Nanyang Technological University

Computational Techniques for Modeling Non-Player Characters in Games

Feng Shu
School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree
of Doctor of Philosophy

2014

Abstract

Modeling of non-player characters (NPCs) is an important research area in the development of computer games. Autonomous NPCs by emulating the behavior of human beings, with realistic performance and believable affective variations, in simulated environment, make the games more challenging and enjoyable.

Modeling of NPCs is essentially the problem of creating autonomous agents, which are expected to function and adapt by themselves in a complex environment. The motivation behind this research is thus to create "realistic" and "believable" NPCs with the abilities of autonomy, interactivity, situatedness, learning, and adaptation. Three key problems are considered in this research: (1) how behavior models of NPCs may be learned by mimicking behavior patterns of other players? (2) how behavior models of NPCs may be adapted through interaction and feedback in a dynamic environment? (3) how emotion of NPCs may be modeled and integrated with the behavior system and to create variations of NPCs?

For learning behavior models, this research investigates two classes of self-organizing neural networks. Firstly, the self-generating neural network (SGNN) is investigated to learn behavior rules from specific sample bots in a supervised manner. Further optimization of SGNN is also proposed via a pruning method which improves its performance. Our empirical experiments based on a first person shooting game environment called the Unreal Tournament show that SGNN is able to learn behaviors effectively from their prototype.

Secondly, another class of self-organizing neural networks, known as Fusion Architecture for Learning, COgnition, and Navigation (FALCON), is adapted for imitative learning to learn behavior patterns in the Unreal Tournament game. Benchmark

experiments are conducted to compare FALCON with SGNN in various aspects. The results show that, compared with SGNN, FALCON is able to achieve a higher level of performance with a much more compact network structure and a much shorter learning time.

Moving beyond supervised learning, this research aims to create more versatile NPCs which are able to further learn and adapt during game play in real time. As FALCON is designed to support a myriad of learning paradigms, it is our natural choice for modeling autonomous NPCs in games. Specifically, two hybrid learning strategies, namely the Dual-Stage Learning (DSL) strategy, and the Mixed Model Learning (MML) strategy, are presented to realize the integration of the supervised learning and reinforcement learning in one unified framework. DSL and MML have been applied to creating autonomous non-player characters (NPCs) in the Unreal Tournament game environment. Our experiments show that the NPCs learned with DSL and MML produce a higher level of performance compared with the traditional reinforcement learning and imitative learning.

The last but not the least, human factors such as emotion should be considered in NPC modeling to order to develop a "realistic" and "believable" agent. Based on the appraisal theory and the theory of cognitive regulated emotion, this research proposes a bio-inspired computational model called Cognitive Regulated Affective Architecture (CRAA) to emulate the structure and computations among neurons and regions of brain for emotion process. CRAA consists of a cognitive network, an appraisal network, and an affective network, which are built and work based on the Adaptive Resonance Theory (ART). Specifically, the cognitive network takes charge of decision making and behavior learning task, whereas the affective network encodes the associations from appraisal components to emotion. In addition, an appraisal network is positioned between the cognitive network and affective network to translate cognitive information to emotion appraisal. The model has also been evaluated in the Unreal Tournament game. Comparing with non-emotional NPC, emotional NPC obtains a higher level of user ratings in focus of game playability and interest.

Acknowledgments

First and foremost, I would express my deep gratitude to my supervisor, Associate Professor Tan Ah Hwee, for his trust, support, encouragement and guidance in my research. His patience and understanding coupled with his research and teaching experience has immensely contributed to my knowledge and research ability. I really appreciate him not only for his advice in my research but also instructions and help for my life. I would like to thank Mr. Wang Di, Miss. Wang Wenwen, and Dr. Budhitama Subagdja for their valuable help during my research. I also thank my lab-mates in Centre for Computational Intelligence Laboratory for their kindly help and assistant. Moreover, I want to thank all my friends. Their encouragement and comfort make me brave, especially when I face difficulties. Last but not least, I deeply appreciate my parents, my husband Hui for their selfless giving, support and love. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivations

Non-player Characters (NPCs) is widely used in video games, tabletop games and other virtual world scenarios. It refers to the characters controlled by computer systems instead of human beings. The NPCs make the games interesting and bring fun to players as they are not under the direct control of human players. Technically, NPC is in fact a kind of synthetic character or artificial virtual creature, which is expected to perform specific tasks, express the right behaviors in a particular situation based on its own internal states, and can interact with a person in real-time. NPC is also generally perceived as an autonomous agent which behaves based on its own motivations and desires [72] [18]. Recently, the modeling of NPCs has been intensively studied all over the world, along with the development of computer technologies. The NPCs technologies could be commonly applied in extensive areas such as commercial games, the virtual spaces with education or nursing purposes [63][165], and computer generated forces (CGF) in military command and control simulations [113] [155] [154]. In these areas, NPCs are required to realize artificial and humanoid performance such as behavior ability, affective expression ability, learning ability and adaptability.

This research focuses on the modeling of NPCs in computer games. Modeling of NPC is crucial for the success of commercial games as it improves the playability of games and satisfaction level of players. On the other hand, games is a good platform to test the capabilities of various artificial intelligence (AI) methodologies. For example, first person shooting games may employ AI techniques to make the games more challenging and enjoyable [158]. Players expect more sophisticated behaviors from NPCs in real time game environment. These expectations include not only intelligence, but also human-like attributes, such as personality and emotion. Recently, a stream of research on Human-level AI [78] has been heavily studied, as human-level has been used as a benchmark of estimating NPC's intelligence. For example, the Bot Prize competition held at IEEE Symposium on Computational Intelligence and Games, 2008 used Turing Test to judge whether a virtual agent (NPC) has human-like behaviors[1].

Modeling of NPC is essentially the problem of creating autonomous agents, which are expected to function and adapt by themselves in a complex and dynamic environment [147]. The key characteristics of autonomous agents may include perception, reasoning, learning, ability to reflect and so on. Figure 1.1 shows the interaction between an agent and its environment. Firstly, the agent perceives sensory input from its environment as the current state. Then, depending on the current state and its knowledge and drive, the agent decides and performs the most appropriate action. Finally, the agent may receive the feedback of its action from the environment, as the action affects the environment.

When a NPC is situated in some dynamic environment, how could it know which action to take in this situation in order to meet its objectives? The motivation behind this research is that as the complexity of creature's interactions with its

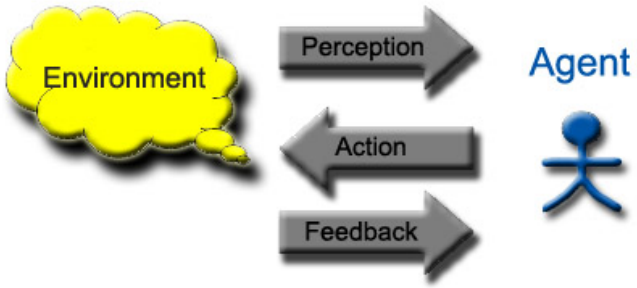

Figure 1.1: Interaction between autonomous agent and its environment

environment increases, there is a need to enrich the creatures' behavior mechanism with the abilities of learning, adaptation, and emotion variation.

1.2 Research Objectives

In the past decade, modeling of NPCs in computer games and virtual environment has made much progress in terms of 3D graphical interfaces and high-fidelity sophisticated modeling of physical aspects in diverse geographical environments. However, NPCs are still limited in the dimensions of affective modeling and behavior modeling in dynamic environments.

To create autonomous NPCs, we must first define our expectation of such artificial beings. The NPCs, as explained earlier, are autonomous agents which need to be provided with the following abilities: the autonomy, which is required for the NPCs to be self-awareness so as to initiate their actions and increase the dynamics of the virtual environment; the interactivity, which refers to the ability to respond or communicate [150] so as to improve the game's playability; the situatedness, which facilitates the NPCs to develop in their environment [5]; the learning ability, which includes the capabilities of imitating, as well as acquiring new knowledge to improve

its performance. Further, this research addresses the advanced requirements of NPCs as "believable" and "realistic" agents [127], in focus of how human-like [16] and how convincing the agent is from the view of the users [13].

Considering the above requirements, this dissertation research begins with the study of behavior modeling, which is crucial for creating the autonomous NPCs and is also the most popular topic in analyzing such artificial creatures. Especially in first person shooting games (FPS), different approaches to behavior modeling led to different levels of NPCs' performance. At the lowest level, designers script their behavior models with hard-coding to realize the primitive behaviors such as pick up items, performing gestures, using objects, and so on. The second level deals with the NPCs' movement problems, such as path finding, handling obstacles or platforms. The highest level is decision making, which works to select the right actions to achieve the NPC's tasks. For example, in a fighting scenario, the NPC's task is to beat the opponent, but what should the NPC do when he's wounded? Should he engage the enemy or look for a health package? Decision making is more complex than and built upon the previous two levels. In the above example, if the enemy's decision is to collect health package, the NPC must have the capabilities of path finding, obstacle handling (at the second level) and hard-coded animations, such as walking, jumping, pick up, etc (at the lowest level).

In order to create autonomous NPCs with the autonomy, interactivity, situatedness and learning abilities, this research focuses on the highest level of behavior modeling, so as to improve the NPC's decision making ability. While many techniques, such as machine learning, knowledge representation, and reasoning, have been proposed for modeling decision making [5], this research takes the approach of imitative learning for learning from other individuals or existing behavior patterns

and reinforcement learning for acquiring the action policy through interacting with the environment in a dynamic process.

Another focus of this dissertation research is emotion modeling, which, in a simulated world, is very crucial for making agents "realistic" and "believable" [127]. In fact, emotion has an important role in human cognition and serves as a substantial function in behavior decision. Consequently, modeling of emotion could enrich virtual agents in the abilities of expressing with lively facial expressions, presenting motivated responses to environment, and intensifying their interactions with human users. Also, interactive agents with an emotion model could form a better understanding of user's moods and preferences and can adapt itself to the user's needs [39]. Moreover, as emotion is indispensable in maintaining rational behaviors, a robust and accurate model of emotion-to-response relations is needed in modeling virtual humans.

1.3 Issues and Challenges

1.3.1 Learning Behavior Models

Learning defines the ability of obtaining knowledge automatically. There are many forms of learning, including unsupervised learning, supervised learning and reinforcement learning. Among the various learning paradigms, supervised learning is probably the most effective, due to its use of explicit teaching signals.

Learning from existing user patterns is a new research trend in the field of modeling NPC. It could be accomplished through instruction or by mimicking based on observing. Just like infants learn their initial actions by imitation, learning from user patterns is a form of "mimicking", which provides the basis for higher intelligence.

During the past years, learning by imitation has been a popular method of acquiring complex behaviors [152] [12]. This kind of behavior learning can be applied in very broad areas like recognizing human action sequences [88] [75], creating learning robots [173] [95] and creating humanoid virtual characters [124] [91]. Imitative learning is also useful in solving problems, such as dynamically generating the gaming intelligence according to user's play patterns [168].

For behavior learning, many supervised learning methods exist, including fuzzy and neural networks, Hidden Markov Model, and data mining methods. Noda *et al.* [102] applied Markov Model in implementing the robot's behavior and teaching the robot by Q-learning algorithm. However, their focus was not on recognizing complex behaviors by observing other soccer characters. Lee *et al.* [83] used data mining methods based on sequential database to find association rules of behavior patterns. Although this method can find interrelationship between sequential attributes or actions, it is not suitable for learning behavior rules which associates states consisting of parallel attributes. Self-organizing neural networks are a special class of neural networks that may learn without explicit teaching signals. Recent development in self-organizing neural networks has extended them for supervised learning tasks. This research will explore applying the self-organizing models in learning behavior models, with the view that the behavior patterns learned can provide the seeds or initial knowledge for autonomous agents.

1.3.2 Behavior Adaptation

Non-player characters in computer games are essentially autonomous agents, which are expected to be able to learn, reflect and adapt in real-time. In particular, adaption and learning abilities are very important in the mechanism of behavior

selection. If learning defines the ability of to obtain the knowledge of how to behave, adaption can be defined as the ability of refining behavior models according to the user's command or according to the changes in the environment.

Traditional methods of integrating knowledge and learning in neural networks are mainly using supervised learning. This kind of learning is superior in acquiring prior knowledge, but has limitations in adaptation when facing dynamic environment. Reinforcement learning [142] is an interaction based paradigm wherein an autonomous agent learns to adjust its behavior according to feedback from the environment. Classical solutions to the reinforcement learning problem generally involve linking a given state to a desired action (action policy), or associating a pair of state and action to a utility value (value function), using temporal difference methods, such as SARSA [122] and Q-learning [160]. The problem of the original formulation is that mappings must be learned for each and every possible pair of state and action. This causes a scalability issue for continuous and/or very large state and action spaces.

Neural networks and reinforcement learning have had an intertwining relationship. In particular, multi-layer perceptron (MLP) and gradient descent backpropagation (BP) are commonly used to learn an approximation of the value function or action policy [3] [141]. However, MLP and BP are not designed for online incremental learning as they typically require an iterative learning process.

Taking a different approach, self-organizing neural networks are typically used for the representation and generalization of continuous state and action space [135]. The state and action clusters are then used as entries in a Q-value table implemented separately. In view of the above limitations, this research shall aim to incorporate reinforcement learning with self-organizing neural models to realize real-time learning and adaptation in a dynamic environment.

1.3.3 Affective Modeling

Emotion is a complex phenomenon affecting human being's psychological and physiological responses while interacting with the environment. To make the NPC more human-like, modeling of emotion (such as anger, happy, fear, sad, etc) is indispensable, since it is an important aspect of the human mind.

A key challenge in emotion modeling is to understand and explain the mechanism of emotion. In fact, this topic has been studied over a long period. Early psychological theories of emotion, such as the James-Lang Theory, the Cannon-Bard Theory, and the Two-Factor Theory, focus on how emotion arises. Despite the differing views, it is generally agreed that emotion is the combinative outcome of numerous internal factors and external circumstances. Lately, several evolutionary theories argue that emotion is a developed mechanism belonging to advanced species as their solutions for adaptation problem, in order to survive under a natural selection environment [137] [37]. These theories focus on the physical displays of emotions and they believe that emotions express a person's appraisal of a person-environment relationship which may demonstrate a harm or benefit for the individual. On the other hand, cognitive theories of emotion argue for the role of cognitive activity in emotion generation, since the occurring and intensity of emotion are controlled through cognitive judgement and evaluation [108] [137] [126]. Meanwhile, neurobiological theories point to the phenomenon that the occurring of emotions relates to distributed neural activations in human brains [112]. Specifically, numerous experiments by functional magnetic resonance imaging (fMRI), electroencephalography (EEG), electromyography (EMG), and skin conductance (SC) demonstrate that the mechanism of human emotion is highly associated with cognition and processed by multiple regions in human brains [112].

Inspired by the existing theories of emotion, researchers have begun to develop computational models of emotions as part of complex, interactive agents. For example, interactive agents with a model of emotions can understand the user's moods and adapt itself to the user's requirements [39]. Non-player characters in game environment with a model of emotions could effectively enhance their believability by simulating and expressing more realistic emotional responses [13] [121].

In the recent decades, "appraisal theories" has become the leading theory for modeling emotion, which states that a person's emotion is his/her personal assessment of "person-environment relationship" based on events. Consequently, many computational emotion models such as EM [101], FLAME [38], FAtiMA [34], ALMA [48], PEACTIDM [90] and THESPIAN [131] are proposed based on various appraisal theories [108] [126] [137]. These models are proposed based on psychological grounding aiming at simulating real human emotion processes. However, most of these models rely on the measurement of appraisal parameters only. Based on the above consideration, this research aims to incorporate cognitive processing with affective process to realize an integrated cognitive-affective architecture for autonomous agents.

1.4 Approach and Methodology

Based on self-organizing neural networks, this dissertation research shall focus on behavior modeling, behavior adaptation, affective modeling, and the integration of the behavior system and affective system.

1.4.1 Behaviour Modeling Using Self-Organizing Neural Networks

Self-organizing neural networks are a special class of neural networks that learn without explicit teaching signals. Recent development in self-organizing neural networks has extended them for supervised learning tasks. Compared with gradient descent based neural networks, they offer fast and real-time learning as well as self-scaling architectures that grow in response to signals received from their environment. Although there have been extensive works on applying self-organizing neural networks to modeling NPCs such as traditional self-organizing map [12], and other self-organizing models [47] [159] [163], most of them make use of a fixed architecture. In other words, the structure and the number of nodes in the network have to be determined before training. In addition, SOM performs iterative weight tuning, which is not suitable for real time adaptation. The following introduces two classes of self-organizing neural models focused by this research.

Self-generating neural network (SGNN) was firstly developed by Wen *et al.* [161, 162] based on self-organizing maps (SOM) and implemented as a self-generating neural tree (SGNT) architecture. Self-generating neural network is appealing, as it does not require a designer to specify the structure of network and the class parameters. In addition, it has efficient learning ability and reasonably good adaptive ability. Because of these good attributes, SGNN is a suitable candidate for learning behaviour rules. Compared with traditional neural networks, the structure of SGNN does not require to be pre-determined according to the particular application in hand. A good performance of SGNN in behavior modeling is expected.

FALCON (Fusion Architecture for Learning, Cognition, and Navigation) was first proposed by Tan *et al.* [147] based on Adaptive Resonance Theory (ART)

[23], as a natural extension of self-organizing neural network architecture. In the past years, although various models of ART have been widely applied to pattern analysis and recognition tasks, there have been very few attempts to used ART-based networks for building autonomous systems, especially for behavior modeling. Comparing with SGNN, which may ignore low frequency training data, FALCON enables fast one-shot and incremental learning.

1.4.2 Behavior Adaptation

Reinforcement learning has been widely used in behavior learning tasks of NPCs for creating self-adapting agents. However, reinforcement learning typically requires a process of exploration. As such, some initial knowledge is very necessary for a learning NPC in order to avoid geek and weak performance at the beginning. Directly inserting prior knowledge is the commonly used method to solve the problem. On the other hand, supervised learning is efficient in learning rate but strictly relies on the training data. This work will focus on integrating the reinforcement learning with supervised learning in order to combine their merits for behavior learning and adaptation.

The neural model of FALCON (Fusion Architecture for Learning, Cognition, and Navigation) integrates reinforcement learning and multi-channel mappings of patterns in an online and incremental manner. FALCON network learns cognitive codes encoding multi-channel mappings of multimodal input patterns involving sensory input, actions, and rewards. By using competitive coding as the underlying adaptation principle, the network dynamic encompasses a myriad of learning paradigms, including unsupervised learning, supervised learning, as well as reinforcement learning. Therefore, FALCON is a natural choice for behavior learning and adaptation of NPCs.

1.4.3 Affective Modeling

It has been a commonly accepted view that cognition and emotion are two closely intertwined systems in human brains. Specifically, the prefrontal cortex (PFC) and anterior cingulate cortex (ACC) not only have a central role in cognitive control but also involve in assessing emotions. The core of affective regions, namely amygdala (AMYG), also has been confirmed to respond based on attention and is closely linked to the function of perception [112]. These findings suggest that the mechanism of emotion involves not only the "affective" brain regions but also the "cognitive" brain regions, and that cognitive information need to be integrated with affective information in order to regulate the emotion. This view has also been supported by functional magnetic resonance imaging (fMRI) examination [106], which reveals three aspects of the cognitive regulation process: strategy generation for cognitively coping with emotional events associated with working memory processes localized in the lateral PFC; interference between reappraisals and evaluations to generate an affective response associated with the dorsal anterior cingulate cortex; and re-evaluating between internal states and external stimuli associated with medial PFC. This view has inspired us to explore a model which integrates multiple brain functions and processes such as perception, mental state, learning, and decision making within a network topology, in order to simulate the cognition and emotion functions in human brains.

Following the conceptual framework, this research takes the following positions: (1) cognition and emotion are two separated but interacting systems; (2) emotion is generated through the interaction among multiple neural networks and regions in human brains; and (3) emotion responses are a consequence of the cognitive appraisal of both the internal mental states and the signals generated by a cognitive

decision making system connected to external environment. Consequently, our aim is to develop a biologically-plausible computational model that is able to emulate the structure as well as the processes of behavior learning, and the processes of emotion based on parallel and distributed computations in multiple interacting neural networks.

1.5 Summary of Contributions

As mentioned earlier, new and effective behavior modeling and affective modeling approaches need to be developed to create "realistic" and "believable" NPCs. At architecture level, this dissertation research proposes a biologically-inspired neural model called Cognitive Regulated Affective Architecture (CRAA) to realize an integrated system with interactive cognitive and affective modules. At the computational level, this research contributes in imitative behavior learning by adopting two classes of self-organizing neural models; improves the NPCs adaption and learning abilities in a dynamic environment by proposing the Dual-Stage Learning (DSL) and the Mixed Model Learning (MML) strategies; and realizes affective modeling based on emotion appraisal and integrated with the behavior system in the Cognitive Regulated Affective Architecture (CRAA).

- First of all, this research studies a specific class of self-organizing neural networks, namely Self-Generating Neural Networks (SGNN), for the behavior learning task. A pruning method is proposed to optimize the SGNN network in order to overcome the increased computational time, while maintaining the learning performance. For the application of behavior rule learning, a novel rule extraction method is further proposed to extract useful knowledge from

SGNN and translate them into symbolic rules. This research investigates how this class of self-organizing models can be used to build autonomous players by learning the behaviour patterns of sample bots in a first person shooting game, known as Unreal Tournament 2004 (UT2004). Benchmark experiments are conducted to examine SGNN method and the pruning method in the aspects of generalization capability, learning efficiency, and computational cost, under the same set of learning conditions. The experiment results show that the pruning method effectively reduces the number of neurons in SGNN, while maintaining the learning accuracy. Moveover, online testing of NPC is conducted in the game UT2004 with the Deathmatch scenario. The online testing results demonstrate that SGNN can be used to build autonomous players by learning the behaviour patterns of sample bots. Specifically, the SGNN based bot could achieve a respectable level of performance when fighting against the sample bot. The bots created based on pruned SGNN and SGNN rules also successfully learned similar behavior patterns from the sample bot, and achieve good performance.

- Secondly, this research investigates how the class of self-organizing model FALCON can be used to build autonomous players by learning the behaviour patterns of sample bots in a first person shooting game, known as Unreal Tournament 2004 (UT2004). Benchmark experiments are conducted to compare FALCON with SGNN (in Chapter 3) in various aspects, including generalization capability, learning efficiency, and computational cost, under the same set of learning conditions. Our benchmark experiments show that, comparing with SGNN, FALCON learns faster and produces a higher level of generalization capability with a much smaller set of nodes. Online testing of NPC in

the Deathmatch scenario also confirms that FALCON Bot produces a similar level of fighting competency to the Hunter Bot, which is not matched by bots based on SGNN. It is also demonstrated that as FALCON is designed to support a myriad of learning paradigms, including unsupervised learning, supervised learning and reinforcement learning, it is our natural choice for modeling autonomous NPCs in games.

- To improve the NPCs adaption and learning abilities, this research further develops strategies for complementing imitative learning (IL) with reinforcement learning (RL) for adaptive ability in real time. Specifically, the Dual-Stage Learning (DSL) and the Mixed Model Learning (MML) are presented to realize their integration of the two learning paradigms to one framework. The DSL strategy contains two learning stages. In the imitative learning stage, FALCON learns prior knowledge to built the initial behaviour model of an autonomous agent. Later in the reinforcement learning stage, the agent further adapts to the environment through Q-learning and by applying the prior knowledge for exploitation. The MML strategy combines the two learning paradigms into one process, in which IL and RL work simultaneously by activating their corresponding input fields and sharing a common knowledge field. The DSL and MML have been applied to creating autonomous non-player characters (NPCs) in the Unreal Tournament. Our experiments show that both DSL and MML are effective in enhancing learning ability in terms of increasing learning speed and accelerating the convergence. The NPCs build by DSL and MML also produce better performance comparing with the traditional RL and IL methods.

- The last but not the least, based on the appraisal theory, the Adaptive Resonance Theory (ART) [27], and the theory of cognitive regulated emotion [104], this research has proposed a biologically inspired computational model called Cognitive Regulated Affective Architecture (CRAA). Specifically, CRAA consists of a Cognitive Network, an Appraisal Network and an Affective Network, designed to simulate the functions and interactions of prefrontal cortex (PFC), anterior cingulate cortex (ACC) and amygdala (AMYG) respectively. As a natural extension of reinforcement learning, this architecture shows how appraisal signals produced by a cognitive system may be used in generating emotional responses by an affective system.

  The CRAA model has been integrated into autonomous Non-Player Characters (NPCs) for playing in the Unreal Tournament. Our experimental results show that CRAA demonstrates a reasonably good level of accuracy in emotion modeling based on human assessment. Furthermore, the questionnaire based experiments also show that NPCs with emotions are significantly more appealing comparing with emotionless NPCs in terms of cognitive absorption, social presence and enjoyment.

1.6 Thesis Organization

The organization of this thesis is as follows. Chapter 1 introduces the background and motivation of this research and outlines the main research issues. Chapter 2 provides a detailed review of the related works. Chapter 3 and chapter 4 describe two specific approaches to learn behavior rules according to recorded patterns. Specifically, chapter 3 proposes a learning method based on the self-generating neural

network (SGNN). Chapter 4 proposes another learning method based on Fusion Architecture for Learning and Cognition (FALCON) to learn behavior mechanism from behavior patterns. In both chapters, we investigate how these two models may learn behavior rules from specific sample bots in the Unreal Tournament game in a supervised manner. Chapter 5 proposes a universal learning architecture which allows the NPCs to autonomously learn and adapt during game play in various manners. Chapter 6 proposes a bio-inspired computational model called Cognitive Regulated Affective Architecture (CRAA) to emulate the structure and computations among neurons and regions of brain in emotion process, as well to develop a "realistic" and "believable" agent. Chapter 7 summarizes work completed and discusses future work.

Chapter 2

Literature Review

Refer to our research tasks, this chapter presents an interdisciplinary study across the fields of artificial intelligence, machine learning, cognitive science, and neuroscience relevant to the modeling of NPCs. The specific works related to the research tasks of behavior modeling and affective modeling are reviewed in the following sections.

2.1 Behavior Modeling

In this Section, we first describes the simplest approach to model the non-player characters through rule based methods in Section 2.1.1. Section 2.1.2 introduces the behavior modeling method based on finite state machine. Section 2.1.3 reviews the related works of using artificial neural networks to model behaviors. Section 2.1.4 introduces the method of applying fuzzy logic in modeling the behaviors of NPCs. Finally, Section 2.1.5 describes the research trend of utilizing confabulation theory in modeling NPCs. It should be noted that the reviews provided in this chapter do not include hybrid or integrated methods.

2.1.1 Rule-Based Method

For the behavior modeling of NPCs, rule-based systems is probably the most primary and widely used method [31]. Rule-based systems are constructed based on a set of If-Then rules which are of the form: Situation→Action [167] [38]. It is easy to transfer expert experience into rule-based knowledge base of the systems.

