Creating Autonomous Adaptive Agents in a Real-Time First-Person Shooter Computer Game

Article *in* IEEE Transactions on Computational Intelligence and AI in Games · June 2015

DOI: 10.1109/TCIAIG.2014.2336702

2 authors:

Di Wang
Nanyang Technological University
43 PUBLICATIONS **280** CITATIONS
SEE PROFILE

Ah-Hwee Tan
Nanyang Technological University
232 PUBLICATIONS **3,946** CITATIONS
SEE PROFILE

Creating Autonomous Adaptive Agents in a Real-Time First-Person Shooter Computer Game

Di Wang, *Member, IEEE* and Ah-Hwee Tan, *Senior Member, IEEE*

***Abstract*—Games are good test-beds to evaluate AI methodologies. In recent years, there has been a vast amount of research dealing with real-time computer games other than the traditional board games or card games. This paper illustrates how we create agents by employing FALCON, a self-organizing neural network that performs reinforcement learning, to play a well-known first-person shooter computer game called Unreal Tournament. Rewards used for learning are either obtained from the game environment or estimated using the temporal difference learning scheme. In this way, the agents are able to acquire proper strategies and discover the effectiveness of different weapons without any guidance or intervention. The experimental results show that our agents learn effectively and appropriately from scratch while playing the game in real-time. Moreover, with the previously learned knowledge retained, our agent is able to adapt to a different opponent in a different map within a relatively short period of time.**

***Index Terms*—Reinforcement learning, real-time computer game, Unreal Tournament, Adaptive Resonance Theory operations, temporal difference learning.**

I. Introduction

MODERN video games have become a core part of the entertainment world today. The rapidly growing global game industry is valued about 78.5 billion US dollars in 2012, approximately 62% of the global movie industry [1].

Traditionally, game developers tend to utilize scripting techniques, finite state machines, rule-based systems, or other such knowledge intensive approaches to model Non-Player Characters (NPCs) [2]. These approaches often lead to two major limitations. First of all, no matter how skilled the developers are and no matter how long the games have been play-tested before release, the existence of unseen circumstances is unavoidable [3]. In such situations, the decisions of the NPCs are unpredictable and often undesired. As such, it is common nowadays that popular games periodically release series of patches or updates to correct those previously undiscovered loopholes. The second limitation is that these knowledge intensive approaches are static in nature. The behaviors of the NPCs usually follow a few fixed patterns. Once the player learns their patterns and discovers their weaknesses, the game is considered boring, less fun, and less challenging [4].

Furthermore, from the player point of view, invincible NPCs are not preferred [5], [6]. People find it is more enjoyable to play with or against NPCs that try to imitate human players who have flaws but are able to learn from mistakes. If NPCs are able to evolve and dynamically adjust themselves according to the interaction outcomes between different players, the level of player satisfaction increases drastically [7].

Modern video games, especially First-Person Shooter (FPS) computer games, involve complex and frequent interactions between players and NPCs. Among all the machine learning approaches, FPS game environments are naturally suited for reinforcement learning rather than unsupervised or supervised learning. As reinforcement learning enables the NPCs to be rewarded or be penalized according to different interaction outcomes that are directly derived from the game environment, this trial-and-error philosophy resembles the natural way of human learning and suits complex environments well [8].

We create agents (in this paper, the term NPC is interchangeably used as agent or bot) that employ FALCON (Fusion Architecture for Learning, COgnition, and Navigation) networks [9] to play an FPS computer game. FALCON is a generalization of the Adaptive Resonance Theory (ART) [10] to perform reinforcement learning. By utilizing respective FALCON networks, our agents learn and apply rules for both behavior modeling and weapon selection during run time.

Our game of choice is Unreal Tournament 2004 (UT2004), which is a well-known commercial FPS computer game. UT2004 has been directly used as the application domain of much research work (elaborated in Section II). The UT2004 game server provides interfaces for two-way communications, such as passing the relevant game information to the user and receiving user commands to control the avatar in the game. To make our implementation work easier, we use Pogamut [11], which is a freeware to facilitate rapid developments of agents embodied in UT2004.

To deal with the partially-observable information (elaborated in Section IV-C) provided by UT2004, in this paper, we propose a set of combinatorial operations for FALCON networks to replace the conventionally applied operations. Our agents employ two respective FALCON networks to learn how to select appropriate behaviors and how to choose effective weapons under different circumstances. In the experiment section, we first show that the proposed combinatorial learning operations enable our agents to learn (from scratch in real-time) more appropriately in terms of behavior modeling and effectively in terms of weapon preferences. After demonstrating how different values of the two critical FALCON parameters affect the overall performance in one game configuration, we apply the same set of parameter setting to three more game configurations, where our agents play against different opponents in different maps, to evaluate the performance and more importantly, to show the general set

The authors are with the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (email: {wangdi, asahtan}@ntu.edu.sg)

of parameter values can be commonly adopted in different game configurations. Furthermore, we conduct experiments of our agent playing against the new opponent in the new map with pre-inserted knowledge retained from playing against the previous opponent in the previous map. The statistical test shows that after convergence, the agent adapting the pre-inserted knowledge performs at the same level as the agent learning from scratch, however, within a much shorter period of time. Therefore, based on the carefully defined learning algorithms, we show that the general set of parameter values can be commonly adopted in different game configurations and our agents are able to quickly adapt to new opponents in new maps if the previously learned knowledge is retained.

The rest of this paper is organized as follows. First, we review related work. Then we talk about the chosen FPS computer game (UT2004) and its corresponding development software (Pogamut). Next, we introduce FALCON as the reinforcement learning methodology and the variations we made to enable our agents to learn appropriately and effectively. Subsequently, we provide the experimental results with ample illustrations and discussions. Finally, we conclude this paper.

II. Related Work

In computer games, adaptive NPCs are beneficial to both developers and players. Integrating various intelligent methodologies into NPCs is one feasible way to meet this demand [12]. In this section, we review the related research work.

Spronck *et al.* [13], [14] use dynamic scripting to create different NPCs (with different scripts, i.e. rules, according to both tactics and level of experiences of the opponents) to fight against different players. This approach could increase the satisfaction level of the human players because their opponents are dynamic. However, the NPCs are fixed after creation. They do not evolve in real-time.

Cook *et al.* [15] use a graph-based relational learning algorithm to extract patterns from human player graphs and apply those patterns to agents. This work shows how human knowledge could be transferred, but it cannot improve the intelligence of the agent.

Many researchers use Genetic Algorithms (GAs) to automatically tune the parameters used in games [16]–[19]. These parameters include weapon preferences, priorities of targets, and different level of aggressiveness. Although the performance is getting better and better through generations, similar to reinforcement learning with function approximation, GA does not guarantee the final solution to be globally optimal. Moreover, even for a satisfactory sub-optimal solution, GA often takes an unnecessarily long time for a real-time computer game.

Among the aforementioned work, both commercial computer games and self-implemented platforms are applied. There are also much research work that focus on the same application domain as ours, the UT2004 FPS computer game.

Hy *et al.* [20] define a way to specify various behaviors of the agents, such as aggressive, caution, and normal. These behaviors could be tuned by adjusting the corresponding probability distributions, which is a straightforward way to use only probability equations to determine the next action instead of writing and maintaining scripts. However, the parameters defined are all based on heuristics or experiences, which often involve human bias because different people may have different cognitions and judgments for different behaviors.

Kim [21] proposes a finite state machine to switch between different behaviors of the agents according to the context-sensitive stimuli received from the game environment. However, this architecture is rigidly defined and the agents are hard-coded. They always perform the same action under similar circumstances and are not able to evolve.

There are also research work [8], [22] closely related to our approach, wherein reinforcement learning is applied to create agents in UT2004. However, they study the collaboration strategies for an entire team of agents in the Domination game scenario. In contrast, we focus on the behavior selections and the weapon preferences of a single agent in the 1-on-1 DeathMatch game scenario. Furthermore, there are a series of research work related to the BotPrize competition[1] [5], which is like a Turing Test for bots that evaluates whether computer programs are able to convince human judges that they are the actual human players. The basic idea behind one of the two winner bots is to mimic the actions of the other opponents (human players and bots) that have been recorded in the short-term memory module. The other winner bot (previous version presented in [23]) applies evolutionary algorithms to optimize combat behaviors and follows the traces of human players to navigate in the map. Much work on human-like bots are presented in [24]. To numerically evaluate humanness, Gamez *et al.* [25] propose a measure to combine scores from various sources into a single number. Their bot utilizes a relatively large spiking neural network with a consequently large number of connections as the global workspace to resemble the human brain architecture and to perform human-like behaviors in real-time. Different from many bots that focus on the human-like behaviors, our bot focuses on the capability to select appropriate behaviors and choose effective weapons under different circumstances. The rewards used for learning are either directly obtained from the interaction outcomes in the game environment or estimated using the temporal difference learning scheme. Therefore, our bot does not need any role model to learn from. Although focusing on how to defeat the opponents, our bot is also human-like in the sense that it learns how to select appropriate behaviors and choose effective weapons according to the perceived interaction outcomes through trial-and-error, like humans do.

III. UT2004 and Pogamut

UT2004 is a popular FPS computer game, which provides an environment for embodying virtual agents. There are many game scenarios available and the most commonly played ones are the DeathMatch, Domination, and Capture The Flag (CTF) scenarios. Fig. 1 shows a screen snapshot of the game taken from the spectator point of view.

[1]All details and archives of the BotPrize competition are available online: http://www.botprize.org/

Fig. 1. A screen snapshot of the Unreal Tournament 2004 game. `FALCONBot` (the bottom one) is engaging fire with `advancedBot` in the Idoma map.

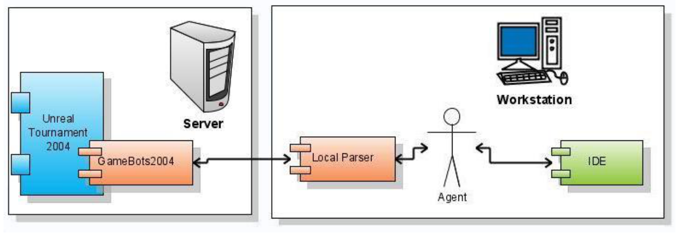


Fig. 2. Pogamut architecture overview (picture excerpted from Pogamut website, https://artemis.ms.mff.cuni.cz/pogamut/).

In UT2004, a human player uses a keyboard and a mouse to control the actions of an avatar in a 3-D confined virtual world. On the other hand, the actions of an agent are performed when specific commands are received by the game server. These actions are primitive ones: run, jump, turn, shoot, change weapon, etc. Both players and agents have many high-level tasks in the game: killing opponents, collecting items, securing areas, etc. The winning criterion is different for different game scenarios. In terms of the DeathMatch scenario, the first individual who obtains a predefined game score wins the game. One point is awarded when any individual kills an opponent and deducted when any individual commits suicide (killed by the collateral damage of own weapon or killed by the environment, e.g. by falling from high ground, or by entering a pool of lava). Getting killed in the game is a minor setback, since an individual re-enters the game with a health level of 100 points (which can be boosted by collecting special health items to 199) and the two most primitive weapons (those collected in the previous life are removed). Therefore, always staying healthy and possessing powerful weapons with ample ammunition results in better chances of winning.

