Creating Human-like Autonomous Players in Real-time First Person Shooter Computer Games

Di Wang, Budhitama Subagdja
Intelligent Systems Centre
Nanyang Technological University
Nanyang Drive, Singapore 637553
{wangdi, budhitama}@ntu.edu.sg

Ah-Hwee Tan
School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, Singapore 639798
asahtan@ntu.edu.sg

Gee-Wah Ng
DSO National Laboratories
20 Science Park Drive
Singapore 118230
ngeewah@dso.org.sg

Abstract

This paper illustrates how we create a software agent by employing FALCON, a self-organizing neural network that performs reinforcement learning, to play a well-known first person shooter computer game known as Unreal Tournament 2004. Through interacting with the game environment and its opponents, our agent learns in real-time without any human intervention. Our agent bot participated in the 2K Bot Prize competition, similar to the *Turing test* for intelligent agents, wherein human judges were tasked to identify whether their opponents in the game were human players or virtual agents. To perform well in the competition, an agent must act like human and be able to adapt to some changes made to the game. Although our agent did not emerge top in terms of human-like, the overall performance of our agent was encouraging as it acquired the highest game score while staying convincing to be human-like in some judges' opinions.

Introduction

Games can be used to test the capabilities of various AI methodologies, and the focus of research is changing from traditional board games or card games to modern games. Real-time computer games such as first person shooter games may employ AI techniques to make games more challenging and enjoyable. To achieve this goal, Non-Player Characters (NPCs) in games should be dynamically evolving according to different human players and different game scenarios. Moreover, NPCs should exhibit human-like behaviors to improve the satisfactory level of the players.

The 2K Bot Prize competition was held at IEEE Symposium on Computational Intelligence and Games, 2008. It is like the *Turing test* that human judges need to identify whether their opponents in the game are human players or virtual agents. The platform of this competition is Unreal Tournament 2004 (UT2004), a well known real-time first person shooter computer game. To win this competition, an agent must act like human and be able to adapt to some changes made to the game. In particular, weapons hints were disabled and their effects were modified during the final competition. The agent must acquire all the necessary information directly from the game environment on its own.

In this paper, we describe how we make an agent to address the challenges. The agent passed the pre-final qualification trials and entered into the final of the 2K Bot Prize competition. Although we did not win the prize, the overall performance of our agent was encouraging as it acquired the highest total game score and was convincing to be human-like in the opinions of some judges.

Our agent is based on a class of self-organizing neural models called Fusion Architecture for Learning, COgnition, and Navigation (FALCON) (Tan 2004; Tan, Lu, and Xiao 2008), which is a generalization of Adaptive Resonance Theory (ART) (Carpenter, Grossberg, and Rosen 1991; Carpenter and Grossberg 2003; Tan, Carpenter, and Grossberg 2007) for real-time reinforcement learning. Specifically, we build two FALCON networks, one for behavior selection and the other for weapon selection, and integrate them into a software agent using Pogamut, a freeware for rapid development of embodied agents in UT2004. By utilizing FALCON networks, the agent bot learns knowledge in the form of cognitive nodes, each can be translated into a rule associating a pair of state and action to an estimated reward value. Results show that our agent is able to perform competitively through learning to play the game in real-time.

The rest of this paper is organized as follows. First, we review related works. Then we introduce FALCON and TD-FALCON as the models for reinforcement learning. Next, we describe UT2004 and the Pogamut development tool. Subsequently, we provide the details of the competition and the results. Finally, we conclude and discuss future works.

Related Works

Real-time computer game is one of the leading trends in applying intelligent methodologies. In this section, we shall review some related research works:

Spronck et al. (2006) use dynamic scripting to enable Non-Player Characters (NPCs) to be evolving according to different players. This means NPCs created in different time have different scripts (rules), according to both the player's tactics and level of experiences. By doing so, it could improve the satisfactory level of the human players as their opponents are dynamic. However, the NPCs are fixed after creation. They do not evolve in real-time.