Commonly, a rule is of the form: Situation→Action. When the situation is matched, it causes the action to execute. One rule might contain multiple Boolean conditions for the situation. For example, a rule may state that if the enemy is visible, the agent's health is at high level, and the agent's weapon is better than the enemy, then the agent should attack. The action might simply be a single movement, or a set of complex operations. This action will be executed until the next rule is selected based on the updated environment. We use the following examples to show that how a single character agent is controlled by the engine through operating rules.

John *et al.* [77] designed a well known agent for Quake II commercial game, called Quakebot based on Soar. Figure 2.1 shows the construct of applying Soar to Quakebot. Soar is designed as an engine to make decision and execute decisions. Soar Quakebot develops a series of tactics using one hundred operators, of which twenty have sub-states, and more than seven hundred rules including collecting rules (eg: collect items to power up), attacking rules, retreating rules etc. The reasoning module runs on a separate computer and interacts with the game by Quake II interface. The implementations of the Soar Quakebot's sensors and motor actions are embedded in a dynamically linked library. The knowledge of how to play the game and how to use the internal map is encoded by rules. Soar transmits all information such as the perception and motor information between the Quakebot and the game. Then the Quakebot can operate depending on these rules. We see

that, even the complex agent in large scale simulation should be implemented based upon the operation of rules.

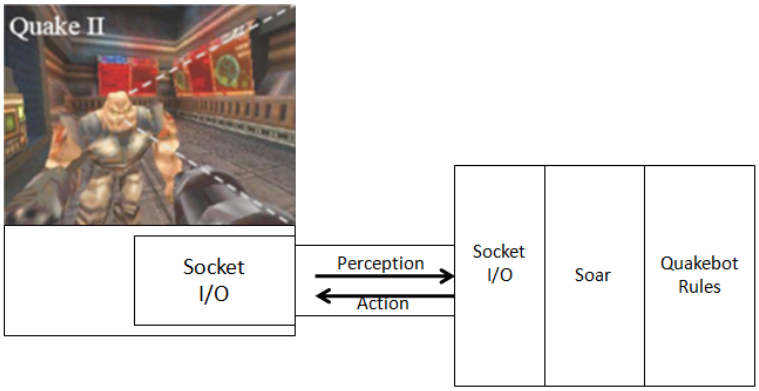

Figure 2.1: The interface between QuakeII and the Soar Quakebot (Adopted from [77]).

Prieditis *et al.* [116] once proposed a model-based system, which was an extension of rule-based systems. He applied this approach in the Locust AI engine to QuakeIII and got preliminary results.

Figure 2.2 shows the example structure of the model-based method. In this structure, the states or situations are represented by nodes. The moves from old states to new states are represented by directed arcs. Each arc emanating from the root node represents the actual choices at hand. And those below represent future choices. The best choice is the one which maximizes the expected outcome.

Simple rule-based systems is not sufficient to handle the complex problem of NPC modeling in computer games. The knowledge base constructed by experts always includes human bias. It is difficult to create a knowledge base with concise but effective rules. Although researchers have tried to combine rule-based systems with computational intelligence methods, such as decision tree, finite state machine

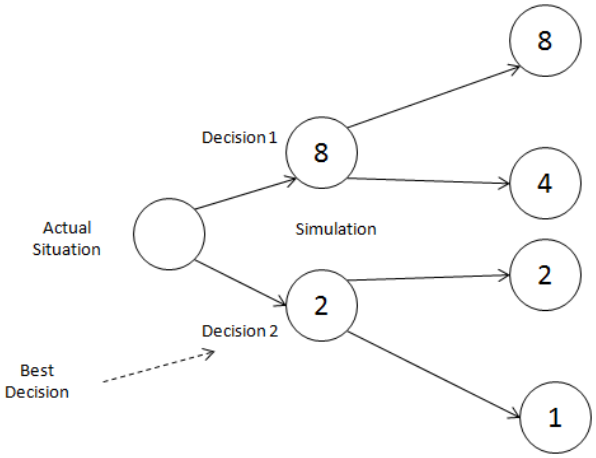

Figure 2.2: A sample structure of model-based method (Adopted from [116]).

or genetic algorithms [44] [8] [116], the virtual characters created by traditional rule-based system are still lacking in flexibility and adaptability to environment.

2.1.2 Finite State Machine

Finite state machine is another simple but popular way to modeling AI behaviors in non-player characters [30] [117] [68]. In finite state machine, the transition of behaviors is based on context sensitive stimulations received from the game. When a valid input event generates a transition, the state machine will move from the present state to the next state. It is regarded as a mathematical model of system that has discontinuous inputs and outputs.

FSMs are often depicted graphically using flowchart-like diagrams in which states and transitions are drawn as rectangles and arrows. Figure 2.3 shows a simple example of FSM-based behavior model. In this structure, the states and actions describe how the NPC will act. The transitions between states represent the decisions made by the NPC. The conditions represent the situations which could trigger the transition from one state to another. In a way, FSM makes the modeling of NPCs

behavior more easily to be implemented.

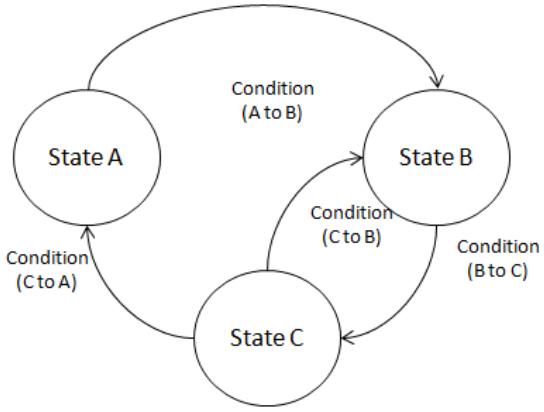

Figure 2.3: Example of an FSM as a diagram.

A lot of research has been done in developing the technologies of implementing FSM in commercial games. Dan *et al.* [30] defined FSMs by using macro-assisted FSM language created by Rabin [117], which is readable and maintainable. Then the FSM can be realized by scripting with an FSM language. The behavior can be modeled as a sequence of different character "mental states". The transitions among states are driven by the actions of the player or other characters, or possibly the game environment. Later, Eric [40] created a high-level custom scripting language with finite-state machine qualities built into the grammar. This language allows the designer to create states and transitions in an easy way. Furthermore, Paul [111] provided a stack-based finite-state machine, which operates by adding a state stack and new transition types. This method allows game developers to specify how the stack should be manipulated when a transition occurs. Moreover, Gilberto [49] implemented a data-driven state machine as an FSM class that allows developers to define state transition logic through external data. This method allows dynamically operating behaviors as well as quickly changing the state machine's logic.

To integrate FSMs into a specific game environment, the designers need to select an FSM processing model and consider how to optimize FSM architecture for efficient execution. Many different ways have been applied to implement it. According to Kwon *et al.* [76], FSM can be realized in forms of deterministic FSM, non-deterministic FSM and hierarchical FSM. Kwon *et al.* made use of hierarchical structure finite state machine, that used the previous planned state and the behavior mode to control the NPCs action in a first person shooting (FPS) game. Cho *et al.* [28] described a real life behavior framework in simulation games based on probabilistic state machine with Gaussian random distribution. According to the dynamic environment information, NPCs can generate behavior planning autonomously associated with defined FSM. However, the architecture is rigidly defined and the agents are hard-coded. It follows that the agents will always perform the same action under similar circumstances and are lacking in evolving ability.

2.1.3 Artificial Neural Network

Artificial neural network (ANN) is made up of units and weights, which represent biological neurons and synapses, respectively. ANN can be seen an electronic simulation based on a simplified human brain. Our brains are made up of approximately 100 billion neurons, each with up to 10,000 connections to other neurons. These connections to other neurons are called synapses and are the channels for the transfer of chemical signals between neurons. Inspired by the way biological nervous systems process information, each ANN includes a large number of highly interconnected neurons working in unison to solve specific problems [6]. The common use of ANNs includes memory, pattern recognition, learning, and so on. The remarkable ability of ANN is to approximate functions or to solve certain tasks by learning from ex-

amples, such as learning, self-organization, real time operation and fault tolerance abilities.

It is very important to apply ANN in the modeling of NPC in virtual environment in term of improving NPCs learning ability. A diagram of a simple network used for behavior model in computer game, adopted from [143] (see Figure 2.4), shows the general structure of ANN. Commonly, ANN consists of an input layer, zero or more hidden layers, and an output layer. The input layer consists of units that represent the input to the network. Each input unit has one or more weights that feed into the first hidden layer or the output layer. Each weight allows an input unit to provide an excitatory or inhibitory influence on the unit to which it is connected. The output layer consists of one or more units that compose the output of the network. With multiple output units, the system can classify the input into one of many categories. The hidden layers in ANN are used to make the network more powerful by extracting features from the data. Each hidden unit also has an associated set of weights that feed into the next hidden layer or the output layer.

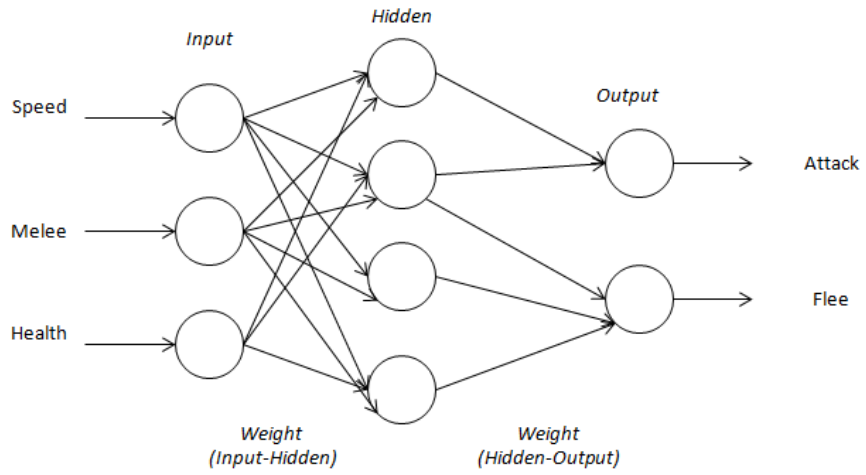

Figure 2.4: Demonstration of a simple artificial neural network (Adopted from [143]).

Each unit in the hidden and output layers has an adder for combining their input signal and an activation function for determining their level of activation. The adder sums the input entering the unit, where each input value is calculated by mutiplying the input unit by its associated weight. The activation of the unit is shown as:

$$\alpha = \sum_{i=1}^{n} x_i w_i \qquad \text{(Eq. 2.1)}$$

where α is the activation, n is the number of units, x_i is the i^{th} unit and w_i is the i^{th} weight. After the sum of the input to the unit has been calculated, this value is then fed into an activation function that determines the level of activation of the unit. The unit then "fires" and propagates its activation onto the next layer.

Basically, ANN could be used to make decisions or interpret data depending on previous input and output examples [65]. Neural network was used in game, by Sweetser [144], to realize weighted sum, analyze the data from the influence map, and make a decision based on the current state. Influence maps is composed by several layers, in which each layer represents various variables of the computer game. The weighted sum is used to combine these layers. Some or all of these layers are combined via a weighted sum to provide an overall idea of the suitability of each area on the map [132]. And the agent system will made decisions based on these information. The training dataset is built up based on the decisions that human players made in different game states, and then this data could be used to train the neural network, so that the system learns behaviors from human players.

To train an ANN, the training data can be composed of many different types that represent various types of events, characters, or states [65]. In a computer game environment, the input is a set of variables from the game world, similar to that

used by a state machine, that represents attributes of the game world. The output from the ANN can be a decision, a classification, or a prediction. For example, the input to the ANN could represent the attributes including health, hitpoint, strength, stamina, attack, and so on, which the NPC may encounter in the game world. The output could be a set of possible actions that the NPC takes, such as talking, running away, attacking, or avoiding.

Generally, there are two kinds of training methods. First, the ANN can be used in environment by retriving experienced knowledge of environment and the corresponding outcomes in these situations. Alternatively, a training set could be input to the ANN during development, either hand-crafted by the developer or built up by exploring during playing. The learning method would allow the agent (NPC) to adapt to the player and have different outcomes dependent on different experiences. However, this method requires the ANN to perform computation during the game running.

So far, ANN has achieved good performance in modeling NPCs in computer games. The game "Battlecruiser: 3000AD (BC3K)" is the first computer game which uses artificial neural networks. ANN is used to control the non-player characters as well as to guide negotiations, trading, and combat. ANN is also used for basic decision making and pathfinding, with a combination of supervised and unsupervised learning. In the game of "Black and White", the player has a creature that learns from the player and other creatures. The creature's mind is realized by ANNs [42]. The series of games "Creatures" makes use of ANN techniques, including heterogeneous ANNs, in which the creatures utilize ANNs to learn behavior and preferences over time. The game "Heavy Gear" uses an ANN as a part of the control mechanisms. Each specialized ANN is used for a particular aspect [143].

2.1.4 Fuzzy Logic

Fuzzy logic proposed by Lotifi Zadeh in 1965 [171] is a another approach to construct behavior rules for NPC [84]. Zadeh demonstrated that "fuzzy logic is a means of presenting problems to computers in a way akin to the way humans solve them." A fuzzy set is considered to be a family of pairs, $R = \{(x, \mu_R(x))\}$, and membership function $\mu(\cdot)$. The degree of membership is viewed as some kind of weighting on elements of the underlying reference space X. A fuzzy restriction is the mapping μ. $\mu : X \to [0, 1]$.

Fuzzy techniques provides a design methodology that is convenient to transfer human knowledge. First, it allows input-output rules to be built upon the expert's knowledge and used instead of mathematical models. Sometimes, the complex mathematical models may be impossible to derive. The implementation of fuzzy logic can be simplified to IF-AND-THEN statements. Second, fuzzy logic supports linguistic formate, which provides a simple platform for developers and users to design and customize, since rules are derived from language descriptions rather than mathematical formulas. Furthermore, fuzzy logic provides an alternative solution to describing the nonlinearities in a system through the developer's intuitive understanding of the system. Thus, fuzzy logic can be used as a basis for controlling a complex process, such as real-time controlling of NPCs.

Fuzzy rules have been widely used in games. Yifan *et al.* [169] described a general structure of fuzzy rule-based system, which includes a fuzzifier, that transfers input into fuzzy sets; a rule base, in which each rule is expressed in a IF-THEN form to present the relations between the inputs and the desired output; a reasoning unit that applies inputs to the rules; and a defuzzifier to convert the output into crisp sets. The four components and their relationship are shown in Figure 2.5. By

using fuzzy logic, the human experts experience can be realized in the design of the controller.

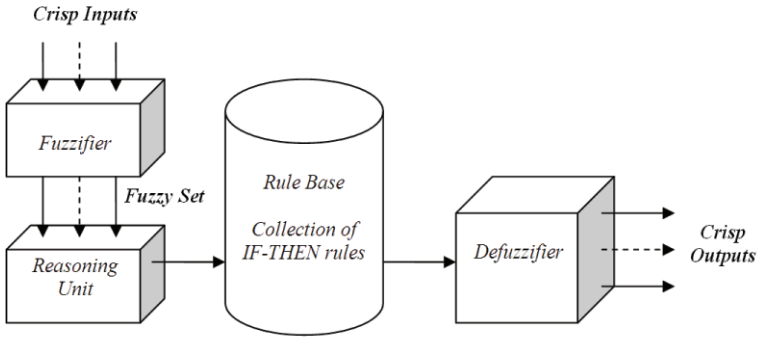

Figure 2.5: Four components of fuzzy rule-based system.

Kenneth *et al.* [55] modeled autonomous mobile robot's emotions and behaviors by a set of Fril's support logic fuzzy rules, Fril [10]. Using Fril, Kenneth *et al.* models emotion as a set of hierarchical fuzzy rules. Each emotion state is defined by a fuzzy rule based on its previous state. These fuzzy rules allow the NPC to control and to select different behaviors depending on the environment and its experience. The fuzzy rules in games work like a behavior controller and generator. Leong *et al.* [84] developed a fuzzy cognitive goal net intelligent virtual agent within a massive multi-player online games. This agent was applied to control typical non-player characters by exhibiting goal-oriented behavior and non-deterministic behavior.

In summary, fuzzy logic could help game agents to take responses with a certain degree of unpredictability by inference and non-deterministic behaviors. However, as fuzzy logic operations are still based on predefined experience, NPCs created based on this method are also lacking in learning ability and cannot evolve during the game-playing.

2.1.5 Confabulation Theory Based Method

Confabulation theory is proposed by Hecht [53] to explain the cognitive mechanism of human brains. Recently, some researchers simulate this kind of theory in the modeling of synthetic characters. The confabulation theory based modeling method is essentially a kind of probabilistic method based on Bayesian theory. Although it was rarely used, it has shown some success in this research area.

Confabulation theory comprehensively and concretely explains the animal's cognition. According to confabulation theory by Hecht-Nielsen, all aspects of cognition such as seeing, hearing thought processes etc, are all implemented by four fundamental elements: "a universal modular system" for representing the objects of the mental world; "knowledge links" which connect the neuron collection representing one symbol to neurons; "confabulation operation" which is like a winner-take-all competition process among the symbols; and the "origin of behavior" [53].

Confabulation theory "hypothesizes the specific underlying mathematical mechanism of cognition; as well as the human neuronal implementation of that mechanism" [53]. Applying confabulation theory in games is helpful to construct NPCs behavior selection mechanism and interactive model [70] [73]. Robot intelligence technology lab in Korea focuses on the research of applying confabulation theory to synthetic character's behavior selection. Kim *et al.* [71] combined a deterministic behavior selection and a probabilistic behavior selection together to make decision of every behavior based on both of characters internal states and external sensor information. For example, Cho *et al.* proposed a behavior selection approach for a synthetic character based on confabulation method [28]. Kim *et al.* proposed a architecture which had two layers for synthetic characters' behavior selection based

on confabulation and learning method [69]. However, for probabilistic based methods, the parameters are usually defined based on personal suggestions. As different people have different judgment for different kind of behaviors, human bias may be included in probabilistic methods [158].

2.2 Affective Modeling

In the second part of the review, Section 2.2.1 presents some traditional emotion theories. Section 2.2.2 reviews the most popular psychological emotion models in recent decades. Finally Section 2.2.3 elaborates the popular computational models in recent research.

2.2.1 Traditional Emotion Theories

Emotion is a complex psychological and physiological phenomenon involving an individual's state of mind and its interaction between that individual and its environment. The topic of how emotions are generated has been studied over a long period. The following lists several traditional and well known emotion theories:

James-lange theory, which is proposed by James in 1884 and Lange in 1887 [62], says the emotion is not the cause but a result of the human beings physiological event. Therefore once a human has experiences, his/her autonomic nervous system creates physiological events such as muscular tension, heart rate increases, perspiration, dryness of the mouth, etc [2].

Cannon-bard theory, which is proposed by Cannon in 1927 [21], points out that people feel emotions and physiological changes (such as muscular tension, sweating, etc.) because a stimulating event happens [2].

The Two-factor theory is proposed by Schachter and Singer in 1962 [125]. It argues that emotion is firstly a experience of physiological arousal. That is, when an individual experiences a feeling, he/she tries to find a label to explain this feeling [2].

Lately, several theories explain emotion from the evolutionary view, arguing that emotion is a developed mechanism belonging to advanced species as their solutions for adaptation problem, in order to survive under a natural selection environment [137] [37]. These theories focus on the physical displays of emotions and they believe that emotions express a person's appraisal of a person-environment relationship which may demonstrate a harm or benefit for the individual.

On the other hand, the cognitive theories argue for the role of cognitive activity in emotion generation, since the occurring and intensity of emotion are controlled through cognitive judgement and evaluation [108] [137] [126]. Meanwhile, the neurobiological theories point to the phenomenon that the occurring of emotions relates to distributed neural activations in human brains [112].

Despite the differing views, the above theories generally agree that emotion is the combinative outcome of numerous internal factors and external circumstances.

2.2.2 Psychological Emotion Models

This section takes a review of the psychological emotion models proposed in the past decades. In early 1989, Ortony, Clore and Collins [108] proposed a model called OCC, which has been widely utilized in the following years. In this model, emotions are organized in groups according to what kind of aspect they focus on of the world, which includes events, agents and objects (see Figure 2.6). Twenty-two emotions are defined based on their eliciting conditions. The work of OCC also focused on the relationship between appraisal variables and specific emotion types.

The characterization of each emotion includes "local" or "global" variables, that influence the intensity of emotions. OCC theory proposed detailed descriptions of how the emotions are structurally related based on their eliciting conditions; how the emotions are triggered based on appraisal of events; and how the emotions could be computational tractable. The most obvious contribution of structure theory in OCC is that it provides us a direct and systematic way to understand the various emotions and their relations with each other. Thereafter, OCC model is widely adopted in modeling of emotions as a prototype as it is easy to be applied.

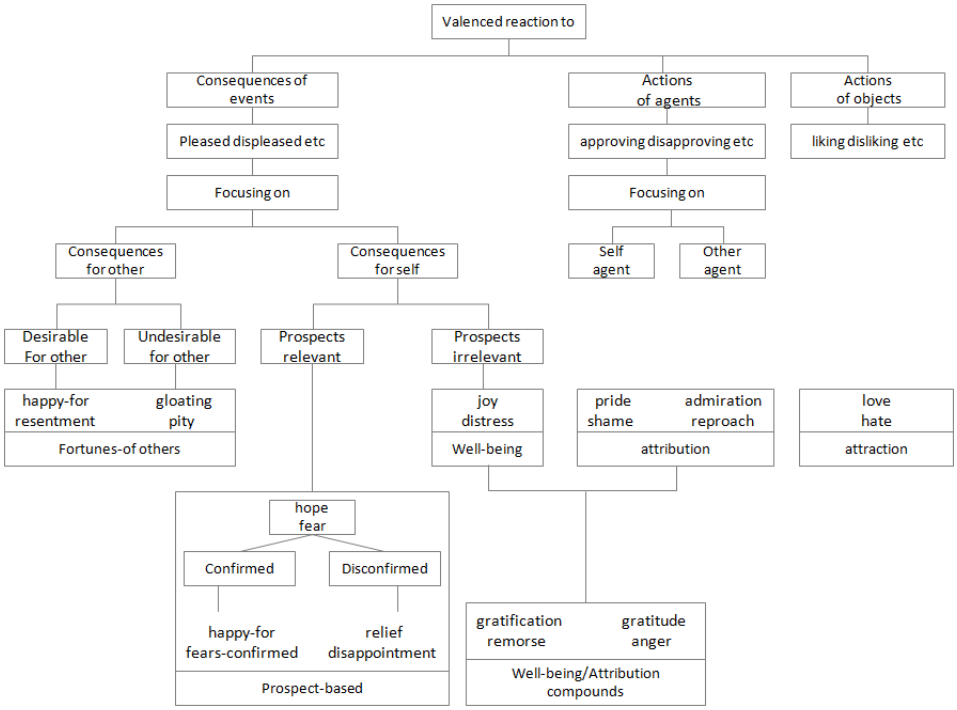

Figure 2.6: A diagram of OCC emotion model [108].

Later in 1994, Damasio [29] distinguished emotions into two classes: "primary emotion" and "secondary emotions". The primary emotion is supposed to be innate and trigger reactive response behaviors, whereas secondary emotions arise from high level cognitive processes, associating with the ability to evaluate the outcome and

expectations [15]. Secondary emotions utilize the machinery of primary emotions. The two level architecture of emotion have been applied in a lot of simulation about emotion affection. His model also described the role of the body in emotions and feelings as well as the high-level architecture of the human affective system.

Smith and Lazarus [137] depicted a cognitive-motivational-emotive model in which emotion was viewed as a two stage control system: appraisal and coping. Smith and Lazarus assume that human beings are constantly engaged in appraisals of their relationship with environment. The emotion appraisal model is defined as a dynamic appraisal-coping-reappraisal process: appraisal promotes the evaluation of conditions and determines the emotion state, and then, the emotion motivates individuals to cope with their environment.

A Core Relational Themes is mentioned in [137], saying that emotion is the identification of individual's core relation with environment in terms of harm or benefit. Six appraisal components are proposed for primary appraisal and secondary appraisal: motivational relevance, motivational congruence, accountability, problem-focused coping potential, emotion-focused coping potential and future expectancy. For the appraisal of each of the emotions, the six appraisal components are combined into core relational themes. For example, anger motivates the person to eliminate, or undo a source of harm, and anxiety motivates the person to avoid potential harm.

Figure 2.7 depicts the appraisal process. In this three-level model, firstly individuals build up interpretation of the objective conditions based on their own core relational theme of harm or benefit. In this stage, the personal goals/need, beliefs or knowledge will be considered comprehensively. Then, in the appraisal process, a number of features are applied to characterizes the interpretation of person-environment relationship. The following is an emotional response, which is at the

center of system including a distinctive subjective experience, action tendency, and physiological motor. The coping action will finally suggest changes to the objective conditions. Thus the reactions constantly maintain the person-environment relationship.

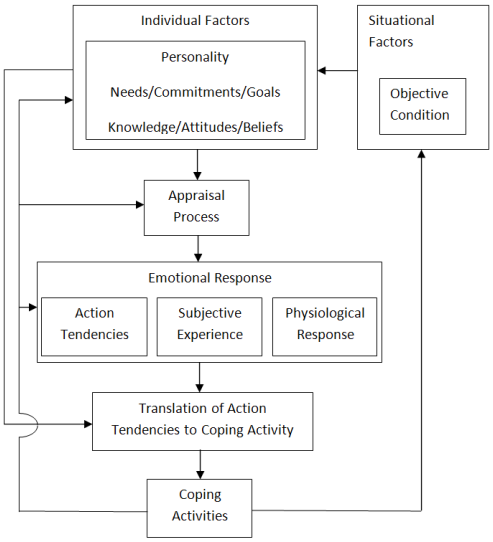

Figure 2.7: The cognitive-motivational-emotive system by Smith and Lazarus (adopted from [137]).

Later in 2001, Scherer *et al.* [126] proposed a multi-level sequential check model. Scherer describes emotion as a continuous mechanism that adapts to environment by taking responses to stimulus. To evaluate these stimulus events, this mechanism checks four appraisal objectives: how relevant is this event for me? what is the implication of consequences of this event and how it affect my goals? how well can I cope with these consequence? and what is the significance of this event? These four objectives are called relevance, implications, coping potential and normative significance respectively. The checks suggest that appraisal of stimulus events should be described in detail, and grouped by multiple objectives. Moreover, the appraisal of emotion is not isolated but interacting with personal inner states.

In this model, the process of appraisal is built based on the claim that the stimulus evaluation checks are processed in sequence. This sequential check theory of emotion is regarded as a part of a dynamic model of emotion. Emotion is argued as an multi-shot process, in which appraisal is followed by re-appraisal. This point of view is similar to the feedback based cycle model by Smith and Lazarus [137]. The re-appraisal serves to correct the evaluation results based on new information. However, Scherer also admitted existence of parallel processing in the sequence assumption.

Emotion is described as a three-level process, including sensory-motor level, schematic level and conceptual level by Scherer [126]. A neural network architecture has been chosen to demonstrate the potential architecture of the appraisal process (see Figure 2.8). This architecture explains the relationships among action tendencies, appraisal objectives, and appraisal registers. However, the neural network architecture lies in only proposal level, has not been realized in a practical system.