Pogamut [11] is an Integrated Development Environment (IDE) and a plug-in for the NetBeans Java development environment. Pogamut communicates to UT2004 through Gamebots 2004 (GB2004) [26]. GB2004 is an add-on modification written in UnrealScript (the scripting language of UT2004), which delivers information from the game to the agent and vice

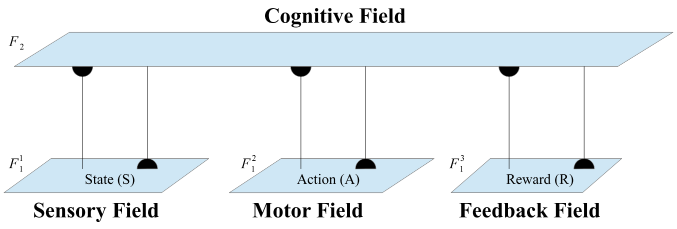


Fig. 3. FALCON network structure.

versa. Because GB2004 only exports and imports text messages, a parser is required for the translation tasks. Pogamut has a built-in parser module, which automatically converts the text messages to Java objects and vice versa. The architecture of Pogamut is shown in Fig. 2.

It is convenient to implement an agent that plays UT2004 using Pogamut, which provides templates for various types of simple agents. The user only needs to define and override a few functions to develop a limited but functional agent. Other than templates, there are also competitive agents provided by Pogamut. They are good examples to follow and could be used for benchmarking purposes. In the experiment section, we employ two sample bots, namely `advancedBot` and `hunterBot`, as the opponents of our `FALCONBot`.

IV. FALCON AND TEMPORAL DIFFERENCE LEARNING

A Fusion Architecture for Learning, COgnition, and Navigation (FALCON) is proposed in [9]. It employs a 3-channel architecture (see Fig. 3), comprising a cognitive (high-level) field (F_2) and three input (low-level) fields. The three input fields include a sensory field (F_1^1) representing states, a motor field (F_1^2) representing actions, and a feedback field (F_1^3) representing reinforcement values. FALCON is a self-organizing neural network based on the Adaptive Resonance Theory (ART) [10], which means it is able to evolve systematically to incorporate new information. FALCON networks can either learn from scratch or learn with pre-inserted knowledge. When learning from scratch, the FALCON network consists of only one uncommitted code (all weight values are set to 1) in the cognitive field. When learning with pre-inserted knowledge, the FALCON network consists of a number of committed codes (each code represents an inserted rule, whose values vary in the $[0, 1]$ interval) and one uncommitted code.

There are two major processes to utilize FALCON networks, namely knowledge retrieval and knowledge update. When we need to retrieve certain knowledge, we present the currently known vectors of the input fields to retrieve the winner code in the cognitive filed. For example, if we want to find out which action according to the current state will receive the maximum reward of 1, we present the state vector and the reward vector $(1, 0)$ to the FALCON network and consequently read out the action vector of the winner code retrieved from the cognitive field. The process to find the winner code is briefly described as follows. According to the current input vectors, for each code in the cognitive field, the activation value is computed. The winner code is then selected to be the one having the largest activation value among all the codes that fulfil the vigilance criterion. The equations are defined in such a way that an uncommitted code will always be selected as the winner if all the committed codes do not fulfil the vigilance criterion. Following the aforementioned example, if a committed code is retrieved, the action stored in the winner code will be performed. If an uncommitted code is retrieved, a random action will be performed instead.

When we need to learn certain knowledge, we present all vectors of the three input fields and perform the knowledge retrieval process to find the winner code in the cognitive filed. Either a committed code (closely similar knowledge has been learned before) or an uncommitted one (the presented information is new) will be retrieved. The currently presented information is then updated to the knowledge stored in the winner code. The update of information procedure is named template learning. If template learning is performed, the uncommitted code becomes committed and FALCON will automatically create a new uncommitted code in the cognitive field for future usage. Thus, FALCON expands its network architecture dynamically in response to the ongoing new input patterns. A detailed example is given later in Section V-F.

Most existing FALCON networks [9], [27]–[30] apply fuzzy ART operations [31] (elaborated in Section IV-A), which require the input vectors to include the complement values of the originals. For example, we can assume the weight vector of the winner code for a particular field is retrieved as $(0.2, 0.8)$. This value only represents one exact point: 0.2 with its complement of 0.8. If the currently presented input vector of that field $(0.1, 0.9)$ is to be learned with the learning rate $\beta = 0.1$. The updated weight vector will become $(0.19, 0.8)$ (detailed equations are given in Section IV-A). Thus, it now represents a region from 0.19 to 0.2 (the complement of 0.8). Because each code learned with fuzzy ART operations is generalized to represent a certain area instead of an exact point, in general, fuzzy ART operations produce a smaller number of cognitive codes than other types of operations do. Applying fuzzy ART operations to reduce the number of learned codes is referred as the method to avoid the code proliferation problem [32]. FALCON networks can also apply ART2 operations [33], [34] (elaborated in Section IV-B), which use exact values to represent knowledge. However, in UT2004, we find FALCON networks learn more appropriately and effectively with our proposed combinatorial operations (elaborated in Section IV-C), which combines fuzzy ART and ART2 operations. Furthermore, to estimate the reward received in the current state if the selected action is performed, temporal difference learning is incorporated in TD-FALCON networks [35] (elaborated in Section IV-D).

A. FALCON Network with Fuzzy ART Operations

The generic network dynamics of FALCON, based on fuzzy ART operations [31], is described as follows:

1) Input vectors: Let $\mathbf{S} = (s_1, s_2, \ldots, s_n)$ denote the state vector, where $s_i \in [0, 1]$ indicates the sensory input i. Let $\mathbf{A} = (a_1, a_2, \ldots, a_m)$ denote the action vector, where $a_i \in [0, 1]$ indicates the preference of action i. Let $\mathbf{R} = (r, \bar{r})$ denote the reward vector, where $r \in [0, 1]$ is the reward signal value and $\bar{r} = 1 - r$. Complement coding serves to normalize the magnitude of the input vectors and has been found effective in fuzzy ART systems in preventing the code proliferation problem [32]. Since all the values used in FALCON are within the $[0, 1]$ interval, normalization is often required.

2) Activity vectors: Let $\mathbf{x}^k$ denote the F_1^k activity vector for $k = 1, 2, 3$. Let $\mathbf{y}$ denote the F_2 activity vector.

3) Weight vectors: Let $\mathbf{w}_j^k$ denote the weight vector associated with the j^{th} code in F_2 for learning the input patterns in F_1^k for $k = 1, 2, 3$.

4) Parameters: The dynamics of FALCON is determined by choice parameters $\alpha^k > 0$ for $k = 1, 2, 3$; learning rate parameters $\beta^k \in [0, 1]$ for $k = 1, 2, 3$; contribution parameters $\gamma^k \in [0, 1]$ for $k = 1, 2, 3$, where $\sum_{k=1}^{3} \gamma^k = 1$; and vigilance parameters $\rho^k \in [0, 1]$ for $k = 1, 2, 3$.

5) Code activation (Fuzzy ART): A bottom-up propagation process first takes place in which the activities (the choice function values) of the cognitive codes in F_2 are computed. Specifically, given the activity vectors $\mathbf{x}^1$, $\mathbf{x}^2$, and $\mathbf{x}^3$ (in F_1^1, F_1^2, and F_1^3, respectively), for each F_2 code 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 $p_i \wedge q_i \equiv min(p_i, q_i)$, and the norm $|.|$ is defined by $|\mathbf{p}| \equiv \sum_i p_i$ for vectors $\mathbf{p}$ and $\mathbf{q}$. In essence, T_j computes the similarity between the input vectors and the weight vectors of the F_2 code j with respect to the norm of the weight vectors.

6) Code competition: A code competition process follows under which the F_2 code with the highest choice function value is identified. The winner is indexed at J where

$$T_J = \max\{T_j : \text{for all } F_2 \text{ code } j\}. \tag{2}$$

When a choice is made at code J, $y_J = 1$ and $y_j = 0, \forall j \neq J$. This indicates a winner-take-all strategy.

7) Template matching: Before code J can be named as the winner, a template matching process checks whether the weight templates of code J are sufficiently close to their respective activity patterns. Specifically, resonance occurs if for each channel k, the match function m_J^k of the chosen code J meets its vigilance criterion (defined in Eq. (3)). Choice and match functions work co-operatively to achieve stable coding and control the level of code compression.

$$m_J^k = \frac{|\mathbf{x}^k \wedge \mathbf{w}_J^k|}{|\mathbf{x}^k|} \geq \rho^k. \tag{3}$$

If any of the vigilance constraints are violated, mismatch reset occurs in which the value of the choice function T_J is set to 0 for the duration of the input presentation. The search and evaluation process is guaranteed to end because FALCON will either find a committed code that satisfies the vigilance criterion or activate the uncommitted code (all weight values equal to 1) that definitely satisfies the criterion.

8) Template learning (Fuzzy ART): Once a code J is selected, for each channel k, the weight vector $\mathbf{w}_J^k$ is updated by the following learning rule:

$$\mathbf{w}_J^{k(\text{new})} = (1 - \beta^k)\mathbf{w}_J^{k(\text{old})} + \beta^k(\mathbf{x}^k \wedge \mathbf{w}_J^{k(\text{old})}). \tag{4}$$

The learning function adjusts the weight values towards the fuzzy norm (AND) result of their original values and the respective input values. The rationale is to learn by encoding the common attribute values of the input vectors and the weight vectors. For an uncommitted code, the learning rates β^k are typically all set to 1. For committed codes, β^k are set to high values for fast learning in straightforward situations or small values for slow learning in noisy environments.

B. FALCON Network with ART2 Operations

Instead of fuzzy ART operations, FALCON network can incorporate ART2 operations [33], [34] to avoid possible over-generalization (elaborated in Section IV-C). To change from fuzzy ART to ART2 operations, only two functions need to be altered.

1) Code activation (ART2): The ART2 code activation function defines the difference between the input vector $\mathbf{x}^k$ and the weight vector $\mathbf{w}_j^k$ as their cosine similarity measure:

$$T_j = \sum_{k=1}^{3} \gamma^k \frac{\mathbf{x}^k \cdot \mathbf{w}_j^k}{\|\mathbf{x}^k\| \|\mathbf{w}_j^k\|}, \tag{5}$$

where the operation $\cdot$ is the dot product and the norm $\|.\|$ is defined by $\|\mathbf{p}\| \equiv \sqrt{\sum_i p_i^2}$.

2) Template learning (ART2): While the Fuzzy ART template learning function (Eq. (4)) updates the new weight vector $\mathbf{w}_J^{k(\text{new})}$ with the generalization of the input vector and the old weight vector $\mathbf{x}^k \wedge \mathbf{w}_J^{k(\text{old})}$, the ART2 template learning function only learns towards the input vector $\mathbf{x}^k$:

$$\mathbf{w}_J^{k(\text{new})} = (1 - \beta^k)\mathbf{w}_J^{k(\text{old})} + \beta^k \mathbf{x}^k. \tag{6}$$

C. Combinatorial Operations for Effective Learning

Fuzzy ART operations work well in generalization, especially in deterministic scenarios such as the mine-field navigation [9] and the mission-on-mars exploration [29]. In those applications, the same choice of action made by the agent under similar circumstances always gets similar rewards. Therefore, generalization of similar information does not incur information loss and it helps to maintain a smaller set of rules.

