\bar{\mathbf{I}}^k$ such that $\bar{I}_i^k = 1 - I_i^k$ for $k \in \{1, 3\}$. Particularly, in FALCON, only input state ($\mathbf{I}^1$) and input Q ($\mathbf{I}^3$) are complement coded. Specifically, complement coding in FALCON with fuzzy operations is necessary to prevent *category proliferation* and to enable generalization of input (output) attributes (Tan et al., 2007).

Activity vectors: Let $\mathbf{x}^k = (x_1^k, x_2^k, x_3^k)$ be the F_1^k activity vector of FALCON input fields, for $k = 1$, 2, and 3. Initially, $\mathbf{x}^k = \mathbf{I}^k$. Let $\mathbf{y}$ be the F_2 activity vector of the FALCON category field.

Weight vectors: Let $\mathbf{w}_j^k$ be the weight vector associated with jth node in F_2 for learning the input patterns in F_1^k. Initially, F_2 contains only one *uncommitted* node and its weight vectors contain all 1's.

Parameters: The neural network is characterized by learning rate parameters $\beta^k \in [0, 1]$ that sets how much the update is applied to the weight vector of the corresponding k-channel, contribution parameters $\gamma^k \in [0, 1]$ that corresponds to the importance of field k during bottom-up activation, and vigilance parameters $\rho^k \in [0, 1]$ that indicates how sensitive field k towards differences during top-down matching operation. For $k \in \{1, 2, 3\}$, the choice parameter $\alpha^k > 0$ indicates the significance of field k among the others. It is also used to avoid division by zero.

Based on the definitions, the FALCON neural network operates in two different modes, namely, action selection to select an action to take by the agent and learning based on the reward feedback as received from the environment. These modes are described as follows.

4.1.1. Action Selection

Action selection is a process of finding the best action to take by the agent based on the current input state and the existing learned code that best matches with the state and may provide the maximum accumulated values in the long run if its action is performed. The action selection in FALCON is carried out in four basic steps, namely, code activation, code competition, template matching, and activity readout. As a special kind of fusion ART neural network, those steps can be described in general to select a node in F_2 category field and to readout the value to a particular field as follows.

Code activation: A bottom-up activation takes place in which the activities of the nodes in the F_2 field are computed. Given the activity vectors $\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3$, for each F_2 node j, the choice function T_j is computed as follows:

$$T_j = \sum_{k=1}^{3} \gamma^k \frac{|\mathbf{x}^k \wedge \mathbf{w}_j^k|}{\alpha^k + |\mathbf{w}_j^k|}, \tag{1}$$

where the fuzzy AND operation $\wedge$ is defined by $(\mathbf{p} \wedge \mathbf{q})_i \equiv min(p_i, q_i)$, the norm $|.|$ is defined by $|\mathbf{p}| \equiv \sum_i p_i$ for vectors $\mathbf{p}$ and $\mathbf{q}$, and the operation $\cdot$ is a vector dot product.

Code competition: A code competition process follows under which the F_2 node with the highest choice function is identified. The winner is indexed at J where $T_J = \max\{T_j : \text{for all } F_2 \text{ node } j\}$. When a category choice is made at node J, $y_J = 1$; and $y_j = 0$ for all $j \neq J$. In this case, a winner-take-all strategy is applied.

Template matching: Before the node J can be selected and retrieved, a template matching process checks if the weight templates of node J are sufficiently close to their respective input patterns. Specifically resonance occurs if, for each channel k, the match function m_J^k of the chosen node J meets its vigilance criterion:

$$m_J^k = \frac{|\mathbf{x}^k \wedge \mathbf{w}_J^k|}{|\mathbf{x}^k|} \geq \rho_s^k, \tag{2}$$

where ρ_s^k is the vigilance parameter of the corresponding field k during the action selection process. If any of the vigilance constraints for a field k is violated, the

search process then selects another F_2 node J. This search and test process is guaranteed to end as it will either find a *committed* node that satisfies the vigilance criterion or activate an *uncommitted* node that definitely satisfies the criterion as its weight vectors contain all 1's. This cycle of *code activation*, *code competition*, and *template matching* to find a node with resonance condition is also called *resonance search*.

Activity readout: The chosen F_2 node J performs a readout of its weight vectors into the input field F_1^k such that $\mathbf{x}^{k(new)} = \mathbf{x}^{k(old)} \wedge \mathbf{w}_J^k$. The resultant activity vectors in F_1^k are thus the fuzzy AND of the original value and their corresponding weight vectors.

The resonance search is the core process of fusion ART wherein bottom-up activation and top-down matching interact together with simple Fuzzy operations of Choice activation (Equation 1) and Template matching (Equation 2) to settle at the choice of extending an learned cluster or creating a new one. This continuous matching process is analogue to the use of acceptance probability in Metropolis-Hasting algorithm to determine the inclusion of a sample in random walk process (Martino & Elvira, 2017).

Action selection in the reinforcement learning of FALCON neural network is employed to get the action to perform which can be readout from an existing code that can provide the desirable reward values. In Direct Access method of FALCON (Tan, 2007), this can be achieved by selecting a node with the highest Q value with the matching state s. Upon input presentation, the state vector is initialized as $\mathbf{x}^1 = \mathbf{S} = (s_1, s_2, \ldots, s_n)$ where $s_i \in [0, 1]$ indicates the value of sensory input i. Direct Access method simplifies the selection of the best action by directly retrieving the code with maximum Q value. The search can be conducted by simply presenting the state $\mathbf{S}$, the action $\mathbf{A} = (1, ..., 1)$ (to allow complete overriding of values by activity readout), and the maximum value vector $\mathbf{Q}^*$, in which $\mathbf{Q}^* = (1, 0)$, to the corresponding fields to retrieve the code with the closest (the highest possible) match of the reward value through the four steps in action selection process.

However, the agent still needs to explore the possible outcomes of different

Algorithm 1 FALCON Action Selection

Sense the environment and formulate a state representation in $\mathbf{S}$
According to exploration-exploitation policy, make a choice between exploration and exploitation
if exploration **then**
    select a random action a from the set of possible actions
    set $x_a^2 \leftarrow 1$ and every other $x_b^2 \leftarrow 0, b \neq a$ in action vector $\mathbf{x}^2$
    set $\mathbf{Q}$ to the default value (for example $\mathbf{Q} \leftarrow (0.5, 0.5)$)
else if exploitation **then**
    present state, action, and reward value vector $\mathbf{S}$, $\mathbf{A} = (1, ..., 1)$, and $\mathbf{Q}^*$ to the corresponding $\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3$ vector in F_1 respectively
    repeat
        for all node j in F_2 field, compute $T_j \leftarrow \sum_{k=1}^{3} \gamma^k \frac{|\mathbf{x}^k \wedge \mathbf{w}_j^k|}{\alpha^k + |\mathbf{w}_j^k|}$ (Code activation)
        select node J in F_2 based on $T_J = \max\{T_j : \text{for all } F_2 \text{ node } j\}$ (Code competition)
        compute $m_J^k \leftarrow \frac{|\mathbf{x}^k \wedge \mathbf{w}_J^k|}{|\mathbf{x}^k|}$ (Template matching)
    until $m_J^k \geq \rho_s^k$ for every channel k in F_1
    set $Q \leftarrow \mathbf{w}_J^3$ (Q weight vector)
    if node J is uncommitted (no matching code can be found in F_2) **then**
        select a random action a from the set of possible actions
        set $x_a^2 \leftarrow 1$ and every other $x_b^2 \leftarrow 0, b \neq a$ in action vector $\mathbf{x}^2$
        set $\mathbf{Q}$ to the default value (for example $\mathbf{Q} \leftarrow (0.5, 0.5)$)
    end if
    readout $\mathbf{x}^{2(new)} \leftarrow \mathbf{x}^{2(old)} \wedge \mathbf{w}_J^2$ (Activity readout)
end if
set $\mathbf{A} \leftarrow \mathbf{x}^2$