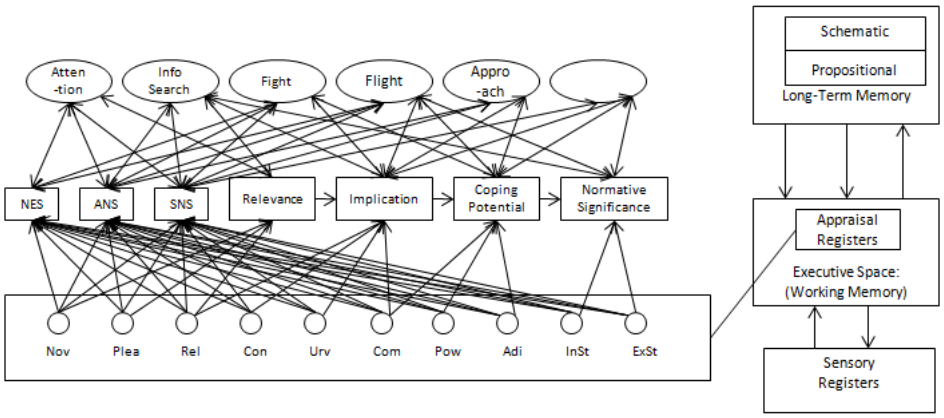

Figure 2.8: The architecture of appraisal process by Scherer. (adopted from [126])

2.2.3 Computational Emotion Models

Although extensive research has been done in exploring the psychological mechanism of emotion, they do not propose detailed computational models that are suitable to be applied in practical level. Nevertheless, these achievements has inspired researchers to implement emotional cognitive systems for virtual characters.

Recently, many computational models are proposed for emotions of virtual agents. Some of them are built based on one or more of psychological models, aiming at simulating the process of emotion. The others are built within an architecture which integrates cognition, emotion, and behaviors together, aiming at realizing accurate emotional responses for interactive agents. The former characterizes a computational model as a process operating on a set of causal representations [92], whereas the latter treats emotions as labels when integrated with cognition or decision making. Appraisal of emotion works like an evaluation system which estimates and analyzes the cause of emotion via a set of IF THEN rules. Depending on the different application objective, they tend to cooperate with different appraisal mechanisms and variables.

The psychological prototypes of emotion are widely adopted in computational models. For example, Ortony, Clore and Collins [108] model is cited in EM by Neal Rilly *et al.* [101], FLAME by El-Nasr *et al.* [38], FAtiMA by Dias *et al.* [34], ALMA [48] by Gebhard and so on. Based on Smith and Lazarus's appraisal theory [137], Gratch *et al.* focused on the dynamic evaluation of emotion and proposed the process model of emotion appraisal [52]. Other appraisal models were inspired by the Scherer's sequential-checking theory, such as WASABI by Becker-Asano *et al* [15], and PEACTIDM by Marinier, Laird *et al.* [90].

El-Nasr *et al.* presented FLAME based on the OCC model [108] and the "event-appraisal model" [120] [38]. FLAME maps the assessments of events with goals into emotions using a fuzzy method. Action selection is then associated with specific emotions by fuzzy rules in a simple but friendly relationship. FLAME, however, is designed for virtual pets and not for simulating human emotions.

FAtiMA, a two-layer architecture, was proposed by Dias *et al.* [34] to create virtual agents based on the OCC theory [108]. In FAtiMA, the appraisal of emotion is mainly based on the evaluation of desirability, desirability for other, praiseworthiness and like relation. This method is however rather simplistic and is not able to deal with complex situations.

Also based on the OCC model [108], Gebhard [48] proposed ALMA which describes emotion as a short term affect. ALMA makes use of personality, mood and emotion for generating the emotions of virtual characters. However, without a general appraisal principle, its appraisal specifically depends on the domain knowledge and personality profiles.

Marinier *et al.* [90] described PEACTIDM, a computational structure to support emotional appraisal based on Scherer's theory [126]. Following Scherer's appraisals, PEACTIDM's emotions are decided by six dimensions: suddenness, goal relevance, intrinsic pleasantness, conduciveness, control and power. The variables and rules need to be defined and evaluated subjectively.

Marsella and Gratch [93] build EMA based on the dynamic of emotional appraisal. The appraisal frame of EMA is designed with a set of variables which are used to evaluate the "significant events", which can facilitate or inhibit a state. Emotions are categorized mainly by the variables of relevance, desirability, likelihood, expectedness, causal attribution, controllability and changeability. EMA is an en-

gineered model instead of a learning model. Therefore it's limited in emulating the personalized human emotions in a dynamic environment.

Becker-Asano *et al.* [15] developed WASABI, which simulates the appraisal processes based on the pleasure-arousal-dominance (PAD) space and the Scherer's sequential-checking theory [126]. The three-dimensional emotion space describes all events in terms of three dimensions, named pleasure, arousal and dominance. WASABI makes a distinction between primary and secondary emotion and has been used to model the emotion of a virtual human called MAX in a card playing scenario. However, the PAD values of each event have to be defined beforehand. It is thus also not easy to use the model for situated agents in a dynamic environment.

THESPIAN proposed by Si *et al.* [131] is based on the "appraisal-coping-reappraisal" loop described by Smith and Lazarus [137]. Appraisal in this model is designed as a continuous process considering the multi-agent situation. THESPIAN is also a dimension based appraisal model including motivational relevance, incongruence, accountability, control and novelty requiring objective evaluation. However, the modeling of emotion instances, for which a certain state of expressible emotion, is not considered in THESPAIN.

Chapter 3

Learning Behavior Models from User Patterns by Self-Generating Neural Networks

Computer games has been a good platform for testing the capabilities of various computational intelligence techniques. Following the development of games from traditional card games to modern games, such as the first person shooting (FPS) games, one key focus of modern computer game research has been on the use of machine learning techniques to make games challenging and enjoyable [158]. Specifically, the modeling of Non-player characters (NPC) is crucial for the success of commercial FPS games as it improves action capability of NPCs and the playability of games. However, designing a complex behavior mechanism for NPC can be difficult, as all possible conditions and consequences should be taken into consideration of the design.

Learning from behavior patterns, or imitative learning [33] [50], is a novel and promising approach to modeling NPCs' behaviors, as long as training data are available. The knowledge acquired by imitative learning can lead to embedded and reusable knowledge base of behavior mechanism. Different from traditional rule-based approaches [107], behavior learning avoids the need for manually defining

the behavior rules from scratch. Compared with reinforcement learning, imitative learning has the benefit of learning explicit signals provided in the behavior patterns observed.

This chapter studies a specific classes of self-organizing neural networks, namely Self-Generating Neural Networks (SGNN) in behavior learning task. Compared with traditional neural networks, SGNN does not require a designer to determine the structure of the network according to the particular application in hand. However, the computational time of SGNN increases dramatically due to the continual creation of neural nodes. To overcome this problem, we propose a pruning method to optimize the SGNN network while maintaining the learning performance. We conduct benchmark experiments to examine the SGNN model and the pruning method in the aspects of generalization capability, learning efficiency, and computational cost. In addition, a novel rule extraction method is proposed to extract useful knowledge from SGNN and translate them into symbolic rules. Online testing of N-PC is conducted in a first person shooting game, known as Unreal Tournament 2004 (UT2004) with the Deathmatch scenario. The online testing results demonstrate that SGNN can be used to build autonomous players by learning the behaviour patterns of sample bots. We also compare the behavior learning ability of SGNN Bot, SGNN Bot with pruning, and SGNN Rule Bot.

3.1 Introduction

SGNN was firstly developed by Wen *et al.* [161, 162] based on self-organizing maps (SOM) and implemented as a self-generating neural tree (SGNT) architecture. Later, Inoue *et al.* [61] [60] improved the accuracy of SGNN by applying a multiple system and ensemble method. SGNN is appealing, as it does not require a designer

to specify the structure of network and the class parameters. In addition, it has an efficient learning ability and reasonably good adaptive ability. Because of these good attributes, SGNN is a suitable candidate for learning behaviour rules.

SGNN has been used widely in pattern recognition and time series signal prediction [57] [58] [86]. Since all input training data are inserted into the SGNT leaf neurons, the weights of each node neuron n_j are the averages of the corresponding weights of all its children. Inoue *et al.* [57] predicted the next time signal output by searching the winner for every input test data. Further more, Li *et al.* [87] applied ISGNN to predict time interval in software failure therefore making contribution in research of software reliability. Following the similar consideration in [57], Li *et al.* [85] predicted the output by matching the input test data to the nearest leaf node of SGNT. Although they used different methods such as ensemble in [59] and iteration in [85] to improve the accuracy of SGNN, the prediction methods of matching test data with leaf neuron are similar, and both do not consider the combination role of centre weight with leaf nodes.

For NPC modeling, the type of data we use is different from what they used in [57] [58] [86]. Moreover, although SGNN is effective in fast learning and effective classification, the computational time of it increases due to continual creation of nodes. To overcome this problem, an optimization of SGNN is proposed via a pruning method which improves its performance. This method is proved to be efficient for real-time experiment in UT2004 environment.

3.2 Structure of Self-Generating Neural Networks

SGNN is self-generating in the sense that there is no need to determine the network structure and the parameters beforehand. In this kind of neural network, not only

the weights of the neurons connections but also the structure of the network are all learned from the training instances directly [58]. That is to say, the number of the neurons in the network, the interconnections among the neurons, and the weights on the connections are automatically constructed from the given training instances. A neural tree generated in this self-generating way is called a self-generating neural tree (SGNT). Generating a SGNT is defined as a tree construction problem of how to construct a tree structure from the given instances which consists of multiple attributes under the condition that the final leaf neurons correspond to the given instances [58]. The generated SGNT is a tree structure for hierarchical classification. The sub-network of the SGNT can be seen as rooted by a root neuron. So, every neuron linked directed to root neuron represent the centre of a class, and each leaf node represents the typical value obtained from training data.

3.2.1 Generation of SGNT

To explain the self-generation process, we first define the relevant notations [161] as follows:

Definition 1: Each input example e_i is a vector of real attributes: $e_i =< e_{i1}, ..., e_{im} >$, where e_{ik} represents the k^{th} attribute of e_i.

Definition 2: The j^{th} neuron n_j is expressed as an ordered pair (w_j, c_j), where w_j is the weight vector $w_j = (w_{j1}, w_{j2}, ..., w_{jm})$, and c_j is the number of the child neurons of n_j.

Definition 3: Given an input e_i, the neuron n_k in a neuron set $\{n_j\}$ is called a *winner* for e_i, if $\forall j$, $d(n_k, e_i) \leq d(n_j, e_i)$, where $d(n_j, e_i)$ is the Euclidean distance between neuron n_j and e_i. The *winner* can be the root neuron, a node neuron, or a leaf neuron.

Table 3.1: Inputs of SGNN.

Inputs	*value*
A set of training example	$E = \{e_i\}$, $i = 1, ...N$
A threshold value	$\xi > 0$
A distance measure	$d(e_i, n_j)$

The building process of SGNT is, in the nutshell, a hierarchical clustering algorithm. Initially, the neural network is empty. The key input parameters of SGNN are summarized in Table 3.1. The generating algorithm is governed by a set of rules described as follows:

Node creation rule 1: Given an input example e_i, if $d(n_{winner}, e_i) > \xi$, where ξ is a predefined threshold, a new node n is generated by copying the weights (attributes) from the current example. Then the new neuron is connected to the *winner* as its child.

Node creation rule 2: If the *winner* node n_{winner} is also a leaf node, another new node n is generated by copying the weights (attributes) from n_{winner}. A neuron can be called a leaf node only if it has no child.

Weight updating rule: The weight vector of neuron n_j is updated by the attribute vector of e_i according to (Eq. 3.1):

$$w_{jk} = w_{jk} + \frac{1}{c_j + 1}(e_{ik} - w_{jk}), \qquad \text{(Eq. 3.1)}$$

where w_{jk} is the weight of n_j after learning the first k examples covered by n_j.

The above procedure builds a self-organizing growing network based on a set of training examples. However, the network may not have the best structure and there may be some dead neurons which will never be visited by any example in the future. Therefore, further optimizations may be necessary for the generated network.

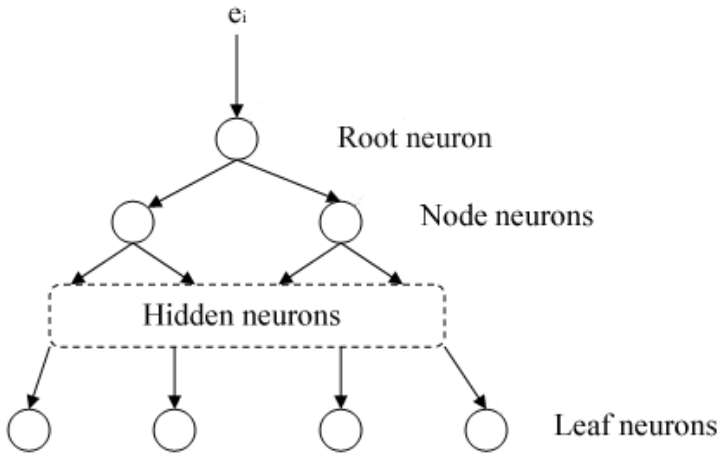

Figure 3.1: The structure of self-generating neural tree

Horizontal optimization rule: For a neuron n, if $d(n,p) > d(n,s)$, where p denotes the parent of n and s denotes the sibling of n sharing a common parent, n is moved down by one layer under s if s is not its only sibling. If s is a terminal neuron, a new neuron is generated as another child of s with the same weights as those of s. The weights of s is then updated by (Eq. 3.2):

$$w_{sk} = \frac{N_s \cdot w_{sj} + N_n \cdot w_{nj}}{N_s + N_n}, \qquad \text{(Eq. 3.2)}$$

where N_s is the number of examples covered by s. The number of examples covered by s should be increased to $N_s' = N_s + N_n$ accordingly.

Vertical optimization rule: For a neuron n, if $d(n,p) > d(n,g)$, where g denotes the grandparent of n, n is moved up by one layer as a child of g. The weights of p are then updated according to: (Eq. 3.3):

$$w_{pi} = \frac{N_p \cdot w_{pj} - N_n \cdot w_{nj}}{N_p - N_n}. \qquad \text{(Eq. 3.3)}$$

After updating, the number of examples covered by p is decreased to $N_p' = N_p - N_n$ accordingly.

After applying the optimization methods, the given data set would be hierarchically classified by the SGNT. In the final network, the number of children for the root neuron is the same as the number of classes in the data set. The attributes of the root's child neurons represent the weights of the cluster centers.

3.2.2 Class Prediction

In our application, the training data set D is collected from a sample robot in Unreal Tournament 2004 (UT2004) environment. The training data record the actions of the sample robot in various situations in real time. Each data sample includes a set of ten state attributes and an action. The data set D is presented as $\{(x_i, y_i), i = 1, ..., N\}$, where x_i is the input attribute vector and y_i is the desired output, the action's identification number.

When predicting the action output, given an input vector, a search process takes place in the SGNN to find the winner node. When the winner node is a leaf node, the class label of the leaf node is retrieved as the output for action decision. Considering the desired output class are not continuous but discrete variables, if the winner node is not a leaf node, we determine the output action by the following functions:

$$p(y_k) = \frac{N_{\mathbf{w}k}}{N_{\mathbf{w}}}, \qquad \text{(Eq. 3.4)}$$

$$p(y_{\mathbf{w}}) = \max_{k=1,...,K} \{p(y_k)\}, \qquad \text{(Eq. 3.5)}$$

where K and $N_{\mathbf{w}}$ indicates the number of actions and leaf nodes under the winner node respectively, and $N_{\mathbf{w}k}$ denotes the number of leaf nodes having the same referent output y_k under the winner node. Thus if most members of the winner node have the output $y_{\mathbf{w}}$, we set $y_{\mathbf{w}}$ as the desired output.

3.2.3 Pruning

An obvious weakness of self-generating neural network is the continual increase of the number of nodes as the number of samples increases. When the number of samples is very large, the training speed will slow down dramatically when the number of nodes is extremely large. In prior work, researchers also consider this problem [61, 60] and propose pruning algorithm for a multi-classifier system comprising of multiple SGNN. However, the multi-classifier system (MCS) is aimed at improving classification accuracy at the cost of more learning time and it is difficult to apply MCS in real time. Here we introduce a novel pruning method for single SGNN systems, which is aimed at improving learning and classification efficiency.

During the generating period of SGNN, the number of nodes increases continuously. Let N_o denote the total number of input training samples and S denote the current number of nodes. In this pruning method, we introduce a threshold S_T, which is defined as:

$$S_T = \eta * N_o \quad (\eta > 0). \tag{Eq. 3.6}$$

As the number of input samples increases during the training period, the pruning procedure kicks in when the number of nodes exceeds this threshold ($S > S_T$), which can be set according to a function of the learning speed. When pruning occurs, it checks the connections among the root neuron, the node neurons, and the leaf neurons for redundancies. If there is a hidden node h positioned between the leaf node and its corresponding node neuron, h will be deleted. The leaf node is then connected to the node neuron directly as its child. The weights of node neuron c are then updated according to (Eq. 3.7):

$$w_{ci} = \frac{N_c \cdot w_{cj} - w_{hj}}{N_c - 1}, \tag{Eq. 3.7}$$

where N_c is the number of examples covered by c. The number of examples covered by c is decreased to $N_c^{'} = N_c - 1$ accordingly.

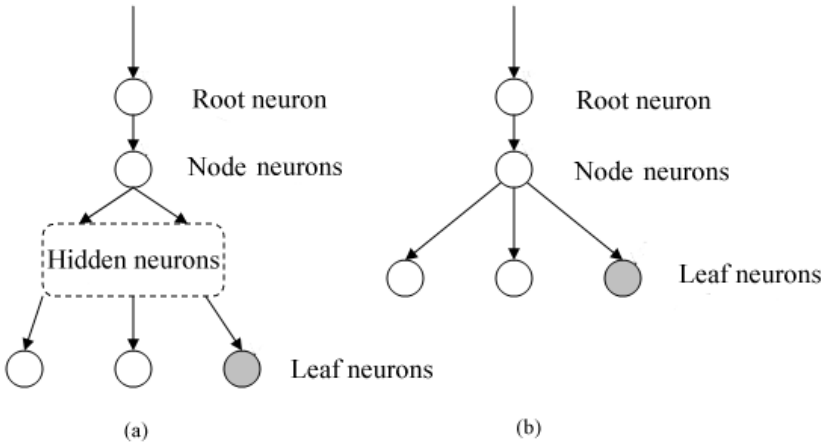

Figure 3.2: Structure of nodes in SGNN (a) Before pruning; (b) After pruning.

Figure 3.2 shows an example of the structure before pruning and after pruning.

3.2.4 Multi-class SGNN

In our application of behavior learning, SGNN is used to learn the sample user's behavior model. Although SGNN can perform well in terms of accuracy, real-time decision making is quite complex due to the exuberant tree structure of SGNN. To simplify the action selection process, we introduce a method to extract behavior rules from SGNN by constructing a multi-class SGNN system. This method also provides a way to extract useful knowledge while avoiding redundant computation.

To construct a multi-class SGNN, the training data set is first divided into several partitions according to their referent output actions. Each partition of the data samples (belonging to the same class) is used to build a SGNN tree. The individual SGNN trees built are then joined by their root nodes into a single SGNN. In the

merged SGNN, each child of the root node represents a center of a class and the number of classes corresponds to the number of children of the root node. Therefore, for each sub-SGNN, a set of rules can be extracted based on the centers of classes. Suppose that the training data set has been clustered into K classes according to K actions. A total of $\sum_{k=1}^{K} r_k$ rules can be obtained, where $r_k(k = 1, ..., K)$ denotes the number of rules for each action.

3.3 Learning Behavior Patterns in Unreal Tournament

3.3.1 UT2004 Environment

Unreal Tournament 2004 (UT2004) is a first person shooting game featuring close combat fighting between robots and human. Figure 3.3 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.

UT2004 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 [56], is available to developers for building agents for the UT environment. This means the developers can implement their own agents (or bots) using any specific algorithms and run them in UT.

Figure 3.4 (adapted from [56]) shows the interface between Pogamut and the Unreal game server. Pogamut runs as a plug-in for the NetBeans Java development

Figure 3.3: Unreal Tournament 2004 game environment

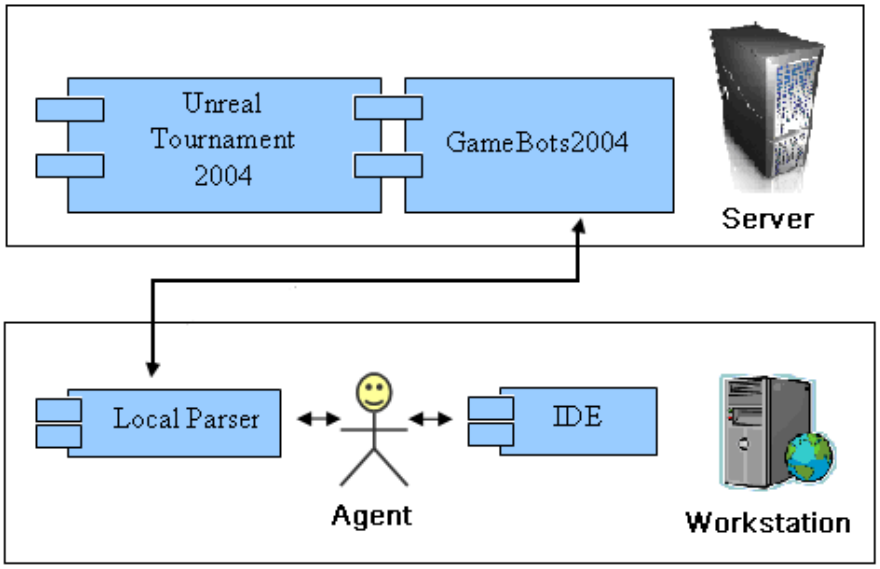

Figure 3.4: The interface between Pogamut and the Unreal Tournament 2004 (adapted from [56]).

Table 3.2: 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 3.3: The ten 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?

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.

3.3.2 The Behavior Learning Task

We focus on the task of learning from the behaviour patterns from a sample bot called Hunter Bot provided in UT2004. *Hunter Bot* is a rule-based sample bot,

Table 3.4: 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 $CanRunAlongMedKit$	$GetMedicalKit$

which exhibits a full range of combat competency, including fighting with enemies and making use of resources such as weapons and medical kits. Hunter has eight types of behaviors (shown in Table 3.2) which he switches from one to the other based on ten state attributes (shown in Table 3.3). With the exception of the health attribute, all attributes are boolean. There are in total eight main rules captured in the Hunter's behavior mechanism based on these state attributes, which are summarized in Table 3.4.

When playing the UT2004 game, the internal states, external states, and behavior patterns of Hunter are recorded as training data. Each training example consists of a vector of the state attribute values as well as the behaviour (action chosen). The collected data are then used to train the self-organizing neural models using the supervised learning paradigm. After learning, the behavior pattern rules can be utilized as the embedded knowledge of a new bot. By assimilating the behaviour of the sample bot, the new bot is expected to exhibit similar behavior patterns in the same environment and produce comparable fight competency to the sample bot.

To evaluate the effectiveness of SGNN in learning NPC, we first conduct benchmark experiments based on off-line learning to compare their performance in terms

of learning time, generalization capability and computational cost. Online testing of bots are subsequently conducted, wherein we investigate the competency of the new bots when fighting against the sample bot which they learn from.

3.4 Experiments

3.4.1 Off-line Testing

We first conduct empirical experiments to evaluate the performance of SGNN in off-line learning. The data set consists of a total of 8000 training samples and 8000 test samples generated by the Hunter Bot. By training on a varying number of training examples, we test the generalization capability of SGNN on an equivalent number of test samples. We also measure their efficiency in terms of the number of internal nodes/rules created and the time taken for learning.

In our experiments, SGNN uses a standard set of parameter values. For SGNN, we adopt $\xi = 0$ and $\eta = 1.5$.

Figure 3.5 summaries the performance of the baseline SGNN (without pruning), SGNN with pruning, in terms of the classification accuracy on the test set. As the size of the training data increases, the accuracies of the two models converge to 100%. In general, SGNN with pruning achieves roughly the same accuracy level as SGNN without pruning.

Figure 3.6 depicts the performance of the baseline SGNN (without pruning), SGNN with pruning, in terms of average numbers of neurons/nodes created during learning. We see that the pruning method greatly reduces the number of neurons in SGNN.

Figure 3.7 shows the learning time taken by baseline SGNN (with pruning), SGNN with pruning. For SGNN with pruning, as the number of nodes is reduced,

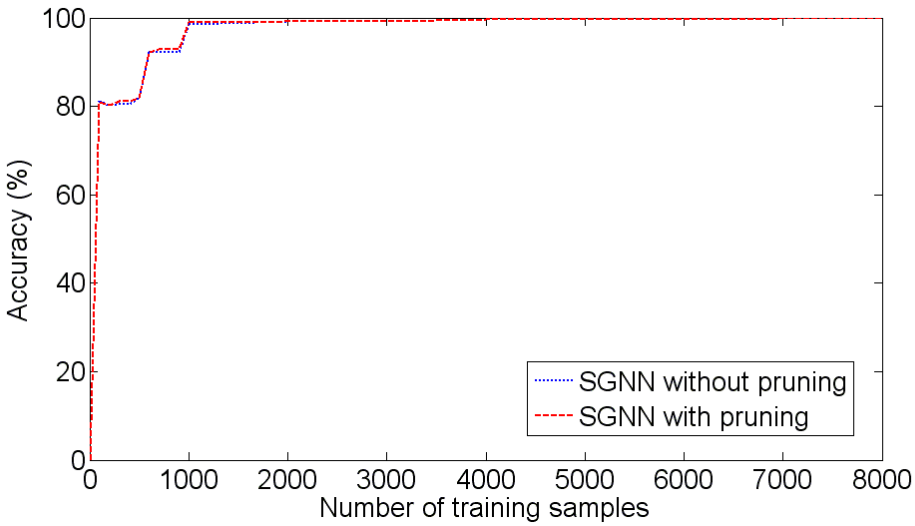

Figure 3.5: The accuracy of SGNN using pruning method compared with SGNN without using pruning

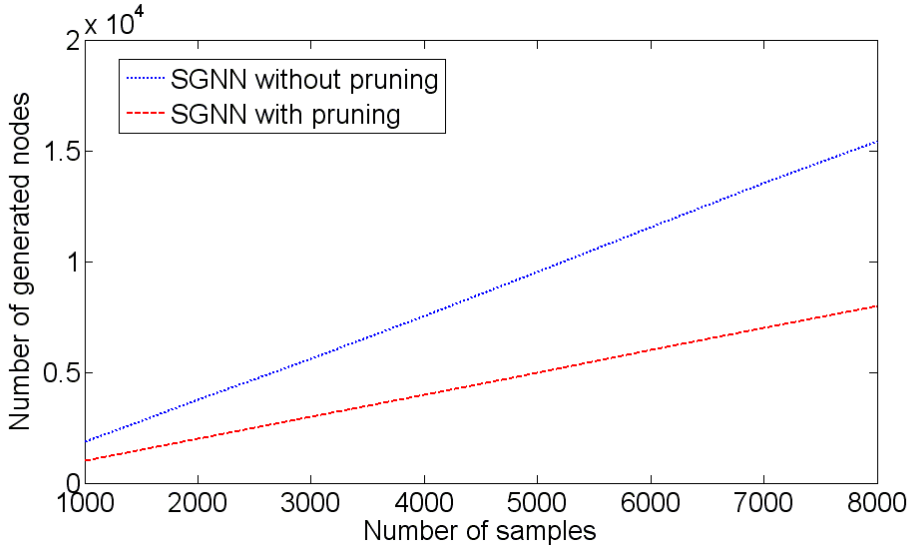

Figure 3.6: The number of node generated by SGNN using pruning method compared with SGNN without using pruning

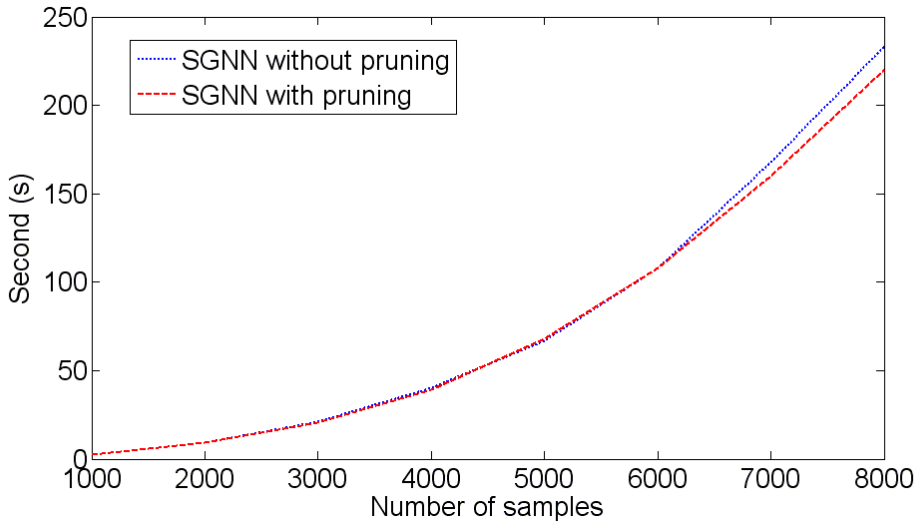

Figure 3.7: The running time of SGNN using pruning method compared with SGNN without using pruning method

so is the time for learning. Moreover, the amount of time saved increases as the number of samples increases. Considering all aspects, the pruning method is proved to be effective for SGNN, as it effectively reduces the number of neurons and the learning time, while maintaining the learning accuracy.

