9-2016

Towards autonomous behavior learning of non-player characters in games

Shu FENG

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

Citation

FENG, Shu and TAN, Ah-hwee. Towards autonomous behavior learning of non-player characters in games. (2016). *Expert Systems with Applications*. 56, (1), 89-99.
Available at: https://ink.library.smu.edu.sg/sis_research/5247

Towards Autonomous Behavior Learning of Non-Player Characters in Games

Shu Feng[a,b,*], Ah-Hwee Tan[a,**]

[a]School of Computer Engineering, Nanyang Technological University, Singapore
[b]NEC Laboratories Singapore, Singapore

Abstract

Non-Player-Characters (NPCs), as found in computer games, can be modelled as intelligent systems, which serve to improve the interactivity and playability of the games. Although reinforcement learning (RL) has been a promising approach to creating the behaviour models of non-player characters (NPC), an initial stage of exploration and low performance is typically required. On the other hand, imitative learning (IL) is an effective approach to pre-building a NPC's behavior model by observing the opponent's actions, but learning by imitation limits the agent's performance to that of its opponents. In view of their complementary strengths, this paper proposes a computational model unifying the two learning paradigms based on a class of self-organizing neural networks called Fusion Architecture for Learning and COgnition (FALCON). Specifically, two hybrid learning strategies, known as the Dual-Stage Learning (DSL) and the Mixed Model Learning (MML), are presented to realize the integration of the two distinct learning paradigms in one framework. The DSL and MML strategies have been applied to creating autonomous non-player characters (NPCs) in a first person shooting game named Unreal Tournament. Our experiments show that both DSL and MML are effective in producing NPCs with faster learning speed and better combat performance comparing with those built by traditional RL and IL methods. The proposed hybrid learning strategies thus provide an efficient method to building intelligent NPC agents in games and pave the way towards building autonomous expert and intelligent systems for other applications.

Keywords: behavior learning; reinforcement learning; imitative learning; self-organizing neural network; intelligent agent;

1. Introduction

Intelligent non-player characters (NPCs) in computer games can potentially make the games more challenging and enjoyable. As such, behavior modeling of non-player character (NPC) has become an important component in computer games, especially in first person shooting games (FPS) (Wang et al., 2009; Wang & Tan, 2015).

In the game environment, each NPC is essentially an autonomous agent, which is expected to function and adapt by themselves in a complex and dynamic environment. Consequently, a popular approach to developing intelligent agents is through machine learning algorithms.

In particular, reinforcement learning (RL) is considered by many to be an appropriate paradigm for an agent to autonomously acquire its action policy through interacting with its environment in a dynamic process. In general, an RL agent makes responses to the environment in order to maximize the future expected rewards with respect to its goals and motivations. However, in a first person shooting game, an NPC without prior knowledge will perform poorly at the initial stage as they have to spend substantial time in exploring and learning the environmental information. Playing with these NPCs certainly takes the fun out of the game. Moreover, specific types of knowledge may be too complex to learn through reinforcement feedback.

To overcome these drawbacks, a possible remedy is to pre-insert domain knowledge into the learning agents, in order to increase learning efficacy, shorten convergence time as well as enhance NPCs' performance. Although there have been extensive works towards improving RL with prior knowledge, the methods for obtaining and integrating knowledge are still an open problem. Most of the earlier works complement reinforcement learning by direct inserting prior knowledge

*Email: feng0027@e.ntu.edu.sg
**Email: asahtan@ntu.edu.sg

through either encoding domain knowledge in the learning architecture (Shapiro et al., 2001; Busoniu et al., 2010), adding prior knowledge as a rule base Song et al. (2004), or using an added-on module to provide prior knowledge (Dixon et al., 2000; Moreno et al., 2004). An obvious drawback of direct insertion is that the prior knowledge cannot be used in exploitation during learning and cannot adapt to changes in the environment.

In contrast to reinforcement learning, imitative learning with explicit supervisory teaching signals is a promising approach to acquiring complex behavior for autonomous agents. The knowledge learnt by imitation can be used readily as the agent's behavior model (Feng & Tan, 2010). Imitative learning and reinforcement learning can been seen as two complementary learning paradigms. While the former is effective and fast in acquiring patterns, it strictly relies on the training data and typically is not used in real time adaptation. On the other hand, reinforcement learning is good in learning from experience and adapting to the environment in real time. However, it is less effective for fast learning due to the lack of explicit teaching signals. In view of their complementary strengths, this work aims to combine the fast learning capability of IL with real-time adaptive ability of RL for a better performance.

Specifically, this paper presents two hybrid learning strategies, known as Dual-Stage Learning (DSL) and Mixed Model Learning (MML) to realize the integration of the two learning paradigms in one unified framework based on a class of self-organizing neural networks, namely Fusion Architecture for Learning and COgnition (FALCON) (Tan, 2004; Xiao & Tan, 2007). FALCON learns cognitive codes encoding multi-dimensional mappings simultaneously across the multi-modal pattern channels. By using competitive coding as the underlying adaptation principle, FALCON is capable of supporting multiple learning paradigms, including unsupervised learning, supervised learning and reinforcement learning (Tan et al., 2007).

The DSL strategy combines imitative learning and reinforcement learning in two stages. In the imitative learning stage, FALCON learns from opponent behaviour patterns to build the initial behaviour model of an autonomous agent. Subsequently, in the reinforcement learning stage, the agent further adapts in real time through Q-learning while applying the prior knowledge for exploitation. Compared with DSL, the MML strategy combines the two learning paradigms tightly into one model, in which both IL and RL work in an interleaving manner sharing a common knowledge field.

We have evaluated various learning methods and strategies in a first person shooting game named Unreal Tournament(UT). Our experiments show that the NPCs learned with DSL and MML produce a higher level of performance compared with the traditional RL and IL methods.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 defines the problems addressed by this work. Section 4 explains the issues and challenges. Section 5 and Section 6 introduce the learning model and presents the methods of DSL and MML. Section 7 reports the learning tasks and the experiments. Finally, section 8 concludes and discusses the future work.

2. Related Work

In the past years, NPCs in computer games have developed significantly in terms of behavior modeling. This section reviews some of the related work.

For commercial games, the most commonly used method is via rule based approaches. Ji *et al.*(2014) proposed a behavior tree to manage the controlling of behaviors. By designing complex intelligent role behaviors in logical ways, this method could easily integrate expert experience into the intelligent system. However, as it inherited the use of action and condition rules based on finite state machine, the agent is not able to adapt to different environment.

Akbar *et al.*(2015) applied Gaussian distribution with fuzzy logic to create natural variation actions of each NPC and select the optimal action. As the fuzzy method still has to follow logic rules, this method doesn't help NPCs to evolve and capture new knowledge.

In view of the limitations of the rule-based approaches, learning based techniques have attracted much attention for behavior modeling. David *et al.*(2014) used genetic algorithm for evaluation and search mechanism for decision makings. Stanley *et al.*(2005) applied evolution algorithm to enable the NPCs evolve behaviors through interacting with players, thus keeping the game interesting. Although evolutionary algorithms can improve the performance continuously, the final solution cannot be guaranteed as the global optimal solution. Moreover, tuning parameters during learning is time consuming and not suitable for real-time video games.

On the other hand, many have applied imitative learning to create NPCs by mimicking the behaviour patterns of human beings in order to achieve the human-like behaviors (Zanetti & Rhalibi, 2004; Bauckhage et al., 2003; Feng & Tan, 2010). These imitation based learning enables fast learning and is capable of acquiring complex behavior patterns. However, imitative learning requires specific observations to be available. Furthermore, in real time processing, imitative learning cannot associate the behaviour with the underlying motivations or goals.