Cook et al. (2007) use 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 (GA) to automatically tune the parameters used in game (Louis and Miles 2005; Stanley, Bryant, and Miikkulainen 2005; Miles and Louis 2006). These parameters are like weapon preferences, following tasks, and level of aggressiveness. Although the performance is improving through generations, GA does not guarantee the final solution to be global optimal. Even for a satisfactory sub-optimal solution, it often takes an unnecessarily long time for a real-time computer game.

In the above mentioned papers, both commercialized computer games and own developed platforms were studied. There are also research works that have the same domain of application as ours, the Unreal Tournament game.

Hy et al. (2004) define a way to specify various behaviors of the agent. These behaviors could be tuned by adjusting the pre-defined probability distributions. This is a straightforward way to use only probability equations to determine the next action rather than write and maintain scripts. However, the parameters defined are all based on personal experience which includes human bias that different people have different judgment for different kinds of behaviors (aggressive, caution, and etc.).

Kim (2007) defines a finite state transition machine to switch behaviors of the agent based on context sensitive stimulations received from the game. However, this architecture is rigidly defined and the agent is hard-coded. The agent will always perform the same action under similar circumstances and will never evolve.

Unlike the above mentioned works, we adopt reinforcement learning. Therefore, our agent is able to learn in real-time as rewards received currently may affect its future decisions and the learning time is not long. The agent learns by getting rewards directly derived from the game, which does not involve any human supervision and intervention. The rewards derived are based on the outcomes when interacting with the opponents. In this way, the agent does not need a perfect model to learn from. The agent is able to adjust its behaviors when confronting different opponents, if provided with enough training. The reinforcement learning tool we employed is called FALCON, which will be introduced in the following section.

FALCON Dynamics

FALCON (Fusion Architecture for Learning, COgnition, and Navigation) was first proposed in (Tan 2004) as a kind of reinforcement learning mechanism. FALCON employs a 3-channel fusion ART (Tan, Carpenter, and Grossberg 2007) architecture (see Figure 1), comprising a cognitive field F_2^c and three input fields, namely a sensory field F_1^{c1} for representing current states, an action field F_1^{c2} for representing actions, and a reward field F_1^{c3} for representing reinforcement values. The generic network dynamics of FALCON, based on fuzzy ART operations (Carpenter, Grossberg, and Rosen 1991), is described as follows:

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.

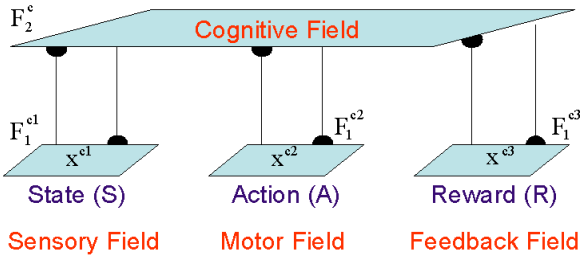

Figure 1: FALCON architecture.

Let $\mathbf{A} = (a_1, a_2, \ldots, a_m)$ denote the action vector, where $a_i \in [0, 1]$ indicates a possible action i. Let $\mathbf{R} = (r, \bar{r})$ denote the reward vector, where $r \in [0, 1]$ is the reward signal value and $\bar{r}$ (the complement of r) is given by $\bar{r} = 1 - r$. Complement coding serves to normalize the magnitude of the input vectors and has been found effective in ART systems in preventing the code proliferation problem. As all input values of FALCON are assumed to be bounded between 0 and 1, normalization is necessary if the original values are not in the range of $[0, 1]$.

Activity vectors: Let $\mathbf{x}^{ck}$ denote the F_1^{ck} activity vector for $k = 1, \ldots, 3$. 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 patterns in F_1^{ck} for $k = 1, \ldots, 3$. 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 $k = 1, \ldots, 3$; learning rate parameters $\beta^{ck} \in [0, 1]$ for $k = 1, \ldots, 3$; contribution parameters $\gamma^{ck} \in [0, 1]$ for $k = 1, \ldots, 3$ where $\sum_{k=1}^{3} \gamma^{ck} = 1$; and vigilance parameters $\rho^{ck} \in [0, 1]$ for $k = 1, \ldots, 3$.