3.4.2 On-line Testing

In this section, a series of experiments are conducted in UT2004 game environment to check if the bots created based on SGNN, SGNN with pruning, and rules extracted from SGNN, could learn the behavior patterns and contend against the sample Hunter Bot. In this set of the experiments, all bots are training using the 8000 training samples data recorded from the Hunter Bot.

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 one of the bots reaches a maximum score of 25.

During the battles, the scores, updated in intervals of 25 seconds, indicate fighting competency of the robots. For benchmark purpose, we run the game for ten times and record the average scores obtained by the bots.

- **Experiment 1: Battle between Hunter and SGNN Bot**

This experiment examines the competency of the bot created using the baseline SGNN (without pruning). As shown in Figure 3.8, the SGNN Bot can achieve a respectable level of performance but its scores are always 2 to 3 points lower than those of Hunter.

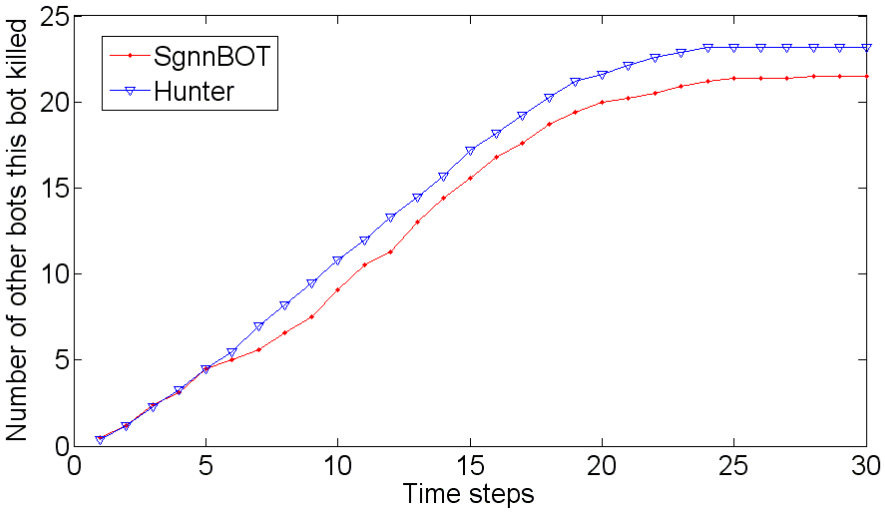

Figure 3.8: Performance of SGNN Bot fighting against Hunter.

- **Experiment 2: Battle between Hunter and Pruned SGNN Bot**

This experiment examines the competency of the bot created using SGNN with pruning. As shown in 3.9, after applying SGNN pruning, the new bot largely maintains the same level of performance, which is still inferior to Hunter.

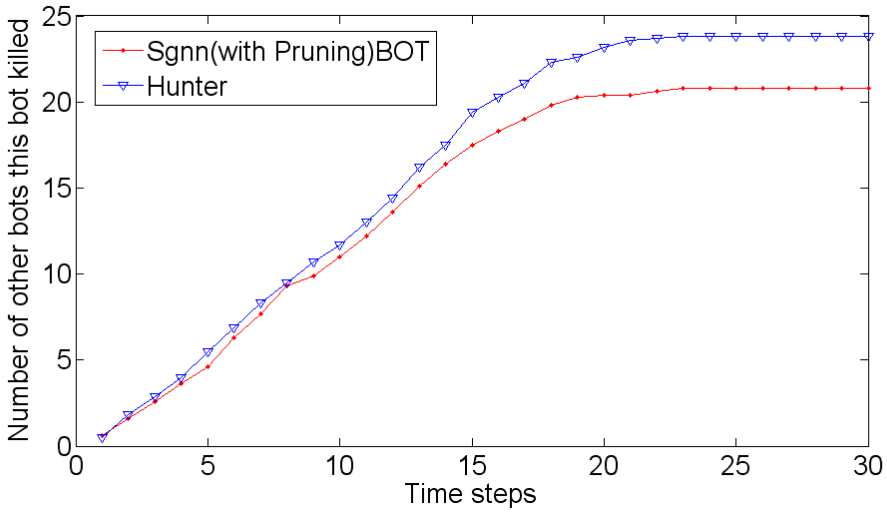

Figure 3.9: Performance of SGNN Bot (with pruning) fighting against Hunter

Table 3.5: SGNN rule examples of learning Hunter in UT2004.

No.	Att_1	Att_2	Att_3	Att_4	Att_5	Att_6	Att_7	Att_8	Att_9	Att_{10}	*Action*
R_1	1	1	1	0.0017	0.0332	0.0035	0.1311	0.1049	0.4374	1	A_1
R_2	1	0	1	0	0.0512	0.6354	0.1677	0.0562	0.4520	1	A_2
R_3	0	0	1	1	0.2795	0.9933	0.2130	0.0337	0.4528	1	A_3
R_4	0	0	1	0	1	1	0	1	0.2067	1	A_4
R_5	0	0	1	0	0	1	1	0.0717	0.4666	1	A_5
R_6	0	0	1	0	0	0	1	1	0.4286	1	A_6
R_7	0	0	1	0	0	0	0	1	0.1709	1	A_7
R_8	0	0	1	0	0	0	0	0	0.3718	1	A_8

- **Experiment 3: Battle between Hunter Bot and SGNN Rule Bot**

As mentioned before, SGNN, as tree structure neural network, is not suitable to be used in real-time games. Although it can handle small sets of training data, it does not scale up to the requirement of online learning and adaption due to its continuously expanding architecture. In view of this limitation, rules are further extracted from SGNN network to simplify the computation incurred.

In this test, a multi-class SGNN system is constructed by dividing the training data into eight parts according to their reference output actions. In each sub-system, the number of extracted rules equals to the number of child node of the root, i.e. the center node. Each center node could be extracted as a rule which demonstrates the node's attributes as conditions and the node's reference output as the result. Table 3.5 shows the sample set of rules, which are extracted from the multi-class system and using the class prediction approach mentioned in Section 3.2.2.

For the ease of presentation, the ten state attributes are denoted by Att_1 to Att_{10} and the eight behaviors are denoted by A_1 to A_8. According to the rule extraction method, each rule indicates a center weight in the self-generating neural network and its corresponding output. During online operation in the game environment, the actions of the SGNN Rule Bot can be determined through these rules based on the Euclidean distance similarity measure.

Before on-line testing, we conduct a benchmark evaluation on this multi-class method using the same training data and testing data as in Section 3.4.1. The SGNN multi-class method achieves the accuracy of 81.75%. This result shows that the rules extracted by this method could maintain a good accuracy level while simplifying the exuberant tree structure of SGNN into simple rules. After that, we similarly examine the competency of the bot created based on the class rules of SGNN. As shown in Figure 3.10, the SGNN Rule Bot achieves a slightly lower level of performance, widening the gap between the sample bot and the learned bot.

3.5 Summary

This chapter has shown how SGNN can be used to build autonomous players by learning the behaviour patterns of sample bots in a first person shooting game,

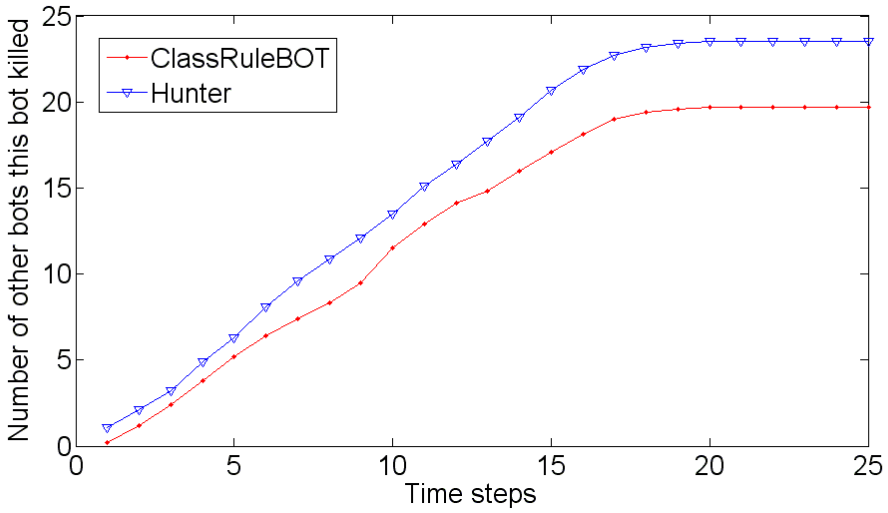

Figure 3.10: Performance of SGNN Rule Bot fighting against Hunter

known as Unreal Tournament 2004 (UT2004). We have conducted benchmark experiments to examine the SGNN model and the pruning method in the aspects of generalization capability, learning efficiency, and computational cost. Our experiments demonstrate that the pruning method effectively reduces the number of neurons in SGNN, while maintaining the learning accuracy. For the application of behavior rule learning, a novel rule extraction method is further proposed to extract useful knowledge from SGNN and translate them into symbolic rules. The empirical experiment based on the UT2004 also shows that the SGNN based bot could achieve a respectable level of performance when fighting against the sample bot. The bots created based on pruned SGNN and SGNN rules also successfully learn similar behavior patterns from the sample bot, and achieve good performance.

Chapter 4

Learning Behavior Models from User Patterns by FALCON

Modeling of Non-player characters (NPC) is crucial for the success of commercial games as it improves the playability of games and the satisfaction level of players. Especially, in first person shooting games (FPS), autonomous NPC modeled by machine learning techniques make games more challenging and enjoyable [158].

In Chapter 3, we have investigated the Self-Generating Neural Networks (SGNN) in behavior learning task. In this chapter, we study another specific class of self-organizing neural networks, namely Fusion Architecture for Learning and COgnition (FALCON).

Fusion Architecture for Learning and COgnition (FALCON) [147, 166] is a three-channel fusion Adaptive Resonance Theory (ART) network [149] that incorporates temporal difference methods [142, 160] into Adaptive Resonance Theory (ART) models [23, 22] 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. It has

displayed superior learning capabilities, compared with gradient descent based reinforcement learning systems in various benchmark experiments [151, 166].

This chapter investigates how FALCON can be used to build autonomous players by learning the behaviour patterns of sample bots in a first person shooting game, known as Unreal Tournament 2004 (UT2004). We conduct benchmark experiments to compare FALCON with SGNN (presented in Chapter 3) as they have the flexibilities of self-growing architecture and fast supervised learning manner in common. The terms of comparison, such as the number of internal nodes/rules created and the time taken for learning, are selected in view of their self-growing mechanisms, and thus not suitable for other models with a fixed architecture. Comparison between FALCON with other traditional neural network models in other domains has been reported in some previous works [148] and [151]. Our benchmark experiments show that, comparing with SGNN, FALCON learns faster and produces a higher level of generalization capability with a much smaller set of nodes. Online testing of NPC in the Deathmatch scenario also confirms that FALCON Bot produces a similar level of fighting competency to the Hunter Bot, which is not matched by bots based on SGNN.

4.1 Introduction

FALCON (Fusion Architecture for Learning, COgnition, and Navigation) was first proposed by Tan [147] based on ART Theory (adaptive resonance theory) [23] as a natural extension of self-organizing neural network architecture. Compared with traditional neural network, this neural architecture is superior in terms of integrating reinforcement learning and multi-channel pattern mappings in an online and incremental manner. FALCON network learns cognitive codes encoding mappings

of multimodal input patterns involving sensory input, actions, and rewards. By using competitive coding as the underlying adaptation principle, the network dynamic encompasses a myriad of learning paradigms, including unsupervised learning, supervised learning, as well as reinforcement learning.

Although various models of ART have been widely applied to pattern analysis and recognition tasks, there have been very few attempts to used ART-based networks for building autonomous systems. In view that, FALCON has good characteristics in forms of generalization capability, learning efficiency, and computational cost, in this research, we study how this class of self-organizing neural network can be used to build the autonomous agents.

The rest of this chapter is organized as follows. Section 4.2 introduces the FALCON architecture and the associated rule learning algorithm and action selection policy. Section 4.3 examines FALCON with offline testing and online testing in Unreal Tournament game. The comparison between FALCON and SGNN (in chapter 3) is provided through benchmark experiments. We also provide a discussion of utility of FALCON rules. The final section concludes and discusses limitations and future work.

4.2 FALCON Architecture and Learning

Fusion Architecture for Learning, Cognition, and Navigation (FALCON) employs a three-channel architecture (Figure 4.1) comprising a category field F_2 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 feedback field F_1^{c3} for representing the reward value. The dynamics of FALCON based on fuzzy ART operations [24][146], is described below.

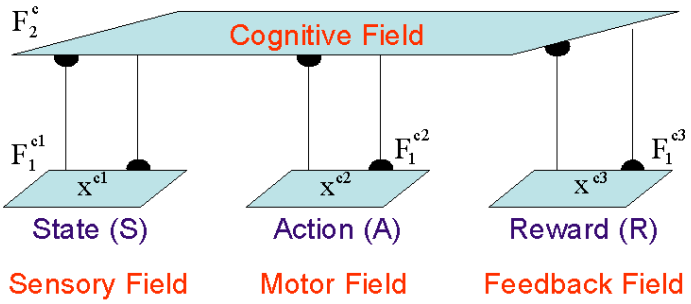

Figure 4.1: FALCON architecture

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 $k = 1, ..., 3$.

Given the state vector $\mathbf{S}$, the system performs code competition and selects an action based on the output activities of action vector $\mathbf{A}$. The detailed algorithm is presented below.

4.2.1 Supervised Learning

In the supervised learning mode, FALCON learns an action policy which maps directly from states to desired actions. Given the state vector **S** and an action vector **A**, the activity vectors are set as $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{R} = (1, 0)$. FALCON then performs code activation to select a category node J in the F_2 field to learn the association between **S** and **A**. The detailed algorithm is presented as follows.

Code activation: A bottom-up propagation process first takes place in which the activities (known as choice function values)of the category nodes in the F_2 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_2^{c2}, and F_3^{c3}, respectively), for each F_2 node j, the choice function T_j is computed as follows:

$$T_j^c = \sum_{k=1}^{K} \gamma^{ck} \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_j^{ck}|}{\alpha^{ck} + |\mathbf{w}_j^{ck}|}, \qquad \text{(Eq. 4.1)}$$

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

$$T_J^c = \max\{T_j^c : \text{for all} \quad F_2^c \quad \text{node} \quad j\}. \qquad \text{(Eq. 4.2)}$$

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 node 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 node J meets its vigilance criterion

$$m_J^{ck} = \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{x}^{ck}|} \geq \rho^{ck}. \qquad \text{(Eq. 4.3)}$$

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_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 ρ^{-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:

$$\mathbf{w}_J^{ck(new)} = (1 - \beta^{ck})\mathbf{w}_J^{ck(old)} + \beta^{ck}(\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck(old)}). \qquad \text{(Eq. 4.4)}$$

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.

Rule creation: Our implementation of FALCON maintains ONE uncommitted node the F_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_2^c field. FALCON thus expands its network architecture dynamically in response to the input patterns.

4.2.2 Action Selection

Given a state vector $\mathbf{S}$, FALCON selects a category node J in the F_2 field which determines the action. For action selection, 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, FALCON searches for the cognitive node which matches with the current state using the same code activation and code competition processes according to Eq. 4.1 and Eq. 4.2.

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

$$\mathbf{x}^{c2(new)} = \mathbf{x}^{c2(old)} \wedge \mathbf{w}_J^{c2}. \qquad \text{(Eq. 4.5)}$$

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

$$x_I^{c2} = \max\{x_i^{c2(new)} : \text{for all} \quad F_1^{c2} \quad \text{node} \quad i\}. \qquad \text{(Eq. 4.6)}$$

4.3 Experiments

Experiments in this section are conducted in the same environment as in Chapter 3, for the same behavior learning task. The training data is collected from Unreal Tournament 2004 (UT2004) game environment. The data recorded the behavior of a sample bot, called "Hunter", provided by UT2004. We collect 28000 sample data as training data with 10 attributes shown in Table 3.3. The behaviors are shown in Table 3.2. The ten attributes above are denotes as Att_1 to Att_10 respectively. The outputs consist of eight possible actions: change to better weapon, engage, stop shooting, response to hit, pursue, walk, grab item, and get medical kit, denoted by A_1 to A_8 respectively.

To evaluate the effectiveness of FALCON and SGNN in learning NPC, we first conduct benchmark experiments based on off-line learning to compare their performance in terms of learning time, generalization capability and computational cost. Online testing of bots are subsequently conducted, wherein we investigate the competency of the new bots when fighting against the sample bot which they learn from.

4.3.1 Off-line Testing

We first conduct empirical experiments to evaluate the performance of SGNN and FALCON in off-line learning. The data set consists of a total of 8000 training samples and 8000 test samples generated by the Hunter Bot. By training on a varying number of training examples, we compare the generalization capability of SGNN and FALCON on an equivalent number of test samples. We also measure their efficiency in terms of the number of internal nodes/rules created and the time taken for learning.

In our experiments, SGNN and FALCON use a standard set of parameter values. For SGNN, we adopt $\xi = 0$ and $\eta = 1.5$. For FALCON, we adopt the parameter setting as follows: choice parameter α=(0.1, 0.1, 0.1); learning rate parameter β=(1, 1, 1); contribution parameter γ=(1, 0, 0); and vigilance parameter ρ=(1, 1, 0).

Figure 4.2 summaries the performance of the baseline SGNN (without pruning), SGNN with pruning, and FALCON, in terms of the classification accuracy on the test set. As the size of the training data increases, the accuracies of all three models converge to 100%. In general, SGNN with pruning achieves roughly the same accuracy level as SGNN without pruning. Comparing with SGNN, FALCON shows a faster rate of convergence by obtaining a higher accuracy with small data sets.

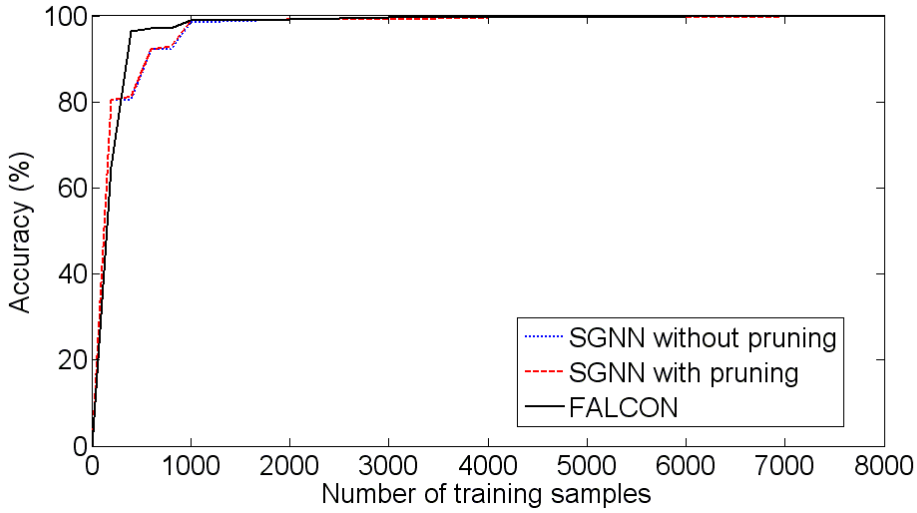

Figure 4.2: The accuracy of baseline SGNN, SGNN with pruning, and FALCON in classifying the test samples.

Figure 4.3 depicts the performance of the baseline SGNN (without pruning), SGNN with pruning, and FALCON, in terms of average numbers of neurons/nodes created during learning. We see that the pruning method greatly reduces the number of neurons in SGNN. However, the number of nodes created by FALCON is

significantly less than those of SGNN.

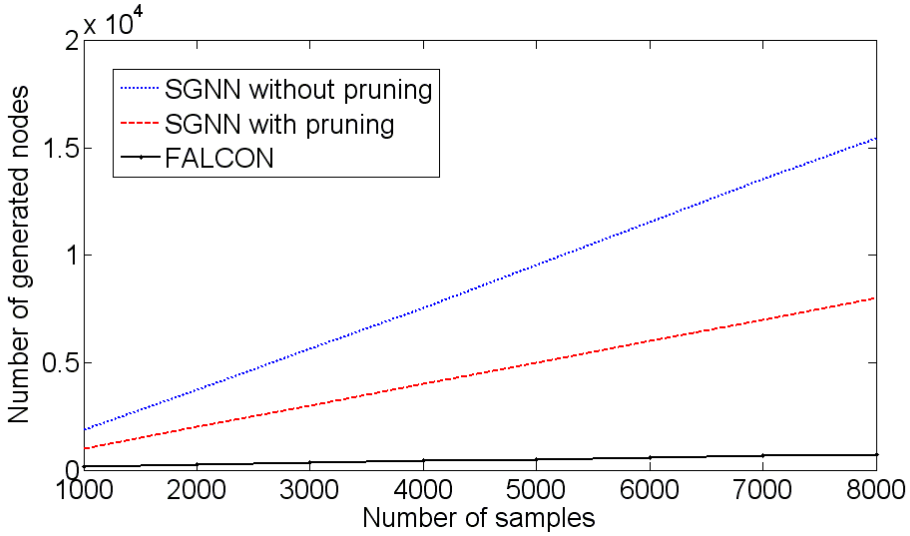

Figure 4.3: The number of nodes generated by baseline SGNN, SGNN with pruning, and FALCON.

Figure 4.4 shows the learning time taken by baseline SGNN (with pruning), SGNN with pruning, and FALCON. For SGNN with pruning, as the number of nodes is reduced, so is the time for learning. Moreover, the amount of time saved increases as the number of samples increases. Considering all aspects, the pruning method is proved to be effective for SGNN, as it effectively reduces the number of neurons and the learning time, while maintaining the learning accuracy. Partly because the number of nodes generated by FALCON is the least among these three systems, the required learning time is also significantly less than those of SGNN. This suggests that FALCON is clearly a more suitable candidate for real time learning and performance.

4.3.2 Online Testing of FALCON Bot

In this section, a series of experiments are conducted in UT2004 game environment to check if the bots created based on FALCON could learn the behavior patterns and

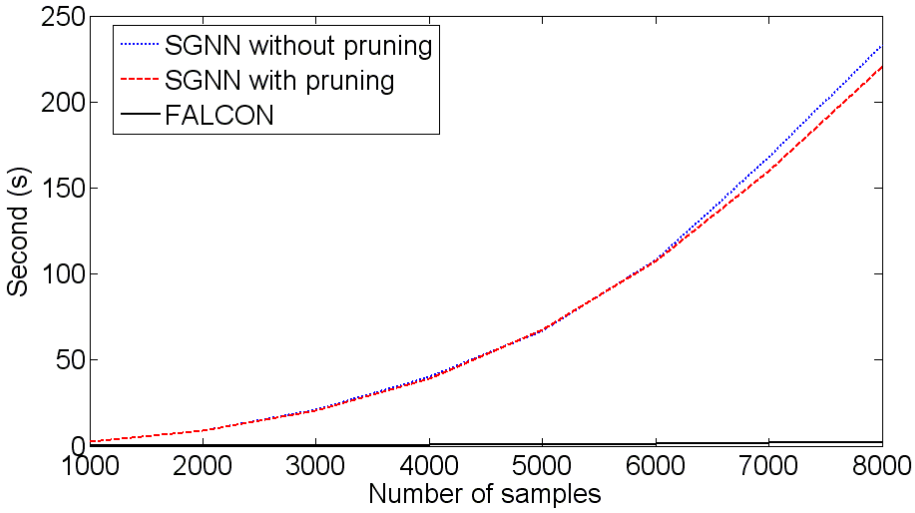

Figure 4.4: The learning time of baseline SGNN, SGNN with pruning, and FALCON

contend against the sample Hunter Bot. The FALCON Bot, trained using the same 8000 training samples data recorded from the Hunter Bot, consists of 647 behavior rules.