Besides imitative learning, many have applied reinforcement learning (RL) successfully to autonomous NPCs for learning strategies and behaviors in a dynamic environment. Especially in combat scenarios games, RL is good at helping NPCs to learn through experience with the supplement of necessary initial knowledge. Glavin *et al.*(2015) applied reinforcement learning to enable NPCs to get experience from gaming experience, and improve their fighting skills over time based on the damage given to opponents. Ponce *et al.*(2014) applied MaxQ-Q based reinforcement learning within a hierarchical structure to enhance user's experience. Wang *et al.*(2015) utilized reinforcement learning in a first person shooting game to learn behavior strategy and effectiveness of different weapons. However, using pure reinforcement learning, an initial stage of exploration is required and depending on the problem domain, this process may take a long time.

In view of the above issue, a popular way to initialize the learning agent is with human knowledge (Unemi, 2000; Dixon et al., 2000). They improve the performance of reinforcement learning by introducing relatively simple human knowledge such as intrinsic behaviors to reduce learning time. However, the prior knowledge is embedded and not represented in the same form as the learned knowledge. As such, the knowledge cannot be further modified in real-time learning. Bayesian based method is another principle way to incorporate prior knowledge into reinforcement learning (Ghavamzadeh et al., 2015; Doshi-Velez et al., 2015; Jonschkowski & Brock, 2014). Here the prior knowledge mostly refers to probability distributions. Taken from prior observations, the knowledge helps to start up the learning as well as to continuously guide decision makings. However, they need to be designed beforehand based on the specific problem domains, not to mention the experience based knowledge may include objective bias. Kengo *et al.*(2005) enhanced the reinforcement learning agent by applying goal state prior knowledge to the agent in order to modulate the decision making by giving priorities to the goal oriented actions. In the chosen game scenarios, prior knowledge is designed by the domain expert. Again, once it is applied, the knowledge fixed and can't be further adapted. Framling(2007) introduced a reinforcement learning model with pre-existing knowledge. A bi-memory system including the concepts of short term and long term memories is proposed to modulate the exploration in state space in order to make a faster learning. However, this model is not a universal architecture. Consequently, specific heuristic rules are still required for proposing the pre-existing knowledge.

In summary, although there have been efforts to address the limitation of reinforcement learning using domain knowledge, none of the prior approaches has attempted to combine imitative learning with reinforcement learning. This forms the unique focus of our work.

3. Problem Statement

A learning NPC is essentially an autonomous agent which strives to acquire a desirable behavior model through its interaction with the environment with respect to its goals. In order to define our problem statement, we review the following definitions as used in the field of reinforcement learning.

Definition (State Space): State space S of an agent is a set of states $\{s_1, s_2, ..., s_n\}$, where each state s_i represents a snapshot of the environment in which the agent resides.

Definition (Action Space): Action space A of an agent is a set of actions $\{a_1, a_2, ..., a_m\}$, where each action indicates an unique response to the environment.

Definition (Reward): Reward r is a real-valued evaluative feedback received by an agent from its environment.

Definition (Behaviour Model): The behaviour model F of an agent is a an internal model or function mapping from the state space S to the action space A of the agent. More formally, the behaviour model is defined by

$$\mathcal{F} : \mathcal{S} \longrightarrow \mathcal{A}, \tag{1}$$

where each state $s_i \in S$ is mapped to an action $a_i \in A$. A behaviour model dictates how the NPC responds to the situations in its environment. It is thus akin to the action policy as used in the literature of reinforcement learning.

There are two distinctive approaches to acquiring a behaviour model. One is to learn the behaviour function $\mathcal{F}$ directly from a given set of sample pairs (s_i, a_i), where a_i indicates the action to be taken in state s_i. In the context of first person shooting games, such training samples can be acquired readily through observing the behaviour of the agent's opponents. This paradigm of learning from observations is known as imitative learning (IL).

Definition (Imitative learning): Imitative learning is a learning process, wherein an agent infers its behaviour model function from a set of observations, each of which contains an input state $s_i \in S$ and an action $a_i \in A$.

The basic assumption of imitative learning is that each observed behaviour is appropriate for the specific environment in which the agent resides.

Another approach to building a behaviour model is through learning a value function, which specifies the payoff value of performing an action in a given situation. More formally, the value function is defined by

$$\mathcal{V} : \mathcal{S} \times \mathcal{A} \longrightarrow \mathcal{R}\pounds \tag{2}$$

where each state-action pair (s_i, a_i) is associated with a reward value $r \in R$. During decision making, the agent can then take the action a which has the maximal reward in a state s. This is known as reinforcement learning (RL).

Definition (Reinforcement learning): Reinforcement learning is a learning process, wherein an agent learns a value function or an action policy and adjusts its behaviour patterns so as to maximize the future payoff, based on the reward signal $r_i \in R$ when the action $a_i \in A$ is performed in the state $s_i \in S$.

RL assumes there's always a best choice of action for the specific surroundings in which the agent is situated, among all the possible choices.

4. Issues and Challenges

Imitative learning and reinforcement learning both learn the associations among the states (S), actions (A), and values (R) but do so in distinct ways. As imitative learns the mapping ($S \rightarrow A$) from existing patterns, the knowledge acquired is limited by the quality of the observations available. Reinforcement learning focuses on learning action policies and estimating the values to indicate the goodness of action-state pairs. However, exploration in the initial stage can be time consuming. The challenge is how to integrate the two learning methods into one unified framework, so as to combine their complementary merits for better performance.

4.1. Unifying Knowledge Representation

The knowledge learned via imitative learning and reinforcement learning are distinct in nature. By imitative learning, the knowledge is in the form of a series of state-action pairs $f_i(s_i, a_i)$ with the logic that when the state s_i is satisfied, the action a_i will be taken consequently. On the other hand, the knowledge acquired by reinforcement learning is a value function, associating each of the state-action pairs with a reward value. Given a 3-tuple $v_i(s_i, a_i, r_i)$, the logic states that if an action a_i is taken in state s_i, the estimated expected reward value is given by r_i. The challenge is how to derive a unified knowledge structure which can fuse and represent these different types of knowledge that can be learned through either imitative learning and reinforcement learning.

4.2. Unifying decision making

Note that the knowledge learned through imitative learning is a behaviour function $\mathcal{F}$ from input states to actions. During action selection, given the current state, an action can be chosen by simply feeding the input state vector into the behaviour function. On the other hand, the knowledge acquired by reinforcement learning is a value function, associating each of the state-action pairs with a reward value. Given the current state, the agent evaluates the value of performing each possible action and selects the action with the maximal reward value. Integrating these two distinct decision making processes is non-trivial as the knowledge learned by one method may not perform well with another decision making process. Simply combining the two types of knowledge may even degrade the overall performance. Therefore, a key challenge is how to derive an integrated decision making process, so that the different types of knowledge can be used in an interchangeable manner.

4.3. Unifying learning Process

Note that imitative learning relies on a large quantity of training data labelled with explicit input states and output actions. The behaviour function inferred by imitative learning can map new inputs to a desired decision. In contrast, learning by reinforcement feedback focuses on online performance without presenting explicit supervisory input-output patterns. In addition, reinforcement learning needs to balance between exploration and exploitation. At the initial stage, exploration is typically performed to explore the utility of new actions. As the learning progress, the agent tends to perform exploitation, by selecting actions with the highest Q-value. Whereas ultimately exploitation is necessary for maximizing the rewards, exploration is necessary to search or discover an optimal solution to the problem. When an agent is trained by imitative learning followed by reinforcement learning, for example, the "prior rules" acquired by imitative learning earlier may tip the balance between exploration and exploitation in reinforcement learning. The challenge in unifying learning is how to derive a unified model that is compatible for different types of learning methods and is able to find a good trade-off between exploration and exploitation.

5. The Learning Model

This work proposes an integrated behavior learning approach based on a class of self-organizing neural networks, known as Fusion Architecture for Learning and COgnition (FALCON) (Tan, 2004; Xiao & Tan, 2007). FALCON is a three-channel fusion Adaptive Resonance Theory (ART) network (Tan et al., 2007) that incorporates temporal difference methods (Sutton & Barto, 1998; Watkins & Dayan, 1992) into Adaptive Resonance Theory (ART) models (Carpenter & Grossberg, 1987b,a) for reinforcement learning. By inheriting the ART code stabilizing and dynamic network expansion mechanism, FALCON is capable of learning cognitive nodes encoding multi-dimensional mappings across multi-modal input patterns, involving states, actions, and rewards, in an online and incremental manner.