Code activation: A bottom-up propagation process first takes place in which the activities (known as choice function values) of the cognitive 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^{cs} and F_1^{c3} respectively), for each F_2^c node j, the choice function T_j^c is computed as follows:

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

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

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

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 code J can be used for learning, a template matching process checks that 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^{ck} of the chosen code J meets its vigilance criterion:

$$m_J^{ck} = \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{x}^{ck}|} \geq \rho^{ck}. \quad (3)$$

The match function computes the similarity of the activity and weight vectors with respect to the norm of the activity vectors. Together, the choice and match functions work co-operatively 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_J^c 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_J^{c1}. The search process then selects another F_2^c 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 $1s$.

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

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

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. When an uncommitted node is selecting for learning, it becomes *committed* and a new uncommitted node is added to the F_2^c field. FALCON thus expands its network architecture dynamically in response to the input patterns.

TD-FALCON

Selecting which weapon to use is straightforward if the associations are learned by FALCON. However, behavior selection does not only depend on the current state. We need to perform anticipation and select the action which may possibly lead to the maximum reward. Therefore, while a simple form of reactive FALCON (Tan 2004) is applied for weapon selection, TD-FALCON is applied for behavior selection.

TD-FALCON (Tan, Lu, and Xiao 2008) incorporates Temporal Difference (TD) methods to estimate and learn value functions of action-state pairs $Q(s, a)$ that indicates the goodness for a learning system to take a certain action a in a given state s. Such value functions are then used in the action selection mechanism, also known as the *policy*, to select an action with the maximal payoff. The original TD-FALCON algorithm (Xiao and Tan 2007; Tan, Lu, and Xiao 2008) selects an action with the maximal Q-value in a state s by enumerating and evaluating each available action a by presenting the corresponding state and action vectors **S** and **A** to FALCON. The TD-FALCON used in this paper replaces the action enumeration step with a direct code access procedure (Tan 2007).

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 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 of performing the chosen action in the current state. The new Q value is then used as the teaching signal for TD-FALCON to learn the association of 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 equation are elaborated in the following subsections.

Action Selection Policy

Through a direct code access procedure, TD-FALCON searches for the cognitive node which matches with the current state and has the maximal reward value. For direct code access, the activity vectors $\mathbf{x}^{c1}$, $\mathbf{x}^{c2}$, and $\mathbf{x}^{c3}$ are initialized by $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = (1, \ldots, 1)$, and $\mathbf{x}^{c3} = (1, 0)$. TD-FALCON then performs code activation and code competition according to equations (1) and (2) to select a cognitive node.

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

$$\mathbf{x}^{c2(\text{new})} = \mathbf{x}^{c2(\text{old})} \wedge \mathbf{w}_J^{c2}. \quad (5)$$

An action a_I is then chosen, which has the highest activation value

$$x_I^{c2} = \max\{x_i^{c2(\text{new})} : \text{for all } F_1^{c2} \text{ node } i\}. \quad (6)$$

Learning Value Function

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

$$\Delta Q(s, a) = \alpha T D_{err} \quad (7)$$

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

TD-FALCON employs a Bounded Q-learning rule, wherein the temporal error term is computed by

$$\Delta Q(s, a) = \alpha T D_{err} \left(1 - Q\left(s, a\right)\right). \quad (8)$$

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'.

Unreal Tournament 2004

Unreal Tournament 2004 (UT2004) is a commercial game, which could be used as an environment for embodying virtual agents. One great thing about this game is its extensibility. Users can make modifications to the game, like the one used in the 2K Bot Prize competition. In this competition, the DeathMatch game type is used. Figure 2 shows a screen shot from the recorded video of the 2K Bot Prize competition (from the view of the judge).

Figure 2: A snapshot of Unreal Tournament.

Pogamut

Pogamut[1] is an Integrated Development Environment (IDE) to facilitate the development of agents in the UT2004 game. This freeware is a plug-in for the NetBeans Java development environment. Pogamut communicates to UT2004 through Gamebots 2004 (GB2004). GB2004 is a built-in server inside UT2004, which exports information from the game to the agent and vice versa. As GB2004 only exports and imports text messages, a parser is needed for translation purpose. Pogamut has a built-in parser module. It automatically converts text messages to Java objects and vice versa.