However, FPS games are not identical to those application domains. In this kind of game, certain information such as the health of the opponent is often not provided to the players. In this sense, UT2004 is a Partially-Observable Markov Decision Process (**POMDP**) [36]. As a consequence, perceptual aliasing occurs and distinct states are sensed via identical feature vectors. For example, whether an agent kills its opponent or gets killed does not only depend on the agent itself, but also depends on the status of the opponent (such as health), which is unknown to the agent. In this case, if the agent is facing closely similar (self-sensed) situations at different times and performs the same action, the outcomes of the interactions between it and its opponent can be different, sometimes exactly opposite. Generalization of the contradictory information is dangerous and may lead to incorrect results (because information learned is ambiguous) or lead to the creation of unnecessarily large number of redundant rules (because rules become less-representative if over-generalized).

Mathematically speaking, we can derive Eq. (7) from Eq. (4):

$$\mathbf{w}_J^{k(\text{new})} = \mathbf{w}_J^{k(\text{old})} \wedge (\mathbf{w}_J^{k(\text{old})} - \beta^k(\mathbf{w}_J^{k(\text{old})} - \mathbf{x}^k)). \tag{7}$$

It is obvious to notice from Eq. (7) that the new weight vector $\mathbf{w}_J^{k(\text{new})}$ will always be less than or equal to the old weight vector $\mathbf{w}_J^{k(\text{old})}$. Therefore, using fuzzy ART template learning function can only generalize information. Moreover, once generalized, it cannot be specialized afterwards.

It is our intention to generalize the sensory field input vectors of the FALCON network (see Fig. 3). Therefore, similar situations are grouped together and only one vector is required to represent all of them. The input vectors are Boolean for the motor field and they require exact matches. Therefore, applying either Eq. (4) or Eq. (6) makes no differences here. Problems occur when the feedback field input vectors are generalized. In the FPS game domain, for the same pair of state and action, the reward can vary drastically. A good example is assuming the agent kills its opponent at one time. For another time, the agent may have the same state vector (because the health level of the opponent is unknown) as before and choose the same action, but this time itself gets killed. In this POMDP [36], the quality of one rule may highly depend on its initial values. Recall that when an uncommitted code becomes committed, the weight vector is assigned to the input vector $\mathbf{w}_J^{k(\text{new})} = \mathbf{x}^k$ (learning rates all equal to 1). After setting the initial weight vector, a small β means the learning is slow and the correctness of this code highly depends on its initial value. If β is set to a high value, when contradictory outcomes arrive, the code will soon become over-generalized, less-representative, or even misleading. Moreover, no matter how β is set, a code will become redundant eventually if contradictory input vectors keep coming in. Because once a code is over-generalized, it will always fail the vigilance criterion defined in Eq. (3).

Therefore, in order to make our agents generalized and at the same time to ensure they learn appropriately and effectively, we propose to combine the fuzzy ART and ART2 operations. Fuzzy ART operations are applied to the state ($k = 1$) and action ($k = 2$) vectors to preserve generalization and ART2 operations are applied to the reward ($k = 3$) vectors for effective learning. The updated code activation and template learning functions are respectively defined as follows:

$$T_j = \sum_{k=1}^{2} \gamma^k \frac{|\mathbf{x}^k \wedge \mathbf{w}_j^k|}{\alpha^k + |\mathbf{w}_j^k|} + \gamma^3 \frac{\mathbf{x}^3 \cdot \mathbf{w}_j^3}{\|\mathbf{x}^3\| \|\mathbf{w}_j^3\|}, \tag{8}$$

$$\mathbf{w}_J'^k = \begin{cases} (1 - \beta^k)\mathbf{w}_J^k + \beta^k(\mathbf{x}^k \wedge \mathbf{w}_J^k) & \text{for } k = 1, 2, \\ (1 - \beta^k)\mathbf{w}_J^k + \beta^k \mathbf{x}^k & \text{for } k = 3, \end{cases} \tag{9}$$

where $\mathbf{w}_J'^k$ denotes $\mathbf{w}_J^{k(\text{new})}$ and $\mathbf{w}_J^k$ denotes $\mathbf{w}_J^{k(\text{old})}$.

The improvement of utilizing the combinatorial operations (Eqs. (8) and (9)) instead of fuzzy ART operations (Eqs. (1) and (4)) is illustrated in Section V-C.

TABLE I
TD-FALCON ALGORITHM WITH DIRECT CODE ACCESS.

Step 1	Initialize the TD-FALCON network.
Step 2	Sense the environment and formulate the corresponding state vector **S**.
Step 3	Following an action selection policy, first make a choice between exploration and exploitation. If exploring, take a random action. If exploiting, identify the action a with the maximum Q-value by presenting the state vector **S**, the action vector **A**=(1,...1), and the reward vector **R**=(1,0) to TD-FALCON. If no satisfactory action is retrieved, select a random one.
Step 4	Perform the action a, observe the next state s', and receive a reward r (if any) from the environment.
Step 5	Estimate the value function $Q(s,a)$ following the Temporal Difference formula defined in Eq. (12).
Step 6	Present the corresponding state, action, and reward (Q-value) vectors (**S**, **A**, and **R**) to TD-FALCON for learning.
Step 7	Update the current state by $s = s'$.
Step 8	Repeat from Step 2 until s is a terminal state.

D. TD-FALCON Network

FALCON networks learn associations among states, actions, and rewards. However, decision making sometimes does not only depend on the current state. We need to perform anticipations on future outcomes and select the action that may lead to the maximum reward. Furthermore, not every state-action pair is guaranteed to receive a reward. We need to carefully estimate the reward for learning. For those purposes, Temporal Difference (TD) learning is incorporated into FALCON.

TD-FALCON is first proposed in [37]. It estimates and learns value functions of the state-action pair $Q(s,a)$, which indicates the estimated reward when performing action a in state s. Such value functions are then used to select an action with the maximum payoff. The TD-FALCON network used in this paper applies the direct code access procedure [32] to select the best action associated with the state.

Given the current state s, TD-FALCON first decides between exploration and exploitation by following an action selection policy. For exploration, a random action is picked. For exploitation, TD-FALCON searches for the optimal action through a direct code access procedure. Upon receiving a feedback from the environment after performing the action, a TD formula is used to compute a new estimate of the Q-value to assess the chosen action in the current state. The new Q-value is then used as the teaching signal for TD-FALCON to associate the current state and the chosen action to the estimated Q-value. The details of the action selection policy, the direct code access procedure, and the temporal difference learning equation are elaborated as follows.

1) Action Selection Policy: There is a trade-off between *exploration* (the agent should try out random actions to give chances to those seemingly inferior actions) and *exploitation* (the agent should stick to the most rewarding actions to its best knowledge). The agent takes a random action with a probability of ϵ or selects the action with the highest reward with a probability of $1-\epsilon$. In practice, we prefer the agent explores more in the initial training phases and exploits more in the final phases. Therefore, ϵ decreases after each training trial. This is known as the ϵ-greedy policy with decay.

2) Direct Code Access Procedure: In exploitation mode, an agent, to the best of its knowledge, searches for the cognitive code that matches with the current state and has the maximum reward value. To perform the direct code access procedure, we set the activity vectors $\mathbf{x}^1 = \mathbf{S}$, $\mathbf{x}^2 = (1,\ldots,1)$, and $\mathbf{x}^3 = (1,0)$. TD-FALCON then performs code activation (Eq. (8)) and code competition (Eq. (2)) to select the winner code J in F_2. Consequently, a readout is performed on J in its action field F_1^2:

$$\mathbf{x}^{2(\text{new})} = \mathbf{x}^{2(\text{old})} \wedge \mathbf{w}_J^2. \tag{10}$$

An action a_I is then chosen, which has the highest value in the action field F_1^2:

$$x_I^2 = \max\{x_i^{2(\text{new})} : \text{for all } F_1^2 \text{ code } i\}. \tag{11}$$

3) Learning Value Function: TD-FALCON employs a bounded Q-learning rule, the equation for the iterative estimation of value function $Q(s,a)$ adjustments is given as follows:

$$\Delta Q(s,a) = \alpha TD_{err}\left(1 - Q\left(s,a\right)\right), \tag{12}$$

$$TD_{err} = r + \gamma \max_{a'} Q(s',a') - Q(s,a), \tag{13}$$

where $\alpha \in [0,1]$ is the learning parameter; TD_{err} is a function of the current Q-value predicted by TD-FALCON and the Q-value newly computed by the TD formula; r is the immediate reward value; $\gamma \in [0,1]$ is the discount parameter; and $\max_{a'} Q(s',a')$ denotes the maximum estimated value of the next state s'.

The Q-learning update rule is applied to all states that the agent traverses when the maximum value of the next state s' could be estimated. By incorporating the scaling term $1 - Q\left(s,a\right)$ in Eq. (12), the adjustment of Q-value is self-scaling so that they could not increase beyond 1 [32]. If the immediate reward value r is constrained within the $[0,1]$ interval, we can guarantee the Q-values are bounded in the $[0,1]$ interval [32].

The overall TD-FALCON algorithm with the direct code access procedure is summarized in Table I.

V. EXPERIMENTS AND DISCUSSIONS

Our agents employ two networks for behavior modeling (TD-FALCON) and weapon selection (FALCON) respectively. First of all, we conduct experiments on how to appropriately and effectively model the behaviors of our agents through reinforcement learning. In this set of experiments, we evaluate different learning operations and parameter values to show how they affect the performance of our agents in several aspects with discussions. Moreover, to obtain stable performance evaluation, the weapon selection network is by-passed during behavior modeling, which means all the change-to-the-best-weapon commands are governed by the game server wherein the pre-defined expert knowledge is stored. After presenting and evaluating the experimental results on behavior modeling, the general set of parameter values is then applied to all the other experiments presented in this paper. Actually, these values can be roughly estimated based on the basic understanding of the application domain, which could serve as the guidelines for applying similar reinforcement learning methodologies in other similar FPS game domains.

In the following set of experiments, we challenge our agents to effectively learn the weapon preferences. Before conducting experiments on weapon selection, we choose a set of learned behavior modeling rules (from previously conducted experiments) for appropriate behavior modeling. To obtain stable performance evaluation, the learning of the behavior modeling network is turned off in the weapon selection experiments. Therefore, the performance change is certainly determined by the quality of the learned weapon selection rules.

After evaluating the individual performance of the behavior modeling and the weapon selection networks respectively, we enable the learning of both networks and conduct three more sets of experiments, wherein our agents play against different opponents in different maps. In these three sets of experiments, our agents learn from scratch, which means both networks start with zero knowledge (i.e. only one uncommitted code in the cognitive field). Furthermore, to show our agents can quickly adapt the learned knowledge to a new opponent in a new map, we repeat one set of the conducted experiments wherein our agents play against a new opponent in a new map with the learned (after playing against the previous opponent in the previous map) sets of rules (both behavior modeling and weapon selection) pre-inserted.