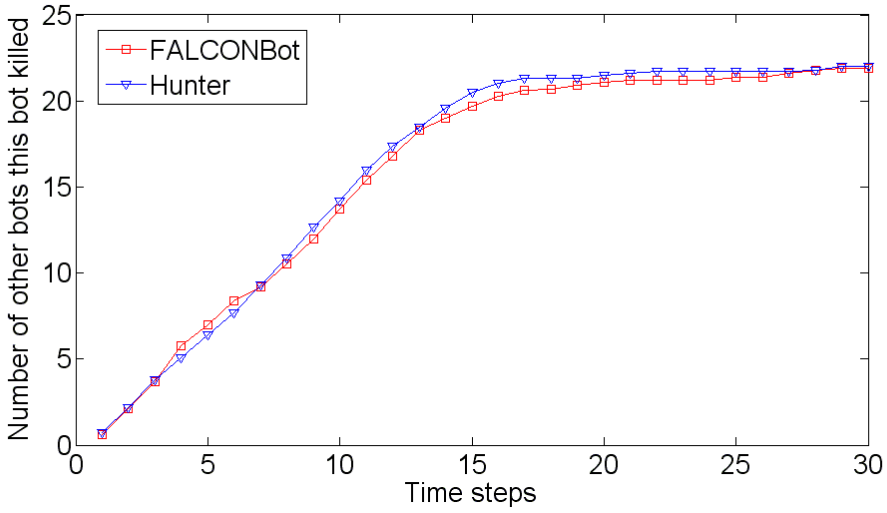

Figure 4.5: Performance of FALCON Bot fighting against Hunter

Table 4.1 summarizes the offline and online performance of FALCON, based on

Table 4.1: Testing results of FALCON rules with different vigilance settings

Vigilance Setting	Number of Cognitive Nodes	Offline Test Accuracy (%)	Average Game Score Difference	Standard Deviation
(1.00, 1, 0)	647	99.9	-0.05	3.84
(0.75, 1, 0)	32	100	0.50	2.90
(0.50, 1, 0)	11	100	-0.55	4.47
(0.25, 1, 0)	11	100	-0.35	3.87
(0.00, 1, 0)	9	100	0.45	3.96

different vigilance settings. The average game score difference and their standard deviation are calculated according to the final scores of FALCON Bot fighting against Hunter across twenty games. We see that a high vigilance of 1.0 for the State space produces 647 cognitive nodes (equivalence of rules). With minimal code compression, FALCON is able to classify 99.9% of the test samples correctly and achieve almost an identical level of fighting competency as Hunter Bot. Lowering the vigilance all the way to 0.0 reduces the number of cognitive nodes to 9, while maintaining the same level of offline and online performance. This shows that the match tracking mechanism of FALCON is able to produce maximal compression as well as generalization.

Figure 4.5 shows the scores of FALCON Bot fighting against Hunter averaged across ten games. In contrast to those of SGNN, we can clearly see that the fighting competency of FALCON Bot is almost identical to that of Hunter. This shows that FALCON has learned most, if not all, of the Hunter's knowledge perfectly. Comparing with bots created based on SGNN, FALCON Bot is thus obviously a much better learner in assimilating the behavior patterns from the Hunter Bot.

Table 4.2 shows the weight vectors of the nine cognitive nodes learned by FALCON. We see that the weight values of all boolean attributes take the values of 0

Table 4.2: Weight vectors of FALCON's cognitive nodes by learning Hunter in UT2004.

No.	Att_1	Att_2	Att_3	Att_4	Att_5	Att_6	Att_7	Att_8	Att_9	Att_{10}	*Action*
R_1	1	1	1	0	0	0	0	0	0.021-0.558	1	A_1
R_2	1	0	1	0	0	0	0	0	0.005-0.579	1	A_2
R_3	0	0	1	1	0	1	0	0	0.011-0.558	1	A_3
R_4	0	0	1	0	1	0	0	0	0.011-0.526	1	A_4
R_5	0	0	1	0	0	1	0	0	0.011-0.558	1	A_5
R_6	0	0	1	0	0	0	1	0	0.042-0.558	1	A_6
R_7	0	0	1	0	0	0	0	1	0.011-0.579	1	A_7
R_8	0	0	1	0	0	0	0	0	0.021-0.558	1	A_8

or 1, whereas the values of the only real-valued attribute are in the range of 0 and 1. Following a rule extraction procedure as used in ARTMAP system [25], these weight vectors may be converted into symbolic rules that are more interpretable. By applying weight quantization, the AgentHealth may be described in symbols, namely $VeryLow$, Low, $Medium$, $High$ and $VeryHigh$. The translated symbolic form of the FALCON rules is exemplified in Table 4.3. Comparing with the SGNN rules (Table 3.5), FALCON rules are close to the original rules of Hunter and are easy to interpret.

4.3.3 Discussion: Utility of FALCON Rules

In a dynamic environment, generalization of learning is very important as the generalization ability could enable the non-player characters to adapt to various environments involving multiple opponents/companions and different game maps. The first issue of generalization of learning is whether the NPCs built by imitative learning could adapt to unseen data correctly. The second consideration is whether the NPC can be built by concise and interpretable behavior knowledge.

Table 4.3: FALCON rules in symbolic form.

No.	*IF (Condition)*	*THEN (Behavior)*
R_1	$SeeAnyEnemy$ AND $AgentHealth(VeryLowToMedium)$ AND $HasAnyLoadedWeapon$ AND $HasBetterWeapon$ AND $CanRunAlongMedKit$	$ChangeToBetterWeapon$
R_2	$SeeAnyEnemy$ AND $HasAnyLoadedWeapon$ AND $AgentHealth(VeryLowToMedium)$ AND $CanRunAlongMedKit$	$Engage$
R_3	$HasAnyLoadedWeapon$ AND $IsShooting$ AND $LastEnemy$ AND $AgentHealth(VeryLowToMedium)$ AND $CanRunAlongMedKit$	$StopShooting$
R_4	$HasAnyLoadedWeapon$ AND $IsBeingDamaged$ AND $AgentHealth(VeryLowToMedium)$ AND $CanRunAlongMedKit$	$RespondToHit$
R_5	$HasAnyLoadedWeapon$ AND $LastEnemy$ AND $AgentHealth(VeryLowToMedium)$ AND $CanRunAlongMedKit$	$Pursue$
R_6	$HasAnyLoadedWeapon$ AND $IsColliding$ AND $AgentHealth(VeryLowToMedium)$ AND $CanRunAlongMedKit$	$Walk$
R_7	$HasAnyLoadedWeapon$ AND $AgentHealth(VeryLowToMedium)$ AND $CanRunAlongMedKit$	$GrabItem$
R_8	$HasAnyLoadedWeapon$ AND $AgentHealth(VeryLowToMedium)$ AND $CanRunAlongMedKit$	$GetMedicalKit$

In this research, the issues of generalization have been investigated to a certain extent through our simulations. For example, the behavior learning models have been evaluated in terms of the NPC's performance in real time and the number of behavior rules learned (see Section 3.4.1 for SGNN and Section 4.3.1 for FALCON).

Compared with SGNN, FALCON provides the additional flexibility in its learning efficiency by controlling the number of nodes to be learned. This is achieved by adjusting the vigilance parameter values used in the state, action and feedback fields. In this section, we discuss the effect on FALCON by varying the vigilance values. For simplicity, we assume that ρ^{c1}, ρ^{c2} and ρ^{c3} use the same value ρ.

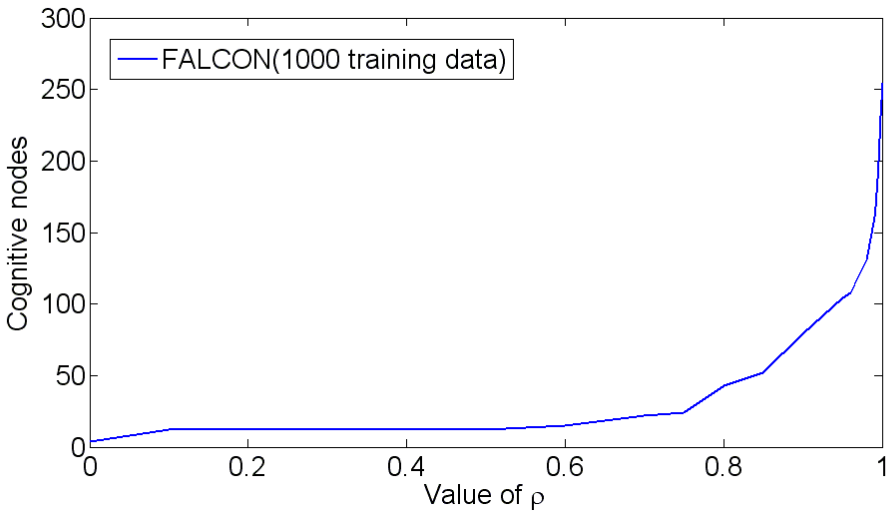

Figure 4.6: The number of nodes learned by FALCON as ρ increases.

During learning, the vigilance parameter affects the generation of nodes. Figure 4.6 shows that as ρ increases, FALCON becomes more vigilant and the number of nodes generated also increases. In general, the bot's performance could be better with a larger number of nodes, as additional rules may contribute in capturing detailed behavior patterns.

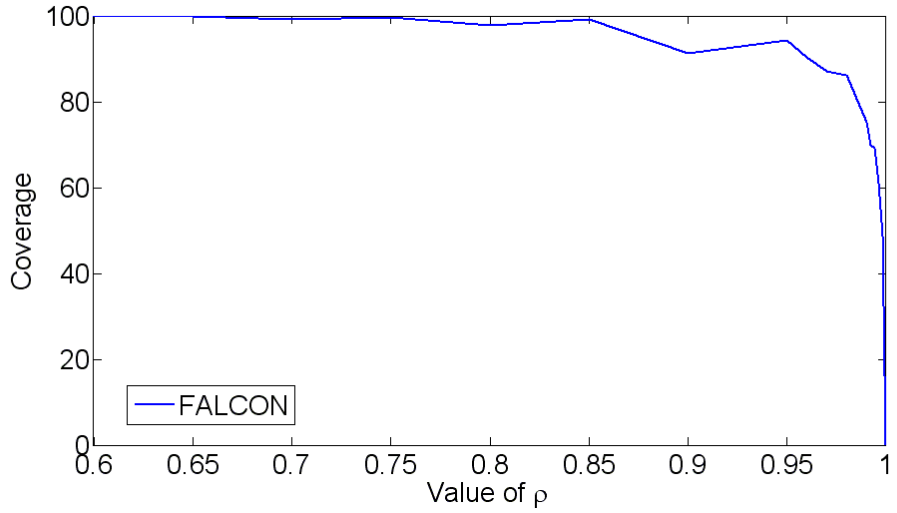

Figure 4.7: The utility of FALCON as ρ increases.

Besides affecting the number of nodes learned, the vigilance parameter also influences the decision making ability of the FALCON Bot during performance. To measure the usage of FALCON knowledge, we introduce an index called *coverage*, which computes the utility rate of FALCON rules as follows:

$$C = \frac{N_{match}}{N_{dc}}, \qquad \text{(Eq. 4.7)}$$

where N_{dc} denotes the number of decision cycles and N_{match} denotes the number of decision cycles in which the agent's state causes a FALCON node/rule to fire. During online testing, the FALCON Bot always tries to identify a desirable response given the current state. However, sometimes the bot may not find a rule satisfying the excessive vigilance. Therefore, a higher coverage indicates a better use of the FALCON's knowledge. Figure 4.7 shows that the coverage decreases as the vigilance ρ increases. To achieve a 100% coverage, a low vigilance ρ of less than 0.65 is needed.

From the above discussion, we can see that a high ρ value is a double-edged sword. In applications, the choice of vigilance parameter is made based on the

requirement of learning and performance. Therefore, the typical setting, as used in our experiments, is to use a high ρ for node/rule generation and a lower ρ in knowledge retrieval during performance.

4.4 Summary

This chapter has demonstrated how FALCON can be used to build autonomous players by learning the behaviour patterns of sample bots in a first person shooting game, known as Unreal Tournament 2004 (UT2004). The benchmark experiments show that, compared with SGNN (in Chapter 3), FALCON is superior in all aspects of generalization capability, learning efficiency, and computational cost. The online experiments based on the UT2004 also show that the FALCON based bot achieves a similar level of fighting competency as the sample bot. Compared with SGNN, FALCON is able to achieve a higher level of performance with a much more compact network structure and a much shorter learning time.

Chapter 5

Autonomous Behavior Learning of Non-Player Characters in Games

While the previous two chapters have focused on creating autonomous agents by learning from user patterns in an off-line mode, the focus of this chapter is on how the agents could learn, reflect and adapt in a real-time environment.

Reinforcement learning (RL) has been a promising approach to behavior learning for creating autonomous agents in games. Although learning by evaluative signals enables an agent to adapt in real time, an initial stage of exploration is required. While it is possible to incorporate prior knowledge by direct insertion, such knowledge has to be available in the first place. In contrast to reinforcement learning, imitative learning (IL) is an effective approach to acquiring behavior model by learning from opponent's actions. However, learning by imitation limits the agent's performance to that of its opponents. In view of their complementary strengths, this chapter 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 enhancing learning ability in terms of faster learning speed and better convergence. The NPCs build by DSL and MML also produce better performance comparing with those by traditional RL and IL methods.

5.1 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) [158].

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, a NPC without prior knowledge will perform poorly at the initial stage as they have to spend substantial time in exploring and learning the environment. 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 through either encoding domain knowledge in the learning architecture [129] [20], adding prior knowledge as a rule base [138], or using an added-on module to provide prior knowledge [35] [97]. 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 [43]. 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 chapter 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) [147, 166]. 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.

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 chapter is organized as follows. Section 5.2 reviews the related works. Section 5.3 defines the problems addressed by this work. Section 5.5 introduces the learning model and presents the methods of DSL and MML. Section 5.6 reports the learning tasks and the experiments. Finally, section 5.7 concludes and discusses the future work.

5.2 Related Works

In the past years, NPCs in computer games have advanced significantly in terms of their behavior modeling. For commercial games, the most commonly used method is via rule based system (RBS) approach, in which the behavior patterns are constructed primarily based on a set of If-Then rules representing the antecedences and consequences [123]. Once the observations of NPCs fulfil the conditions stated in the antecedence clause, the corresponding action as stated in consequence clause is taken. Although RBS supports a straightforward way of designing NPC behavior models, the manual way of specification does not scale well to complex environment due to the difficulty of deriving concise and effective rules.

To improve upon the RBS approach, many other techniques have been put forward, such as finite state machine (FSM) [170], fuzzy logic [7], and Bayesian method [9]. Finite state machine (FSM) models a finite number of states, actions and transitions rules between these states. As valid input events generate the transitions, FSM can move from one state to another. FSM has been widely applied in commercial games providing more complex transitions than RBS. Fuzzy logic supports rules reasoning and can infer conclusions based on the facts with uncertainty. In addition, it is possible to design rules in linguistic form or transferred easily from human knowledge. Bayesian method allows inference among Bayes's rule which model the prior probabilities of taking specific actions. Decision making is done based on calculating the action related probabilities. The above techniques have since been widely applied in commercial games. However, they are as limited in learning ability and the rules cannot evolve in real time during game-playing.

In view of the limitations of the rule-based approaches, learning based techniques, such as evolutionary algorithm, imitative learning, and reinforcement learning, have

attracted much attention for behavior modeling. Inspired by biological processes, evolutionary algorithm performs well in optimization and solution approximation. However, as long computing time is needed in evolving the generations, it is not suitable for modeling NPCs in real time [156] [164].

Imitative learning is a popular method to create NPCs by mimicking the behaviour patterns of human beings in order to achieve the human like behaviors [172] [14] [43]. The imitation based learning enables fast learning and is capable of acquiring complex behavior patterns. However, different from reinforcement learning, 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 of the .

In the recent years, reinforcement learning (RL) has been applied 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 [94] [153]. The most popular way to initialize the learning agent is with human knowledge. For instance, Unemi improves the performance of reinforcement learning by introducing relatively simple human knowledge such as intrinsic behaviors [157]. Dixon *et al.* introduced a general approach to incorporate prior knowledge with the reinforcement learning and achieve reduced learning time for mobile-robot and grid- world domains [35]. However, the prior knowledge are embedded in the controller and not represented in the same form as the learned knowledge and cannot be further modified in real-time learning. Later, Shapiro *et al.* [129] applied background knowledge in ICARaUS architecture to create a knowledgeable agent in order to perform better than unknowledgable agent. The hard coded background knowledge can speed up learning.

However, again, it cannot be used in subsequent exploration. Kengo *et al.* enhanced the reinforcement 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 [67]. In the chosen game scenarios, the prior knowledge needs to be designed beforehand by considering the specific problem domain. Again, once the knowledge is applied, they are fixed and can't be further adapted. Moreno *et al.* applied a control module to combine the expert knowledge with the reinforcement information learnt to date. Prior knowledge is decided by the experts recommending the actions to be carried out in a number of representative positions within the environment [97]. The control module decides whether the recommended action or the information learnt to date is suitable to the current situation. However, the prior knowledge is hard coded and obtained subjectively instead of learning. These knowledge are unable to be updated through the dynamic learning process. Framling introduces a reinforcement learning model with pre-existing knowledge [45]. 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 an universal architecture, specific heuristic rules are still required for proposing the pre-existing knowledge.

5.3 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): The 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): The 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): The 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

$$\mathcal{F} : \mathcal{S} \longrightarrow \mathcal{A}, \qquad \text{(Eq. 5.1)}$$

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

$$\mathcal{V} : \mathcal{S} \times \mathcal{A} \longrightarrow \mathcal{R} \qquad \text{(Eq. 5.2)}$$

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.

5.4 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.

5.4.1 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.

5.4.2 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.

5.4.3 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.

5.5 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) [147, 166]. FALCON is a three-channel fusion Adaptive Resonance Theory (ART) network [149] that incorporates temporal difference methods [142, 160] into Adaptive Resonance Theory (ART) models [23, 22] 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 [149], 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 4.1, 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 [24] [146], has been presented in Section 4.2 for supervised learning. This section shows how FALCON can be used for integrating IL and RL.

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 $\mathbf{F} = (\mathbf{w}_j^{c1}, \mathbf{w}_j^{c2}, \mathbf{w}_j^{c3})$ representing an association among the template state vector ($\mathbf{w}_j^{c1}$, the template action vector $\mathbf{w}_j^{c2}$) and the template reward vector $\mathbf{w}_j^{c3}$.

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 **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 **S** and the action vector are presented simultaneously in a predicting mode, FALCON will read out the reward vector **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.

5.5.1 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 5.1 shows the pattern configuration of FALCON in imitative learning. Given a pair of state vector **S** and action vector **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 (Eq. 4.1) and (Eq. 4.2), to select a category node J in the F_2^c field. Once the template matching condition is satisfied, the category node J undergoes template learning, wherein it learns the association between **S** and **A** (Eq. 4.4).

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

Table 5.1: 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.

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 5.1.

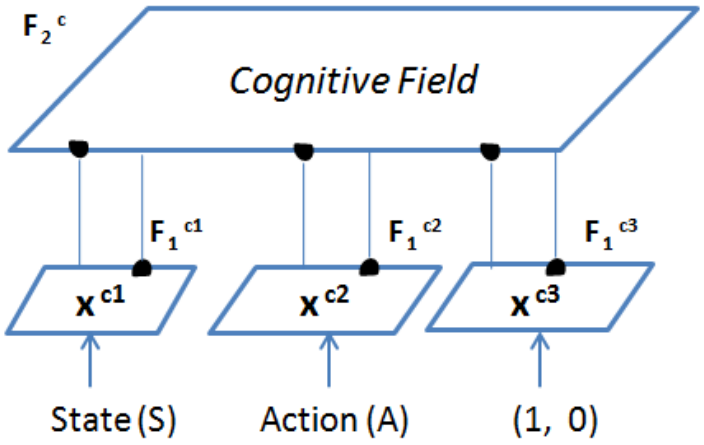

Figure 5.1: FALCON in imitative learning.

For action selection, FALCON receives a state vector **S** in performing mode and selects a category node J in the F_2^c 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 [148], FALCON searches for the cognitive node which matches with the current state using the code activation and code competition processes according to equations (Eq. 4.1) and (Eq. 4.2).

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

$$\mathbf{x}^{c2(new)} = \mathbf{x}^{c2(old)} \wedge \mathbf{w}_J^{c2}. \quad \text{(Eq. 5.3)}$$

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

$$x_I^{c2} = \max\{x_i^{c2(new)} : \text{for all} \quad F_1^{c2} \quad \text{node} \quad i\}. \quad \text{(Eq. 5.4)}$$

5.5.2 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 5.2 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 $\mathbf{S}$, 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

Table 5.2: Reinforcement learning by FALCON with direct code access.

1. Initialize the TD-FALCON network.
2. Sense the environment and formulate a state vector **S**.
3. Select the action a with the maximal Q(s,a) value by presenting the state vector **S**, action vector **A**=(1,...1) and the reward vector **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 $\Delta Q(s, a) = \alpha \text{TD}_{err}$.
6. 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:
7. Update the current state by s=s'.
8. Repeat from Step 2 until s is a terminal state.

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 [148] is summarized in Table 5.2.

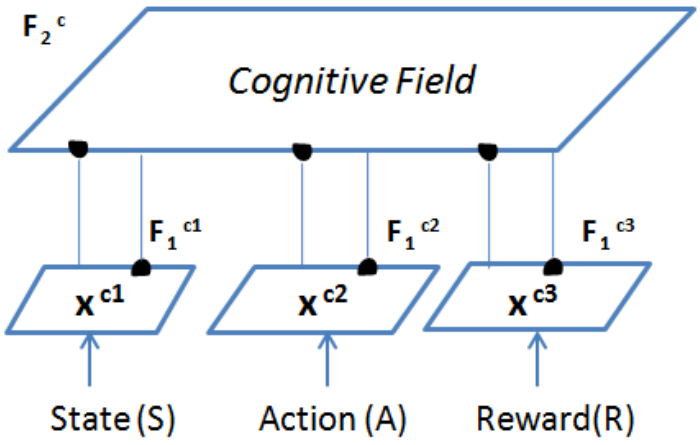

Figure 5.2: TD-FALCON in reinforcement learning.

A typical Temporal Difference (TD) equation for iterative estimation of value

functions Q(s,a) is given by

$$\Delta Q(s,a) = \alpha TD_{err} \quad \text{(Eq. 5.5)}$$

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

$$\Delta Q(s,a) = \alpha TD_{err}\left(1 - Q\left(s,a\right)\right). \quad \text{(Eq. 5.6)}$$

where $TD_{err} = r + \gamma \text{max}_{a'} Q(s',a') - Q(s,a)$, of which r is the immediate reward value, $\gamma \in [0,1]$ is the discount parameter, and $\text{max}_{a'} Q(s',a')$ denotes the maximum estimated value of the next state s'. It is important to note that the Q values involved in estimating $\text{max}_{a'} Q(s',a')$ 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 $r + \gamma \text{max}_{a'} Q(s',a')$ over time. By incorporating the scaling term $1 - Q\left(s,a\right)$, 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.

5.5.3 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.

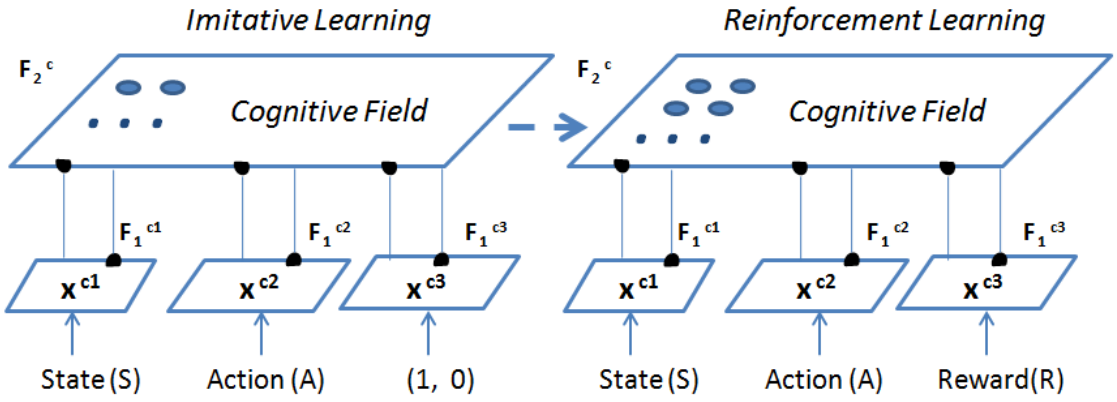

Figure 5.3: The Dual-Stage Learning procedure.

The process of the Dual-Stage Learning strategy is illustrated in Figure 5.3. During imitative learning, given 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 5.1 and Table 5.2 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 (Eq. 4.4) and the corresponding state-action value q_J will be estimated again and revised by equation (Eq. 5.6).

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

$$m_J^{c3} = \frac{|\mathbf{x}^{c3} \wedge \mathbf{w}_J^{c3}|}{|\mathbf{x}^{c3}|} = q_J < \rho^{c3}. \quad \text{(Eq. 5.7)}$$

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:

$$min\{Q_n(s, a) = q_n\} \geq \rho^{c3} \quad (n = 1, 2, ..., N). \quad \text{(Eq. 5.8)}$$

5.5.4 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 5.4. 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 **S**, **A**, and **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 summarized in Table 5.3, TD-FALCON first decides between the imitative learning mode and the reinforcement learning mode and then activates the learning of the corresponding input patterns.

Reinforcement learning performs regularly upon receiving the reward signals, and 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 $\mathbf{x}^{c3} = \mathbf{R} = (Q, \overline{Q})$.

5.6 Benchmark Evaluation

5.6.1 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 3.3 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

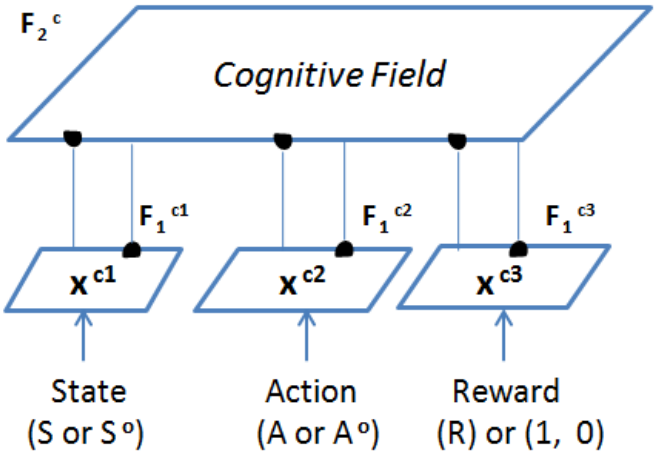

Figure 5.4: The Mixed Model Learning strategy.

Table 5.3: 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 $\mathbf{S}$, action vector $\mathbf{A}$=(1,...1) and the reward vector $\mathbf{R}$=(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 $\mathbf{S}$, 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.
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.

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 [56], is available to developers for building agents for the UT environment. This means the developers can implement their own agents (or bots) using any specific algorithm and run them in UT.

Figure 3.4 (adapted from [56]) 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.

5.6.2 Behavior Learning Tasks

There are in total eight types of behaviors designed for the agent (as shown in Table 3.2). Upon receiving the information from the environment, the agent is designed to make responses by choosing one of the eight behaviors based on ten state attributes (shown in Table 3.3). 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 3.4.

5.6.2.1 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 behaviour (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.

5.6.2.2 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 3.2 and Table 3.3 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.

5.6.2.3 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 3.2 and Table 3.3 respectively, the MML Bot is also expected to show a higher level of fighting competency than its opponent.

5.6.3 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.

5.6.3.1 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.

5.6.3.2 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.

5.6.3.3 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.

5.6.3.4 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.

5.6.3.5 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.

5.6.3.6 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.

5.6.4 Results

Figure 5.5 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 5.5, 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 5.6 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 5.6, 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 5.5, 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 5.6 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 5.4.