As FALCON is designed to support a myriad of learning paradigms, including supervised learning, unsurprised learning and reinforcement learning (Tan et al., 2007), it makes a natural choice for achieving the integrations of imitative learning and reinforcement learning.

A summary of the FALCON model and the basic learning and performing algorithms is given below. In the following section, we shall show how FALCON may unify the reasoning and the learning processes of imitative learning and reinforcement learning using various hybrid learning strategies.

As shown in Figure 1, FALCON employs a three-channel architecture comprising a category field F_2^c and three input fields, namely a sensory field F_1^{c1} for representing current states, a motor field F_1^{c2} for representing actions, and a feedback field F_1^{c3} for representing the reward values. The dynamics of FALCON based on fuzzy ART operations (Carpenter et al., 1991; Tan, 1997), is described below.

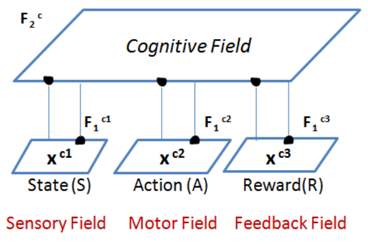

Figure 1: The FALCON architecture.

Input vectors: Let $\mathbf{S} = (s_1, s_2, ..., s_n)$ denote the state vector, where s_i indicates the sensory input i. Let $\mathbf{A} = (a_1, a_2, ..., a_m)$ denote the action vector, where a_i indicates a possible action i. Let $\mathbf{R} = (r, \bar{r})$ denote the reward vector, where $r \in [0, 1]$ and $\bar{r} = 1 - r$.

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

Weight vectors: Let $\mathbf{w}_j^{ck}$ denote the weight vector associated with the jth node in F_2^c for learning the input representation in F_1^{ck} for $k = 1, ..., 3$. Initially, F_2^c contains only one *uncommitted* node, and its weight vectors contain all 1's. When an *uncommitted* node is selected to learn an association, it becomes *committed*.

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

Code activation: A bottom-up propagation process first takes place in which the activities of the category nodes in the F_2^c field are computed. Specifically, given the activity vectors $\mathbf{x}^{c1}$, $\mathbf{x}^{c2}$, and $\mathbf{x}^{c3}$ (in the input fields F_1^{c1}, F_1^{c2}, and F_1^{c3}, respectively), for each F_2^c node j, the choice function T_j is computed as follows:

$$T_j^c = \sum_{k=1}^{K} \gamma^{ck} \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_j^{ck}|}{\alpha^{ck} + |\mathbf{w}_j^{ck}|}, \qquad (3)$$

where the fuzzy AND operation $\wedge$ is defined by $(p \wedge q)_i \equiv \min(p_i, q_i)$ and the norm $|\cdot|$ is defined by $|p| \equiv \sum_i p_i$ for vectors p and q. In essence, the choice function T_j computes the similarity of the activity vectors with their respective weight vectors of the F_2^c node j with respect to the norm of individual weight vectors.

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

$$T_J^c = \max\{T_j^c : \text{for all} \quad F_2^c \quad \text{node} \quad j\}. \qquad (4)$$

When a category choice is made at node J, $y_J^c = 1$; and $y_j^c = 0$ for all $j \neq J$. This indicates a winner-take-all strategy.

Template matching: Before node J can be used for learning, a template matching process checks that the weight templates of node J are sufficiently close to their respective activity patterns. Specifically, resonance occurs if for each channel k, the *match function* m_J^{ck} of the chosen node J meets its vigilance criterion

$$m_J^{ck} = \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{x}^{ck}|} \geq \rho^{ck}. \qquad (5)$$

Whereas the match function computes the similarity between the input and weight vectors with respect to the norm of the weight vectors, the match function computes the similarity with respect to the norm of the input vectors. The choice and match functions work cooperatively to achieve stable coding and maximize code compression.

When resonance occurs, learning ensues, as defined below. If any of the vigilance constraints is violated, mismatch reset occurs in which the value of the choice function T^c_J is set to 0 for the duration of the input presentation. With a *match tracking* process, at the beginning of each input presentation, the vigilance parameter ρ^{c1} equals a baseline vigilance $\bar{\rho}^{c1}$. If a mismatch reset occurs, ρ^{c1} is increased until it is slightly larger than the match function m^{c1}_J. The search process then selects another F^c_2 node J under the revised vigilance criterion until a resonance is achieved. This search and test process is guaranteed to end as FALCON will either find a *committed* node that satisfies the vigilance criterion or activate an *uncommitted* node which would definitely satisfy the criterion due to its initial weight values of all 1*s*.

Template learning: Once a node J is selected, for each channel k, the weight vector $\mathbf{w}^{ck}_J$ is modified by the following learning rule:

$$\mathbf{w}^{ck(new)}_J = (1-\beta^{ck})\mathbf{w}^{ck(old)}_J + \beta^{ck}(\mathbf{x}^{ck} \wedge \mathbf{w}^{ck(old)}_J). \quad (6)$$

The learning rule adjusts the weight values towards the fuzzy AND of their original values and the respective weight values. The rationale is to learn by encoding the common attribute values of the input vectors and the weight vectors. For an uncommitted node J, the learning rates β^{ck} are typically set to 1. For committed nodes, β^{ck} can remain as 1 for fast learning or below 1 for slow learning in a noisy environment.

Code creation: Our implementation of FALCON maintains ONE uncommitted node the F^c_2 field at any one time. When an uncommitted node is selecting for learning, it becomes *committed* and a new uncommitted node is added to the F^c_2 field. FALCON thus expands its network architecture dynamically in response to the input patterns.

6. Learning in FALCON

Note that FALCON is designed to learn cognitive nodes encoding multi-dimensional mappings across multi-modal input patterns, involving states, actions, and rewards, in an online and incremental manner. Specifically, each FALCON cognitive node encodes a 3-tuple knowledge structure $\mathbf{F} = (\mathbf{w}^{c1}_j, \mathbf{w}^{c2}_j, \mathbf{w}^{c3}_j)$ representing an association among the template state vector ($\mathbf{w}^{c1}_j$, the template action vector $\mathbf{w}^{c2}_j$) and the template reward vector $\mathbf{w}^{c3}_j$.

Using competitive coding as the underlying adaptation principle, FALCON supports a variety of learning paradigms depending on the operating mode and the activity patterns presented across the three pattern channels. For example, when the state vector $\mathbf{S}$ and the action vector, representing the state s and the action a respectively, are presented simultaneously in a learning mode, FALCON will learn a new cognitive node or refine an existing cognitive node to associate the state s and the action a. However, when the state vector $\mathbf{S}$ and the action vector are presented simultaneously in a predicting mode, FALCON will read out the reward vector $\mathbf{R}$, indicating the reward value r of performing the action a in the given state s. We present each of these cases, especially for imitative learning and reinforcement learning, in the subsequent sections.

6.1. Imitative Learning

Imitative learning involves the learning of an action policy which maps directly from current states to desired actions. This can be realized in FALCON by learning the observed behavior patterns directly. Figure 2 shows the pattern configuration of FALCON in imitative learning. Given a pair of state vector $\mathbf{S}$ and action vector $\mathbf{A}$, the activity vectors across the three pattern channels are set as $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{x}^{c3} = \mathbf{R} = (1, 0)$. FALCON then performs the code activation and code competition processes, according to equations (3) and (4), to select a category node J in the F^c_2 field. Once the template matching condition is satisfied, the category node J undergoes template learning, wherein it learns the association between $\mathbf{S}$ and $\mathbf{A}$ (6).

Note that when the uncommitted node is selected to encode a novel state-action pair, a new uncommitted node will be created. As a result, FALCON expands its architecture by learning the association between the observed states and actions continuously. As shown in subsequent sections, imitative learning in FALCON can be performed in a batch mode from behavior patterns recorded over period or in an incremental mode from behaviour observed in real time. The procedure for imitative learning is summarized in Table 1.