A. Designing Experiments

All experiments are conducted in the 1-on-1 DeathMatch scenario of UT2004. Each game trial completes when either bot obtains a game score of 25. We run each set of experiments for 20 times and average the results to remove randomness.

Two sample bots (`advancedBot` and `hunterBot`) provided by Pogamut are selected as the opponents of our bot named `FALCONBot`. `AdvancedBot` collects items along the way while exploring the map and shoots at its opponent once spotted. It always uses the same weapon, named assault rifle, which is a primitive weapon given to every new born player. Because `advancedBot` performs all basic high-level tasks that a competent agent should do and its combat performance is steady (always shoots with assault rifle unless it runs out of ammunition), it is probably the most appropriate opponent in terms of providing baseline comparisons for performance evaluation. There is one and only one modification that we made to keep the experiments more consistent: Whenever `advancedBot` runs out of ammunition (less than 50 bullets left for assault rifle), it sends specific commands directly to the game server for refill. Therefore, `advancedBot` always sticks to the same weapon and thus becomes a steadier opponent. Moreover, such a modification actually increases the performance of `advancedBot`. Because based on our observation, changing weapons during battles (after running out of ammunition rather than proactively) is a noticeable disadvantage (wasting critical time). However, we do not automatically refill the ammunition for `FALCONBot`. If `FALCONBot` runs out of ammunition for its weapon in use, it is forced to change to another weapon. During battles, if the weapon selection network is enabled to learn and retrieve knowledge, `FALCONBot` changes to the best weapon in possession (has ammunition) based on the learned knowledge (changes to a random weapon if no relevant knowledge has been learned or decides to try another weapon for exploration). `HunterBot` also performs all basic high-level tasks that a competent agent should do and it has been used in much research work [25], [38]–[40]. Different from `advancedBot`, `hunterBot` changes its weapon based on the expert knowledge stored in the game server.

Two maps ("DeathMatch 1-on-1 Idoma" or Idoma in short and "DeathMatch 1-on-1 Spirit" or Spirit in short) provided by UT2004 are selected to be the combat arenas. Both maps are ideal for 1-on-1 DeathMatch games because they are relatively small in size but have all kinds of collectible items for bots to pick up. Furthermore, the Idoma map has several levels of ground in different altitudes and rich terrain types: slopes, tunnels, obstacles, etc. Therefore, the bots are naturally prevented from excessive encounters and escaping becomes relatively easier. On the other hand, the Spirit map also has levels of ground but its number of terrain types is limited and the central space of the virtual arena is hollow (one bot can spot the other from a different level of ground in the other side of the map and bots can directly drop to the lower ground from a higher level), which increase the chances of encounters. Therefore, in the Spirit map, bots have relatively less amount of time to collect useful items. In summary, the abilities of all bots can be thoroughly evaluated in these two maps.

All game configurations are listed in Table II to show explicitly how we conduct our experiments. In the first row of Table II, BM denotes the learning of behavior modeling and WS denotes the learning of weapon selection.

Our experiments are conducted concurrently on various computers. One of the computers is equipped with Intel Pentium Dual Core processor of 3.4 GHz and 2 GB RAM. When we use it to run the experiments shown in Section V-H, we find that the decision cycle (for all experiments, the time budget between cycles are given as 200 ms) is never postponed (occasionally delayed by server communications) due to any excessive computation time taken by our agents. This shows that our agents are able to learn and make decisions in real-time while playing the FPS game.

A recorded video on how we set up the experiments and how our agents learn in real-time while playing the game has been uploaded online[2]. From the demo video, one may notice that sometimes `FALCONBot` erratically switches its direction. There are mainly two circumstances under which

[2]The demo video is available online: http://youtu.be/i5AEXZ1KOgQ

TABLE II
GAME CONFIGURATIONS OF ALL THE CONDUCTED EXPERIMENTS

Experiment	Opponent	Map	BM	WS	ϵ	decay	# trials	Purposes
Section V-C	`advancedBot`	Idoma	on	off	0.5	0.02	30	To evaluate different operations and different parameter values.
Section V-E	`advancedBot`	Idoma	off	on	0.5	0.02	30	To evaluate whether learned weapon effects are better than expert knowledge.
Section V-G.1	`advancedBot`	Spirit	on	on	0.5	0.0125	45	To evaluate the performance under a different game configuration.
Section V-G.2	`hunterBot`	Idoma	on	on	0.5	0.0125	45	To evaluate the performance under a different game configuration.
Section V-G.3	`hunterBot`	Spirit	on	on	0.5	0.0125	45	To evaluate the performance under a different game configuration.
Section V-H	`hunterBot`	Spirit	on	on	0.5	0.05	15	To show `FALCONBot` can adapt quickly with pre-inserted knowledge.

`FALCONBot` erratically changes its direction. In the first circumstance, `FALCONBot` is momentarily switching between behavior states. Such behavior is expectated in the earlier trials of the experiments when `FALCONBot` is purposely exploring all the possible options. However, `FALCONBot` gets more determined in the later trials as it gradually learns how to select appropriate behavior states through trial-and-error. The second circumstance is that the opponent of `FALCONBot` is out of sight. Therefore, `FALCONBot` looks around and occasionally jumps to locate its opponent.

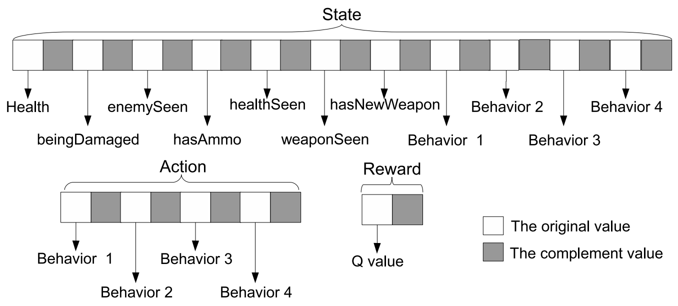


Fig. 4. Information vector for behavior modeling.

B. Modeling Behavior Selection Network

In order to avoid having a hard-coded agent that will always perform the same action under similar circumstances, we enable our agent to learn in real-time by employing a TD-FALCON network to model various behaviors.

We define four behavior states for our agent. In each state, the agent performs a sequence of primitive actions described as follows. This decomposition of high-level tasks to low-level actions is similar to the Hierarchical Task Network (HTN) discussed in [8], [22]. These four states are:

1) Running around state, wherein the agent explores the map with a randomly selected reachable location. Once the location is chosen, if our agent remains in this behavior state, it will reach the determined location before the next one can be chosen. However, a new random reachable location will be chosen when our agent switches to this behavior state from another state.
2) Collecting item state, wherein the agent runs to a particular place and picks up a collectible item. Different priorities are assigned to all the collectible items in sight, which can be briefly described as follows. If the health of our agent is low (less than 100 points), then go for health boost. Else if the weapon has not been collected, then go to pick it up. Else if certain ammunition is low for its corresponding weapon, then go to pick it up. Else our agent just picks up items along its moving direction.
3) Escaping from battle state, wherein the agent flees the battle field and collects health boosts nearby. Whenever our agent decides to escape from the battle, it chooses a navigation point in its $[90°, 135°]$ or $[-135°, -90°]$ direction. Then, it runs along that direction and picks up health boosts (if available) nearby.
4) Engaging fire state, wherein the agent tries to kill its opponent and avoids being hit at the same time. There are a number of movements that our agent performs during battles: moving towards or away from the opponents (according to the effective range of the weapon in use), jumping, crouching (when the agent is on a higher level of ground than its opponent), dodging (when the agent detects an incoming projectile), strafing, etc. Although in this paper, certain movements are performed randomly, we propose another FALCON network to learn the effective tactic movements (according to the strategies or abilities of its opponents) to avoid being hit as part of the future work. Although, only in this state, our agent fires on its opponents, it is programmed to collect useful items while firing (by sending the strafe-to-location command).

The information vector used for behavior modeling is shown in Fig. 4. Inputs in grey indicate they are the complements of their respective preceding inputs. These complements are added to facilitate FALCON operations (see introduction in Section IV) as well as to avoid code proliferation [32].

The state vector $\mathbf{S}$ comprises eleven inputs. Therefore, the length of $\mathbf{S}$ is 22 (including eleven complements). These inputs indicate (1) the current health level of the agent (normalized), (2) whether the agent is being damaged, (3) whether the opponent is in sight, (4) whether the agent has adequate ammunition, (5) whether any collectible health boost is available nearby, (6) whether any collectible weapon is available nearby, (7) whether the agent possesses weapons other than the two primitive ones given when born, and (8) another four Boolean values such that one and only one of them is 1, indicating the current behavior state of the agent. The length of the action vector $\mathbf{A}$ is eight (including four complements). One and only one of the four inputs is 1, indicating which behavior state the agent should switch to based on $\mathbf{S}$ and $\mathbf{R}$. The reward vector $\mathbf{R}$ only has two values, namely the reward and its complement.

The aim of the DeathMatch game scenario is to get high game scores. However, we cannot directly use the scores as the rewards for agents. Instead, by intuition, the highest reward of 1 is given whenever the agent kills its opponent and the lowest reward of 0 is given whenever the agent gets killed. The reward of 0 is actually a form of punishment because a reward of 0.5 is the neutral value in the $[0, 1]$ interval. Furthermore, an immediate reward of 0.75 (the median number between the neutral reward and the highest reward) is given whenever the agent successfully hits its opponent, increases health, or collects new weapon.

C. Experiments on Behavior Modeling

Before conducting experiments on behavior modeling, we need to set TD-FALCON parameters. The default parameters used are listed as follows:

Vigilance parameters $\{\rho^1, \rho^2, \rho^3\}$ are set to $\{0.8, 1, 0.2\}$. Because most (10 out of 11) of the attributes in the state vector are Boolean, we set ρ^1 to a relatively high value to preserve certain level of integrity of the learned knowledge. At the same time, ρ^1 is not set to a extremely high value to allow compaction on the learned knowledge to obtain certain level of generalization. Setting ρ^2 to 1 for action field means that we want an exact match for all the Boolean variables. The reason for setting ρ^3 to a low value for the reward field is to ensure effective learning for all similar state-action pairs.

Learning rate parameters $\{\beta^1, \beta^2, \beta^3\}$ are set to $\{0.2, 0.2, 0.1\}$. Because we have immediate rewards, β^1 and β^3 are set to low values for slow learning. Actually, β^2 for the action field could be assigned to any value in the $[0, 1]$ interval, because we constrain the learning in that field to be an exact match.

The contribution factors $\{\gamma^1, \gamma^2, \gamma^3\}$ are set to $\{0.3, 0.3, 0.4\}$. We slightly favor the reward field not because we intend to be goal driven, but because the code activation values are calculated based on different functions. In general, cosine similarity measurement (Eq. (5)) is smaller than fuzzy measurement (Eq. (1)) in terms of numerical values.