actions at the initial stage of learning when the existing learned knowledge are still insufficient to direct the agent towards the best possible values. To deal with the trade-off between *exploitation*, like selecting the action based on a learned knowledge so far, and exploration, like trying out an action less familiar, an exploration strategy can be applied providing options besides selecting the action based on learned knowledge. In Direct Access method (Tan, 2007), for example, ϵ-greedy policy is employed so that the selection based on the learned knowledge (exploitation) is applied with probability of ϵ and random action selection is taken with probability of $1-\epsilon$. The ϵ parameter is decayed gradually over time until the policy turns into the full exploitation.

Algorithm 1 shows the action selection process in FALCON wherein an action is selected through resonance search in fusion ART neural network (line 8 to 12) during the mode of exploitation. It is also shown that a random action selection is still conducted when no match can be found or an uncommitted code is selected instead (line 13 to 15). The outputs of Algorithm 1 are the selected action vector $\mathbf{A}$ and the expected $\mathbf{Q}$ vector for taking the action of $\mathbf{A}$.

4.1.2. Learning

As a type of fusion ART neural networks, FALCON conducts learning by *template learning* when a matching F_2 node J is found through the resonance search during action selection.

Template learning: Given a selected node J , for each input channel or field k in F_1 of FALCON, the weight vector $\mathbf{w}_J^k$ is modified by the following learning rule.

$$\mathbf{w}_J^{k(\text{new})} = (1-\beta^k)\mathbf{w}_J^{k(\text{old})} + \beta^k(\mathbf{x}^k \wedge \mathbf{w}_J^{k(\text{old})}), \tag{3}$$

where $\beta^k \in [0,1]$ is the learning rate parameter for the particular field k in F_1.

The template learning rule implies that the weight vector $\mathbf{w}_J^k$ is updated towards the values of its corresponding input vector $\mathbf{x}^k$ with the rate of β^k. β^k can be set to 1 as the maximum rate for fast learning or below 1 for a slower learning to deal with noisy inputs from the environment. When the field

k is complement coded, the updated weights become generalized so that the corresponding node J will match with more variations of similar inputs in the particular field k.

However, when the selected node J is uncommitted implying that the previous resonance search failed to find a matching code, the entire patterns of all the input fields must be stored as they are in weight vectors under node J. In that case, the learning rate β^k is typically set to 1, for the uncommitted node J.

On the other hand, reinforcement learning in FALCON is intended to acquire an action policy as the mapping of a state to its corresponding desirable action. Once the selected action a is performed, the agent will receive a feedback from the environment as the reward signal r and may end up in another state s'. Based on these information, FALCON makes use of Temporal Difference method of learning to update the value of the current policy. Let $Q(s, a)$ be the current value of taking action a in state s. When receiving reward r, ending up in state s', $Q(s, a)$ must be updated based on Temporal Difference equation such that $\Delta Q(s, a) = \alpha TDerr$, where $\alpha \in [0, 1]$ is the learning rate parameter and $TDerr$ is the temporal error term given state s and action a so that

$$TD_{err} = r + \gamma \text{max}_{a'} Q(s', a') - Q(s, a), \tag{4}$$

where r is the immediate reward value, $\gamma \in [0, 1]$ is the discount parameter, and $\text{max}_{a'} Q(s', a')$ denotes the maximum estimated value of the next state s'. The update function is then becomes $Q^{(\text{new})}(s, a) \leftarrow Q^{(\text{old})}(s, a) + \alpha TDerr$. This Q-learning update rule is applied to all states that the agent traverses. With many iterations of update, the value function $Q(s, a)$ is expected to converge to $r + \gamma max_{a'} Q(s', a')$ over time.

As the fusion ART neural network can only process a normalized input value from 0 to 1, a Bounded Q-Learning rule can be defined so that it can also be applied in FALCON as follows.

$$\Delta Q(s, a) = \alpha TD_{err}(1 - Q(s, a)). \tag{5}$$

By incorporating the scaling term $1 - Q(s, a)$, the adjustment of Q values will be self-scaling so that they will not increase beyond 1. Based on the Bounded Q-learning update rule, the **Q** input field can be updated in FALCON so that the code for the policy can be learned.

Algorithm 2 shows the steps taken in FALCON reinforcement learning. It is assumed that the action selection process has taken place prior to the learning and selected action has been performed. In that case, the state, the selected action, the reward, the successor state, and the expected Q value are available at the start (line 1 of Algorithm 2). Resonance search processes are conducted twice in Algorithm 2, firstly to obtain the maximum expected Q value in the future (line) from the next state s' (line 2) and secondly to select a node J in F_2 to learn the code or rule of behavior (line 8).

Algorithm 2 FALCON Learning

1: Given the previous state vector **S**, the performed action vector **A**, the expected **Q** vector of taking action **A** from **S**, the reward r received, and the next state vector **S′** after taking action **A**
2: Obtain the maximum expected future **Q′** (or $max_{a'}Q(s', a')$) by resonance search with $\mathbf{x}^1 \leftarrow \mathbf{S}'$, $\mathbf{x}^2 \leftarrow (1, ..., 1)$, and $\mathbf{x}^3 \leftarrow \mathbf{Q}^*$
3: **if** uncommited node is found for **Q′ then**
4: set **Q′** to default value (like $\mathbf{Q}' \leftarrow (0.5, 0.5)$)
5: **end if**
6: Based on **Q′**, **S**, **A**, **Q**, and r, obtain $Q^{(\text{new})}(s, a) \leftarrow Q^{(\text{old})}(s, a) + \alpha TDerr$
7: Update **Q** according to $Q^{(\text{new})}(s, a)$ just obtained
8: Find the best match node J in F_2 by resonance search with $\mathbf{x}^1 \leftarrow \mathbf{S}$, $\mathbf{x}^2 \leftarrow \mathbf{A}$, and $\mathbf{x}^3 \leftarrow \mathbf{Q}$ and ρ_l^k as the vigilance parameter
9: **if** node J is uncommitted **then**
10: $\mathbf{w}_J^{k(\text{new})} = \mathbf{x}^k \wedge \mathbf{w}_J^{k(\text{old})}$ for all k in F_1
11: set J to be committed and allocate a new uncommitted node
12: **else**
13: Readout action vector $\mathbf{x}^2 \leftarrow \mathbf{x}^2 \wedge \mathbf{w}_J^2$
14: $\mathbf{w}_J^{k(\text{new})} = (1 - \beta^k)\mathbf{w}_J^{k(\text{old})} + \beta^k(\mathbf{x}^k \wedge \mathbf{w}_J^{k(\text{old})})$ for all k in F_1
15: **end if**