For action selection, FALCON receives a state vector $\mathbf{S}$ in performing mode and selects a category node J in the F^c_2 field which determines the action. Specifically, the activity vectors $\mathbf{x^{c1}}$, $\mathbf{x^{c2}}$, and $\mathbf{x^{c3}}$ are initialized by $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = (1, ..., 1)$, and $\mathbf{x}^{c3} = (1, 0)$. Through a direct code access procedure (Tan, 2007), FALCON searches for the cognitive node which matches with the

Table 1: Imitative learning in FALCON network.

1.	Initialize the FALCON network.
2.	Observe the opponent's state and formulate a state vector **S**.
3.	Observe the opponent's action and formulate a action vector **A**.
4.	Present the state vector **S**, the action vector **A**, and the reward vector **R**=(1,0) to FALCON for learning:
6.	Repeat from Step 2.

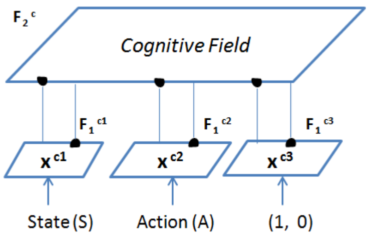

Figure 2: FALCON in imitative learning.

current state using the code activation and code competition processes according to equations (3) and (4).

Upon selecting a winning F_2^c node J, the chosen node J performs a readout of its weight vector into the action field F_1^{c2} such that

$$\mathbf{x}^{c2(new)} = \mathbf{x}^{c2(old)} \wedge \mathbf{w}_J^{c2}. \quad (7)$$

FALCON then examines the output activities of the action vector $\mathbf{x}^{c2}$ and selects an action a_I, which has the highest activation value

$$x_I^{c2} = \max\{x_i^{c2(new)} : \text{for all} \quad F_1^{c2} \quad \text{node} \quad i\}. \quad (8)$$

6.2. *Reinforcement learning*

Reinforcement learning can be realized in FALCON through learning the value policy by associating the input activity patterns across the sensory, action and reward fields. Learning from delayed evaluative feedback is further enabled by incorporating a Temporal Difference (TD) method to estimate and learn the value functions of action-state pairs $Q(s, a)$ that indicates the goodness to take a certain action a in a given state s.

Figure 3 shows the pattern configuration of the TD-FALCON network model in the reinforcement learning paradigm. Given the current state s, the FALCON network first chooses an action a to perform by following an action selection policy. For action selection, the state vector **S**, the action vector $\mathbf{A} = (1, ...1)$ and the reward vector $\mathbf{R} = (1, 0)$ are presented to FALCON. The setting of the action and reward vectors serves to select an action which is expected to lead to the maximal reward in the given state.

After performing the chosen action, a reward may be received from the environment and a TD formula is used to compute a new estimate of the Q value of performing the chosen action a in the current state s. The new Q value is then used as the teaching signal for TD-FALCON to learn the association of the current state s and the chosen action a to the estimated Q value. The procedure for reinforcement learning in TD-FALCON (Tan, 2007) is summarized in Table 2.

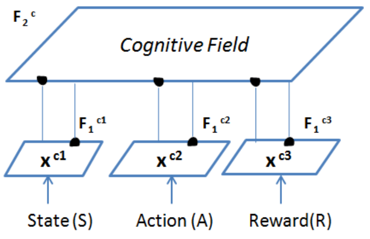

Figure 3: TD-FALCON in reinforcement learning.

A typical Temporal Difference (TD) equation for iterative estimation of value functions Q(s,a) is given by

$$\Delta Q(s, a) = \alpha TD_{err} \quad (9)$$

where $\alpha \in [0, 1]$ is the learning parameter and TD_{err} is a function of the current Q-value predicted by FALCON and the Q-value newly computed by the TD formula.

For used with ART-based neural networks, a Bounded Q-learning rule is generally employed, wherein the temporal error term is computed by

$$\Delta Q(s, a) = \alpha TD_{err} (1 - Q(s, a)). \quad (10)$$

where $TD_{err} = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$, of which 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'. It is important to note

Table 2: Reinforcement learning by FALCON with direct code access.

```
1. Initialize the TD-FALCON network.
2. Sense the environment and formulate a state vector $\mathbf{S}$.
3. Select the action $a$ with the maximal Q(s,a) value by presenting the state vector $\mathbf{S}$, action
   vector $\mathbf{A}$=(1,...1) and the reward vector $\mathbf{R}$=(1,0) to TD-FALCON:
4. Perform the action $a$, and receive a reward $r$ from the environment.
5. Estimate the revised value function $Q(s,a)$ following a Temporal Difference formula such as
   $\Delta Q(s,a) = \alpha \mathrm{TD}_{err}$.
6. Perform learning in TD-FALCON, by presenting the state vector $\mathbf{S}$, action vector
   $\mathbf{A}$=$(a_1, a_2, ..., a_n)$, where $a_I$=1 if $a_I$ corresponds to the action $a$, $a_i = 0$
   for $i \neq I$, and reward vector $\mathbf{R}$=($Q(s,a)$,1-$Q(s,a)$) to TD-FALCON for learning:
7. Update the current state by s=s'.
8. Repeat from Step 2 until $s$ is a terminal state.
```

that the Q values involved in estimating $\max_{a'} Q(s', a')$ are computed by the same FALCON network itself and not by a separate reinforcement learning system. The Q-learning update rule is applied to all states that the agent traverses. With value iteration, the value function $Q(s,a)$ is expected to converge to $r + \gamma \max_{a'} Q(s', a')$ over time. By incorporating the scaling term $1 - Q(s,a)$, the adjustment of Q values will be self-scaling so that they will not be increased beyond 1. The learning rule thus provides a smooth normalization of the Q values. If the reward value r is constrained between 0 and 1, we can guarantee that the Q values will remain to be bounded between 0 and 1.

6.3. Dual-Stage Learning (DSL)

The Dual-Stage Learning (DSL) strategy is proposed for hybrid learning based on TD-FALCON, combining the complementary merits of the two learning paradigms, namely fast supervised learning ability of imitative learning and continual real-time adaptation ability of reinforcement learning, in a sequential manner.

Under the DSL strategy, imitative learning, based on a collection of data samples, is first used to set up the knowledge structure of the learner. Using the TD-FALCON's competitive coding principle, an action policy is learned in the form of associative pattern chunks $(\mathbf{S}, \mathbf{A})$, and stored in the cognitive field of TD-FALCON.

During reinforcement learning, the associative pattern chunks serve as the base knowledge for immediate performance and subsequent knowledge refinement. With a Temporal Difference (TD) learning method, TD-FALCON refines and expands the existing knowledge in real-time according to the feedback received from its interaction with the environment.

The process of the Dual-Stage Learning strategy is illustrated in Figure 4. During imitative learning, giv-

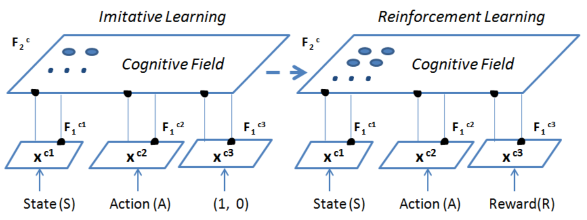

Figure 4: The Dual-Stage Learning procedure.

en an associative pattern pair, namely the state vector $\mathbf{S}$ and the action vector $\mathbf{A}$, the activity vectors of TD-FALCON are initiated to $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{x}^{c3} = \mathbf{R} = (q, 1-q)$, where $q \in [0,1]$ is the default expected reward of the inserted knowledge. During the second stage of reinforcement learning, TD-FALCON learns from a 3-tuple, consisting of the state vector $\mathbf{S}$, the action vector $\mathbf{A}$, and the reward vector $\mathbf{R} = (Q, \overline{Q})$. The algorithms of imitative learning and reinforcement learning follow those presented in Table 1 and Table 2 respectively.