Choice parameters $\{\alpha^1, \alpha^2, \alpha^3\}$ are set to $\{0.1, 0.1, 0.1\}$. It is introduced to avoid any possible invalid calculation in Eq. (1).

TD learning rate parameter $\alpha = 0.5$ in Eq. (12) and TD learning discount parameter $\gamma = 0.5$ in Eq. (13), which are the general values used in TD-FALCON networks [32].

The action selection policy threshold ϵ is initially set to 0.5 and decays 0.02 after each game trial until it reaches 0. Therefore, our agents are expected to explore as well as exploit in the first 25 game trials and rely only on their best knowledge in the last 5 game trials.

Three sets of experiments are conducted for behavior modeling. In these experiments, we always use the default parameter values except the one being evaluated to ensure consistent performance comparisons.

1) Comparisons on different operations: As discussed in Section IV-C, using combinatorial operations instead of fuzzy ART operations is expected to improve the performance of our agents in this FPS game domain. Performance comparisons on different operations are shown in Fig. 5.

The game score difference is defined as the game score of our agent minus the game score of `advancedBot` at the end of each game trial. The agent with combinatorial operations (Eq. (8) and Eq. (9)) uses the default parameter values as given earlier. It is obvious from Fig. 5(a) that it performs better (the highest average game score difference is 6.9) than all the other agents with fuzzy ART operations.

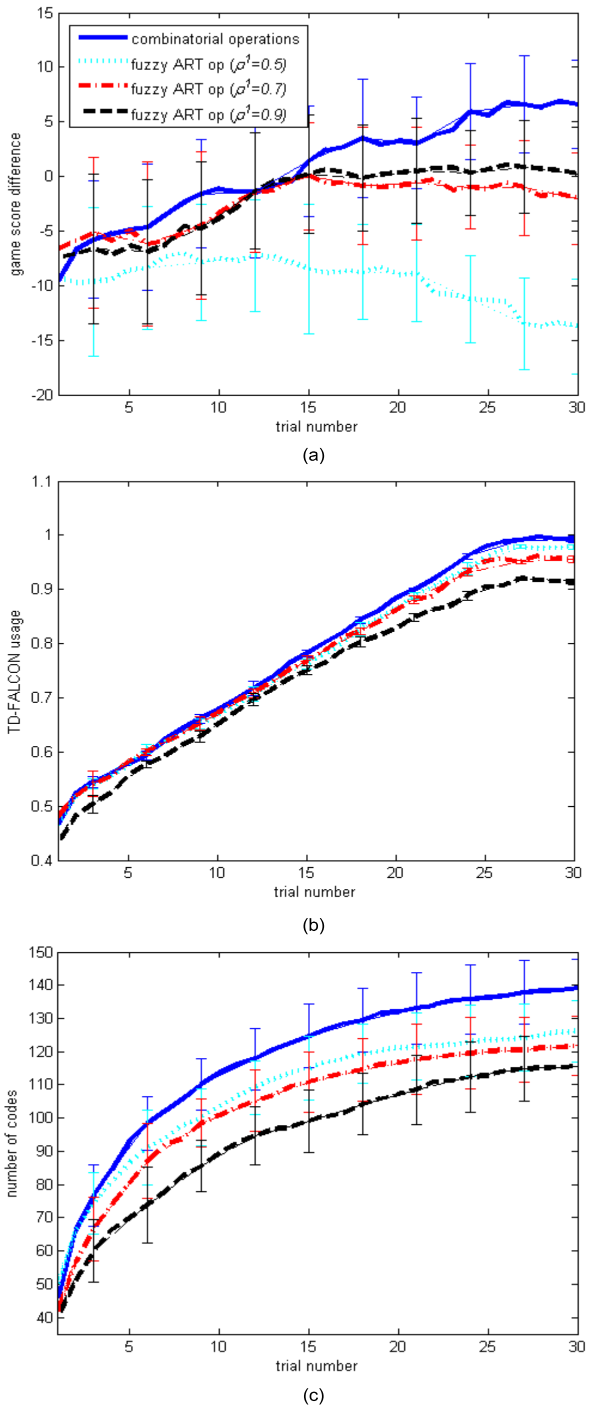
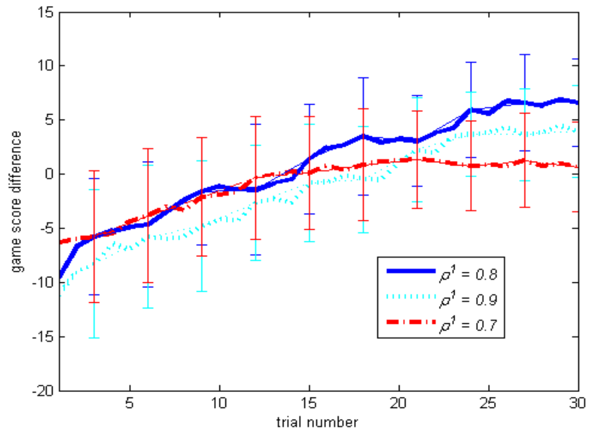


Fig. 5. Results of `FALCONBot` with different TD-FALCON operations playing against `advancedBot` in the Idoma map. (a) Averaged game score difference between `FALCONBot` and `advancedBot`. (b) Averaged percentage of actual TD-FALCON usage of `FALCONBot`. (c) Averaged number of codes generated in the F_2 cognitive field. (Note: in this paper, all figures visualizing the experimental results are plotted with the confidence intervals at every third trials to avoid too much overlaps that could potentially mess up the figures.)

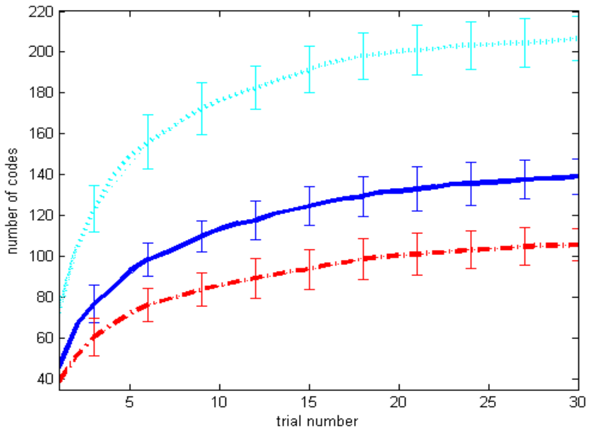
This statement is further supported by the statistical test. We run single factor ANOVA [41] on the averaged game score differences obtained by using combinatorial operations and fuzzy ART ones ($\rho^1 = 0.9$ is used for comparison since it performs better than $\rho^1 = 0.7$ and $\rho^1 = 0.5$) from the 26th to 30th game trial (after convergence). P-value is computed as $3.43 \times 10^{-10} < 0.05$ and $F = 1337.37 > F_{\text{crit}} = 5.32$. Both measures indicate that FALCONBot significantly improves its performance in the FPS game domain by using the combinatorial operations than using the fuzzy ART ones. In terms of fuzzy ART operations, if vigilance parameter for the state vector $\rho^1 = 0.5$, the learning completely fails because the performance keeps dropping after a slight increase in the early trials. This indicates that over-generalization with fuzzy ART operations learns incorrect knowledge in this FPS game domain. The agent learns if $\rho^1 = 0.7$, but the performance is not encouraging because the highest game score difference is only 0 and it still seems to be decreasing in the later trials. Further increase of ρ^1 to 0.9 shows improvements. However, the highest game score difference is 1.05, which is much smaller than 6.9 obtained by applying the combinatorial operations (corresponding statistical test result is shown earlier in this paragraph). All agents using fuzzy ART operations stop improving their performance after the 16th game trial. In contrast, the agent using combinatorial operations keeps improving until it converges in the last five game trials.

The agent using combinatorial operations utilizes its learned knowledge well. Its decisions at the last five trials (converged) are nearly 100% instructed by the TD-FALCON network (see Fig. 5(b)). Although the agent is learning from scratch, in the end, its knowledge set is complete to instruct nearly every decision making and achieve satisfactory performance. The learning is thus shown to be successful and effective. For agent using fuzzy ART operations with $\rho^1 = 0.5$, it relies heavily (over 97.5% after convergence) on its learned knowledge but its performance is disappointing. The decrease in performance when applying fuzzy ART operations with $\rho^1 = 0.5$ again shows that applying over-generalized fuzzy ART operations learns incorrect rules. When ρ^1 used for fuzzy ART operations increases, the utilization of learned knowledge decreases and there is a significant drop when $\rho^1 = 0.9$. There are two reasons for this finding. The first one is that the agent is not good at associating similar situations because it might be over-specific. Therefore, the knowledge learned may not be complete. The second reason is that once the previously learned functional knowledge gets corrupted, it is not easy to make corrections due to the high ρ^1 value.

The agent using combinatorial operations generates more rules (codes in the F_2 cognitive field) than all the others using fuzzy ART operations. This is expected because the agent using combinatorial operations is less generalized and requires more rules for competent decision making. It is interesting to note in Fig. 5(c) that as ρ^1 for fuzzy ART operations increases, the size of the rule set decreases. Intuitively speaking, when the vigilance parameters decrease, the agent is supposed to use less rules because those rules are more generalized. This contradictory proportional relation is due to the reason that there are many redundant rules created (grouping inconsistent knowledge together makes the rule unreliable) for being over-generalized. This problem has been discussed in Section IV-C.



(a)



(b)

Fig. 6. Results of FALCONBot with different vigilance parameter values playing against advancedBot in the Idoma map. (a) Averaged game score difference between FALCONBot and advancedBot. (b) Averaged number of codes generated in the F_2 cognitive field.

2) Comparisons on different ρ values: After showing that the agents should apply combinatorial operations (Eqs. (8) and (9)) for this FPS game domain, in this subsection, we compare the performance of the agents based on different vigilance parameter values. This is an important parameter because it controls the level of knowledge generalization. Performance comparisons on different vigilance parameter values are shown in Fig. 6.

It is clear in Fig. 6(a) that the agent with $\rho^1 = 0.9$ performs worse than the one with $\rho^1 = 0.8$ at all times (single factor ANOVA test shows that the difference between $\rho^1 = 0.8$ and $\rho^1 = 0.9$ from the 26th to 30th game trial is significant since P-value is computed as $5.44 \times 10^{-7} < 0.05$ and $F = 205.80 > F_{\text{crit}} = 5.32$). This is because $\rho^1 = 0.9$ is too specific in this game domain and it leads to ineffective learning. When $\rho^1 = 0.7$, the agent learns well in the first 15 game trials. However, its performance does not improve ever since. This is because $\rho^1 = 0.7$ is too general that the learned knowledge becomes more and more ambiguous and eventually gets corrupted. Actually, we did perform experiments with different ρ^1 values around 0.8 (0.77 and 0.83), the results are not noticeably different with that of using 0.8 (unlike 0.7 and 0.9). This means a rough estimation on the parameter values is sufficient to create competent agents in the FPS game domain.

The biggest difference among all the three settings of vigilance parameters in terms of TD-FALCON utilization is less than 1%. Therefore, the comparisons in this aspect are not shown and discussed in this paper.