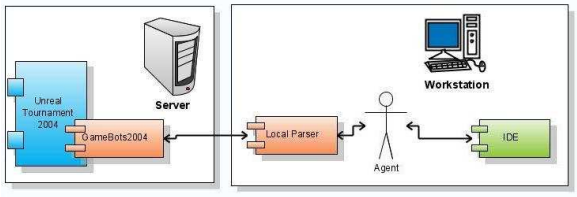

Figure 3: Pogamut architecture.

The architecture of Pogamut is shown in Figure 3[2]. It is very easy to implement an agent that plays UT2004 using Pogamut, as it has templates for various types of virtual agents. A bot developer only needs to define and override a few methods for a simple agent. Other than templates, there are also a number of sample agents available in Pogamut. They are good examples to follow when one starts to create his or her own agents, and they are also very useful for benchmarking purpose.

[1]The Pogamut homepage is available online. URL: http:/artemis.ms.mff.cuni.cz/pogamut/

[2]This figure is taken from Pogamut Homepage.

The 2K Bot Prize

About the Competition

The 2K Bot Prize competition[3] was held at IEEE Symposium on Computational Intelligence and Games, 2008. The aim of this competition is to find out whether a virtual agent could convince a panel of judges that it is actually a human player. The panel of judges included experts in various areas: AI experts, game designers, and game players.

To enter this competition, teams must send their agents to the competition organizer for a pre-final qualification trial before the conference date. In total, only five teams entered the final competition.

When playing UT2004, some weapon hints will be received by the agent. These hints include the type of each weapon (close range melee type or long distance sniping type), its effective range, and its maximum range. However, these hints were disabled during the final competition. Moreover, the weapon effects were changed.

On the day of final competition, in the morning, teams were allowed to install their agents on the local computers and train the agents. At this stage, teams were no longer allowed to modify their agents. Only the agents themselves were allowed to create or modify programs or files. This was when the modified game (new map, no hints, and weapon effects were changed) was released to the teams for the first time. Therefore, the agents were totally on their own to play in the modified game. That is why it is crucial to incorporate learning modules in our agent.

The actual competition was held in the afternoon. All the five teams of agents played five rounds, each round with one judge and one human confederate. Therefore, the five agents, the five judges, and the five human confederates all played with every opponent exactly once. At the beginning of each round, the agent and its human confederate joined the game first. The judge needed to join later as it was his or her job to identify which opponent was a virtual agent and which opponent was a human player. At the end of each round, the judge rated the level of humanness for both opponents. A rating of 0 is the lowest score as it indicates this opponent is not a very human-like agent. A rating of 4 is the highest score as it indicates this opponent is a human player in the judge's opinion.

Behavior Selection

In order to avoid having a hard-coded agent that will always perform the same action under similar circumstances, we enable our agent to be evolving in real-time by employing TD-FALCON network. The agent learns and utilizes the association of its current state, behavior selection, and reward.

[3]All information about this competition is available online. URL: http://www.botprize.org/

We define four behavior states for our agent. In each state, the agent will perform a set of predefined actions based on some hard-coded heuristics. The four states are listed as follows:

- Running around state, where the agent will explore randomly on the map.
- Collecting items state, where the agent will go and pick up collectible items.
- Escaping from battle state, where the agent will try to run away from the battle field.
- Engaging fire state, where the agent will try to kill its opponent and avoid being hit at the same time.

The state vector $\mathbf{S}$ comprises eight inputs. Therefore, the length of $\mathbf{S}$ is 16 in total with complement coding (Carpenter, Grossberg, and Rosen 1991). These inputs include the current health level of the agent (discretized), whether the agent is being damaged, whether the opponent is in sight, whether the agent has adequate ammo, and another four Boolean variables, indicating the current behavior state.

The action vector $\mathbf{A}$ consists of four attributes. Using a binary representation, only one of them is 1 at any one time, indicating which behavior state should the agent switch to based on $\mathbf{S}$ and $\mathbf{R}$. The reward vector $\mathbf{R}$ consists of only two variables, namely the reward and its complement.