4.2. FALCON with ART2 Extension

With the Fuzzy AND operation $\wedge$, Fuzzy ART allows generalization of the input vectors to be learned. Equation 3 suggests that, based on the learning rate β^k parameter, weight values may decrease to the minimum if they are different from the corresponding elements of the input vector. This implies that some features that are irrelevant or changed significantly often, like in the state field, may vanish or be filtered out in the corresponding vector to zero. In a complement coded field, its corresponding weight vector may also be generalized ignoring varying features while staying focus on invariant parts.

However, this kind of generalization cannot hold for the reward field as the value may fluctuate inconsistently especially when there is a hidden or partially-observed state. In a later section on the experiments in this paper, it is also shown that FALCON reinforcement learning with all the Fuzzy operations cannot cope with inconsistent changes in rewards due to the partial observality and uncertainties in the game environment. The vanishing values and overgeneralization in rewards result in low performance of learning.

A modification for effective learning was therefore proposed to handle the uncertainties in an environment, similar to the partially-observable Markov decision process (POMDP) (Di & Tan, 2015). In POMDP, performing the same action in a similar situation may not lead to the same outcome. Therefore, in order to allow the agents to perform generalization and concurrently ensure the correctness of learning, a combination of fuzzy ART and ART2 operation was proposed. Specifically, fuzzy ART operations are applied to the state (k=1) and action (k=2) vectors for state and action space generalization and ART2 operations are applied to the reward (k=3) vectors for value approximation. In this hybrid model of FALCON, the choice activation function applied in Algorithm 1 as defined in Equation 1 can be replaced with the new hybrid function as follows.

$$T_j = \sum_{k=1}^{2} \gamma^k \frac{|\mathbf{x}^k \wedge \mathbf{w}_j^k|}{\alpha^k + |\mathbf{w}_j^k|} + \gamma^3 \frac{\mathbf{x}^3 \cdot \mathbf{w}_j^3}{||\mathbf{x}^3||\ ||\mathbf{w}_j^3||}, \tag{6}$$

where the norm $||.||$ is defined by $||\mathbf{p}|| \equiv \sqrt{\sum_i p_i^2}$. The input vector $\mathbf{I}^3$ is

augmented with a complement vector $\bar{I}_i^3 = \sqrt{1-(I_i^3)^2}$ instead.

In the case of learning when performing ART2 operation on the reward field, the following learning rule can be used.

$$\mathbf{w}_J^{k(\text{new})} = \begin{cases} (1-\beta^k)\mathbf{w}_J^{k(\text{old})} + \beta^k(\mathbf{x}^k \wedge \mathbf{w}_J^{k(\text{old})}), & \text{for } k\text{=1,2} \\ (1-\beta^k)\mathbf{w}_J^{k(\text{old})} + \beta^k\mathbf{x}^k, & \text{for } k\text{=3.} \end{cases} \tag{7}$$

The fuzzy ART learning rule adjusts the weight values using the Fuzzy AND operation which allows generalization in complement coded state vector but the weight values are monotonically decreasing. On the other hand, the ART2 template learning, based on dot product operation, still allows the weight values to increase or decrease according to the input vector $\mathbf{x}^k$. This hybrid model of learning can be applied in FALCON by replacing template learning method that changes weight vectors in Algorithm 2 defined in Equation 3 with the one in Equation 7.

4.3. Knowledge Insertion

In FALCON, a code associating state, action, and Q (reward) value is learned or clustered as a node in its F_2 category field. The codes learned in F_2 are compatible with a class of IF-THEN rules that maps a set of input attributes (antecedents) in one pattern channel (field) to a disjoint set of output attributes (consequents) and the estimated reward value in the other channel. In this way, instructions in the form of IF-THEN rules (accompanied by reward values) can be readily translated into the recognition categories at any stage of the learning process. Consequently, the learning outcomes can also be observed and analysed directly in terms of the rules.

Specifically, each corresponding rule can have the following format:

$$\textbf{IF } c_1 \wedge c_2 \wedge \cdots \wedge c_n \textbf{ THEN } a_j \ (\mathbf{Q} = r)$$

where $\wedge$ indicates the logical AND operator. Each conditional attribute c_i and action attribute a_j correspond to each element of the state and action vector respectively. On the other hand, $\mathbf{Q}$ corresponds to the reward values

in FALCON input vectors. In other words, The rule format above contains n number of condition (c_1, c_2, and so on until c_n) to match from the state input that each condition refers to the corresponding element of the state vector, an action a_j to select wherein j may correspond to the index of the element in the action vector, and r value corresponding to the value in the reward vector in FALCON input fields.

Besides analyzing the learned code explicitly, the rules can also be used as a form of instructions that they can be directly inserted to the network allowing the agent to follow and act without firstly conducting many iterations of trials-and-errors in reinforcement learning. A set of rules can be defined explicitly either by hand or based on the results of previous learning episodes (transferred knowledge) and used as initial knowledge to direct the behavior of the agent at the early stage of learning.

To insert rules into the network, the IF-THEN clauses and reward values of each rule can be translated into corresponding input vectors. Knowledge insertion is an important feature where expert knowledge can be inserted to speed up the learning process. This feature works hand in hand with dynamic exploration by exploiting the existing knowledge prior to exploring.

In this paper, the rules insertion feature is applied in the experiments to guide the initial behavior of the agents both at micromanagement unit level and macromanagement commanding level of the hierarchical control structure. The detail of rules to be inserted will be given in later section about the experimental settings.

5. The proposed model

As mentioned earlier, existing AI bots typically employ macromanagement for handling research management, building placement, and strategy planning, but leaving combat to be handled at the micromanagement level. However, such micromanagement of units does not provide for cohesiveness in the action and movement among the agents, impeding the overall fire power and unit

placement.

In this section, a hierarchical learning and control model is presented, comprising a commander agent at the upper level and multiple unit agents at the lower level. The commander and unit agents are each modeled as a reinforcement learning agent, namely FALCON, as described in the previous section. The state vector, action vector and reward system used in the next section will be illustrated.

5.1. The Unit Agent Model

In the unit agent model, an individual agent is allowed to make its own action decision based on the sensory input obtained from the environment. An example of the unit agent information vector is shown in Figure 4. Each state vector is piped through a channel of the fusion ART at the F_1 input layer, which is connected to the F_2 category field.