Fig. 6(b) well demonstrates that when the vigilance parameters increase, the agent needs more rules for decision making. With $\rho^1 = 0.9$, the agent needs over 200 rules which is significantly large considering its inferior performance. With $\rho^1 = 0.7$, the agent keeps a small number of rules (less than 106). However, it fails to further improve its performance.

3) Comparisons on different β values: The learning rate β^k in Eq. (9) is also an important parameter. If the agent learns too fast in this POMDP [36], it may not learn well in the end or even under-perform due to excessive learning. To facilitate discussion, we denote low learning rates of $\{0.2, 0.2, 0.1\}$ as small β values, medium learning rates of $\{0.5, 0.5, 0.2\}$ as medium β values, and high learning rates of $\{1, 1, 0.5\}$ as big β values. The performance comparisons on different learning rates are shown in Fig. 7.

The performance of the agent applying small β values is not worse than that applying medium β values (single factor ANOVA test shows no real difference from the 26th to 30th game trial since P-value is computed as $0.21 > 0.05$ and $F = 1.86 < F_{\text{crit}} = 5.32$) as shown in Fig. 7(a). However, the agent applying big β values is not comparable to applying small β values (P-value is computed as $3.62 \times 10^{-9} < 0.05$ and $F = 738.79 > F_{\text{crit}} = 5.32$).

As being comparable in performance, the preference between small and medium β values depends on their rule set sizes. According to Occam's Razor (which states that when all other things being equal, the simplest solution is the best), we prefer small β values since the rule size is noticeably smaller as shown in Fig. 7(b). When big β values are used, an excessive number of rules are generated fast and continuously. This again shows that over-generalization introduces redundant rules and is not preferred in this FPS game domain.

Before conducting the other experiments, we would like to emphasize again in here that although we evaluate various operations and parameter values in this section, our purpose is to show that the combinatorial operations should be applied instead of fuzzy ART operations and how different parameter values of the vigilance threshold and the learning rate affect the performance of our agent, rather than sweeping parameter values to determine the most appropriate ones. We believe that based on the basic understanding of any application domain, any developer would be able to apply the proposed methods with roughly estimated parameter values to achieve satisfactory performance without a relatively complete sweep. The default parameter values are used to conduct the rest of the experiments presented in this paper to show that the parameter values are general for different game configurations. Therefore, no tuning is required on any learning parameter.

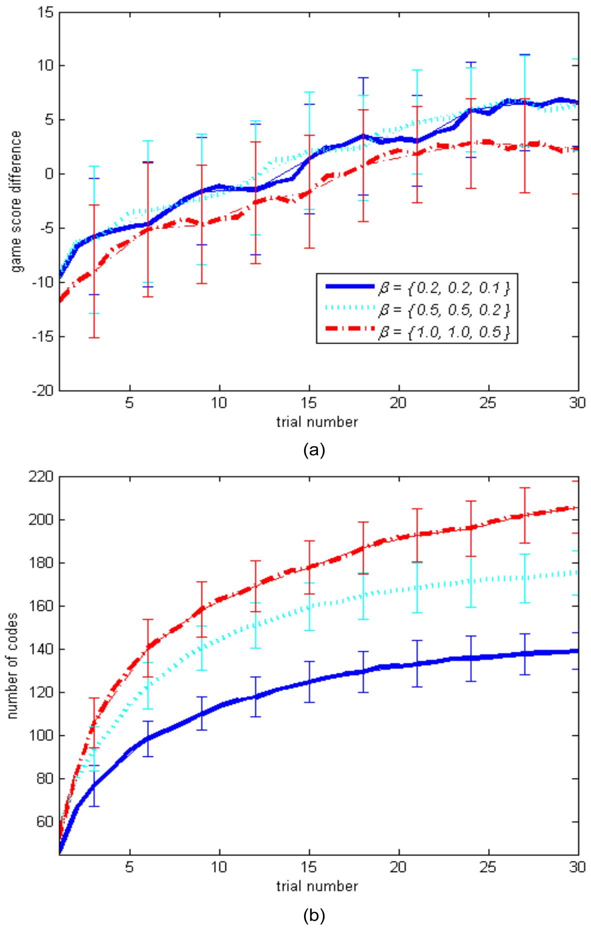


Fig. 7. Results of `FALCONBot` with different learning rates playing against `advancedBot` in the Idoma map. (a) Averaged game score difference between `FALCONBot` and `advancedBot`. (b) Averaged number of codes generated in the F_2 cognitive field.

D. Modeling Weapon Selection

Each weapon in UT2004 has several attributes that can be queried. Some important ones are the effective distance, the maximum range, and the weapon type (close-range melee or long-distance sniper). In Section V-C, weapon selections are performed by sending specific commands to the game server. Based on the predefined expert knowledge, the server then selects the most powerful weapon in possession for the agent.

The idea of incorporating a weapon selection network into our agent is inspired by [5] that during the competition, weapon effects are changed and no weapon hints are provided by the game server. Therefore, all weapon specifics are not available and the agent must learn all the modifications by itself. By incorporating a weapon selection network that learns from scratch, we enable our agent to deal with such challenge. Furthermore, our agent learns any possessed weapon directly from the outcomes during battles at run time without any human supervision and its performance is shown (later) to be as good as using the predefined expert knowledge.

To model the behaviors of our agent in this weapon selection experiment, we select one of the best performing sets of knowledge (133 rules) obtained from the previous experiment.

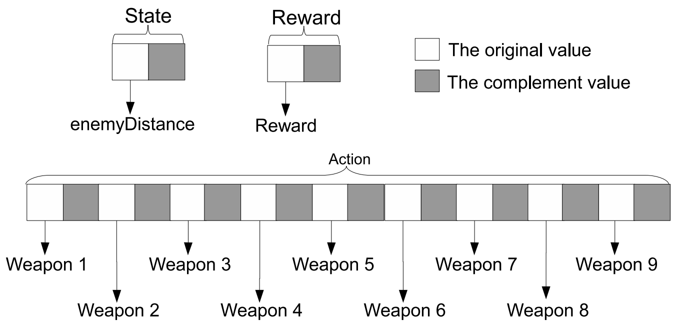


Fig. 8. Information vector for weapon selection.

The learning of the behavior modeling network is now turned off to obtain consistent performance comparisons.

The information vector for the weapon selection network is shown in Fig. 8. There are only two values used for the state vector $\mathbf{S}$, namely the (normalized) distance between the agent and the opponent and its complement. There are nine inputs (with their complements) for the action vector $\mathbf{A}$, which is the same as the total number of available weapons. One and only one of them is 1, indicating which weapon the agent uses currently (the alternative firing mode of each weapon is not considered). The reward vector $\mathbf{R}$ only has two values, the reward and its complement. This weapon selection network is only learning reactively, which means the weapon in use does not affect any other weapon (no chains of effects). Therefore, the rewards perceived from the game environment are directly applied for learning without any Q-value estimation.

Intuitively, if the agent kills its opponent with certain weapon, we give that weapon the highest reward of 1. If a weapon is fired but missed, we give a reward of 0 (as a form of punishment). The reward value for a successful hit by a certain weapon is calculated according to Eq. (14).

$$r = min(1, 0.5 + \frac{d}{100}), \tag{14}$$

where r represents the reward value and d represents the actual amount of damage received by the opponent.

Eq. (14) is designed in such a way that the reward for a successful hit should be larger than the neutral reward of 0.5. Furthermore, if the amount of damage made is greater or equal to 50 (a new born player has a health level of 100 points), which means the particular weapon is extremely powerful, the reward value is capped at 1.

Although it would be cheating if we directly query the health level of the opponent (because the health level of any avatar in UT2004 other than oneself is not shown during the game play), the amount of damage received by the opponent is public information. Whenever someone gets hit, the game server will broadcast a global message containing information such as who has been hit, by whom, with which weapon, at what time, and the exact amount of damage. Furthermore, we do not modify the shooting command for our bot (unlike in [6], our bot shoots towards the opponents with random deviations purposely to behave like a non-perfect human player) as well as its opponents. Moreover, we do not include the tendency of movements or high-level bullet reflections when our bot fires its weapon in hand. We just leave the shooting command unmodified to examine the effectiveness of each weapon.

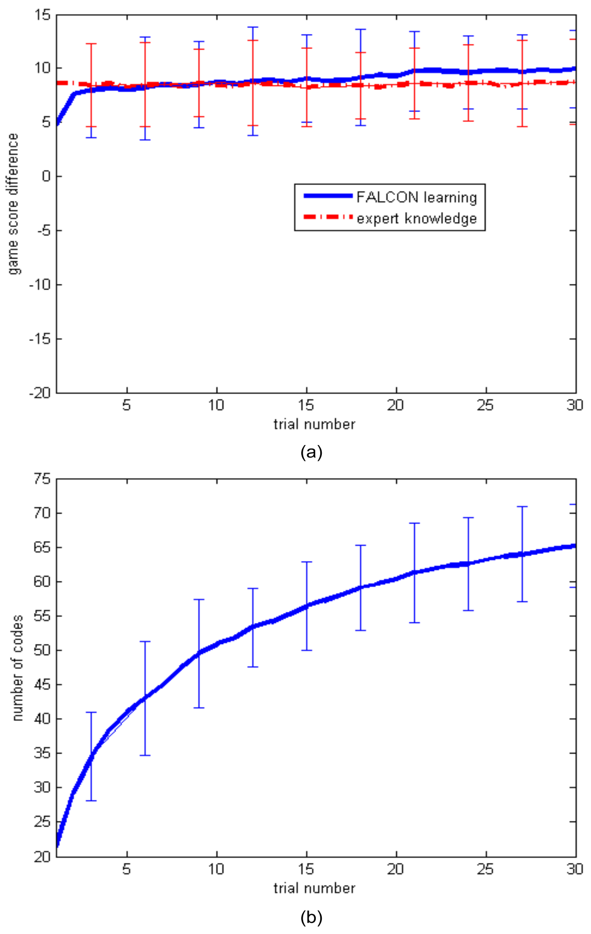


Fig. 9. Results of `FALCONBot` playing against `advancedBot` in the Idoma map to learn weapon preferences. (a) Averaged game score difference between the FALCON learned rules and expert knowledge. (b) Averaged number of codes generated in the F_2 cognitive field.

E. Experiments on Weapon Selection

The parameter values used in the weapon selection network are identical to those default ones used in the previous experiments (given in Section V-C) and the experiment in this subsection is conducted in the same manner as the previous experiment. The experimental results are shown in Fig. 9.

The game score difference between our agent who selects weapons based on the predefined expert knowledge (provided by the game server) and `advancedBot` is represented with the dash line in Fig. 9(a). Because the predefined knowledge of weapons is static, this game score difference is consistent and the average is approximately 8.5. The other game score difference between our agent who selects weapons using the dynamically expanding weapon selection network and `advancedBot` is represented with the solid line in Fig. 9(a). Although the statistical test does not show significant improvements (P-value of single factor ANOVA is computed as $0.058 > 0.05$ and $F = 3.75 < F_{\text{crit}} = 4$), it still

TABLE III
TRANSLATED EXAMPLE RULES FOR BEHAVIOR MODELING

IF	health is around [87, 109], not being damaged, opponent is in sight, has adequate ammo, has health boost nearby, has no weapon nearby, possessing only primitive weapons, and currently in RUN_AROUND state;
THEN	go into ENGAGE state;
WITH	reward of 0.729.
IF	health is around [2, 21], being damaged, opponent is in sight, has adequate ammo, has no health boost nearby, has no weapon nearby, possessing only default weapons, and currently in ENGAGE state (0.9);
THEN	go into ESCAPE state;
WITH	reward of 0.05.