To enhance learning, our agent also memorizes its previous state and action. When a reward is given (for example, after killing an opponent), it does not only learn the current $\mathbf{S}$ and $\mathbf{A}$ with $\mathbf{R}$, but also learn its previous $\mathbf{S}$ and $\mathbf{A}$ with a reduced version of $\mathbf{R}$ (please refer to temporal difference learning for details).

Weapon Selection

As discussed earlier, a reactive FALCON network is used for weapon selection. The state vector $\mathbf{S}$ consists of two attributes, namely the distance between the agent and the opponent (discretized) and its complement. The action vector $\mathbf{A}$ consists of ten attributes, the same number as the total number of available weapons. Using a binary representation, only one of the attributes is 1 at any one time, indicating the weapon should the agent use based on the state and reward vectors $\mathbf{S}$ and $\mathbf{R}$. The reward vector $\mathbf{R}$ consists of only two attributes, namely the reward and its complement.

Human Behaviors

Although it is crucial for the agent to be able to learn in real time, the competition judging criteria is still on the humanness of the agent. Other than behavior selection module and weapon selection module, we also defined series of actions when creating the agent. These human-like behaviors were incorporated in different behavior states and some of them are listed as follows:

- If the agent is hit by its opponent and the opponent is not in sight, the agent will wait for a randomly short time and then turn around in a random direction with a random degree (between $\pm 90^o$ to $\pm 180^o$) to find out the location of its opponent.
- During battle, the agent will try to kill its opponent while trying to avoid being damaged.
- Some obstacles can be jumped over (like log on the ground), while some cannot (like wall). The agent senses the environment in its heading direction and decides which type of obstacle it is dealing with.

Implementation Issues

"One vs one" in a DeathMatch game is a simple scenario in which the agent only needs to focus on killing the only opponent and surviving from the combat. A game consisting of more players will be more complicated. Besides identifying each opponent, the agent faces other challenges. For example, which opponent should the agent attack first, when is the appropriate time to engage fire, and how to find a position that receives the least assault. Our agent does not take considerations of these strategies so far.

We also need a more accurate reward system. Rather than giving 0.5 for a successful hit and 1 for a successful kill in the current weapon reward system, we should provide reward values in proportion to the exact damage done to the opponent. However, this feature is limited by the development tool Pogamut, since it currently only provides information on whether the opponent is hit, but not on the exact amount of the damage.

Competition Results

Table 1 shows the final competition results. Our agent (ISC[4]) came in the third place in terms of the humanness rating, while we tied in the first place in terms of the number of judges convinced[5]. The performance of our agent was satisfactory and encouraging.

Agent ID	Mean Rating	Judges Convinced
AMIS	2.4	2
ICE	2.2	1
ISC	2.0	2
Utexas	0.8	0
Underdog	0.4	0

Table 1: Humanness ratings.

The top three teams were very close in terms of humanness rating. On average, there was only a difference of 0.4 point (i.e., 2 points out of the total 20 points)[6] between the winner and our team, and a difference of 0.2 point (1 point out of the total 20) between the runner-ups.

Furthermore, in terms of the number of judges convinced, our team tied in the first place with another team. Both convinced two judges that the agents were actually human.

Table 2 shows the game score (computed as the number of kills minus the number of deaths) in the final competition.

[4]Named after Intelligent Systems Centre.

[5]All the results of this competition as well as video recordings are available on its official website.

[6]There are five judges and each judge could give a maximum score of 4, so the maximum total score is 20. 2 points divided by five judges gives a difference of 0.4 point on average.

Round	AMIS	ICE	**ISC**	UTexas	Underdog
1	1	3	6	2	6
2	0	2	17	3	3
3	1	2	5	4	4
4	0	2	2	4	8
5	1	3	4	1	7
total	3	12	**34**	14	28

Table 2: Game scores.

The total game score of our agent is 34, while all the other team only scored 14.25 on average. Although we did not win the competition in terms of the human-like rating, our bot has obtained the highest total game score.