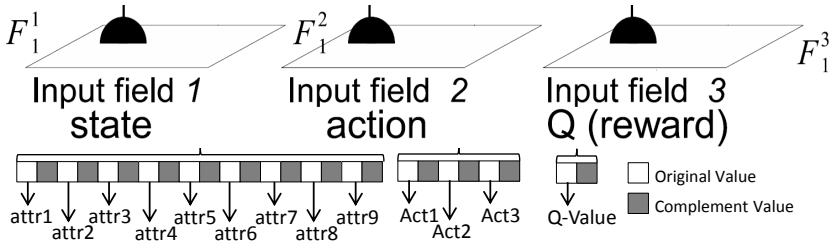


Figure 4: Example of the structure of activity vectors for a unit agent. There are complement coded state, action, and Q (reward) vectors, each consecutively has 9 attributes (attr1 to attr9), 3 actions (Act1 to Act3), and one Q (reward) value (Q-Value). Each element of a vector is shown as a white square for its original value and grey square for its complement.

As shown in Table 2, the state vector of the unit agent model **S** consists of nine input attributes. Using complement coding (Tan et al., 2007), the length of **S** is 18 (including the complement values). These attributes range from the agent-self health condition, enemy's presence to ground height level advantage. In response to the state vector presented, FALCON selects an action among the available choices, namely Advance, Attack, and Fall back. The Advance action is to move towards the next checkpoint. The Attack action is to engage in a

battle with the nearest enemy. The Fall back action is to move towards the previous checkpoint and concurrently lure the enemy down from higher terrain. For the purpose of fall back, checkpoints were selected along the shortest path generated by the terrain analyser in BWAPI.

Table 2: The state attributes of unit agents.

No.	Name	Type/Description
1	Ground height advantage	Real value, disadvantage ground=0.0, even ground=0.5, advantage ground=1.0
2	Enemy in attack range	Boolean
3	Nearest ally closeness	Boolean, true when nearest ally is within half the distance of the unit's seek range
4	Current Health	Real value, normalised against full health
5	Under attack	Boolean
6	Weapon cool down	Boolean
7	Enemy closeness	Boolean, true when enemy is within half the distance of the unit's seek range
8	Ally ratio	Real value, number of existing allies over all existing units including enemy units
9	Average ally health	Real value, average health of existing units normalised against full health

For reinforcement learning, a reward function has to be designed which serves as evaluative feedback to the actions of the agents. For the game scenario, the terminal reward function is defined by the following rules,

if game ends with a win **then**
  reward will be 1.0
else if unit is destroyed **then**
  reward will be 0.0
else if time out occurs **then**
  reward will be 0.0.
end if

During the game play, additional rewards are provided to facilitate learning of the agents. The intermediate reward function is defined by the following rules,

if an enemy is destroyed **then**
  reward will be 1.0
else if an enemy is damaged **then**
  reward will be 0.75
else if unit is damaged **then**
  reward will be 0.25
else if there is a decrease in distance to destination **then**
  reward will be 0.75
else if there is an increase in distance to destination **then**
  reward will be 0.25
else if there is no change in the distance **then**
  reward will be 0.5.
end if

With the intermediate and terminal rewards, FALCON learns by updating or creating new codes using the state and the executed action in the previous action selection cycle. This model shows a standalone action decision model with no interaction among the ally agents. This thus lacks the cohesion and action coordination among existing units that are able to increase the chances of completing the task.

5.2. The Commander Agent Model

The commander agent is also a FALCON model, which gathers the input state information from the environment and the existing units, and makes an action decision to be issued down to individual units. In this model, the action issued by the commander will be strictly followed and executed by all units. This model allows a better judgement of the current situation based on the information gathered from existing units and environment. However, such model

is rigid where actions issued by the commander may not be executable by specific unit (e.g. an attack action is issued by the commander to all units, but some units may not have a target within the attacking range). Therefore these units will remain stationary and waste the precious turn. The information vector of the commander agent model is shown in Figure 5 with the state vector gathered from the environment and individual units. Individual units will strictly execute the action selected (Act1, Act2 or Act3) by the squad model.

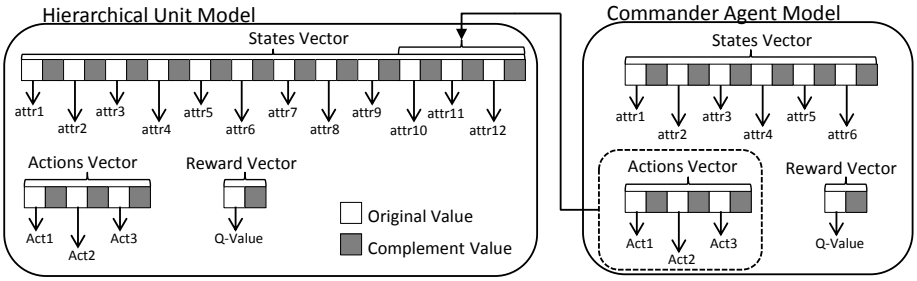


Figure 5: The relationship between actions vector of Commander agent model and the vector of a Unit agent model. The action selected by the Commander in its Actions vector (Act1, Act2, and Act3) is reflected in the last three attributes of the Unit agent (attr10, attr11, and attr12). These attributes correspond to the actions the Unit agent can perform as instructed by the Commander.

In this work, the commander agent's state vector **S** comprises six inputs, and hence the length of **S** is 12 (including 6 complement values), as shown in Table 3.

The actions available are identical to the unit agent actions (Attack, Advance, Fall back). The action from the commander is an overall command or strategy that is passed to the unit level that is strictly followed. The rules for the terminal reward function is given as follows.

if game ends with a win **then**
  reward will be 1.0
else if game ends with a loss **then**

Table 3: State attributes of commander agent.

No.	Attribute	Type/Description
1	Enemy in attack range	Boolean, true when one of the units detected an enemy within it's attacking range
2	Ally ratio	Real value, number of existing allies over all existing units including enemy units
3	Average ally health	Real value, average health of existing units normalised against full health
4	Current strength	Real value, current number of units over initial number of units
5	Ground advantage	Real value, total number of ally units on even or advantageous ground over total number of ally units
6	Ally closeness	Real value, normalised average distance between all existing units against unit seek range, 1.0 when normalised average distance is greater than unit seek range

reward will be 0.0

else if time out occurs **then**

reward will be 0.0.

end if

The rules of the intermediate reward function is given by

if any enemy is destroyed **then**

reward will be 1.0

else if any enemy is injured **then**

reward will be 0.75

else if any ally is injured **then**

reward will be 0.25

else if any ally is destroyed **then**

reward will be 0.0

else if there is a decrease in overall distance **then**

reward will be 0.75

else if there is an increase in overall distance **then**

reward will be 0.25

else if there is no change in distance **then**

reward will be 0.5.