TABLE IV
TRANSLATED EXAMPLE RULES FOR WEAPON SELECTION

IF	distance is very near [108, 317];
THEN	use flak cannon;
WITH	reward of 0.838.
IF	distance is far [1781, 2364];
THEN	use lightning gun;
WITH	reward of 0.781.

indicates that `FALCONBot` is able to quickly acquire weapon selection strategies from scratch and performs at the same level as relying on the expert knowledge. The reasons for the weapon selection network being less successful than the behavior modeling network are three-fold: (1) the choice of an effective weapon is naturally less critical than the choice of an appropriate behavior (which weapon to fire is less influential than whether to shoot), (2) the expert knowledge of weapon effects predefined by the game developers are already good enough, and (3) `FALCONBot` often fights with a limited selection of weapons in possession since it is force to engage fire before getting more effective ones.

The number of rules generated in the FALCON network is shown in Fig 9(b). From the 11st trial to the 20th trial, there are approximately ten rules created. From the 21st trial to the 30th trial, there are approximately four rules created. The curve shows certain level of convergence.

F. Examples of Translated Rules

Our agents learn from zero knowledge in the previously presented experiments. Reliable rules on both behavior modeling and weapon selection are acquired during run time.

Two examples (after translation) of the learned behavior modeling rules are shown in Table III. The first rule basically says that if the health of the agent is high enough, not being hit, with its opponent in sight, and equipped with only the primitive weapons, the agent may start a battle to receive a high reward. The second rule can be highlighted that if the health of the agent is very low, and already engaged with its opponent, the successful rate of running away from the battle is very low. This is true because there is no collectible health boost nearby. The (0.9) at the end of the IF clause means that this current state indication attribute has been generalized.

Two examples (after translation) of the learned weapon selection rules are shown in Table IV. These two rules coincide with our knowledge learned during game plays. Flak cannon is a very powerful melee type of weapon, which is most effective in the short range. Upon firing, it blasts many small shrapnel pieces. Therefore, it seldom misses in a short distance. The first rule is associated with a rather high reward. Lightning gun is a sniper type of weapon with a large maximum shooting range and it is also very powerful in terms of the damage dealt. The second rule is associated with a high reward.

To better illustrate how FALCON works, the first weapon selection rule presented in Table IV is used as an example. If the opponent is spotted at 300 units (in UT2004) away and `FALCONBot` wants to find which weapon is the most effective choice, the direct code access procedure (see Section IV-D.2) is followed to compute the activation values (based on Eq. (8)) for all codes in the cognitive field. Suppose the example rule is found to be the winner (based on Eq. (2)). Because its resonance values fulfil the vigilance criterion (see Eq. (3), $m^1 = 0.93 > \rho^1 = 0.8$ and $m^3 = 0.838 > \rho^3 = 0.2$), the vector in the action field is read out and `FALCONBot` consequently changes to flak cannon (if in possession, has ammunition, and not currently in use). Furthermore, assume for another time, `FALCONBot` fired flak cannon from 900 units away and missed. When this information is presented for learning, assume the example rule is again the winner based on the activation values. However, the vigilance criterion is violated ($m^1 = 0.736 < \rho^1 = 0.8$). Therefore, this new input cannot be updated to the example rule. Moreover, if no other committed codes fulfil the vigilance criterion, an uncommitted node will be selected to incorporate the new knowledge (based on Eq. (9)) and becomes committed after learning.

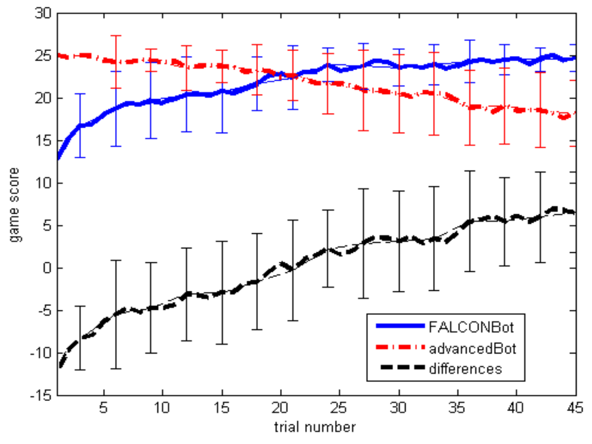
G. Experiments on Performance Evaluation

To evaluate whether our agent (with the same set of parameter setting) learns appropriately and efficiently in different game configurations, we conduct experiments wherein our agent plays against different opponents in different maps. The detailed combinations have been listed in Table II. Because both the behavior modeling and the weapon selection networks are enabled to learn from scratch at the same time, we give our agent more time to learn. The action selection policy threshold ϵ is initially set to 0.5 and decays 0.0125 after each game trial until it reaches 0.

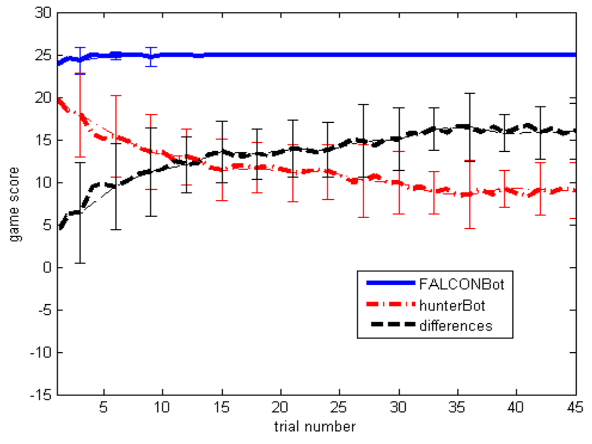
The game scores of our agent and its opponent and their game score differences in the following three game configurations are given in Fig. 10, respectively.

a) `FALCONBot` VS. `advancedBot` in the Spirit map.
b) `FALCONBot` VS. `hunterBot` in the Idoma map.
c) `FALCONBot` VS. `hunterBot` in the Spirit map.

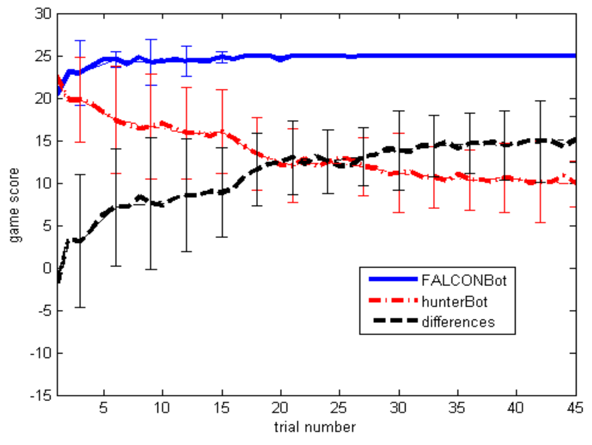
All evaluations plotted in Fig. 10 illustrate clear increasing performance of `FALCONBots` that learn in real-time while playing against different opponents in different maps. To evaluate whether different maps affect the combat outcomes, we run single factor ANOVA on the averaged game score differences taken from the 41st to 45th game trial of the second (b) and third (c) game configurations. P-value is computed as $7.53 \times 10^{-4} < 0.05$ and $F = 27.8 > F_{\text{crit}} = 5.32$. Both measures indicate that the performance of `FALCONBot` playing against the same opponent (`hunterBot`) in different maps is truly different (mainly due to the different frequency of



(a)



(b)



(c)

Fig. 10. Game scores of FALCONBot playing against different bots in different maps. (a) VS. advancedBot in the Spirit map. (b) VS. hunterBot in the Idoma map. (c) VS. hunterBot in the Spirit map.

encounters indirectly determined by the different terrain types and different layouts in the maps, see Section V-A for more details). Similar results can be found when comparing Fig. 10(a) to Fig. 9(a). However, the two experiments are conducted in different manners. The statistical comparisons between the values visualized in Fig. 10(a) and Fig. 9(a) are not performed.

By comparing Fig. 10(c) to Fig. 10(a) (also roughly comparing Fig. 10(b) to Fig. 9(a)), it is clearly shown that when

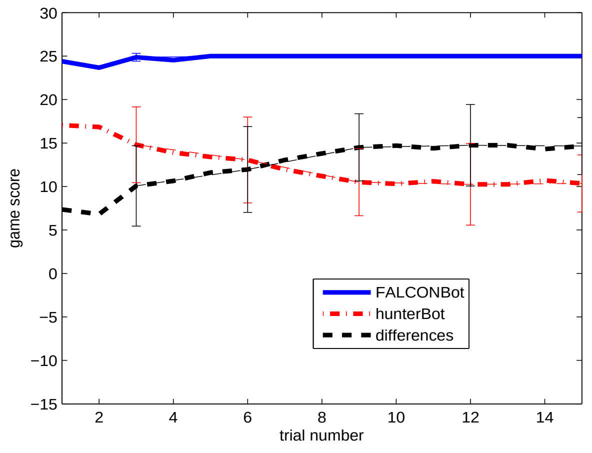


Fig. 11. Game scores of FALCONBot playing against hunterBot in the Spirit map with pre-inserted knowledge that learned when playing against advancedBot in the Idoma map.

FALCONBot adopts the same learning strategies and parameter values, its performance is significantly better when playing against hunterBot than against advancedBot (from the 41st to 45th game trial, P-value of single factor ANOVA on the game score differences is computed as $4.7 \times 10^{-9} < 0.05$ and $F = 691.39 > F_{\text{crit}} = 5.32$). This finding infers that advancedBot is better than hunterBot in terms of the ability to obtain higher game scores. There are mainly two explanations. First, the skill level of hunterBot is deliberately lowered by the developers of Pogamut (FALCONBot and advancedBot have the same default skill level). Therefore, although advancedBot only uses assault rifle (whose fired bullets travel fast in space, therefore, they are naturally hard to miss as long as the aim is accurate), it seldom misses the target. On the other hand, although hunterBot uses all kinds of powerful weapons, it does not shoot as well as the other two bots. Secondly, advancedBot is more determined to find where the opponent is and engage fires. It does not spend unnecessary time on collecting items (only along the path), unlike hunterBot does.

H. Experiments on Knowledge Adaptation

To show that our agent can quickly adapt to a new opponent in a new map, we design the experiments as follows. First of all, we select the same behavior modeling knowledge as used in Section V-E and one set of the learned weapon selection rules from Section V-E as the pre-inserted knowledge. With the knowledge learned when playing against advancedBot in the Idoma map, FALCONBot plays against hunterBot in the Spirit map for a relatively short period of time. The action selection policy threshold ϵ is initially set to 0.5 and decays 0.05 after each game trial until it reaches 0. The performance evaluations are illustrated in Fig. 11.