Note that, in the DSL strategy, action policies can be seen as transferring from an imitative learner to an reinforcement learner as prior knowledge. If there are N association pairs learned in the imitative learning stage, the N pairs will be transferred totally into the second learning stage as pre-existing codes $(C_1, C_2, C_3, ...C_N)$ in the cognitive field F_2^c. In the most general case, the cognitive codes $C_n (n = 1, ..., N)$ can be initialized with different expected reward values $Q_n(s,a) = q_n, (n = 1, ..., N)$, which indicates that the estimated goodness to take the action a in the given state s.

Different from pure reinforcement learning, wherein an agent starts learning by exploring random actions, the prior knowledge transferred enables an agent to start exploitation with the pre-existing codes in F_2^c. When a cognitive code C_J is selected for learning, its weight vector $\mathbf{w}_{C_J}^{ck}$ is updated by the equation (6) and the corresponding state-action value q_J will be estimated again and revised by equation (10).

Lemma: Following the DSL strategy, a given set of pre-existing codes $(C_1, C_2, \ldots, C_N)$ learned via imitative learning is only usable for exploitation in reinforcement learning if their corresponding expected reward values $Q_n(s,a)$ satisfies the reward vigilance criterion, i.e. $q_n \geq \rho^{c3}$, for $n = 1, \ldots, N$.

Proof: Suppose there is a cognitive node J with an initialized reward value of $Q_J(s,a) < \rho^{c3}$. When a category choice is made at code J, the vigilance criterion will be violated because

$$m_J^{c3} = \frac{|\mathbf{x}^{c3} \wedge \mathbf{w}_J^{c3}|}{|\mathbf{x}^{c3}|} = q_J < \rho^{c3}. \quad (11)$$

where $\mathbf{x}^{c3} = (1,0)'$, and $\mathbf{w}_J^{c3} = (q_J, 1-q_J)'$. This causes the code J to be rejected for prediction.

Therefore, in the second stage of reinforcement learning, for the learning agent to perform by doing exploitation with the pre-existing codes $(C_1, C_2, C_3, ...C_N)$ instead of random choices, the following condition should be satisfied: the reward vigilance parameter ρ^{c3} must be lower than the minimum value of all the initialized prior reward values:

$$min\{Q_n(s,a) = q_n\} \geq \rho^{c3} \quad (n = 1, 2, ..., N). \quad (12)$$

6.4. Mixed model learning

The Mixed Model Learning (MML) strategy integrates imitative learning and reinforcement learning in a single knowledge framework in an interleaving manner. The process of ML is illustrated in Figure 5. Note that TD-FALCON comprises a cognitive field F_2^c and three input fields: a sensory field F_1^{c1} for representing current states, a motor field F_1^{c2} for representing actions, and a feedback field F_1^{c3} for representing the reward values. Using the Mixed Model Learning method, the three input fields obtain their state, action and reward patterns based on the behaviour of the agent and its opponents.

Specifically, a set of three input patterns are used for imitative learning, namely $\mathbf{S^o}$, $\mathbf{A^o}$, and $\mathbf{R^o}$, representing the opponent's state, action, and feedback from the environment respectively. Another set of three input patterns are dedicated to reinforcement learning, namely $\mathbf{S}$, $\mathbf{A}$, and $\mathbf{R}$, representing the agent's current state, action, and reward received from the environment respectively.

Note that the six input patterns are not to be active at the same time as TD-FALCON alternates between the two learning methods. As summarised in Table 3, TD-FALCON first decides between the imitative learning mode and the reinforcement learning mode and then activates the learning of the corresponding input patterns.

Whereas reinforcement learning is performance regularly upon receiving the reward signals, imitative learning is done selectively in a strategic manner. Specifically, when a negative reward is received, imitative learning is carried out following reinforcement learning. The rationale is that in a two-player zero sum game, a player's penalty will typically be the outcome of a right action taken by its opponent.

For imitative learning, TD-FALCON senses the opponent's state s^o and action a^o, represented as state vector $\mathbf{S^o}$ and action vector $\mathbf{A^o}$ respectively. The activity vectors of the three input fields are subsequently set as $\mathbf{x}^{c1} = \mathbf{S^o}$, $\mathbf{x}^{c2} = \mathbf{A^o}$, and $\mathbf{x}^{c3} = \mathbf{R^o} = (q, 1-q)$. For reinforcement learning, TD-FALCON follows the typically procedure of setting the activity vectors as $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{x}^{c3} = \mathbf{R} = (Q, \overline{Q})$.

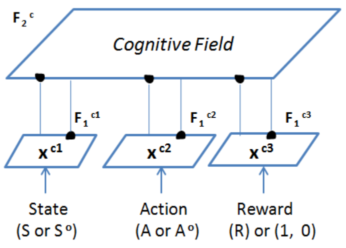

Figure 5: The Mixed Model Learning strategy.

7. Benchmark Evaluation

7.1. The Unreal Tournament Environment

Unreal Tournament (UT) is a first person shooting game featuring close combat fighting between non-player characters and human players in a virtual environment. Figure 6 provides a snapshot of the game environment taken from the view of a human player. The armed soldiers running and shooting in the environment are non-player characters, called Bots. The gun shown at the lower right hand corner is controlled by the human player. In our experiments, we use a "Deathmatch" mode, in which every Bot must fight with any other player in order to survive and win. UT does not merely offer an environment for gaming. More importantly, it also provides a platform for building and evaluating autonomous agents. Specifically, an Integrated Development Environment (IDE), called Pogamut (Pogamut, 2016), is available to developers for building agents for

Table 3: The Mixed Model Learning Method.

1. Initialize the TD-FALCON network.
2. Sense the environment and formulate a state representation s.
3. Obtain the opponent's state and formulate a state representation s^o.
4. Observe the action a^o taken by the opponent.
5. Choose the action a with the maximal Q(s,a) value by presenting the corresponding state vector **S**, action vector **A**=(1,...1) and the reward vector **R**=(1,0) to TD-FALCON.
6. Perform the action a, and receive a reward r from the environment.
7. Observe the next state s'.
8. Estimate the revised value function $Q(s,a)$ following a Temporal Difference formula such as $\Delta Q(s,a) = \alpha \mathrm{TD}_{err}$.
9. Perform learning in TD-FALCON, by presenting the state vector **S**, action vector **A**=$(a_1, a_2, ..., a_n)$, where a_I=1 if a_I corresponds to the action a, $a_i = 0$ for $i \neq I$, and reward vector **R**=($Q(s,a)$,1-$Q(s,a)$) to TD-FALCON for learning.
10. When a negative reward is received, perform imitative learning by presenting the state vector $\mathbf{S^o}$, the action vector $\mathbf{A^o} = (a_1, a_2, ..., a_n)$, where a_I=1 if a_I corresponds to the action a^o and $a_i = 0$ for $i \neq I$, and the reward vector $\mathbf{R^o} = (q, 1-q)$ to TD-FALCON for learning.
11. Update the current state by s=s'.
12. Repeat from Step 2 until s is a terminal state.

the UT environment. This means the developers can implement their own agents (or Bots) using any specific algorithm and run them in UT.

Figure 6: Unreal Tournament 2004 game environment.

Figure 7 (adapted from (Pogamut, 2016)) shows the interface between Pogamut and the Unreal game server. Pogamut runs as a plug-in for the NetBeans Java development environment. It communicates with the UT2004 game through Gamebots 2004 (GB2004), which is a built-in server inside UT2004 for exporting information from the game to the agent and vice versa. Pogamut also has a built-in parser module, which is used for translating messages into Java objects and vice versa.