end if

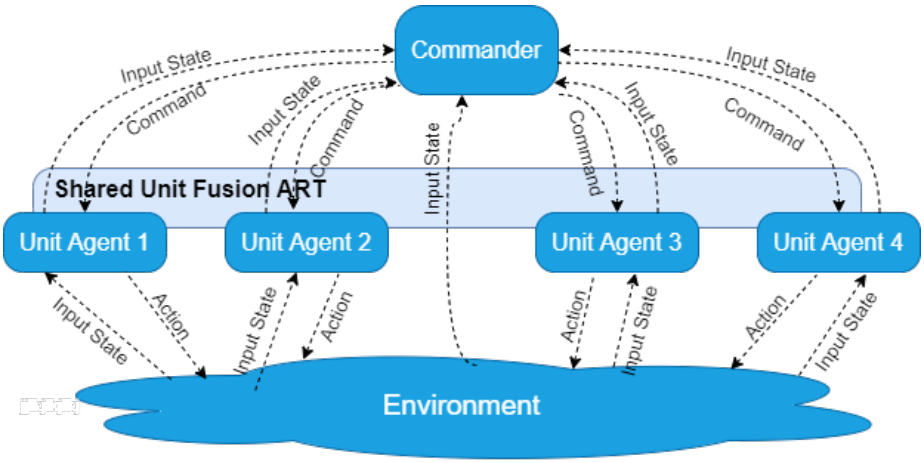


Figure 6: Hierarchical control architecture over four units wherein a Commander agent senses the state of the environment and the state of every Unit agent to issue a command to the Unit agents. Each Unit agent, in turn, senses the environment and perform an action accordingly.

5.3. The Hierarchical Model

As shown in Figure 6, the hierarchical model combines the unit agent model and the commander agent model in a two-level modular architecture. The integration of the two allows the combined model to have the benefits of learning and decision making based on the global situation as well as the flexibility of localised learning and decision making by individual unit agents. Instead of forcing each unit agent to follow strictly the action commands issued by the commander, the action vector of the commander (Act1, Act2, Act3) is fed into the individual units as part of their state vector (attr10, attr11, attr12) as

shown in Figure 5 hierarchical unit model. With the extended state vector, each unit agent performs its action selection cycle and selects the best action to be performed. The hierarchical unit model is allowed to perform the action as per instructed by the squad commander or overwrite the commander action with other more suitable actions.

The reward computation is performed for learning at each level of agent before the next action decision. In this work, the learning cycles for both level of agents are similar, called once at every 20 game frame. Each individual unit agent computes its reward based on the changes on self or environment detected over the period of the 20 game frames. These unit rewards may be included to determine the reward given to the commander agent that in turn refines the learned knowledge of the commander agent model. Reward of the commander may also be determined based on the achievement of the overall objectives of the action strategy. In our implementation, we compute the rewards based on the achievement of overall objectives as discussed described under the commander agent model section.

In Figure 6, it is indicated that at the unit agent level, a shared fusion ART model is employed to capture the knowledge learned by all the unit agents. However, each unit agent has its individual Q-Learning cycle which updates the common fusion ART model. By sharing the knowledge, learning efficiency can be enhanced. When the unit receives a feedback from the environment, it may update and learn the knowledge. The shared knowledge allows the change to be known and used by the other units on the fly.

6. Experiments

In this section, we describe the experiments conducted to test and evaluate the proposed model of hierarchical control for multiagent reinforcement learning. The experiments are based on StarCraft: Brood War game environment customized with particular maps, unit agents characters, and special missions for ally (the side of the reinforcement learning agent) units to accomplish. Using

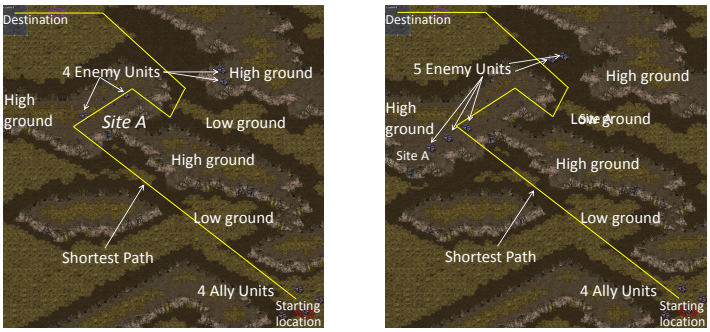
the customized game configurations, the first stage of experiments is to investigate the characteristics of the game environment against the reinforcement learning model in terms of partial observability and uncertainties. Here, we compare the learning performance between the agents with purely Fuzzy ART-based FALCON and those with hybrid Fuzzy ART-ART2 model when they run the mission.

The second stage of the experiments is to evaluate the proposed hierarchical control model for the learning performance. Different organization structures of the agents are compared including purely micromanagement learning by individual units, fully centralized control by a commander agent, and the proposed hierarchical control with unrestrictive directions from the commander. At this stage, the knowledge insertion feature of FALCON is also demonstrated. Two different map configurations with distinct difficulty levels are also applied to investigate the capability of the proposed model.

6.1. Experimental Environment

As a scaled-down version of StarCraft game, the goal (or mission) of the game task is for a team of ally agents to advance from its current location to reach the final destination with as many surviving units as possible. Along the path from the starting point of the ally forces (bottom right) to the destination (top left) are enemy units trying to deter the advancing objective of the ally units as shown in both maps in Figure 7. For simulating the ally and enemy units, siege tanks belonging to the Terran race is used. This is to exclude any effect of bias due to the differing abilities of different unit types in the experiment.

As mentioned earlier, most existing work on the StarCraft domain focused on combat scenarios on a flat terrain (Uriarte & Ontañón, 2015). To make the game more realistic, our experiments have included terrains with different heights, increasing the complexity of the environment. In addition, two different levels of game play were created. Scenario 1 has an equal number of four units for both ally and enemy (4v4), while in the more challenging scenario 2, the enemy force has the benefit of an additional unit (4v5). In both scenarios,



(Map 1) (Map 2)

Figure 7: Customized maps: (Map 1) 4 ally units versus 4 enemy units with two enemies are at a high ground and the other two are at a different high ground position; (Map 2) 4 ally units versus 5 enemy units with three and two enemies strategically placed at a high ground and a low terrain respectively.

ground advantage were given to the enemy side which increases the difficulty level of the simulations. The maps may seem simple, but in actuality, the enemy located on a higher terrain has the advantage of having a larger visibility range due to the fog of war and higher hit rate (see Figure 2). The level of difficulty is further increased in Map 2 by an additional enemy being placed at the higher ground at the *Site A* to defend the advancement route. This greatly increases the fire power of the enemy and reduces the time needed to destroy an ally unit.

6.2. Parameter Settings

For the purpose of consistency, in all our experiments, the default parameter values of FALCON are used without tuning as shown in Table 4. Those parameter values are chosen based on previous empirical investigation on finding the best parameter settings for FALCON reinforcement learning (Teng et al., 2015).