For evaluating learning efficiency, Table 5.5 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.

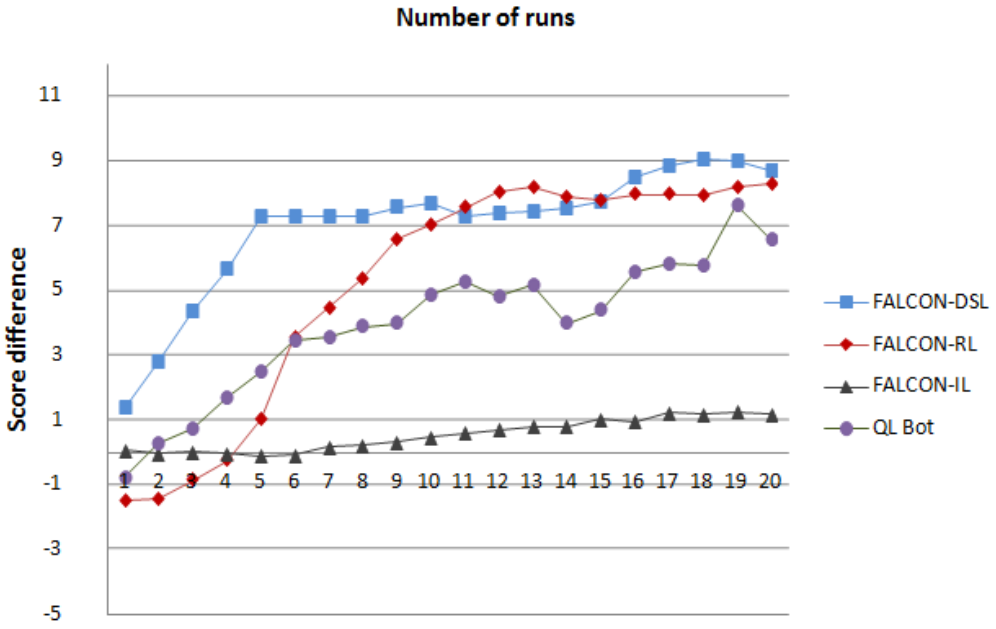

Figure 5.5: The score difference between the learning bots and Hunter Bot.

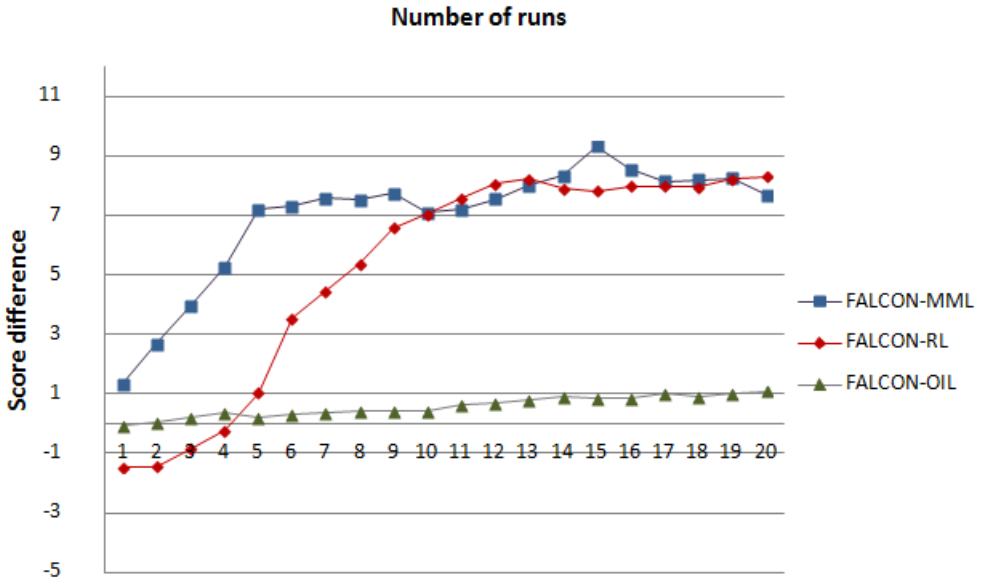

Figure 5.6: The score difference between the learning bots and Hunter Bot.

Table 5.4: 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 5.5: 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

5.7 Summary

This chapter has presented a computational model unifying two popular learning paradigms: imitative learning, and reinforcement learning, based on a class of self-organizing neural networks called Fusion Architecture for Learning and COgnition (FALCON). Specifically, two hybrid learning strategies known as Dual-Stage Learning (DSL) and the Mixed Model Learning (MML) are proposed to realize integration of the two different learning paradigms within one designed framework. DSL and MML have been used to creating the 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 performance comparing with NPCs using the pure reinforcement learning method and the pure imitative learning method.

Moving forward, we aim to enhance the capability of the learning NPCs by incorporating human factors such as emotions and personalities. By integrating affective and motivations with learning methods, the performance of NPC could be more realistic and believable. These, we reckon, will greatly increase the playability of computer games.

Chapter 6

Cognitive Regulated Affective Architecture: A Biologically-Inspired Approach

The previous chapters have presented techniques to improve the NPCs learning and adaption abilities. In order to meet the requirement of NPCs as "believable" and "realistic" agents, this chapter focuses on how to enhance the learning agent by developing emotion models.

Although various emotion models have been proposed based on cognitive appraisal theories, most of them focus on designing specific appraisal rules linking appraisal parameters to emotion instances and do not focus on how the information transfers between cognitive and affective modules. Based on the appraisal theory and the biological basis of emotion, this chapter proposes a biologically-inspired neural model called Cognitive Regulated Affective Architecture (CRAA), which shows how cognitive and affective elements may cooperate to give rise to cognitive regulated emotion responses.

This model is proposed by taking the following positions: (1) cognition and emotion are two separate but interacting systems; (2) emotion is generated through the interaction among multiple neural networks and regions in human brains; and

(3) emotion responses are a consequence of the cognitive appraisal of both the internal mental states and the signals generated by a cognitive decision making system connected to external environment. Comprising a Cognitive Network, an Appraisal Network, and an Affective Network, CRAA simulates the functions and interactions of prefrontal cortex (PFC), anterior cingulate cortex (ACC), and amygdala (AMYG) in human brains and provides an integrated framework which combines emotion appraisal with cognitive decision making in a distributed multi-network architecture.

In our implementation, both the Cognitive Network and Affective Network are built based on a class of self-organizing neural networks called Adaptive Resonance Theory (ART). Specifically, the Cognitive Network performs decision making and reinforcement learning in a real-time environment, whereas the Affective Network learns the associations from appraisal parameters to emotion instances. The Appraisal Network serves as the link between the Cognitive Network and the Affective Network in translating cognitive information to appraisal parameters. This model has been evaluated in a first person shooting game known as Unreal Tournament. Comparing with emotionless NPC, affective NPC obtains a significantly higher user ratings in terms of the game's playability and interestingness. Comparison with a well-known emotion model also shows the strengths of the CRAA model in producing fine-grained combinatory emotion patterns.

6.1 Introduction

Emotion is a complex phenomenon involving human being's psychological and physiological responses while interacting with the environment. While it remains a

challenge to explain the process of emotion generation as part of the "person-environment relationship", the element of emotion has been found useful in many application domains, especially in designing social robots and virtual characters. For example, in a simulated world, modeling of emotion is very crucial for developing a "realistic" and "believable" agent [127], where realistics defines how human-like the agent is [16] and believable refers to how convincing the agent is from the view of the users [13]. Specifically, modeling of emotion could enrich agents with lively facial expressions and behaviors, presenting motivated responses to their environment and intensifying their interactions with human users. Also, interactive agents with emotion modeling capability could form a better understanding of their user's moods and preferences and can adapt themselves to the user's needs [39]. In essence, the presense of emotion provides agents "the illusion of life" [13], making them human-like and appealing.

A key challenge in emotion modeling is to understand and explain the mechanism of emotion. In fact, this topic has been studied over a long period. Early psychological theories of emotion, such as the James-Lang Theory, the Cannon-Bard Theory, and the Two-Factor Theory, focused on how emotion arises. Despite the differing views, it is generally agreed that emotion is the combinative outcome of numerous internal factors and external circumstances. Lately, several evolutionary theories argue that emotion is a developed mechanism belonging to advanced species as their solutions for adaptation problem, in order to survive under a natural selection environment [137] [37]. These theories focus on the physical displays of emotions and they believe that emotions express a person's appraisal of a person-environment relationship which may demonstrate a harm or benefit for the individual. On the other hand, the cognitive theories argue for the role of cognitive activity in emotion generation, since the occurring and intensity of emotion are controlled through

cognitive judgement and evaluation [108] [137] [126]. Meanwhile, the neurobiological theories point to the phenomenon that the occurring of emotions relates to distributed neural activations in human brains [112]. Specifically, numerous experiments by functional magnetic resonance imaging (fMRI), electroencephalography (EEG), electromyography (EMG), and skin conductance (SC) demonstrate that the mechanism of human emotion is highly associated with cognition and processed by multiple regions in human brains [112].

In the recent decades, "appraisal theories" has become the leading theory for modeling emotion, which states that a person's emotion is his/her personal assessment of "person-environment relationship" based on events. Consequently, many computational emotion models have been proposed based on appraisal theories. For example, EM [101], FLAME [38], FAtiMA [34], and ALMA [48] are proposed based on the OCC theory of emotion [108]; WASABI [15], and PEACTIDM [90] were inspired by Scherer's theory [126]; and THESPIAN [131] is proposed based on Smith and Lazarus's appraisal theory [137]. Most of these models focus on designing appraisal rules, which require certain appraisal parameter to be defined beforehand and evaluated in a subjective way [15] [93] [34]. The others focus on logical reasoning about the eliciting factors of emotions [38] [90] [131] [140]. Notably, these models were proposed based on psychological grounding aiming at simulating real human emotion process. However, most of these models rely on the measurement of appraisal parameters and do not focus on how the information transfers between cognitive and affective modules. While some of them involve cognitive processes in emotion, there is a lack of a concrete structure to realize the intrinsic interactions between cognition and affection.

Taking the view that the mechanism of emotion, closely associated with cognition, is the result of the collaborative emergent behaviour of multiple neural systems

in human brains, this chapter proposes a biologically-inspired neural model called Cognitive Regulated Affective Architecture (CRAA) based on the appraisal theory, cognitive regulated emotion theory [104], and the Adaptive Resonance Theory (ART) [27]. It can be seen as a simplification of the cognitive-affective control circuit suggested by Pessoa [112]. The model provides an integrated framework, comprising a Cognitive Network for decision making, an Affective Network for emotion response generation and an Appraisal Network for linking cognitive parameters to emotion appraisal. The three networks emulate the functions of three neural centers in the human brains, namely the prefrontal cortex (PFC), the anterior cingulate cortex (ACC) and the amygdala (AMYG) in human brain respectively. Both the Cognitive Network and the Affective Network are modelled based on a class of self-organizing neural models known as the Adaptive Resonance Theory (ART). Specifically, the Cognitive Network is a fusion ART network which performs situational assessment, decision making, and reinforcement learning through interacting with the environment in which the agent is situated, the Affective Network is a supervised fusion ART network which learns the associative mapping from appraisal parameters, such as expectation, desirability, attribution, power, and match, to emotion instances. The Appraisal Network, a specialized network, in turn serves the linkage between the Cognitive Network and Affective Network for translating cognitive information to emotion appraisal parameters.

The proposed CRAA model has been evaluated in a first person shooting game known as Unreal Tournament [158]. Comparing with emotionless NPC, affective NPC obtains a significantly higher user ratings in terms of the game's playability attributes, including cognitive absorption, social presence and enjoyment. Comparison with a well-known emotion model also highlights the strengths of the CRAA model in producing fine-grained combinatory emotion patterns.

The rest of this chapter is organized as follows. Section 6.2 reviews the related work. Section 6.3 studies the neural substrates of emotion in brains. Section 6.4 explains our approach and design principles. Section 6.5 presents the Cognitive Regulated Affective Architecture (CRAA) in details. Section 6.6 presents our experiments and empirical results on Unreal Tournament. The final section concludes and discusses future work.

6.2 Related Work

In the past decades, researchers have investigated how cognitive processes may affect emotions and proposed various theories of emotion based on cognitive appraisal of events. For example, in the OCC model [108], emotions are organized in groups according to what kind of aspect they focus on the world, such as the consequence of events, action of agents and aspects of objects. The characterization of each emotion depends on the principal variables that influence the intensity of emotions also. Smith and Lazarus [137] define emotion as a dynamic appraisal-coping-reappraisal process and describe emotion as one's core relation with environment in terms of harms and benefits. Damasio [29] distinguishes between primary emotion and secondary emotions, in which the former is supposed to be innate and the later may arise from high level cognitive processes, associating with evaluations of outcomes and expectations. Scherer's theory [126] describes emotion as a continuous mechanism that adapts to the environment by linking responses to stimulus. The evaluation of these stimulus events is based on the their appraisal with respect to the objectives of the agents.

Subsequently, based on these theories, many computational models of emotion have been proposed and applied to non-player characters in virtual worlds. For ex-

Table 6.1: A summary list of emotion models with their respective appraisal variables.

Models	*Appraisal Dimensions*	*Descriptions*
EMA	Relevance	Whether the event is relevant to the agent
	Perspective	The event is judged from the viewpoint of the agent own or others
	Desirability	The event could facilitate or inhibit the agent
	Likelihood	The probability of the outcomes
	Expectedness	The extent to which the truth value of a state could have been predicted from the causal interpretation
	Casual attribution	Who deserves praise or blame
	Controllability	Can the outcome be altered by actions under control of the agent whose perspective is taken
	Changeability	Can the outcome be altered by some other causal agent
FLAME	Desirability	The desirability of an event
	Expectation	The expectation of an event to occur
PEACTIDM	Suddenness	Extent to which stimulus is characterized by abrupt onset or high intensity
	Goal relevance	The importance of event with respect to goal
	Intrinsic pleasantness	The pleasantness of event or stimulus
	Conduciveness	How good or bad the event is for the goal
	Control	Extent to which anyone can influence the event
	Power	Extend to which agent can influence the event
WASABI	Pleasure	How pleasant the emotion is
	Arousal	Measures the intensity of the emotion
	Dominance	Measures the controlling of the emotion
FAtiMA	Desirability	How good or bad is the event for the agent
	Desirability For Other	How good or bad is the event for the target
	Praiseworthiness	The praiseworthiness or blameworthiness of an action
	Like relation	How much the agent like the target
THESPIAN	Motivational relevance	Evaluates the extent to which an encounter touches upon personal goals
	Incongruence	Whether the event is motivationally congruent or incongruent to the agent
	Accountability	Who deserves credit or blame for a given event
	Control	The extent to which an event or its outcome can be influenced or controlled by people
	Novelty	Whether the event is expected from the agent's past beliefs

ample, EM [101], FLAME [38], FAtiMA [34], and ALMA [48] were proposed based on the OCC theory of emotion [108]. WASABI [15], and PEACTIDM [90] were inspired by Scherer's theory [126]. There are also some emotion models proposed based on Smith and Lazarus's appraisal theory [137] such as THESPIAN [131]. Most of these models design specific appraisal rules for evaluating a selected set of appraisal variables. The list of appraisal variables used in these models is summarized in Table 6.1, and a brief review of the selected models is given below.

EMA was proposed by Marsella and Gratch [93] based on the dynamic of emo-

tional appraisal. EMA adopts the appraisal variables such as relevance, desirability, likelihood, expectedness, causal attribution, controllability and changeability. FLAME was presented by El-Nasr *et al.* based on the OCC model [108] and the "event-appraisal model" [120] [38]. Appraisal dimensions of desirability and expectation are used in FLAME. PEACTIDM is a computational structure to support emotional appraisal based on Scherer's theory [126] described by Marinier *et al.* [90]. Emotions in PEACTIDM is decided by six variables: suddenness, goal relevance, intrinsic pleasantness, conduciveness, control and power.

WASABI was developed by Becker-Asano *et al.* [15] to simulates the appraisal processes based on the Scherer's sequential-checking theory [126] and the pleasure-arousal-dominance (PAD) space named pleasure, arousal and dominance. FAtiMA, a two-layer architecture, was proposed by Dias *et al.* [34] to create virtual agents based on the OCC theory [108]. The appraisal of emotion is mainly based on the evaluation of desirability, desirability for other, praiseworthiness and like relation. THESPIAN is proposed by Si *et al.* [131] based on the "appraisal-coping-reappraisal" loop described by Smith and Lazarus [137]. It categorizes the emotions by motivational relevance, incongruence, accountability, control and novelty requiring objective evaluation

There are also models focusing on designing dynamical models to simulate the emotional part of brain, such as the flow model [98], the amygdala simulated model [96], and the BELBIC model [130]. However, these models either do not associate cognition with emotion or do not model expressible emotion instances.

6.3 Neural Substrates of Emotion

The brain mechanisms of emotion is always a serious topic. Early in 1884, James suggested that the emotional brain processes are variously combined processes instead of a ordinary sensorial system devoted to emotional functions [62]. Later, Cannon and Bard push the search down the neocortex, by demonstrating that thalamus, hypothalamus and neocortex are the centre of emotions [21] [11]. Differs from the Cannon-Bard theory, the Papez circuit which involves hypothalamus, anterior thalamus, cingulate gyrus, and hippocampus is proposed as the cortical control of emotion [110]. Later, MacLean defines the limbic system which comprises the Papez circuit together with amygdala, septal nuclei, orbito-frontal cortex, portions of the basal ganglia [89]. However, the limbic system is criticized recently by Brodal [19], Kotter [74], and LeDouzx [82]. Recent research approves the amygdala's critical roles in emotion based on the study of fear [81], whereas suggests there are emotion circuits or emotional networks which may include multiple regions in the brain [82].

Nowadays, a commonly accepted view is that cognition and emotion are two closely intertwined systems in human brain. Specifically, the prefrontal cortex (PFC) and anterior cingulate cortex (ACC) not only have a central role in cognitive control but also involve in assessing emotions. The dorsal-caudal regions of the ACC is concerned with computing reinforcement value of actions and involved in appraisal and expression of emotions [41]. The core of affective regions, namely amygdala (AMYG), also has been confirmed to respond based on attention and is closely linked to the function of perception [112]. These findings suggest that the mechanism of emotion involves not only the "affective" brain regions but also the "cognitive" brain regions, and that cognitive information need to be integrated with affective information in order to regulate the emotion. This view has also been supported by

functional magnetic resonance imaging (fMRI) examination [106], which reveals the LPFC may play a direct part in modulating emotion processing and related with the behavior funtion. In addition, the process of evaluating whether the stimuli are affectively significant is associated with two highly interconnected brain structures: AMYG, which is known as the functional center of affective responses, and medial orbital frontal cortex [104]. The PFC and AMYG are also found to be involved in reappraisal and the PFC can modulate activity in multiple emotion processing systems [106].

Although there are various circuits of emotion proposed in the past years, this work is inspired by looking into the cognitive-affective control circuit which is suggested by Pessoa [112]. Building on the network structure, Pessoa has demonstrated the information flow among several main regions in human brain by a circuit, in which the prefrontal cortex, amygdala and anterior cingulate cortex are involved. Figure 6.1 shows these regions in human brain together with the circuit of information flow. Besides the emotion center AMYG, the PFC, specifically the Orbitofrontal cortex (OFC) and lateral prefrontal cortex (LPFC), and ACC are all involved in generating emotions. Specifically, external stimuli are transmitted in parallel to several regions includes OFC, LPFC and amygdala and their significance are determined by the neural computations among the multiple regions. LPFC has the role of gathering the affective and cognitive information and realizes the affective regulated executive control. On the other hand, the output signals of LPFC are also involved in the affective process. The ACC neurons evaluate the reward signals and assess the effects of actions. Both of the ACC and Lateral PFC interact with amygdala in processing affective information [112].

This proposal also demonstrates several principles for emotion modeling: First of all, emotion is generated not by an isolated region, but through the network of

multiple areas in human brain. Secondly, emotion architecture is a complex network which comprises multiple modules and neural computations. Finally, emotion needs to be integrated with other inner states in order to take further impact on behavior.

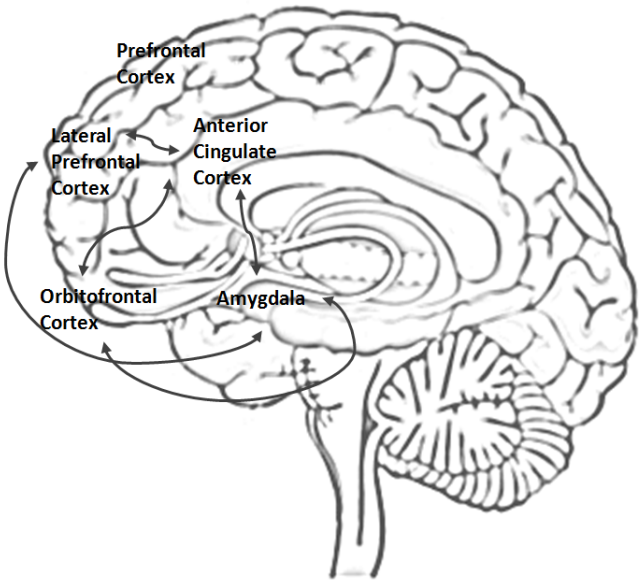

Figure 6.1: The circuit of cognitive-affective interaction in brain. (Adapted from [112]).

6.4 Approach and Design Principles

Our aim is to develop a biologically-plausible computational model that is able to emulate the structure as well as the processes of emotion based on parallel and distributed computations in multiple interacting neural networks.

Based on the Adaptive Resonance Theory (ART) [27], we propose a multi-network architecture to simulate the function and interactions of the prefrontal cortex (PFC) by a Cognitive Network, anterior cingulate cortex (ACC) by an Appraisal Network, and amygdala by the Affective Network. The model called Cognitive Regulated Affection Architecture (CRAA) works to show that affective responses are a

natural consequence of a cognitive system learning through evaluative feedback of rewards and punishments from its environment.

When designing the CRAA model, the following principles have been taken into consideration.

Principle 1: Emotions are "the states elicited by rewards and punishments" [119]

Emotions can be defined as mental states which can be affected by rewarding or punishing stimuli [119]. This statement implies that appraisal involves the assessment of whether a given situation is desirable or to be avoided and that the cognitive learning process and emotion process are tightly coupled. Once the stimuli are recognized by an individual, the emotions are elicited, and at the same time, emotions motivate the individual to respond to the stimuli, such as work ahead of schedule or avoid the dangers [119]. To simulate the impact of cognitive evaluation on emotions, in our architecture, emotions are appraised in parallel with the cognitive learning process.

Principle 2: Emotion is generated through the interaction of multiple neural networks

Based on the suggestions by [112], a model of emotion should involve cooperative computation of multiple neural networks in different brain regions. Figure 6.2 shows a schematic diagram of how cognitive regulated affection may arise based on the information flow among the three main brain regions. This schematic framework serves as the blueprint of our design. Specifically, a Cognitive Network should be able to emulate the executive functions in prefrontal cortex (PFC), such as strategy generating for coping with emotional events and integrating the affective and motivational information. Secondly, an Appraisal Network should simulate the evaluation

functions of anterior cingulate cortex (ACC), such as anticipation of tasks, attention, and evaluating the benefits of actions. Finally, both of the cognitive information and the appraisal information should be processed in the amygdala (AMYG), which is simulated by the Affective Network, to decide the emotion responses.

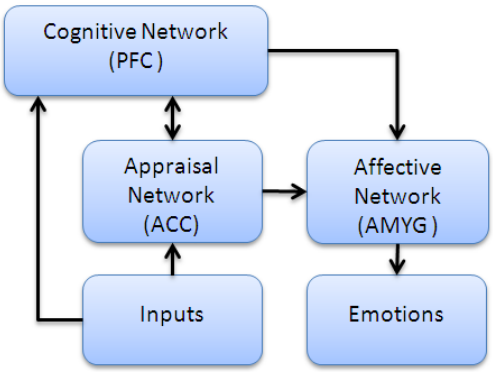

Figure 6.2: The schematic diagram of CRAA.

Principle 3: The expression of emotions is under modulation

We consider the issue that the emotions of agents should be expressed in a natural and modulated way. For example, it is unreasonable for an individual to change emotions back and forth abruptly over a very short period, and the emotion could be enhanced, weakened, and even inhibited sometimes. In fact, emotion expressions can be regulated by cognitive control to alternate the existing or the ongoing ones. This type of cognitive regulation is also known as the reappraisal process, and studies have found that the reappraisal happens based on the activation of anterior cingulate cortex (ACC) and prefrontal cortex (PFC) system. As a result, reappraisal could decrease, increase or maintain the activity in amygdala (AMYG) [105]. Together with previous emotion appraisal process, the "appraisal-reappraisal" progress goes through a top-down and then bottom-up progress to regulate the emotion output. We have considered a simple form of emotion regulation in the CRAA model for

moderating the changes in the intensities of emotion. Based on Picard's suggestion that emotions could be modeled as signals [114], each of the emotion signal will go through a process of intensifying and decaying.

Principle 4: Not all emotions are equal

While there is no unified list of how many different types of emotions exist in human emotion, the discrete emotion theory argues that there exists a set of basic/primary/fundamental emotions, which are biological universal to all humans and could represent the core of emotion. Lately, many scholars support this theory by proposing specific sets of basic emotions, such as the Aronld's eleven emotions, the Gray's four emotions, the Frijda's six emotions [109] and the most widely used Ekman's six basic emotions: anger, disgust, fear, happiness, sadness, and surprise [37]. Some scholars however argue that surprise is not an emotion, as it plays the role of elicitation and intensification of emotions [109]. Furthermore, some researches classify emotions into different categories of primary/basic emotions and secondary/compound emotions, saying that primary emotions are innate for humans and secondary emotions are arisen from higher cognitive processes [29], such as shame, anxiety etc. Some also propose the notion of compound emotions which are obtained by merging particular emotions [108, 34]. For simplicity, we have adopted the Ekman's six basic emotions [37] in our work.

6.5 Cognitive Regulated Affective Architecture

This section proposes the Cognitive Regulated Affective Architecture (CRAA), comprising a Cognitive Network, an Affective Network and an Appraisal Network as a simplification of the cognitive-affective control circuit described by Pessoa [112],

based on the Adaptive Resonance Theory (ART) [27], and the theory of cognitive regulated emotion [104].