7.2. Behavior learning tasks

There are in total eight types of behaviors designed for the agent (as shown in Table 4). Upon receiving the information from the environment, the agent is designed to make responses by choosing one of the eight

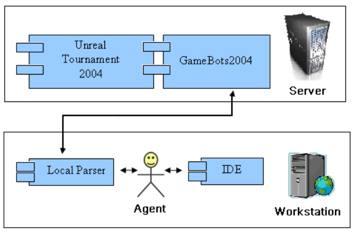

Figure 7: The interface between Pogamut and the Unreal Tournament 2004 (adapted from (Pogamut, 2016)).

behaviors based on ten state attributes (shown in Table 5). The behavior learning task in our experiment platform is to learn from the performance of a sample agent, called *Hunter*, which has the same types of behaviors and attributes as our agent, and exhibits a full range of combat competency based on its rule-based knowledge. There are altogether eight main rules captured in the Hunter's behavior mechanism based on these state attributes. They are summarized in Table 6.

7.2.1. Imitative learning from sample bot

For imitative learning, the empirical training data is obtained from the sample bots available. When playing the UT game, the states and behavior patterns of the Hunter Bot are recorded as training data for off-line learning or sensed by agent in real time during on-line learning. Each training example consists of a vector of the state attribute values as well as the chosen be-

Table 4: The eight behaviors of the Hunter Bot.

No.	*Behaviors*	*Description*
A_1	*ChangeToBetterWeapon*	Switch to a better weapon
A_2	*Engage*	Shoot the enemy
A_3	*StopShooting*	Stop shooting.
A_4	*RespondToHit*	Turn around, try to find the enemy
A_5	*Pursue*	Pursue the enemy spotted
A_6	*Walk*	Walk and check walking path
A_7	*GrabItem*	Grab the most suitable item
A_8	*GetMedicalKit*	Pick up medical kit

Table 5: The state attributes of the Hunter Bot.

No.	*State attributes*	*Type*	*Description*
Att_1	*SeeAnyEnemy*	Boolean	See enemy?
Att_2	*HasBetterWeapon*	Boolean	Has a better weapon?
Att_3	*HasAnyLoadedWeapon*	Boolean	Has weapon loaded?
Att_4	*IsShooting*	Boolean	Is shooting?
Att_5	*IsBeingDamaged*	Boolean	Is being shot?
Att_6	*LastEnemy*	Boolean	Has enemy target to pursue?
Att_7	*IsColliding*	Boolean	Collide with wall?
Att_8	*SeeAnyReachableItemAndWantIt*	Boolean	See any wanted item?
Att_9	*AgentHealth*	[0, 1]	Agent's health level
Att_{10}	*CanRunAlongMedKit*	Boolean	Medical kit can be obtained?

Table 6: The hard-coded rules of the Hunter Bot in UT2004.

No.	*IF (Condition)*	*THEN (Behavior)*
1	*SeeAnyEnemy* AND *HasBetterWeapon*	*ChangeToBetterWeapon*
2	*SeeAnyEnemy* AND *HasAnyLoadedWeapon*	*Engage*
3	*IsShooting*	*StopShooting*
4	*IsBeingDamaged*	*RespondToHit*
5	*LastEnemy* AND *HasAnyLoadedWeapon*	*Pursue*
6	*IsColliding*	*Walk*
7	*SeeAnyReachableItemAndWantIt*	*GrabItem*
8	*AgentHealthisLow* AND *CanRunAlongMedKit*	*GetMedicalKit*

haviour (action). The collected data are then used to train the TD-FALCON network using the supervised learning paradigm. As such, the agent created by imitative learning is expected to exhibit similar behavior patterns and produce comparable level of fighting competency as the Hunter Bot.

7.2.2. Dual-Stage Learning

With Dual Stage Learning, the Bot created by imitative learning is capable of continuously fighting with Hunter Bot and adapt to the environment using reinforcement learning. The DSL Bot uses the same behaviors and state attributes as shown in Table 4 and Table 5 respectively. At the beginning, the Bot performs with only prior knowledge learned by imitative learning. Later on, incorporating with Q-learning method, the Bot is expected to validate and refine the existing knowledge as well as obtain new knowledge. In Unreal Tournament, the DSL Bot is expected to show an enhanced level of fighting competency over its opponent.

7.2.3. Mixed Model Learning

When playing against the Hunter Bot, the Bot created by MML is designed to adapt to the dynamic environment with reinforcement learning as well as to imitate its opponent's behavior patterns in real time. With the same behaviors and state attributes as shown in Table 4 and Table 5 respectively, the MML Bot is also expected to show a higher level of fighting competency than its opponent.

7.3. Learning models in comparison

We conducted a series of experiments in the Unreal Tournament game environment to examine the performance of the non-player characters (Bots) created by various learning methods, namely imitative learning, reinforcement learning, dual stage learning, mixed model learning, and standard Q-learning. The learning configuration of each learning model is presented in details below. Under the Deathmatch scenario, each of the learning Bots enters into a series of one-on-one battles with the Hunter Bot. When a Bot kills its opponent, one point is awarded. The battle repeats until any one of the Bots reaches a maximum score of 25.

7.3.1. FALCON Bot by Imitative learning

FALCON Bot created using imitative learning (IL) is called FALCON-IL Bot. The FALCON-IL Bot is trained using 8000 training samples data recorded from the Hunter Bot. We then examine if FALCON-IL Bot could learn the behavior patterns and play against the sample Hunter Bot.

FALCON-IL Bot adopts the parameter setting as follows: choice parameters $\alpha^{c1} = \alpha^{c2} = \alpha^{c3} = 0.1$; learning rate parameters $\beta^{c1} = \beta^{c2} = \beta^{c3} = 1$ for fast learning; and contribution parameters $\gamma^{c1} = 1$ and $\gamma^{c2} = \gamma^{c3} = 0$. As in supervised learning, TD-FALCON selects a category node based on the input activities in the input state field. The vigilance parameters are set to $\rho^{c1} = \rho^{c2} = 1$ and $\rho^{c3} = 0$ for a strict match criterion in the state and action fields and a zero-match requirement in the reward field.

7.3.2. FALCON Bot by Online Imitative learning

FALCON Bot created using online imitative learning (OIL) is called FALCON-OIL Bot. In the experiments conducted in Unreal Tournament, FALCON-OIL Bot will be trained in real time by sensing the samples data from its opponent, the Hunter Bot. FALCON-OIL Bot adopts the same setting of choice parameters, learning rate parameters, contribution parameters, vigilance parameters, and learning rate α and discount factor γ for the Temporal Difference rule as that of FALCON-IL Bot. (For details of FALCON-IL Bot and FALCON-OIL Bot, please refer to our previous work (Feng & Tan, 2010).)

7.3.3. FALCON Bot by reinforcement learning

FALCON Bot using reinforcement learning only is called FALCON-RL Bot. To examine whether reinforcement learning is effective to enhance the behavior learning, a series of experiments are conducted in Unreal Tournament to examine how the FALCON-IL Bot performs when it plays against the Hunter Bot.

Under the pure reinforcement learning mode, we adopt the parameter setting as follows: choice parameters $\alpha^{c1} = \alpha^{c2} = \alpha^{c3} = 0.1$; learning rate parameters $\beta^{c1} = \beta^{c2} = 0.5$ and $\beta^{c3} = 0.3$ to achieve a moderate learning speed; and contribution parameters $\gamma^{c1} = \gamma^{c1} = 0.3$ and $\gamma^{c3} = 0.4$. During reinforcement learning, a slower learning rate could produce a smaller set of better quality category nodes in FALCON network and lead to better predictive performance, although a lower learning rate may slow down the learning process. The vigilance parameter ρ^{c1} is set to 0.8 for a better match criterion, ρ^{c2} is set to 0 and ρ^{c3} is set to 0.3 for a marginal level of match criterion on the reward space so as to encourage the generation of category nodes. In learning value function with Temporal Difference rule, the learning rate α is fixed at 0.7 and the discount factor γ is set to 0.9.

7.3.4. *FALCON Bot by dual stage learning*

FALCON Bot learned using the DSL strategy is called FALCON-DSL Bot. In the imitative learning stage, FALCON-DSL Bot adopts the same parameter setting as that of the FALCON-IL Bot. For the reinforcement learning stage, FALCON-DSL Bot adopts the same setting of choice parameters, learning rate parameters, and contribution parameters as that of FALCON-RL Bot.