Specifically, the choice parameters $\{\alpha^1,\alpha^2,\alpha^3\}$, as in (6), are set to $\{0.1,0.1,0.1\}$. The contribution factors $\{\gamma^1,\gamma^2,\gamma^3\}$, as in (6), are set to $\{0.33,0.33,0.33\}$ implying that equal weightage is applied to all fields. As fast learning is employed in the experiments, it is reasonable to have learning rate parameters $\{\beta^1,\beta^2,\beta^3\}$ for template learning to be $\{1.0,1.0,1.0\}$. The vigilance parameters $\{\rho_s^1,\rho_s^2,\rho_s^3\}$ are set to $\{0.0,0.0,0.5\}$ for action selection and $\{\rho_l^1,\rho_l^2,\rho_l^3\}$ are set to $\{0.0,1.0,0.75\}$

Table 4: FALCON and Temporal Difference Parameters for Experiments

FALCON Parameters	
Choice parameters (6)	$\{\alpha^1,\alpha^2,\alpha^3\} = \{0.1,0.1,0.1\}$
Contribution parameters (6)	$\{\gamma^1,\gamma^2,\gamma^3\} = \{0.33,0.33,0.33\}$
Learning rate parameters (2)	$\{\beta^1,\beta^2,\beta^3\} = \{1.0,1.0,1.0\}$
Vigilance parameters (action selection)	$\{\rho_s^1,\rho_s^2,\rho_s^3\} = \{0.0,0.0,0.5\}$
Vigilance parameters (learning)	$\{\rho_l^1,\rho_l^2,\rho_l^3\} = \{0.0,1.0,0.75\}$
Temporal Difference Learning	
Learning rate (5)	$\alpha = 0.5$
Discount factor (4)	$\gamma = 0.1$

during learning. These choices of vigilance can still be justifiable since during action selection, the main concern is to find a code or rule that guarantees a good return. The code may only need to match with a few attributes from the input state implying zero state vigilance. In this case, the vigilance for action field is set to zero since its values will be replaced by the readout instead of matched. On the other hand, the vigilance parameters for action is maximum or 1 during learning as it must ensure to find a node that exactly match with the input action to learn.

For Q-learning, the parameters are also chosen empirically. The learning rate α in (5) is set to 0.5 and discount factor γ in (4) is set to 0.1.

6.3. Learning from Scratch Experiments

In this first set of experiments, we evaluate the use of fuzzy ART and ART2 operations in the unit agent model based on the Map 1 shown in Figure 7 as it learns from scratch. In particular, the experiments are designed to compare the model that employs fuzzy ART for all the fields with the hybrid fusion ART employing the fuzzy ART and ART2 operations for state field and reward field respectively.

Note that only the performance of fuzzy ART in the single successful run is shown here as the POMDP nature of the game may cause the reward values of the learned knowledge to be over generalized resulting in continuous time out. In comparison, all five runs of the unit agent model using ART2 operation in the reward field have achieved successful learning behavior as shown in Figure 8. As such, the standard deviation of the five runs are plotted for the unit agent model using ART2 operation only.

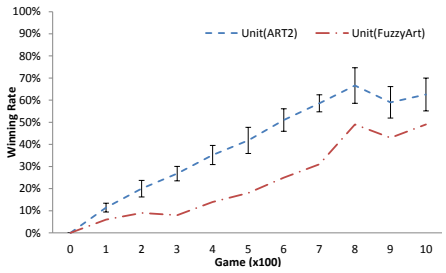


Figure 8: Comparing the performance of unit agent models using fuzzy ART and ART2 operations.

The results have suggested that ART2 operations is a better choice when the domain is not fully observable and has a POMDP nature. The following section further supports this observation.

6.4. Learning Hierarchical Control Experiments

In the second stage of experiment, the purpose is to compare the performance of the three model configurations, namely: the Unit agent model, the commander agent model and the hierarchical model. The unit agent model performs micromanagement of the low-level units and does not maintain cohesion in actions. The commander agent model, on the other hand, performs macromanagement of the high level control, where all low-level units strictly obey the centralized action commands received. Lastly, the hierarchical model combines both the unit agent and commander agent models. On top of evaluating the three agent models, the experiments also evaluate the use of fuzzy ART and ART2 operations in the underlying fusion ART models.

FALCON neural networks have the capability to integrate rule-based knowledge with reinforcement learning (Teng et al., 2015). This set of experiments illustrates the performance of the various models when rules are inserted before the learning. The two sets of rules inserted for the commander agent and the unit agent are shown in Table 5 and Table 6 respectively.

Table 5: Commander Agent Rules

No.	Conditions	Actions
1	Enemy not in range, ally ratio [0.5-1.0], average ally health [0.5-1.0], current strength [0.5-1.0], ground height [0.5-1.0]	Advance
2	Ally ratio [1.0-1.0], average ally health [1.0-1.0], current strength [1.0-1.0], ground height [0.5-0.5], ally closeness [0.5-1.0]	Advance
3	Ally ratio [1.0-1.0], average ally health [1.0-1.0], current strength [1.0-1.0], ground height [0.5-0.5], ally closeness [0.0-0.5]	Advance
4	Enemy in range, ground height [0.0-0.5]	Fall back
5	Enemy in range, ground height [0.5-1.0], ally closeness [0.5-1.0]	Attack
6	Enemy in range, current strength [0.5-1.0], ground height [0.5-1.0], ally closeness [0.0-0.0]	Attack

The reward value r for every inserted rule is set to 0.76 so that it is sufficiently high to be prioritized for selection over other learned nodes whenever the state matches with the input, but still allowing it to learn or create a new rule with a better reward during exploration. In this experiment, the inserted rules are considered as doctrines that must be followed by the units and the commander. In that case, they are made immutable and cannot be changed once they are inserted in the network. However, as Fusion ART allows new uncommitted nodes to be created or recruited, additional rules can still be acquired with similar state and reward value to the inserted ones though it may happen only by chance during exploration.

Table 6: Unit Agent Rules

No.	Conditions	Actions
1	Ground height [0.5-1.0], NO enemy seen	Advance
2	Ground height [0.5-1.0], enemy seen, ally close by, weapon cool down	Attack
3	Ground height [0.0-0.0], enemy seen, under attack, NOT weapon cool down	Fall back
4	Ground height [0.5-1.0], NO enemy seen, ally close by, Commander Attack command	Advance
5	Commander says Advance	Advance
6	Commander says Attack	Attack
7	Commander says Fall back	Fall back

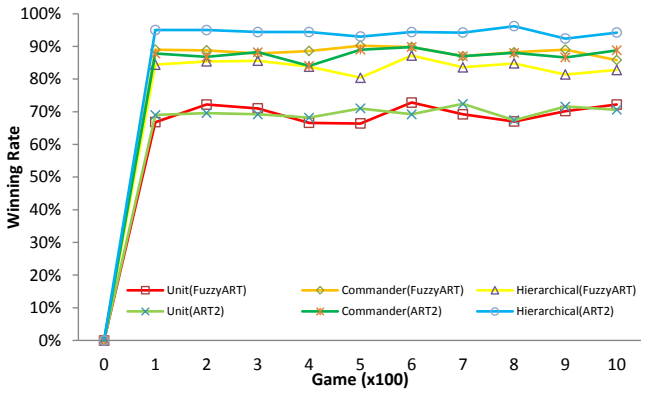


Figure 9: Success (winning) rates over 1000 game trials averaged across 5 runs for the 4v4 scenario in Map 1.