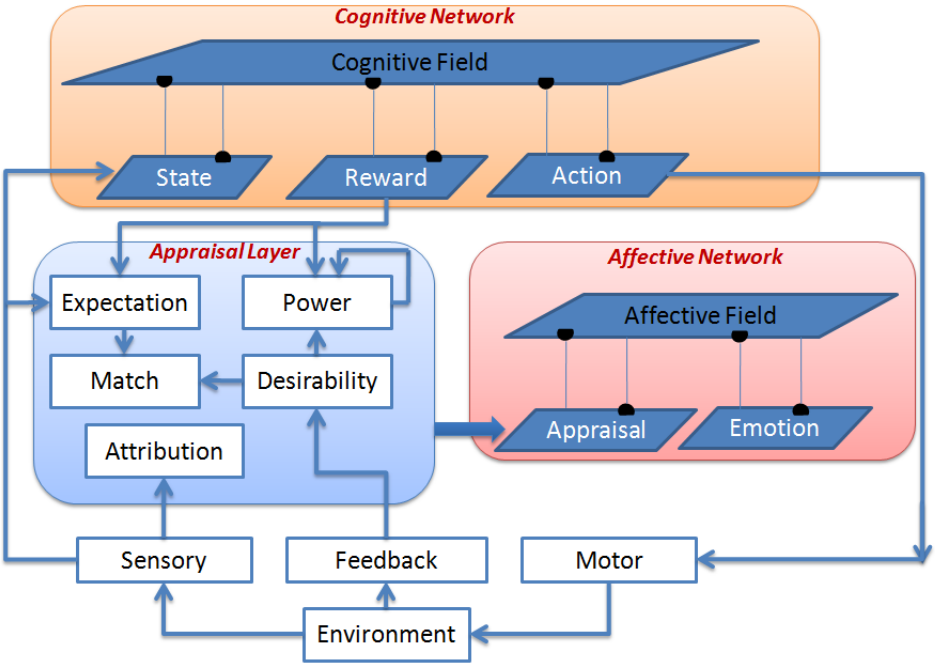

Figure 6.3: The Cognitive Regulated Affective Architecture.

Figure 6.3 shows the integrated cognitive-affective architecture which simulates the functions and interactions of prefrontal cortex (PFC) by the Cognitive Network, anterior cingulate cortex (ACC) by the Appraisal Network, and amygdala by the Affective Network. The Cognitive Network takes charge of the behavior learning tasks and the Affective Network learns the associations from appraisal components to emotion. The Cognitive Network is connected to the Affective Network through the Appraisal Network in real time.

Upon receiving the sensory inputs from the environment, the Cognitive Network performs decision makings based on the content in the working memory. Imitating the role of ACC, the Appraisal Network then evaluates the appraisal variables,

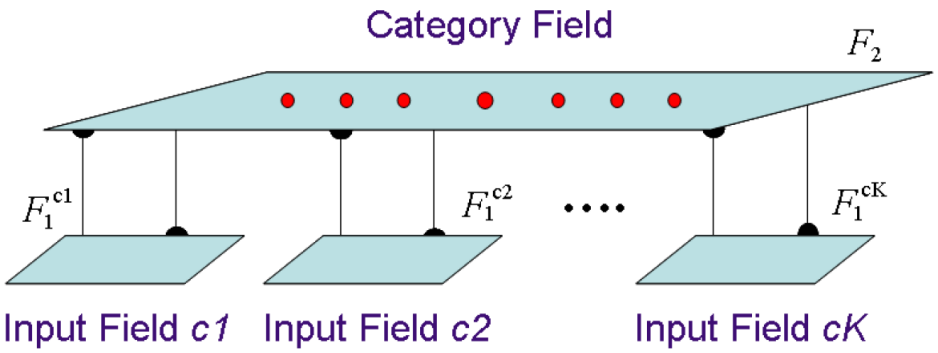

Figure 6.4: The fusion ART model.

consisting of cognition information, namely expectation, desirability and match, and internal mental states, namely power and attribution. The Affective Network then performs the associative mapping from appraisal variables to emotion responses.

In the following sections, we review the building block of Cognitive Network and Affective Network, known as fusion Adaptive Resonance Theory model [149], followed by the three network modules in the CRAA model.

6.5.1 Fusion ART

As a generalization of the Adaptive Resonance Theory (ART) models [27], fusion ART [149] is designed for learning cognitive nodes across multi-channel mappings simultaneously across multi-modal input patterns. The model unifies a number of network designs, most notably Adaptive Resonance Theory (ART) [26, 27], Adaptive Resonance Associative Map (ARAM) [145] and Fusion Architecture for Learning, COgnition, and Navigation (FALCON) [147], developed over the past decades for a wide range of functions and applications.

In its general form, fusion ART comprises a category field F_2 and K input pattern fields (F_1^{ck}, where $k = 1, 2, \ldots, K$) through bidirectional conditionable pathways

(Figure 6.5.1). The general dynamics of fusion ART for learning and prediction, based on fuzzy ART operations [24], is summarized below.

Input vectors: Let $\mathbf{I}^{ck} = (I_1^{ck}, I_2^{ck}, \ldots, I_n^{ck})$ denote the input vector, where $I_i^{ck} \in [0, 1]$ indicates the input i to channel ck. With complement coding, the input vector $\mathbf{I}^{ck}$ is augmented with a complement vector $\bar{\mathbf{I}}^{ck}$ such that $\bar{I}_i^{ck} = 1 - I_i^{ck}$.

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

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

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

Code activation: Given the input vectors $\mathbf{I}^{c1}, \ldots, \mathbf{I}^{cK}$, for each F_2 node j, the choice function T_j is computed as follows:

$$T_j = \sum_{k=1}^{K} \gamma^{ck} \frac{|\mathbf{I}^{ck} \wedge \mathbf{w}_j^{ck}|}{\alpha^{ck} + |\mathbf{w}_j^{ck}|}, \qquad \text{(Eq. 6.1)}$$

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 $\mathbf{q}$.

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

$$T_J = \max\{T_j : \text{for all } F_2 \text{ node } j\}. \qquad \text{(Eq. 6.2)}$$

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

Activity readout: The chosen F_2 node J performs a readout of its weight vectors to the input fields F_1^{ck} such that

$$\mathbf{x}^{ck} = \mathbf{I}^{ck} \wedge \mathbf{w}_J^{ck}. \qquad \text{(Eq. 6.3)}$$

Template matching: Before the activity readout is stabilized and 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 input patterns. Specifically, resonance occurs if for each channel k, the *match function* m_J^{ck} of the chosen node J meets its vigilance criterion:

$$m_J^{ck} = \frac{|\mathbf{I}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{I}^{ck}|} \geq \rho^{ck}. \qquad \text{(Eq. 6.4)}$$

If any of the vigilance constraints is violated, mismatch reset occurs in which the value of the choice function T_J is set to 0 for the duration of the input presentation. Using a *match tracking* process, at the beginning of each input presentation, the vigilance parameter ρ^{ck} in each channel ck equals a baseline vigilance $\bar{\rho}^{ck}$. When a mismatch reset occurs, the ρ^{ck} of all pattern channels are increased simultaneously until one of them is slightly larger than its corresponding match function m_J^{ck}, causing a reset. The search process then selects another F_2 node J under the revised vigilance criterion until a resonance is achieved.

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

$$\mathbf{w}_J^{ck(\text{new})} = (1 - \beta^{ck})\mathbf{w}_J^{ck(\text{old})} + \beta^{ck}(\mathbf{I}^{ck} \wedge \mathbf{w}_J^{ck(\text{old})}). \qquad \text{(Eq. 6.5)}$$

Code creation: Our implementation of Fusion ART maintains ONE uncommitted node in the F_2^c field at any one time. When an uncommitted node is selected for learning, it becomes *committed* and a new uncommitted node is added to the F_2^c

field. Fusion ART thus expands its network architecture dynamically in response to the input patterns.

As a natural extension of ART, fusion ART responds to incoming patterns in a continuous manner. It is important to note that at any point in time, fusion ART does not require input to be present in all the pattern channels. For those channels not receiving input, the input vectors are initialized to all 1s.

6.5.2 Cognitive Network

The Cognitive Network is designed to emulate executive functions in prefrontal cortex (PFC), especially decision making and reinforcement learning for coping with events from the environment. As shown in Figure 6.3, the Cognitive Network is a three-channel fusion ART model called Temporal Difference Fusion Architecture for Learning, COgnition, and Navigation (TD-FALCON) [151]. TD-FALCON is a self-organizing neural network that 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. TD-FALCON consists of a cognitive field and three input fields, namely a sensory field for representing current states, an action field for representing actions, and a reward field for representing reinforcement values.

For decision making, the Cognitive Network first chooses between exploration and exploitation based on an action selection policy. For exploitation, given the current state s, the Cognitive Network determines the best action a to perform based on the fusion ART code activation (Equation Eq. 6.1) and code competition (Equation Eq. 6.2) processes. Upon receiving a feedback from the environment, a TD formula is used to compute a new estimate of the Q value of performing the

chosen action a 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 a to the estimated Q value. For a more detailed description of TD-FALCON, including its action selection policy and Q-learning algorithm, please refer to [148] and [151].

6.5.3 Appraisal Network

The Appraisal Network employs a selected set of appraisal variables to represent the "human-environment relationship", including *expectation of outcome*, *desirability of the event*, *match in expectation*, *attribution* and *power of agent*. Whereas *expectation*, *desirability* and *match* are appraised dynamically based on the cognitive information, such as context and feedback signals, received from the environment, *attribution* and *power* represent internal mental states of the agent. The list of variables is intentionally kept to the minimum as our main aim is to demonstrate that primary emotion can be generated based on a small set of internal attributes and the cognitive signals produced by a cognitive reinforcement learning system.

In the Appraisal Network, an assembly of specialized neurons is used to encode the five appraisal variables and provides a unidirectional translation of information from the cognitive space to the affective space. The list of appraisal variables used in CRAA is summarized in Table 6.2. The detailed description for each of the five appraisal variables is presented as follows.

Expectation: Expectation E is the predicted desirability of what is likely to happen. In the context of reinforcement learning, expectation corresponds to the estimated expected rewards when an agent takes an action in a situated environment. It shows the agent's personal evaluation of a possible outcome which occurs subsequently after taking a specific action. Therefore, expectation should be associated

Table 6.2: List of appraisal variables used in CRAA.

Appraisal Dimensions	*Descriptions*
Expectation	The estimation of the extend to which the outcome can be good for the goal based on agent's belief and action
Desirability	How much the event is good for the agent's goal
Attribution	Is the agent self attribution or other attribution
Power	Extend to which the agent can influence the event
Match	The extend to which the outcome is as agent's pre-estimation

with situations, actions, and must depend on the agent personal experience. Formally, expectation can be estimated using the Q-value function defined as

$$E = Q(s, a), \tag{Eq. 6.6}$$

where $Q(s, a)$ calculates the expected reward for taking action a in the state s. Thus, expectation represents the pre-evaluation of the effect of the action to be taken.

Expectation is considered as a key factor for emotional appraisal. And it has been generally thought that the emotion intensity increases as the expectation of positive result or negative avoidance increases [115]. This variable is similar to the "outcome probability" by Scherer [126] and Roseman [120]. For instance, if we have encountered a similar situation and become aware of its outcome, we form an expectation of the desirability of the future event. On the other hand, expectation also demonstrates the potential uncertainty of outcomes (by Roseman [120]), and if the outcomes are beyond our expectations, the emotions of surprise and fear may incur. The implementation of this appraisal variable can be seen in the computational models of EMA [93] and FLAME [38].

Desirability: Desirability D is the degree of positive outcome of an event. In the context of reinforcement learning, desirability corresponds to reward signals received from the environment. According to OCC model, desirability is the main criterion for evaluating an event [108]. It refers to the degree of goodness of an event for

humans or agent by showing if the event consequence is desirable or undesirable. Following OCC, many models implement the computation of desirability such as FLAME [38], EMA [93] and FAtiMA [34] which measure desirability in terms of whether the event could facilitate or inhibit the agent in the current state, or compute the desirability by associating with the importance of goals. EMA and FAtiMA make a distinction between desirability from the agent's and from the others' perspective.

CRAA evaluates desirability based on the context of reinforcement learning, by measuring the perceived reward or penalty signals received from the environment. This is supported by recent research which shows that amygdala, the core region for emotion in human brain, could mediate an arousal effect, which links the cognitive information of rewards to emotion [99]. Computationally, the reward signal is simply the real-time immediate feedbacks r given by the environment, with a value in the range of [0,1]:

$$D = r. \qquad \text{(Eq. 6.7)}$$

Typically, the reward signal is measured within a time window after an action is taken by the agent.

Attribution: Attribution A indicates one's explanation of the cause of an event or outcome. Considering the actions of agents, the OCC model [108] differentiates different types of emotions based on different attributions. In other words, the consideration of whether one self or other agent is responsible for an outcome leads to different emotional responses. Also, this evaluation could further influence the appraisal of the outcome probability and self ability (see similar component "causal attribution" by Scherer [126] and a reasonable replacement "accountability" by Smith and Lazarus [137]). Attribution has been implemented in EMA [93] as

"causal attribution" and in THESPIAN [131] as "accountability" to indicate who deserves the credit or blame for the event.

CRAA also distinguishes between "self attribution" and "other attribution" to explain whether the agent itself or other agent should take responsibility for the outcome (Formally, the value of Attribution is given by

$$A = \begin{cases} 1 & \text{if self attribution} \\ 0 & \text{otherwise} \end{cases} \quad \text{(Eq. 6.8)}$$

Such interpretation of attribution may influence further evaluations such as self assessment and self esteem.

Power: Power P forms part of one's inner state in real time of how much confidence one's has in its own capability in achieving goals. The appraisal theory by Roseman includes power to differentiate among emotions, pointing out that the appraisal of emotion should be based on the context appraisal by evaluating oneself's competence. For example, perceiving oneself to be weak will lead to the emotion of sadness or fear [120]. This component is also found in Scherer's theory [126], referring to assessments of physical strength, resources and knowledge and directly deciding the coping potential. The appraisal dimensions of "controllability" in EMA [93] and "control" in PEACTIDM [90] and THESPIAN [131] could present the possible replacements. The higher confidence an agent has, the more the agent can influence the events.

In our work, power is measured based on perceiving self, and can be estimated by accumulating one's experience in terms of successes and failures. Specifically, CRAA updates the value of power in real time with an update equation (Eq. 6.9) using evaluative feedback signals and the current power level:

$$P(t+1) = P(t) + \gamma(1 - P(t)) - \delta P(t), \quad \text{(Eq. 6.9)}$$

where $P(t+1)$ and $P(t)$ represent the values of power in time $t+1$ and t respectively, and the parameters $\delta \in [0, 1]$ and $\gamma \in [0, 1]$ are the *power decay rate* and *power gain rate* respectively.

Match: Match M is the measurement of expectedness based on the consistency between one's expected desirability and the desirability of the actual outcome. A similar replacement (but opposite) appraisal known as "novelty" can be found in Roseman and Scherer's theories [120] [126], assessing if the event stimulus is beyond the agent's expectation depending on one's prior experience. The appraisal of "novelty" has been used in the computational model THESPIAN [131]. Here we adopt "match" instead of "novelty" as neural networks provides a natural way of computing the match between the agent's expectation and the desirability. Mathematically, Match can be calculated simply as

$$M = 1 - |E - D|. \qquad \text{(Eq. 6.10)}$$

In real time, expectation, desirability and match are obtained and evaluated dynamically based on the learning and prediction processes of the Cognitive Network. Together with the internal signals of power and attribution, the Affective Network performs the associative mapping from cognitive appraisal to emotional responses.

6.5.4 Affective Network

The Affective Network is designed to simulate the functions of amygdala in producing emotional responses. As shown in Figure 6.5.4, the Affective Network employs a two-channel fusion ART architecture, comprising an affective field and two input pattern fields, namely an appraisal field for representing appraisal variables and the emotion field for representing emotion instances. By learning the association from

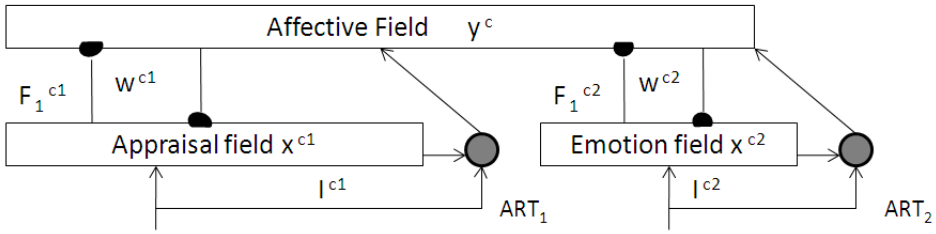

Figure 6.5: The Affective Network model.

appraisal variables to emotion instances, the Affective Network is designed to encode the emotion appraisal function of

$$E_m = f(E, D, A, P, M), \qquad \text{(Eq. 6.11)}$$

where E, D, A, P and M are the cognitive information and internal states maintained by the Appraisal Network, namely expectation, desirability, attribution, power and match respectively and E_m denotes the corresponding emotion instances. The Affective Network thus realizes cognitive regulated emotion via cognitive information infusion through the Appraisal Network.

6.5.4.1 Evaluating Appraisal Variables

Based on fusion ART, the Affective Network is able to respond to incoming signals in a continuous manner. The processing cycle comprises five key stages, namely code activation, code competition, activity readout, template matching, and template learning, as described earlier. At the beginning of each processing cycle, the five neurons encoding incoming information in the Appraisal Network need to be updated. The appraisal information is sensed in the Affective Network in order to realize cognitive regulated emotions. For our discussion, let I_1^{c1} to I_5^{c1} denote the input neuron values for channel F_1^{c1}, which are updated as follows.

Table 6.3: The basic appraisal rules of CRAA.

No.	*Symbolic rules*
1	IF Desirability is low AND Attribution is self AND Power is high THEN Angry
2	IF Desirability is low AND Attribution is others THEN Disgust
3	IF Expectation is low THEN Fear
4	IF Desirability is high THEN Happy
5	IF Desirability is low AND Attribution is self AND Power is low THEN Sad
6	IF Match is low THEN Surprise

Five input signals are regulated synchronously by sensing from the neurons in the Appraisal Network. I_1^{c1} is regulated by the expected reward of the action and updated by $I_1^{c1} = E$. This demonstrates the pre-evaluation of the payoff by taking a specific action. I_2^{c1} is regulated by the real-time feedback from the environment and updated by $I_2^{c1} = D$. I_3^{c1} is regulated by the way of how the agent explains the course of the event, and updated by $I_3^{c1} = A$. I_4^{c1} is regulated by the inner state of power, and updated by $I_4^{c1} = P$. Finally, I_5^{c1} is regulated by the evaluation of how much the action could meet the expectation and updated by $I_5^{c1} = M$. Once the appraisal information transits from the Appraisal Network to the affective network, emotion retrieving can proceed.

6.5.4.2 Emotion Mapping

The Affective Network is designed to realize the emotion appraisal function which maps directly from the appraisal variables (E, R, A, P, M) to the corresponding emotion responses (E_m). Our strategy is to initialize the Affective Network with a set of appraisal rules as listed in Table 6.3 and leverage on the continuity property of neural networks to generalize the responses to situations not covered by the inserted appraisal rules.

- **Modeling Angry and Disgust:** According to Silvia, anger and disgust are aesthetic emotions and have the similarity in goal incongruent [134], which means these emotions should associate with low desirability. To distinguish the two emotions, Silvia thinks anger arises when appraising an event as deliberately caused, saying "this appraisal structure can be expressed thematically as deliberate trespass." The difference between them is that an event is appraised as intentionally goal incongruent for angry and is inherently aversive for disgust [133]. Moreover, based on Roseman's discussion [120], power could be used to discriminate among different types of negative emotions, saying "if one can respond effectively to a negative event, one may feel angry rather than afraid or sad". Therefore low desirability, self attribution and high power are identified as key factors. On the other hand, according to Silvia, "unlike anger, disgust involves appraising something as unpleasant, dirty, or harmful." As such, low desirability and attribution to others are identified to be the key triggers for disgust.

- **Modeling Fear:** Roseman says "uncertain outcome leads to the emotion of fear". Specifically, "Expectation" demonstrates the potential uncertainty of outcomes and if the outcomes are beyond our expectations, the emotions of surprise and fear may incur [120]. As such, we design the rules for fear and surprise based on this consideration. For fear, the Expectation is set to low to emulate that when the agent's expectation of its reward incurred by taking the specific action is not high, the agent may be fearful. This is also consistent with OCC that displeased about the prospect of an undesirable event may incur fear [108].

- **Modeling Happy:** The emotion of happy is often caused by high desirability and arises when some thing desirable happens. This design is consistent with the OCC model, in which the emotion of sadness and happiness are simply the cases of being pleased or being unpleased, resulting from positive or negative reactions to the events that affect one [108]. For happy, which is similar to joy, it is thought to be involved with a desirable event as a necessary condition.

- **Modeling Sad:** According to [108], the emotion of sad relates often to low desirability and arises when a negative event happens. Moreover in our design, sad also relates to attribution and power. The low capability for achieving the goal may strengthen the emotion, as by Roseman [120], perceiving oneself to be weak will lead to the emotion of sad. And the above concerns are based on the attribution of self.

- **Modeling Surprise:** Based on the same consideration by Roseman for Fear, if the outcomes are beyond our expectations, the emotions of surprise and fear may incur [120]. As such, we initialize the Match as low to include both situations of high expectation and low desirability, and low expectation and high desirability.

For rule insertion, each of the appraisal rules is first translated into a pair of input and output patterns. Specifically, given the values of expectation E, desirability D, attribution A, power P, and match M, as indicated by the antecedents of the rules and the target emotion instance E_m as indicated in the consequent, the input appraisal vector $\mathbf{I}^{c1}$ is a complement coded patterns of variables E, D, A, P, and M, wherein each variable with a value *high* is encoded as (1,0); variable with a value *low* is encoded as (0,1); and variable not specified in the rule is encoded as

Table 6.4: Samples of the translated appraisal rules.

No.	*E*	*D*	*A*	*P*	*M*	*Emotion Instances*
1	(0,0)	(0,1)	(1,0)	(1,0)	(0,0)	Anger
2	(0,0)	(0,1)	(0,1)	(0,0)	(0,0)	Disgust
3	(0,1)	(0,0)	(0,0)	(0,0)	(0,0)	Fear
4	(0,0)	(1,0)	(0,0)	(0,0)	(0,0)	Happy
5	(0,0)	(0,1)	(1,0)	(0,1)	(0,0)	Sad
6	(0,0)	(0,0)	(0,0)	(0,0)	(0,1)	Surprise

(0,0). Similarly, the input emotion vector $\mathbf{I}^{c2}$ is a complement coded patterns of E_m, wherein the consequent emotion instance in the rule is encoded as (1,0) and those not specified in the rule are encoded as (0,0). A sample list of the translated rules in pattern form is shown in Table 6.4. For illustration purpose, we only show the complement coded form of the appraisal variables.

Based on the translated appraisal and emotion patterns, the Affective Network network performs the standard fusion ART operations of code activation, code competition, template matching, template learning and code creation, such that each distinct appraisal rule is encoded by a category node in the Affective Field.

6.5.4.3 Emotional Appraisal

During performance, the Affective Network is designed to search for an affective node based on the current situation encoded in the input appraisal pattern. The search is through the same code activation and code competition processes of fusion ART, as in equations (Eq. 6.1) and (Eq. 6.2).

In the standard fuzzy ART algorithm, upon selecting a winning F_2^c node J, the chosen node J performs a readout of its weight vector into the emotion field F_1^{c2}. For affective modeling, however, multiple emotion responses are possible. To enable the model to produce more than a single emotion response for any given situation

when appropriate, we extend the original winner-take-all choice rule of fuzzy ART to allow a combined output by all affective nodes that satisfy the specified input vigilance criterion. Specifically, the output emotion activity vector is a combination of the output weight vectors of the eligible category nodes, weighted by their choice function value:

$$\mathbf{x}^{c2} = \mathbf{I}^{c2} \wedge \sum_{J} \mathbf{T}_J \mathbf{w}_J^{c2} \text{ for all node } J \text{ where } m_J \geq \rho^{c2}. \quad \text{(Eq. 6.12)}$$

6.5.4.4 Reappraisal

Based on the output of the Affective Network, emotion reappraisal is performed by modifying the current emotions based on the newly appraised emotions by

$$\mathbf{E}^{(\text{new})} = \mathbf{E}^{(\text{old})} \vee \mathbf{x}^{c2}. \quad \text{(Eq. 6.13)}$$

Concurrently, the intensities of emotions also undergo a decay process over time. In this work, we use a gaussian decay function to model the decay process of a given emotion E by

$$E(t+1) = E(t) \cdot e^{-\frac{t^2}{2\sigma^2}}, \quad \text{(Eq. 6.14)}$$

where t denotes the time and the parameter σ is used to control the decay rate. After an emotion is triggered, the counter t is reset to 0 and the emotion intensity E is set to the level of intensity as specified in the output emotion pattern $\mathbf{x}^{c2}$, before continuing the decay process again.

6.6 Experiments

6.6.1 The Problem Domain

Unreal Tournament (UT) [158] is a first person shooting game, which does not merely offer an environment for gaming, but also a platform for building and evaluating

autonomous agents. We have built affective Non-Player Characters (NPC) in UT game, by integrating our Cognitive Regulated Affective Architecture (CRAA) into the agent's behavior policies. The NPCs are embedded with perception and sensor modules, which can capture the events happened in their surrounding areas. In the "Deathmatch" game scenario, each NPC must fight with all other players in order to survive and win. Our aim is to design affective NPCs, who can display appropriate emotion responses when they encounter various events. Doing this, we believe, will improve the playability of the game and the enjoyment of the players. In our experiments, we adopt the Ekman's six basic emotions, namely anger, disgust, fear, happy, sad and surprise. For the Unreal domain, the main events that will trigger such emotional responses, include *pick up health*, *pick up weapon*, *damage enemy*, *kill enemy*, *damaged by enemy* and *killed by enemy*.

6.6.2 Experiment Methodology

The appraisal rules shown in Table 6.4 are first inserted into the Affective Network by using the following parameter setting: choice parameter $\alpha^{ck} = 0.1$ for $k = 1, 2$; learning rate $\beta^{ck} = 1$ for $k = 1, 2$; contribution parameter $\gamma^{c1} = 1$, $\gamma^{c2} = 0$; and vigilance parameter $\rho^{ck} = 1$ for $k = 1, 2$. All the input patterns were normalized by complement coding [25]. Parallel triggering of multiple emotions is enabled, as described in the previous section.

6.6.3 Expressions of Emotions

As it is difficult to see the NPC's emotions by facial expressions in a fast pace first-person shooting game, we design customized verbal expressions to show the affective status of the NPCs. As shown in Table 6.5, when a specific emotion is triggered, the affective NPC will utter the corresponding verbal expression.

Table 6.5: The list of six emotions (adopted from [37]) and their corresponding verbal expressions.

No.	*Emotion*	*Emotion expression*
1	Anger	Dammit!
2	Disgust	Disgusting!
3	Fear	Gosh!
4	Happy	Haha!
5	Sad	Oh! No!
6	Surprise	Oops!

Table 6.6: Samples of emotions expressed by NPCs in response to specific events.

Event	*Emotion*	*Expression*
Pick up weapon	Happy	Haha!
Damaged by enemy	Surprise	Oops!
	Disgust	So disgusting!
Pick up health	Happy	Haha!
Killed by enemy	Disgust	So disgusting!
Damage enemy	Happy	Haha!
Kill enemy	Happy	Haha!

Table 6.6 illustrates the sample verbal expressions by the affective NPCs in response to various events. In our implementation, the expressions are shown in a dialog box as part of the UT interface designed for communication. Thus, every player involved in the game can see the messages in real-time. Figure 6.6 (a) and (b) show the snapshots of games with an affective NPC and a typical emotion-less NPC respectively. We see the affective NPC is able to express its affective status dynamically in real time.