The vigilance parameters ρ^{c1} is set to 0.9 which is slightly higher than that of FALCON-RL Bot for a better match criterion, ρ^{c2} is set to 0 and ρ^{c3} to 0.3 for a marginal level of match criterion. For the Temporal Difference rule, FALCON-DSL Bot also adopts the same learning rate α and discount factor γ as that of FALCON-RL Bot.

For knowledge transferred from imitative learning, each of the embedded cognitive codes C_j is initialized with a reward value q_j for $j = 1, \ldots, N$. At the beginning of learning, we assume the embedded codes have a standard reward value of 0.75, to assume them reasonably good rules.

7.3.5. *FALCON Bot by Mixed Model Learning*

FALCON Bot learned with the MML strategy is called FALCON-MML Bot. When imitative learning is activated, the same parameter setting as that of the FALCON-IL Bot is applied.

When the reinforcement learning mode is activated, FALCON-MML Bot adopts the same setting of choice parameters, learning rate parameters, and contribution parameters as that of FALCON-RL Bot.

For the vigilance parameters, ρ^{c1} is set to 0.9 which is slightly higher than that of FALCON-RL Bot for a better match criterion, ρ^{c2} is set to 0 and ρ^{c3} to 0.3 for a marginal level of match criterion. For the Temporal Difference rule, FALCON-DSL Bot also adopts the same learning rate α and discount factor γ as those of FALCON-RL Bot.

7.3.6. *Bot by Standard Q-learning*

For the purpose of comparison, a Bot created by standard Q-learning (called QL Bot) is also realized in Unreal Tournament. QL Bot works by learning the value function for each chosen action in a given state. We conduct a series of experiments to examine how QL Bot performs when it plays against the Hunter Bot. For learning value function using Temporal Difference rule, the learning rate α was fixed at 0.7 and the discount factor γ was set to 0.9.

7.4. *Results*

Figure 8 summarizes the performance of the various Bots in terms of score differences when they play against the Hunter Bot. The game score differences are calculated by averaging across ten sets of 20 continuous runs. As shown in Figure 8, FALCON-IL Bot is able to achieve a similar level of fighting competency as the Hunter Bot. This shows that FALCON-IL Bot is able to learn the behavior patterns from the sample Bot rather well.

On the other hand, FALCON-RL Bot begins with a low level of competency but it is able to acquire the right behavior strategy gradually and defeats the Hunter Bot consistently. In contrast, FALCON-DSL begins with a comparable level of fighting competency with its opponent (Hunter Bot) and continuously improves its performance over runs. The game score difference obtained by FALCON-DSL Bot converges quickly to a decent level above that of FALCON-RL.

The experiment results thus demonstrate that the DSL strategy is effective in enhancing the learning ability of the Bot in terms of faster learning speed and convergence. As a baseline comparison, the performance of QL Bot is visually lower than those of the FALCON Bots.

In the second set of the experiments, we compare the performance of several other Bots, including FALCON-OIL Bot, FALCON-MML Bot, and FALCON-RL Bot. Compared with the Bots evaluated in the first set of experiments, these Bots make use of online real-time learning, doing away with the need to do offline imitative learning before hand. Specifically, imitative learning is done completely in an online fashion for FALCON-OIL Bot, and interleaved with reinforcement learning for FALCON-MML Bot. Figure 9 summarizes the performance of the three Bots in terms of score difference playing against Hunter. As before, the game score differences are calculated by averaging across ten sets of 20 continuous runs.

From Figure 9, it can be seen that demonstrates the FALCON-OIL Bot can learn the behavior patterns very quickly, and have a similar fighting competency as that of Hunter Bot. Comparing with Figure 8, we see that the FALCON-OIL Bot's performance is as good as the FALCON-IL Bot. This result also shows that the online imitative learning is capable of learning behavior patterns fast and accurately.

More importantly, Figure 9 also shows that FALCON-MML Bot produces an significantly higher level of fighting competency than its opponent. As FALCON-MML Bot provides fast learning speed and quick convergence in real time, this result also shows that MML is a powerful strategy to integrate online imitative learning and reinforcement learning. Moreover, the performance of FALCON-MML Bot is also comparably better than that of FALCON-RL Bot, showing an improvement in learning speed and convergence. The details of performance of the six learning strategies are summarized in Table 7.

For evaluating learning efficiency, Table 8 illustrates the number of codes created in the cognitive fields by the six learning methods respectively. We can observe that the number of codes by pure reinforcement learning, standard Q-learning, and MML are about the same and shows a similar rising trend. DSL also has a similar rising trend whereas starting with a relatively small number of codes because the prior knowledge learned by imitative learning could produce compression and generalization. In comparison, the number of codes generated by OIL strategy is much smaller and is almost the same as that of OL strategy. It is notable that the number of codes in DSL remains in a reasonable region although there is prior knowledge encoded beforehand.

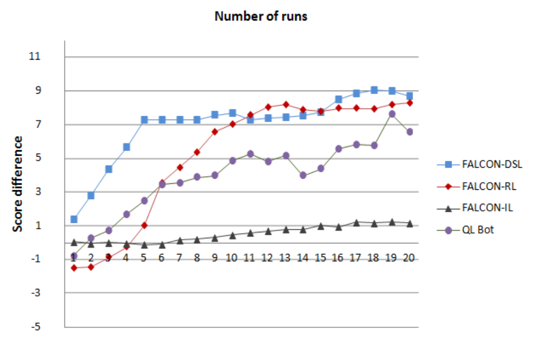

Figure 8: The score difference between the learning Bots and Hunter Bot.

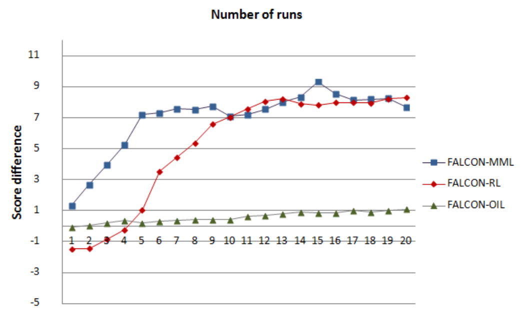

Figure 9: The score difference between the learning Bots and Hunter Bots.

8. Conclusion

This paper has presented a computational model unifying two popular learning paradigms, namely imitative learning and reinforcement learning, based on a class of self-organizing neural networks called Fusion Architecture for Learning and COgnition (FALCON). Addressing the knowledge integration issue, the computational model is capable of unifying states and actions spaces and transferring knowledge seamlessly across different learning paradigms. This enables the learning agent to perform continuous knowledge exploitation, while it enhances the reinforcement learning with complementary knowledge. Specifically, two hybrid learning strategies, known as Dual-Stage Learning (DSL) and the Mixed Model Learning (MML), are proposed to realize the integration of the two different learning paradigms within one designed framework. DSL and MML have been used to create non-player characters (NPCs) in a first person shooting game named Unreal Tournament. A series of experiments shows that both DSL and MML are effective in enhancing the learning ability of NPCs in terms of faster learning speed and accelerating convergence. Most notably, the NPCs built by DSL and MML produce better combat performance comparing with NPCs using the pure reinforcement learning and the pure imitative learning methods. The proposed hybrid learning strategies thus provide an efficient method to building intelligent NPC agents in games and pave the way towards building autonomous expert and intelligent systems for other applications.

Although integration of different learning paradigms appears to be straightforward in our work, note that our integration strategies rely very much on the self-organizing neural network model employed, namely FALCON. As such our work does not provide a general solution for integration of different learning paradigms using any learning algorithm or model.

In terms of algorithm design and experimentation, our main performance metric so far is just the combat performance of the NPC. Moreover, for solving the exploitation-exploration dilemma, we only consider simple direct rewards, such as those given when damaging opponents and collecting weapon. Other more sophisticated aspects of NPCs in first-person shooting scenarios, such as goals, memories, and humanity factors, so far haven't been explored.