For the unit agent model configuration of experiment, only rules 1 to 3 applicable to unit agents will be inserted. On the other hand, for the commander agent model configuration, all rules applicable to the commander agent rules in Table 5 are used. Similarly, in the hierarchical model configuration, the commander agent applies all the listed commander rules.

6.5. Results and Discussion

As shown in Figure 9, the success rate of the various models (averaged over five independent runs) are all above 70%. While both the unit agent models using the fuzzy ART and ART2 operations achieved a stable 70%, the hierarchical model using ART2 operation achieved the best performances of 95%. The other models have also achieved a high performance of roughly 90% success rate with the inserted rules. This shows that the difficulty level of Map 1 may be too low and the inserted rules are sufficient to handle the game.

Note that the hierarchical model with the ART2 operation has achieved the best result compared to the commander agent model with ART2 operation, though they are both using the identical inserted rules. This shows the benefits of flexibility at the unit agent level that allow them to perform other more appropriate actions rather than strictly following the commander's action under the commander agent model.

In addition, the performance of the three different models have shown that unit cohesion may play a part in achieving better performance. A lower success rate of 70% was attained by unit agent model units largely due to the non-cohesive actions taken by units individually. On the other hand, units that either fully obey commander's command (in the commander agent model) or take into consideration the commander's command as additional input (in the hierarchical model) had achieved higher success rates of above 80% due to the unit movement cohesion. This is because, at the commander's level, information of the current observation from all the individual unit contributes to the attributes of the commander agent (e.g. ground advantage). This information can be used to select or learn the best action to take based on the inserted or learned rules.

The final set of experiments is conducted to evaluate the performance of the model in a more challenging configuration of the environment. The difficulty of the simulation is further increased as shown in Map 2, with an additional enemy unit located at *Site A*. The experiments are also independently rerun five times using the same parameters and inserted codes. With the increased difficulty, unit movement and attack cohesion becomes an important factor to achieve a

successful game in this map configuration.

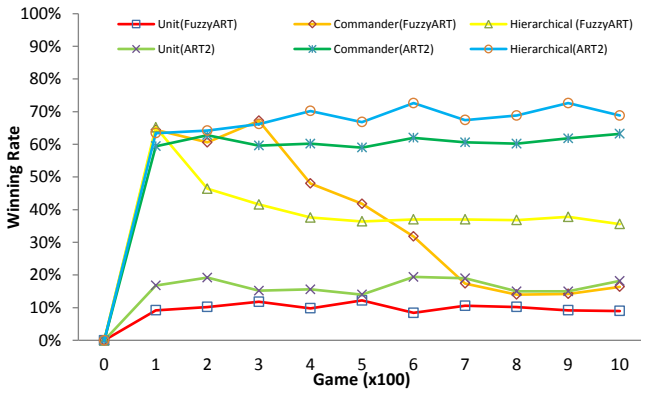


Figure 10: Success (winning) rate over 1000 games averaged across five runs for the 4v5 scenario in Map 2.

Figure 10 shows the performance under the Map 2 configuration. As the difficulty level of the customised map is increased, the overall success rate of the game is well expected to drop. Both the unit agent models using fuzzy ART and ART2 operations are shown to achieve below 20% winning rate. On the other hand, with the underlying mechanism of unit coordination, better performance can be obtained by the hierarchical model and the commander agent model.

The best performance is achieved by the hierarchical model followed by the commander agent model with both using the ART2 operations. These results have shown that with unit coordination, the success rate can be greatly improved, and even better with the hierarchical model over the commander agent model. This is because the hierarchical model retains the flexibility whereby individual units are allowed not to strictly follow the action given by the commander under certain conditions that may lead to a bad outcome at the unit level. Observations were made on situations wherein each unit has chosen an action according to each individual rules, but the result becomes less optimal or even detrimental to the entire team of the agents For example, in situation whereby unit A is on an even ground with the enemy, rule 2 (Table 6) is valid to select or execute. However, another nearby unit B that is at a disadvantageous

ground with the enemy, may instead choose to fall back (according to rule 3 in Table 4), leaving the unit A to face the enemy alone. With the advantages of being on the high ground, the enemy units have a nearly 100% hitting rate of their fire power on ally units on the low ground. Consequently, the ally units typically suffer much health level decline before engaging on any battle. In contrast, both the hierarchical model and the commander agent model mitigate such situations by instilling unit coordination. At the commander level, decisions are made based as seen on the overall situation. However, each individual unit is in its own situation which may not be suitable to follow the commander's instruction all the time. In that case, the flexibility the hierarchical model overcome this rigidity in the commander model.

The performance achieved by both the hierarchical model and commander agent model using fuzzy ART operation is notably poor. As the difficulty of the map is increased (Map 2), the uncertainty within the game is also increased. In particular, the enemy units have a longer range of visibility and higher fire power when they are on a higher ground than the ally units that are mostly placed in low ground. In this case, an action, like attacking, by one ally unit that should lead to a good outcome, may be a bad one for another in a similar situation. As a result, the rules learned using the fuzzy ART operations may be over-generalized, leading to the decline in the success rates. Specifically, it is observed that although generalization based on the complement-coded vectors with fuzzy ART operations is effective for learning the state representation, it is not so for learning the uncertain reward values. On the other hand, the agent models using ART2 operations for learning in the reward field are able to adapt better with the uncertainty, leading to a more stable level of performance.

More observations on the relationship of rules learned with the resulting winning rate have also indicated that striking a balance between generalizing state attributes of a rule and keeping specific attributes of an existing code may determine the effectiveness of the learned rules to achieve the domain objectives. Overgeneralization, even on a single attribute, can be detrimental to the corresponding rule to take effect. For example, as shown in Table 5, rule number 2

and 3 are specific and quite similar in terms of the action to take (Advance) and the state attributes except for the attribute representing the ally closeness (unit cohesiveness). However, making the two as a single rule by generalizing the closeness attribute to ignore any value of it and to match with its entire range of closeness (from 0 to 1) may instead make the rule seldom to be selected. The cause can be that there is another rule with more generalized attributes but still has more chance to be selected as it has always higher activation values in general. In this case, the generalization may instead cause the rule to be ineffective especially during action selection wherein the vigilance $\rho_s^1 = 0$, bypassing the template matching process. In this case, a good rule, either learned or pre-inserted should have most of its attributes generalized (with ranges of values) to a certain extent but not too much (e.g taking the entire range of values).

7. Conclusion

This paper has presented a hierarchical learning and control model for coordinating the adaptation and performance of multiple self-organizing learning agents in real time. The empirical results obtained based on the StarCraft domain have shown that the hierarchical model, combining the learning ability at the macromanagement and micromanagement levels, is more robust than individual unit agent models learning at the micro level. When compared to a commander agent model at the macro level, the hierarchical model has the advantage of flexibility, whereby each individual unit is still able to learn better micro-level strategies rather than strictly following the central commander's instructions. This flexibility can be observed from both the map scenarios in our experiments, wherein the hierarchical model consistently produces a better level of performance.