By utilizing FALCON networks, our agent bot acquires knowledge which can be interpreted as rules during run time. Two sample rules learned during the final competition for behavior and weapon selection are illustrated in Table 3 and 4. These rules indicate that the agent has acquired sensible knowledge when playing the game.

IF	health is 0.4, and being damaged, and opponent is in sight, and has adequate ammo, and currently in collecting item state;
THEN	go into engaging fire state;
WITH	reward of 0.718.

Table 3: Sample rule for behavior selection.

IF	distance between agent and opponent is 0.1;
THEN	use flak cannon;
WITH	reward of 0.826.

Table 4: Sample rule for weapon selection.

Conclusion

In this paper, we have described how we have built an intelligent agent that is capable of playing a well known first person shooter game in real-time. FALCON networks which perform reinforcement learning were employed to overcome certain limitations found in related works. The performance of our agent is satisfactory as it does not only act like human, but also play the game effectively by acquiring and utilizing rules at run time.

In the future, we will extend our work in this game application domain and carry out more empirical experiments. In addition to addressing the implementation issues mentioned in the previous section, we are now incorporating a plan execution module that the agent will identify its goals as well as sub-goals and pursue them in the order of priorities. We are also conducting experiments and analyzing the results to find out how to make the agent perform better.

References

Carpenter, G. A., and Grossberg, S. 2003. Adaptive Resonance Theory. In *The Handbook of Brain Theory and neural Networks*. MIT Press. 87–90.

Carpenter, G. A.; Grossberg, S.; and Rosen, D. B. 1991. Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System. *Neural Networks* 4:759–771.

Cole, N.; Louis, S. J.; and Miles, C. 2004. Using a Genetic Algorithm to Tune First-Person Shooter Bots. In *Proceedings of International Congress on Evolutionary Computation*, 139–145.

Cook, D. J.; Holder, L. B.; and Youngblood, G. M. 2007. Graph-Based Analysis of Human Transfer Learning Using a Game Testbed. *IEEE Transaction on Knowledge and Data Engineering* 19:1465–1478.

Hy, R. L.; Arrigoni, A.; Bessiere, P.; and Lebetel, O. 2004. Teaching Bayesian Behaviours to Video Game Characters. *Robotics and Autonomous Systems* 47:177–185.

Kim, I. C. 2007. UTBot: A Virtual Agent Platform for Teaching Agent System Design. *Journal of Multimedia* 2:48–53.

Louis, S. J., and Miles, C. 2005. Playing to Learn: Case-Injected Genetic Algorithms for Learning to Play Computer Games. *IEEE Transaction on Evolutionary Computation* 9:669–681.

Miles, C., and Louis, S. J. 2006. Towards the Co-Evolution of Influence Map Tree Based Strategy Game Players. In *Proceedings of IEEE Symposium on Computational Intelligence and Games*, 75–82.

Spronck, P.; Ponsen, M.; Sprinkhuizen-Kuyper, I.; and Postma, E. 2006. Adaptive Game AI with Dynamic Scripting. *Machine Learning* 63:217–248.

Stanley, K. O.; Bryant, B. D.; and Miikkulainen, R. 2005. Real-Time Neuroevolution in the NERO Video Game. *IEEE Transaction on Evolutionary Computation* 9:653–668.

Tan, A.-H.; Carpenter, G. A.; and Grossberg, S. 2007. Intelligence through Interaction: Towards a Unified Theory for Learning. In *Proceedings of International Symposium on Neural Networks*, 1094–1103.

Tan, A.-H.; Lu, N.; and Xiao, D. 2008. Integrating Temporal Difference Methods and Self-Organizing Neural Networks for Reinforcement Learning with Delayed Evaluative Feedback. *IEEE Transactions on Neural Networks* 9(2):230–244.

Tan, A.-H. 2004. FALCON: A Fusion Architecture for Learning, COgnition, and Navigation. In *Proceedings of International Joint Conference on Neual Networks*, 3297–3302.

Tan, A.-H. 2007. Direct Code Access in Self-Organizing Neural Networks for Reinforcement Learning. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1071–1076.

Xiao, D., and 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(6):1567–1580.