Moving forward, for the purpose of creating intelligent, believable and attractive NPC agents, we still have to enhance the capabilities of the agents by integrating other high level cognitive factors and human factors. For example, we shall investigate the use of a goal

Table 7: The score difference between our Bots and the enemy Bot during learning

Learning Bots	score difference after 5 runs	score difference after 10 runs	score difference after 20 runs
FALCON-IL Bot	0.10 ± 3.68	0.49 ± 4.24	1.19 ± 3.86
FALCON-OIL Bot	0.22 ± 4.18	0.45 ± 5.12	1.01 ± 2.30
FALCON-RL Bot	1.14 ± 3.52	7.04 ± 3.11	8.30 ± 4.16
QL Bot	2.50 ± 5.34	2.90 ± 3.34	6.60 ± 5.41
FALCON-DSL Bot	7.28 ± 1.96	7.72 ± 4.23	8.72 ± 4.66
FALCON-MML Bot	7.25 ± 3.64	7.10 ± 4.36	7.68 ± 4.62

Table 8: The number of cognitive codes created in learning

Learning Bots	No. of codes after 5 runs	No. of codes after 10 runs	No. of codes after 20 runs
FALCON-IL Bot	36.00 ± 2.12	40.00 ± 2.82	42.00 ± 1.41
FALCON-OIL Bot	35.00 ± 1.73	39.00 ± 2.51	43.00 ± 2.64
FALCON-RL Bot	72.00 ± 5.95	89.00 ± 4.60	111.00 ± 6.10
QL Bot	85.00 ± 5.77	93.00 ± 2.14	113.00 ± 4.34
FALCON-DSL Bot	50.00 ± 0.50	72.00 ± 2.49	112.00 ± 2.51
FALCON-MML Bot	70.00 ± 3.82	88.00 ± 5.19	103.00 ± 5.70

maintenance module, which may help to manage the exploitation-exploration dilemma and predict the outcome of actions. On the other hand, we shall extend our model to be capable of human-like behavior by incorporating personalities and motivations into agents.

Last but not least, it is important to augment the cognitive functions of the agents with affective capabilities, so that the NPCs may display different emotions based on the outcomes (success or failure) of its actions. In addition, it will be interesting to study how the emotion of NPCs may influence their learning experience and future actions. By integrating personal as well as affective attributes, the behaviors of NPC are expected to be more realistic and believable. These, we reckon, will greatly enhance the playability of the computer games.

Akbar, M. A., Praponco, W., Hariadi, M., Mardi, S. et al. (2015). Multi behavior npc coordination using fuzzy coordinator and gaussian distribution. In *Intelligent Technology and Its Applications (ISITIA), 2015 International Seminar on* (pp. 17–22). IEEE.

Bauckhage, C., Thurau, C., & Sagerer, G. (2003). Learning human-like opponent behavior for interactive computer games. In *Lecture notes in computer science* (pp. 148–155).

Busoniu, L., Schutter, B. D., Babuska, R., & Ernst, D. (2010). Using prior knowledge to accelerate online least-squares policy iteration. In *Automation Quality and Testing Robotics (AQTR), 2010 IEEE International Conference on* (pp. 1–6). Cluj-Napoca, Romania volume 1.

Carpenter, G. A., & Grossberg, S. (1987a). ART 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, *26*, 4919–4930.

Carpenter, G. A., & Grossberg, S. (1987b). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, *37*, 54–115.

Carpenter, G. A., Grossberg, S., & Rosen, D. B. (1991). Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, *4*, 759–771.

David, O. E., van den Herik, H. J., Koppel, M., & Netanyahu, N. S. (2014). Genetic algorithms for evolving computer chess programs. *Evolutionary Computation, IEEE Transactions on*, *18*, 779–789.

Dixon, K., Malak, R., & Khosla, P. (2000). Incorporating prior knowledge and previously learned information into reinforcement learning agents. *Technical report, Institute for Complex Engineered Systems, Carnegie Mellon University*, .

Doshi-Velez, F., Pfau, D., Wood, F., & Roy, N. (2015). Bayesian nonparametric methods for partially-observable reinforcement learning. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *37*, 394–407.

Feng, S., & Tan, A.-H. (2010). Self-organizing neural networks for behavior modeling in games. In *proceedings, International Joint Conference on Neural Networks* (pp. 3649–3656). Barcelona, Spain.

Framling, K. (2007). Guiding exploration by pre-existing knowledge without modifying reward. *Neural Networks*, *20*, 736–747.

Ghavamzadeh, M., Mannor, S., Pineau, J., & Tamar, A. (2015). Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, *8*, 359–483.

Glavin, F. G., & Madden, M. G. (2015). Learning to shoot in first person shooter games by stabilizing actions and clustering rewards for reinforcement learning. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on* (pp. 344–351). IEEE.

Ji, L., & Ma, J. (2014). Behavior tree for complex computer game ai behavior. *Simulation and Modelling Methodologies, Technologies and Applications*, *60*, 201.

Jonschkowski, R., & Brock, O. (2014). State representation learning in robotics: Using prior knowledge about physical interaction. In *Robotics: Science and Systems (RSS)*.

Kengo, K., Takahiro, K., & Hiroyuki, N. (2005). Reinforcement learning agents with primary knowledge designed by analytic hierarchy process. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing* (pp. 14–21).

Moreno, D. L., Regueiro, C. V., Iglesias, R., & Barro, S. (2004). Using prior knowledge to improve reinforcement learning in mobile robotics. In *Proc. Towards Autonomous Robotics Systems. Univ. of Essex, UK*.

Pogamut (2016). Retrived from http://artemis.ms.mff.cuni.ca/pogamut/, .

Ponce, H., & Padilla, R. (2014). A hierarchical reinforcement learning based artificial intelligence for non-player characters in video games. In *Nature-Inspired Computation and Machine Learning* (pp. 172–183). Springer.

Shapiro, D., Langley, P., & Shachter, R. (2001). Using background knowledge to speed reinforcement learning in physical agents. In *the 5th International Conference on Autonomous Agents.* (pp. 254–261). Montreal, Quebec, Canada.

Song, M., Gu, G., & Zhang, R. (2004). Behavior control of multi-robot using the prior- knowledge based reinforcement learning. In *the 5m World Congress on Intelligent Control and Automation* (pp. 5027–5030). Hangzhou, China volume 6.

Stanley, K. O., Bryant, B. D., & Miikkulainen, R. (2005). Real-time neuroevolution in the nero video game. *Evolutionary Computation, IEEE Transactions on*, *9*, 653–668.

Sutton, R., & Barto, A. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Tan, A.-H. (1997). Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing. *IEEE Transaction on Neural Networks*, *8*, 237–250.

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

Tan, A.-H. (2007). Direct code access in self-organizing neural architectures for reinforcement learning. In *Proceedings, International Joint Conference on Artificial Intelligence (IJCAI07), Hyderabad, India* (pp. 1071–1076).

Tan, A.-H., Carpenter, G. A., & Grossberg, S. (2007). Intelligence through interaction: Towards a unified theory for learning. *Lecture Notes in Computer Science*, *4491*, 1098–1107.

Unemi, T. (2000). Scaling up reinforcement learning with human knowledge as an intrinsic behavior. In *Scaling up reinforcement learning with human knowledge as an intrinsic behavior* (pp. 511–518).

Wang, D., Subagdja, B., Tan, A.-H., & Ng, G. W. (2009). Creating human-like autonomous players in real-time first person shooter computer games. In *Proceedings of Twenty-First Annual Conference on Innovative Applications of Artificial Intelligence* (pp. 14–16). Pasadena,California.

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

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.

Xiao, D., & Tan, A.-H. (2007). Self-organizing neural architectures and cooperative learning in multi-agent environment. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, *37*, 1567–1580.

Zanetti, S., & Rhalibi, A. E. (2004). Machine learning techniques for FPS in Q3. In *2004 International Conference on Advances in Computer Entertainment Technology* (pp. 239–244).