Regarding the specific model choice of self-organizing neural models, the suitability of ART2 choice and matching functions over the fuzzy ART operations in a POMDP domain, such as StarCraft game, has been observed through the comparative experiments conducted. Due to the unidirectional fuzzy learning

rule, over-generalisation in fuzzy ART has resulted in only one out of five runs successfully achieving a proper learning. It also led to degrading of performance, in terms of winning rates, in the knowledge inserted experiments.

Moving forward, the management of complex combat scenarios in RTS games is still an open research problem as most work in the literature only explores a limited set of actions such as moving, attacking and fleeing. Extending the hierarchical model into a full fledged bot to play the entire game will therefore be a great challenge. The extension may also include mutable inserted rules to investigate the effect of adaptive doctrines to the overall performance during learning. To handle a more complex problem domain, different learning strategies, including deep learning, could be explored in our future work for learning compressed representation in the commander as well as the unit agents.

Acknowledgement

This work was supported by the DSO National Laboratories, Singapore under grant DSOCL16006.

References

Buro, M. (2003). Real-time strategy games: A new AI research challenge. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 2003)* (pp. 1534–1535).

Chu, T., Chinchali, S., & Katti, S. (2020). Multi-agent reinforcement learning for networked system control. In *Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020)*. URL: `https://openreview.net/pdf?id=Syx7A3NFvH`.

Di, W., & Tan, A.-H. (2015). Creating autonomous adaptive agents in a real-time first-person shooter computer game. *IEEE Transactions on Computational Intelligence and AI in Games*, *7*, 123–138.

Doxygen (2017). BWAPI 4.1.2. URL: `https://bwapi.github.io/`.

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H. S., Kohli, P., & Whiteson, S. (2017). Stabilising experience replay for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)* (pp. 1146–1155).

Gabriel, I., Negru, V., & Zaharie, D. (2012). Neuroevolution based multi-agent system for micromanagement in real-time strategy games. In *Proceedings of the Fifth Balkan Conference in Informatics (BCI 2012)* (p. 32–39). New York, NY, USA. URL: https://doi.org/10.1145/2371316.2371324. doi:10.1145/2371316.2371324.

Gronauer, S., & Diepold, K. (2021). Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, . doi:10.1007/s10462-021-09996-w.

Hassabis, D. (2017). Artificial intelligence: Chess match of the century. *Nature*, *544*, 413–414.

Jurenka.sk (2014). Bwmirror javadoc. URL: http://bwmirror.jurenka.sk/javadoc/index.html.

Kumar, S., Shah, P., Hakkani-Tur, D., & Heck, L. (2017). Federated control with hierarchical multi-agent deep reinforcement learning. *ArXiv*, *abs/1712.08266*. URL: http://arxiv.org/abs/1712.08266.

Lowe, R., Wu, Y., Tamart, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS 2017)* (pp. 6382–6393).

Martino, L., & Elvira, V. (2017). *Metropolis Sampling*. doi:10.1002/9781118445112.stat07951.

Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., & Preuss, M. (2013). A survey of real-time strategy game AI research and competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games*, *5*, 293–311.

Ontanon, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., & Preuss, M. (2015). RTS AI problems and techniques. In *Encyclopedia of Computer Graphics and Games*. Springer. doi:10.1007/978-3-319-08234-9_17-1.

Peng, P., Yuan, Q., Wen, Y., Yang, Y., Tang, Z., Long, H., & Wang, J. (2017). Multiagent bidirectionally-coordinated nets for learning to play StarCraft combat games. *ArXiv*, *abs/1703.10069*.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., & Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)* (pp. 4295–4304).

Robertson, G., & Watson, I. (2014). A review of real-time strategy game AI. *AI Magazine*, *35*, 75–104. URL: https://www.aaai.org/ojs/index.php/aimagazine/article/view/2478. doi:10.1609/aimag.v35i4.2478.

Shantia, A., Begue, E., & Wiering, M. (2011). Connectionist reinforcement learning for intelligent unit micro management in StarCraft. In *The 2011 International Joint Conference on Neural Networks* (pp. 1794–1801).

Tan, A.-H. (2004). FALCON: A fusion architecture for learning, cognition, and navigation. In *Proceedings of 2004 IEEE International Joint Conference on Neural Networks (IJCNN 2004)* (pp. 3297–3302).

Tan, A.-H. (2007). Direct code access in self-organizing neural networks for reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI 2007)* (p. 1071–1076). San Francisco, CA, USA.

Tan, A.-H., Carpenter, G., & Grossberg, S. (2007). Intelligence through interaction: Towards a unified theory for learning. In *Proceedings of the International Symposium on Neural Networks (ISNN 2007)* (pp. 1094–1103). Springer. doi:10.1007/978-3-540-72383-7_128.

Tan, A.-H., Lu, N., & Xiao, D. (2008). Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback. *IEEE Transactions on Neural Networks*, *19*, 230–244.

Tan, A.-H., Subagdja, B., Wang, D., & Meng, L. (2019). Self-organizing neural networks for universal learning and multimodal memory encoding. *Neural Networks*, *120*, 58–73.

Teng, T.-H., Tan, A.-H., & Teow, L.-N. (2013). Adaptive computer-generated forces for simulator-based training. *Expert Systems with Applications*, *40*, 7341–7353. doi:10.1016/j.eswa.2013.07.004.

Teng, T.-H., Tan, A.-H., & Zurada, J. M. (2015). Self-organizing neural networks integrating domain knowledge and reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, *26*, 889–902.

Uriarte, A., & Ontañón, S. (2015). A benchmark for StarCraft intelligent agents. In *Artificial Intelligence in Adversarial Real-Time Games: Papers from the AIIDE 2015 Workshop*.

Usunier, N., Synnaeve, G., Lin, Z., & Chintala, S. (2017). Episodic exploration for deep deterministic policies: An application to StarCraft micromanagement tasks. In *Proceedings of the 5th International Conference on Learning Representation (ICLR 2017)*. URL: https://openreview.net/forum?id=r1LXit5ee.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., & Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*, 350–354.

Wender, S., & Watson, I. (2012). Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft:Broodwar. In *2012 IEEE Conference on Computational Intelligence and Games (CIG 2012)* (pp. 402–408).

Wikipedia (2019). StarCraft: Brood war. URL: `https://en.wikipedia.org/wiki/StarCraft:_Brood_War`.

Xiao, D., & Tan, A.-H. (2013). Cooperative reinforcement learning in topology-based multi-agent systems. *Autonomous Agents and Multi-Agent systems*, *26*, 86–119.

Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., Shanahan, M., Langston, V., Pascanu, R., Botvinick, M., Vinyals, O., & Battaglia, P. (2018). Relational deep reinforcement learning. *ArXiv*, *abs/1806.01830*. URL: `http://arxiv.org/abs/1806.01830`.