It is obvious in Fig. 10(c) that at the first game trial, the game score difference between FALCONBot and hunterBot is below 0. However, in Fig. 11, at the first game trial, the same game score difference is above 7. In both experiments, during the first game trials, the action selection threshold ϵ is 0.5. Therefore, this increase in game score difference is contributed by the pre-inserted knowledge.

Furthermore, we run single factor ANOVA on the averaged game score differences taken from the 41^{st} to 45^{th} game trial of the results shown in Fig. 10(c) and from the 11^{th} to 15^{th} game trial of the results shown in Fig. 11. P-value is computed as $0.26 > 0.05$ and $F = 1.45 < F_{\text{crit}} = 5.32$. Both measures indicate that with or without pre-inserted knowledge, after convergence, the performance of our agent playing against the same opponent in the same map is at the same level. This finding is encouraging because it shows our agent can adapt to a new opponent in a new map in a relatively short period of time if previously learned knowledge is retained. Based on this finding, in the future, we could perform more experiments on `FALCONBot` with pre-inserted knowledge fighting against the native bots (given in UT2004 rather than Pogamut). The native bots are challenging to defeat because they are implemented by the experienced game developers (repeatedly play-tested) and their response time and decision cycles are shorter than the bots controlled through Pogamut (because the native bots run directly in the game server, there are no communication delays and connection overheads). More interestingly, because native bots have different specialties and personalities, it would be intriguing to add certain characteristics to `FALCONBot` and investigate how it fights against different native bots.

VI. Conclusion

In this paper, we describe how we create intelligent agents to play a well-known first-person shooter computer game and to learn in real-time from the interaction outcomes perceived in the game environment only. The term intelligent may have different meanings from different perspectives. In our work, it refers to the ability and the quality of self-adaptive learning and the reliability of decision making.

Our agents employ two reinforcement learning networks to learn knowledge on behavior modeling and weapon selection, respectively. Both networks can learn from scratch or with pre-inserted knowledge. In the experiment section, we first show that using our proposed combinatorial operations rather than the conventionally applied operations enables our agents to learn more appropriately and effectively. After that, we demonstrate how different values of certain critical parameters would affect the performance. The general set of parameter values are then applied to all the other experiments presented in this paper. By applying the self-acquired knowledge on weapon effects, our agents can perform at the same level as using the predefined expert weapon preference knowledge stored in the game server. With different game configurations but the same set of parameter setting, we show that the performance of our agents is encouraging when playing against different opponents in different maps. Furthermore, we show that our agents can adapt quickly to a new opponent in a new map if the previously learned knowledge is retained.

Our agents currently do not utilize the information about physical locations based on past experiences. In the future, we could add a new control module to function as the episodic memory [42]. Therefore, the agent could gradually discovery its preferred places in the map where useful items are available for collection or locations suitable for ambush.

In the future, we could also try out different strategies to implement our agents, such as applying the UCB1 [43] or self-regulated action exploration [44] strategies as the new action selection policy.

Last but not least, we could extend the number of agents in control. We can create a team of agents to play in the Domination [8], [22] or Capture The Flag game scenarios. In those scenarios, the difficult problem is to give effective commands to form the best team collaboration rather than to control the individuals. It will be even more challenging if each individual has its own specialties and preferences.

VII. Acknowledgement

This research is supported in part by Interactive and Digital Media Programme Office (IDMPO), National Research Foundation (NRF) hosted at Media Development Authority (MDA) under Grant No.: MDA/IDM/2012/8/8-2 VOL 01. The authors would like to thank the associate editor and the anonymous reviewers for their invaluable comments and suggestions.

References

[1] "Game vs. movie industry," (Accessible till May 2014). [Online]. Available: http://mygaming.co.za/news/features/42864-game-vs-movie-industry.html

[2] J. E. Laird and M. van Lent, "Human-level AI's killer application: Interactive computer games," *AI magazine*, vol. 22, no. 2, pp. 15–25, 2001.

[3] D. B. Fogel, H. Timothy J, and D. R. Johnson, "A platform for evolving characters in competative games," in *Proceedings of the Congress on Evolutionary Computation*, 2004, pp. 1420–1426.

[4] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience for content creation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 54–67, 2010.

[5] P. Hingston, "A Turing Test for computer game bots," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 3, pp. 169–186, 2009.

[6] D. Wang, B. Subagdja, and A.-H. Tan, "Creating human-like autonomous players in real-time first person shooter computer games," in *Proceedings of the Conference on Innovative Applications of Artificial Intelligence*, 2009, pp. 173–178.

[7] G. N. Yannakakis and J. Hallam, "Real-time game adaptation for optimizing player satisfaction," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 2, pp. 121–133, 2009.

[8] H. Wang, Y. Gao, and X. Chen, "RL-DOT: A reinforcement learning NPC team for playing domination games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 17–26, 2010.

[9] A.-H. Tan, "FALCON: A fusion architecture for learning, cognition, and navigation," in *Proceedings of the International Joint Conference on Neural Networks*, 2004, pp. 3297–3302.

[10] G. A. Carpenter and S. Grossberg, "Adaptive Resonance Theory," in *The Handbook of Brain Theory and neural Networks*. MIT Press, 2003, pp. 87–90.

[11] J. Gemrot, R. Kadlec, M. Bda, O. Burkert, R. Pibil, J. Havlicek, L. Zemcak, J. Simlovic, R. Vansa, M. Stolba, and et al., "Pogamut 3 can assist developers in building AI (not only) for their videogame agents," *Agents for Games and Simulations*, pp. 1–15, 2009.

[12] R. Lopes and R. Bidarra, "Adaptivity challenges in games and simulations : A survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 2, pp. 1–14, 2011.

[13] M. Ponsen and P. Spronck, "Improving adaptive game AI with evolutionary learning," *Machine Learning*, pp. 389–396, 2004.

[14] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma, "Adaptive game AI with dynamic scripting," *Machine Learning*, vol. 63, pp. 217–248, 2006.

[15] D. J. Cook, L. B. Holder, and G. M. Youngblood, "Graph-based analysis of human transfer learing using a game testbed," *IEEE Transaction on Knowledge and Data Engineering*, vol. 19, pp. 1465–1478, 2007.

[16] N. Cole, S. J. Louis, and C. Miles, "Using a genetic algorithm to tune first-person shooter bots," in *Proceedings of the International Congress on Evolutionary Computation*, 2004, pp. 139–145.

[17] S. J. Louis and C. Miles, "Playing to learn: Case-injected genetic algorithms for learning to play computer games," *IEEE Transaction on Evolutionary Computation*, vol. 9, pp. 669–681, 2005.

[18] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the nero video game," *IEEE Transaction on Evolutionary Computation*, vol. 9, pp. 653–668, 2005.

[19] C. Miles and S. J. Louis, "Towards the co-evolution of influence map tree based strategy game players," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2006, pp. 75–82.

[20] R. L. Hy, A. Arrigoni, P. Bessiere, and O. Lebetel, "Teaching Bayesian behaviours to video game characters," *Robotics and Autonomous Systems*, vol. 47, pp. 177–185, 2004.

[21] I. C. Kim, "UTBot: A virtual agent platform for teaching agent system design," *Journal of Multimedia*, vol. 2, no. 1, pp. 48–53, 2007.

[22] M. Smith, S. Lee-Urban, and H. Munoz-Avila, "RETALIATE: Learning winning policies in first-person shooter games," in *Proceedings of the National Conference on Innovative Applications of Artificial Intelligence*, 2007, pp. 1801–1806.

[23] J. Schrum, I. V. Karpov, and R. Miikkulainen, "UTˆ2: Human-like behavior via neuroevolution of combat behavior and replay of human traces," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2011, pp. 329–336.

[24] P. Hingston, Ed., *Believable Bots: Can Computers Play Like People?* Springer, 2012.

[25] D. Gamez, Z. Fountas, and A. K. Fidjeland, "A neurally controlled computer game avatar with humanlike behavior," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 1, pp. 1–14, 2013.

[26] G. Kaminka, M. Veloso, S. Schaffer, C. Sollitto, R. Adobbati, A. Marshall, A. Scholer, and S. Tejada, "Gamebots: The ever-challenging multi-agent research test-bed," *Communications of the ACM*, vol. 45, no. 1, pp. 43–45, 2002.

[27] L. D. Nguyen, K.-Y. Woon, and A.-H. Tan, "A self-organizing neural model for multimedia information fusion," in *Proceedings of the Conference on Information Fusion*, 2008, pp. 1738–1744.

[28] B. Subagdja and A.-H. Tan, "A self-organizing neural network architecture for intentional planning agents," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2009, pp. 1081–1088.

[29] Y. Feng, T.-H. Teng, and A.-H. Tan, "Modelling situation awareness for context-aware decision support," *Expert Systems with Applications*, vol. 36, no. 1, pp. 455–463, 2009.

[30] A.-H. Tan and G.-W. Ng, "A biologically-inspired cognitive agent model integrating declarative knowledge and reinforcement learning," in *Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology*, 2010, pp. 248–251.

[31] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, pp. 759–771, 1991.

[32] A.-H. Tan, "Direct code access in self-organizing neural networks for reinforcement learning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2007, pp. 1071–1076.

[33] G. A. Carpenter and S. Grossberg, "ART 2: Self-organization of stable category recognition codes for analog input patterns." *Applied Optics*, vol. 26, no. 23, pp. 4919–4930, 1987.

[34] G. A. Carpenter, S. Grossberg, and D. Rosen, "ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition," *Neural Networks*, vol. 4, no. 4, pp. 493–504, 1991.

[35] A.-H. Tan and D. Xiao, "Self-organizing cognitive agents and reinforcement learning in multi-agent environment," in *Proceedings of the International Conference on Intelligent Agent Technologies*, 2005, pp. 351–357.

[36] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[37] A.-H. Tan, N. Lu, and D. Xiao, "Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback," *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 230–244, 2008.

[38] R. Small and C. B. Congdon, "Agent Smith: Towards an evolutionary rule-based agent for interactive dynamic games," in *Proceedings of IEEE Congress on Evolutionary Computation*, 2009, pp. 660–666.

[39] S. Feng and A.-H. Tan, "Self-organizing neural networks for behavior modeling in games," in *Proceedings of International Joint Conference on Neural Networks*, 2010, pp. 3649–3656.

[40] A. J. F. Leiva and J. L. O. Barragan, "Decision tree-based algorithms for implementing bot AI in UT2004," in *Foundations on Natural and Artificial Computation, Lecture Notes in Computer Science*. Springer, 2011, vol. 6686, pp. 383–392.

[41] R. A. Fisher, "The correlation between relatives on the supposition of mendelian inheritance," *Philosophical Transactions of the Royal Society of Edinburgh*, vol. 52, p. 399433, 1918.

[42] E. Tulving, *Elements of Episodic Memory*. Clarendon Press, 1983.

[43] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[44] T.-H. Teng, A.-H. Tan, and Y.-S. Tan, "Self-regulating action exploration in reinforcement learning," *Procedia Computer Science*, vol. 13, pp. 18–30, 2012.