6.6.4 Evaluating Model Accuracy

We compare the CRAA model with several selected rule sets which are adopted from three recently proposed models, namely EMA [51], FAtiMA [34] and WASABI [15]

Figure 6.6: (a) The affective NPC and (b) the emotionless NPC in Unreal Tournament.

in terms of output accuracy. The three rule sets are chosen for comparison as they have provided clear and specific emotion appraisal which we can use directly. The emotion rules, as used in the EMA, FAtiMA and WASABI, require the evaluation of a selected set of designed appraisal variables respectively, as shown in Table 6.1. For the purpose of comparison, these appraisal variables and emotion rules in above models are customized and applied in the same platform as CRAA. Specifically, events from the Unreal domain are presented to these rules and the performance of the models are evaluated based on human assessment of their output emotions. Using a questionnaire-based approach, we first ask the human assessors to evaluate the appraisal variables in these rule bases. To make those appraisal dimensions easy to be understood, we further formulate them into questions and also cite their explanatory note from their original papers. Before the experiments, we give the subjects a short training so that they understand these models appraisal rules correctly. The questionnaire lists the same situations as in our experiments, leaving out the evaluation of appraisal variables. Then we distribute another questionnaire which lists the same situations as in the previous one, but subjects are asked to state their

Table 6.7: Means and standard deviation of accuracy of CRAA and the other models based on human evaluation.

Model	*Accuracy*	*Standard Deviation*
Emotion rules in EMA [51]	70.1%	15.7%
Emotion rules in FAtiMA [34]	68.8%	14.9%
Emotion rules in WASABI [15]	65.7%	15.3%
CRAA's rules	76.9%	15.2%

own appraised emotions. Thereafter, we compile the results of the questionnaires and calculate the rules' accuracy of each model. Twenty subjects aged between 20 and 30 are randomly selected from the computer school with a gender ratio of 7:3, considering male users are typically more experienced in first person shooting games than females. The results are calculated by averaging across the subjects.

Table 6.7 shows the performance of the rule base in EMA, the rule base in FAtiMA, the rule based in WASABI and CRAA's rules in terms of their average accuracy by human assessment. We observe that the affective NPCs based on CRAA show a reasonable level of accuracy, indicating that the users generally think the cognitive regulated emotions are appropriate. In contrast, most other models achieve less than satisfactory results. The relatively low level of accuracy may be due to two reasons. Firstly, their models are static and limited by the predefined appraisal rules. While the rules may mirror their designers' emotion patterns, they do not generalize well to the context of the Unreal Tournament. For example, in WASABI, the emotion of fear is strictly located in the PAD space of (-80,80,100) and the emotion of sad is fixed at (-50,0,-100) for all people. Secondly, those models haven't considered the real-time regulation of emotion and thus do not display a dynamic range of emotional response during the game play.

6.6.5 Evaluating Effect of Affective NPCs

In addition to measuring the emotion patterns, experiments are conducted to evaluate the qualitative aspects of affective NPCs against the emotionless NPCs. The evaluation is based on the questionnaires originally designed for Second Life [32], which look at virtual worlds along three dimensions of playability: 1) *Cognitive absorption*, which measures whether affective NPC makes player feel more involved [4]; 2) *Social presence*, which measures whether affective NPC improves the sense of interaction [103]; and 3) *Enjoyment*, which measures whether affective NPC makes a difference in the level of enjoyment of the players [100]. As shown in Table 6.8, questions $c1$ to $c6$ are designed to evaluate the cognitive absorption, questions $s1$ to $s6$ are designed to evaluate the social presence, and questions $e1$ to $e6$ are designed to measure the level of enjoyment. For each of the questions, the players' answers can be chosen among "strongly disagree", "disagree", "neutral", "agree", and "strongly agree", each corresponding to a normalized score between 0 and 1.

Two videos were recorded for affective NPCs using CRAA and emotionless NPC playing in the Unreal Tournament game respectively. The human users are requested to complete the questionnaire after viewing the videos. Subsequently, the experiment was repeated for affective NPCs based on the emotion rules used in EMA [51]. The results are calculated by averaging across twenty subjects (the same as in the previous experiments).

The evaluation results for affective NPC using CRAA and affective NPC using EMA emotion rules are summarized in Table 6.9. We also verify these results by Kolmogorov-Smirnov tests confirming the normality distribution hypothesis at the significance level of $\alpha = 0.05$. Both affective NPCs based on CRAA and EMA rules obtain good results over the traditional emotionless NPCs in terms of cognitive

Table 6.8: Questions used in the human user evaluations.

No.	*Questions*
$c1$	I think the affective NPC's verbal message could express its emotions.
$c2$	I think the emotion expressions are appropriate.
$c3$	I could notice the emotional verbal message of NPC during play.
$c4$	I can't help checking the emotional verbal message of NPC during play.
$c5$	I prefer to shot the affective NPC than shot a emotionless NPC because of its reaction.
$c6$	I can play longer with the affective NPC than with a normal NPC.
$s1$	I feel the enemy NPC could react to what happened in the environment.
$s2$	I think the affective NPC is more like a real human than the emotionless NPC.
$s3$	I sense the NPC's current emotion by its expressions during play.
$s4$	I think the affective NPC interacts with you more than the normal NPC.
$s5$	The emotional expression helps me to understand the NPC's status of fighting.
$s6$	I feel a perceptual human-computer interaction in the game with affective NPC .
$e1$	I think the verbal message of affective NPC is an interesting expression.
$e2$	Comparing with the emotionless NPC, affective NPC looks special and different.
$e3$	Sometimes the affective NPC's emotional react could surprise me.
$e4$	I prefer to play the game with affective NPC than without.
$e5$	I feel more emotional when playing the game with affective NPC.
$e6$	The game with affective NPCs gives me more fun.

Table 6.9: Means and standard deviations of human ratings for affective NPCs using CRAA and EMA emotion rules.

Model	*Cognitive absorption*	*Social presence*	*Enjoyment*
CRAA	0.83 ± 0.09	0.82 ± 0.07	0.85 ± 0.08
EMA rules	0.75 ± 0.08	0.76 ± 0.14	0.78 ± 0.18

absorption, social presence and enjoyment, demonstrating that the affective NPCs are significantly more appealing than the emotionless NPCs. By expressing the NPC's affective status explicitly, the emotion models have enhanced the NPCs with a higher degree of interactivity and improved the first person shooting game in terms of playability. In addition, comparing with their counterpart using EMA rules, the NPCs using CRAA model obtain significantly higher scores in all of the three dimensions.

6.6.6 Emotion Pattern Analysis

We conduct studies to analyze the emotion patterns produced by the affective NPCs during the experiments. Figure 6.7 shows the intensity patterns of six emotions produced by CRAA over a period of 100 time steps during the game play. The onsets of events, specifically "kill enemy", "killed by enemy", "damage enemy", "damaged by enemy", "pick up weapon", and "pick up health", are denoted by the letters of K, k, D, d, W and H respectively at the bottom of the graph.

As shown in Figure 6.7, each emotion started with a baseline intensity of zero at the beginning of the trial, which increased at the moment when an event occurred. For example, at $t = 3$, the NPC picked up a new weapon, all the six emotions were aroused but the intensity of happiness was much higher than the others. Consequently, the emotion of "Happy" was expressed by the NPC. After the NPC obtained two new weapons consecutively at $t = 3$ and $t = 4$, the intensities of emotions began to decay toward zero until the next happening of events. When the NPC encountered an enemy and they began to fight, we see "Happy" increased again at $t = 17$ when the NPC damaged its enemy, and "Disgust" was activated when the NPC was damaged by enemy at $t = 18$. As the NPC continued to be

damaged by the enemy, "Happy" decayed to much lower than "Disgust". However, when the NPC killed the enemy at $t = 31$, the intensity of "Happy" shot up and had a higher intensity than ever. At $t = 98$, when the NPC was unexpectedly killed by the enemy, the emotion of "Disgust" was triggered and its intensity reached the highest level. At the same time, the level of "Surprise" was also very high. This is because the NPC had been performing well and had a high expectation and high self-appraisal of its own capability (power). Therefore, the outcome of being killed was totally out of its expectation.

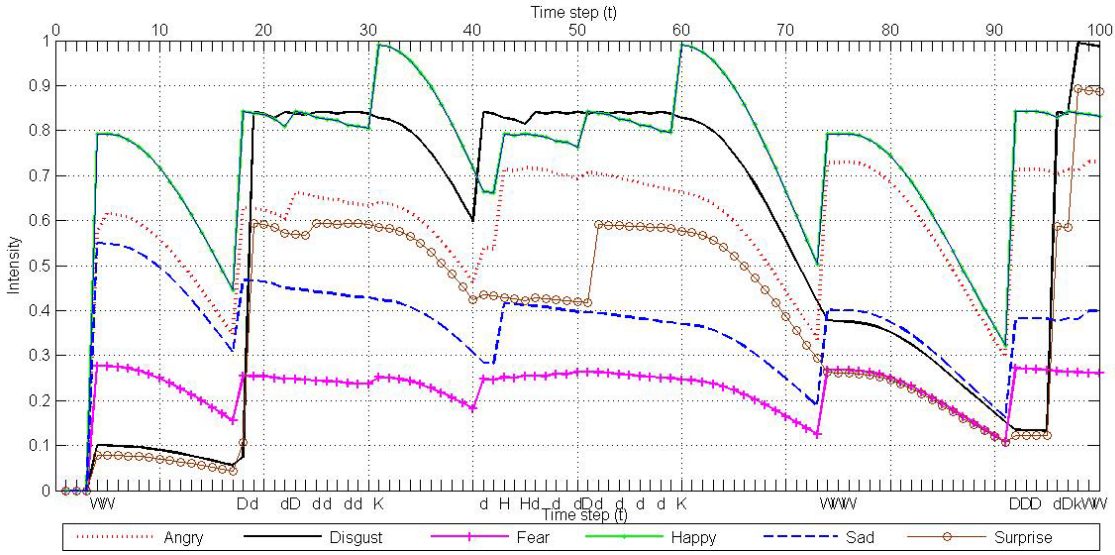

Figure 6.7: Emotion signals recorded from CRAA.

For the purpose of comparison, we also examine the emotion patterns displayed by affective NPCs based on a set of emotion categorization and intensity rules as shown in Table 6.10 [51]. These rules, which were used in an earlier version of EMA, need to be evaluated repeatedly against the occurrence of each of the emotional events. The results are depicted in Figure 6.8, which shows the intensity variation of the six emotions as used in EMA over a period of 100 time steps. EMA adopts a

Table 6.10: Emotion appraisal rules used in EMA.

Appraisal configuration	*Emotion*	*Intensity*
$desirability(p) > 0, likelihood(p) < 1$	Hope	$desirability(p) * likelihood(p)$
$desirability(p) > 0, likelihood(p) = 1$	Joy	$desirability(p) * likelihood(p)$
$desirability(p) < 0, likelihood(p) < 1$	Fear	$\|desirability(p) * likelihood(p)\|$
$desirability(p) < 0, likelihood(p) = 1$	Distress	$\|desirability(p) * likelihood(p)\|$
$desirability(p) < 0$, $causal\ attribution(q) = blameworthy$	Anger	$\|desirability(p) * likelihood(p)\|$
$desirability(q) < 0$, $causal\ attribution(p) = blameworthy$, $causal\ agent = p$	Guilt	$\|desirability(p) * likelihood(p)\|$

slightly different set of emotions, which includes "Hope", "Joy", "Fear", "Distress", "Anger", and "Guilt". We see that EMA is generally able to provide appropriate responses. The emotion "Guilt" was triggered when the agent damaged the enemy, "Joy" was triggered when the agent obtained a weapon, and "Anger" happened when the agent was damaged by enemy. However, the change of emotions in EMA is significantly more abrupt. This is in contrast to that in CRAA, which leverages on the continuity property of neural networks to provide finer-grained emotion outputs. More importantly, EMA does not allow activation of multiple emotion instances at the same time. This limits its ability in producing subtle responses to complex situations.

Table 6.11 summarizes some of the NPC's emotional responses with the feedback reinforce signals, showing that the emotion model could emulate human emotional responses under different reinforcers. For example, a negative or positive reinforcer used to strength the behavior will cause a negative or positive emotion respectively [139].

A human-based evaluation is conducted to examine whether the emotion patterns produced by CRAA could emulate real human emotions. The users are requested to

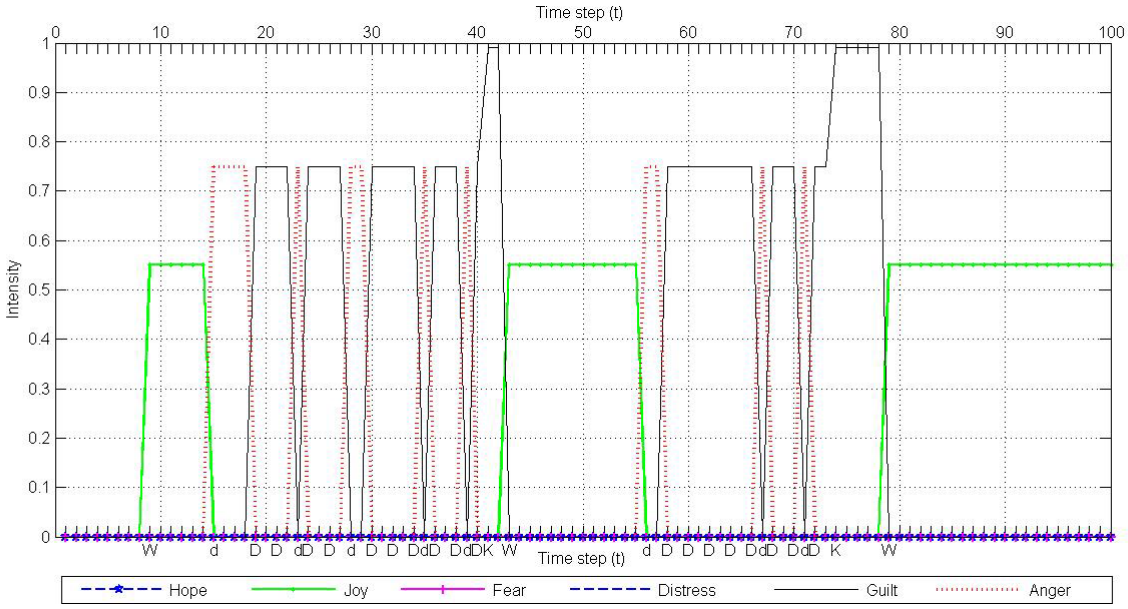

Figure 6.8: Intensities of various emotion signals produced by the emotion rules used in EMA.

estimate the intensity of six emotions after viewing the clips of trial video, imaging they are playing the game. Figure 6.9 shows the intensity of the six emotions by averaging over twenty human users estimations over the same period of 100 time steps during the game play as in Figure 6.7. The onsets of events, specifically "kill enemy", "killed by enemy", "damage enemy", "damaged by enemy", "pick up weapon", and "pick up health", are denoted by the letters of K, k, D, d, W and H respectively at the bottom of the graph. Figure 6.9 demonstrates that the human estimated emotions show similar patterns as produced by CRAA in Figure 6.7. For example, each emotion started with a baseline intensity of zero at the beginning of the trial and increased when an event occurred. Human users consider the emotion "Happy" has the highest intensity at $t = 3$ when a new weapon was picked up and the emotion "Disgust" was activated when it was damaged by the enemy at $t = 18$. Later on, the emotion "Happy" reaches the highest intensity at $t = 31$ when

Table 6.11: Emotion caused by external stimulus which reinforce the behaviors

Time	*Event*	*Reinforcer*	*Emotion (positive/negative)*	*Intensity*
$t = 3$	Pick up weapon	A better weapon (positive)	Happy (positive)	0.79
$t = 17$	Damage enemy	Enemy bleeding (positive)	Happy (positive)	0.84
$t = 18$	Damaged by enemy	Bleeding (negative)	Disgust (negative)	0.84
$t = 31$	Kill enemy	Enemy dead (positive)	Happy (positive)	0.82
$t = 98$	Killed by enemy	Dying (negative)	Disgust (negative)	0.99

the enemy was killed. Moreover, at $t = 98$, "Surprise" was estimated to be the most activated emotion when "killed by enemy" happens which is out of the users expectation.

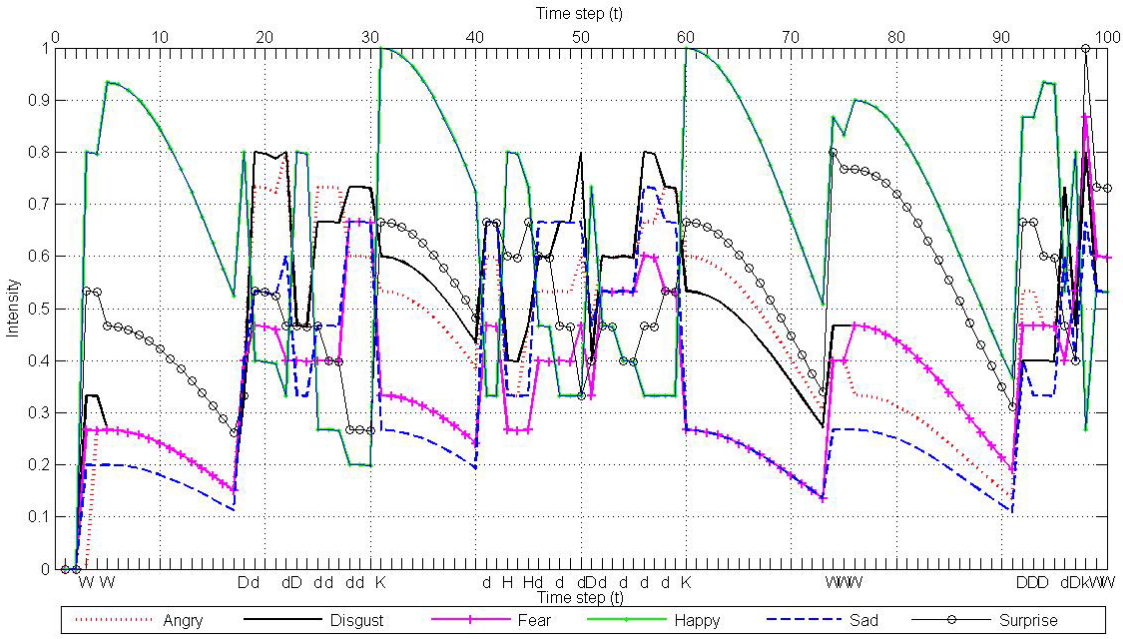

Figure 6.9: Human evaluations for intensities of various emotion signals.

6.7 Summary

Based on the theory of cognitive regulated emotion, this chapter has proposed a biologically inspired computational model called Cognitive Regulated Affective Ar-

chitecture (CRAA) with a multi-network architecture. Specifically, CRAA consists of a Cognitive Network, an Appraisal Network and an Affective Network, designed to simulate the functions and interactions of prefrontal cortex (PFC), anterior cingulate cortex (ACC) and amygdala (AMYG) respectively. As a natural extension of reinforcement learning, this architecture shows how appraisal signals produced by a cognitive system may be used in generating emotional responses by an affective system.

The CRAA model has been integrated into autonomous Non-Player Characters (NPCs) for playing in a first person shooting game. Our questionnaire based experiments shows that NPCs with emotions are significantly more appealing comparing with emotionless NPCs in terms of cognitive absorption, social presence and enjoyment.

Our work thus far has shown the feasibility of an integrated neural system for cognitive appraisal, decision making, reinforcement learning and emotion generation. Going forward, we aim to enhance the capability of the emotion model by expanding the set of emotions and taking into consideration the intricate relations and interaction among emotions. For developing realistic and believable agents, we also aim to further develop the integrated cognitive-affective architecture with more complete high level cognitive functions, such as self-awareness, goal maintenance, and planning.

Chapter 7

Conclusions

7.1 Summary of Contributions

Modeling of NPCs is an attractive topic crucial for the success of computer games. Technically, it is essentially the problem of creating autonomous agents, which are expected to function and adapt by themselves in a complex and dynamic environment. To create "realistic" and "believable" NPCs [127], this research has focused on the issues of learning behavior models from user patterns, behavior adaptation during play time, and emotion modeling.

Specifically, the first part of this research has focused on creating the autonomous players via learning from user patterns. A specific class of self-organizing neural networks, namely Self-Generating Neural Networks (SGNN), is applied in behavior learning task. Compared with traditional neural networks, SGNN does not require a designer to determine the structure of the network according to the particular application in hand. However, the computational time of SGNN increases dramatically due to the continual creation of neural nodes. To solve this problem, we have proposed a pruning method to optimize the SGNN network while maintaining the learning performance. We have conducted benchmark experiments to examine S-GNN method and the pruning method in the aspects of generalization capability,

learning efficiency, and computational cost. For the application of behavior rule learning, a novel rule extraction method is further proposed to extract useful knowledge from SGNN and translate them into symbolic rules. Online testing of NPC is conducted in a first person shooting game, known as Unreal Tournament 2004 (UT2004) with Deathmatch scenario. The online testing result demonstrates that SGNN can be used to build autonomous players by learning the behaviour patterns of sample bots. We also compare the behavior learning ability of SGNN (without pruning) Bot, SGNN (with pruning) Bot, and SGNN Rule Bot.

Beyond SGNN, another class of self-organizing neural networks, namely Fusion Architecture for Learning, COgnition, and Navigation (FALCON), is also applied to building autonomous players by learning the behaviour patterns of sample bots in Unreal Tournament. We have conducted benchmark experiments to compare FALCON with SGNN (in Chapter 3) in various aspects, including generalization capability, learning efficiency, and computational cost, under the same set of learning conditions. Our benchmark experiments show that, comparing with SGNN, FALCON learns faster and produces a higher level of generalization capability with a much smaller set of nodes. Online testing of NPC in the Deathmatch scenario also confirms that FALCON Bot produces a similar level of fighting competency to the Hunter Bot, which is not matched by bots based on SGNN. It has also been demonstrated that as FALCON is designed to support a myriad of learning paradigms, including unsupervised learning, supervised learning and reinforcement learning, it is our natural choice for modeling autonomous NPCs in games.

In order to improve the NPCs adaption and learning abilities, this research further complements the fast learning capability of imitative learning with the continual adaptability of reinforcement learning for a better performance. Specifically, the

Dual-Stage Learning (DSL) and the Mixed Model Learning (MML) strategies are presented to realize their integration of the two learning paradigms into one framework based on FALCON. The DSL learning strategy comprises two learning stages. In the imitative learning stage, FALCON learns prior knowledge to built the initial behaviour model of an autonomous agent. In the reinforcement learning stage, the agent further adapts to the environment through Q-learning and by applying the prior knowledge for exploitation. The MML learning strategy combines the two learning paradigms into one process, in which imitative learning and reinforcement learning work simultaneously by activating their corresponding input fields while sharing a common knowledge field. DSL and MML have been applied to creating autonomous non-player characters (NPCs) in a first person shooting game named Unreal Tournament. Our experiments have shown that both DSL and MML are effective in enhancing learning ability in terms of increasing learning speed and accelerating the convergence. The NPCs built by DSL and MML also produce better performance comparing with the traditional reinforcement learning and imitative learning methods.

The last and not the least, this research has studied the development of the "realistic" and "believable" agents [127], by considering the integration of cognitive functions with affective characteristics. Based on the appraisal theory, the Adaptive Resonance Theory (ART) [27], and the theory of cognitive regulated emotion [104], this research has proposed a biologically inspired computational model called Cognitive Regulated Affective Architecture (CRAA) with a multi-network architecture. Specifically, CRAA consists of a Cognitive Network, an Appraisal Network and an Affective Network, designed to simulate the functions and interactions of prefrontal cortex (PFC), anterior cingulate cortex (ACC) and amygdala (AMYG) respectively.

As a natural extension of reinforcement learning, this architecture shows how appraisal signals produced by a cognitive system may be used in generating emotional responses by an affective system.

The CRAA model has been integrated into autonomous Non-Player Characters (NPCs) for playing in Unreal Tournament. Our experimental results have shown that CRAA demonstrates a reasonably good level of accuracy in emotion modeling based on human assessment. Furthermore, the questionnaire based experiments also show that NPCs with emotions are significantly more appealing comparing with emotionless NPCs in terms of cognitive absorption, social presence and enjoyment.

Our work thus far has shown the feasibility of an integrated neural system for cognitive appraisal, decision making, reinforcement learning and emotion generation. The techniques within this system including autonomous behavior, adaption and decision making, and affective abilities could be applied in many application areas spanning education, training, human-computer interfaces and entertainment.

7.2 Future Work

Going forward, for developing realistic and believable agents, we aim to further develop the integrated cognitive-affective architecture with more complete high level cognitive functions, such as self-awareness, goal maintenance, as well as planning, and more refined affective functions.

7.2.1 Self-Awareness

To "endow" NPCs the ability of self-awareness is one of the key works for developing humanoid NPCs. As humans are highly social animals and can't live without interacting with others, the self-awareness ability helps the human to identify oneself as

an individual separated from the environment and distinguish himself/herself from others. The ability of self-awareness is not innate but developed as a person growing up [118]. It helps a person to recognize his/her own actions from others', and let him/her to percept the relations between an action and its consequence. Research also finds that self-awareness enables a person to imagine an action and its result. Thus the people can aware of his/her intentions to take an action [17]. In the past, although many works modeled self-awareness in NPCs [79], very few of them integrate it with a cognitive-affective architecture. To incorporate the self-awareness ability in our integrated cognitive-affective architecture, one suggestion is to model a self-awareness module which has the cognitive functions of knowing the personal identity, imagining an action, and being aware of "intentions". The self-awareness module could be an add-on of the Cognitive Network in CRAA and should be able to communicate with the Appraisal Network. By interacting with the Cognitive Network and Appraisal Network, the self-awareness module could take a role in adjusting the behavior learning and modulating the emotion generation at the same time.

7.2.2 Goal Maintenance

Modeling the goal maintenance ability is also an interesting research. Goal is an essential element in modeling NPCs, as it assigns the NPCs what their behaviors should aim to. For NPCs, to maintain a goal, some particular actions should be activated when the maintenance condition is no longer met. Comparing with goal achievement, in which the goal could be dropped once it is completed successfully, goal maintenance enables an agent to continuously estimate the possible result by the actions it performs and make a further decision making [36]. In other words,

goal maintenance is more difficult and more complicated than goal achievement. So far, although in most situations, the NPCs are created as goal-oriented, there are still limited works contributing in goal maintenance [54] [36] [66]. The function of goal maintenance could be incorporated in our cognitive-affective architecture as a separate module. This module would senses the necessary information from the Cognitive Network and Affective Network and evaluates the current conditions. At the same time, the self-awareness module could generate intentions to the goal maintenance module. The goal maintenance module then sends these information to both the Cognitive Network and Affective Network. Finally, Cognitive Network makes the decisions and activates the necessary actions.

7.2.3 Planning

Another suggestion for future research is to modeling the planning ability for NPCs. Planning is closely related to goals and refers to the ability of specifying sequentiality of actions in order to achieve a goal. In a real-time game platform, planning could enhance the NPCs' abilities of finding a better behavior strategy and handling the unexpected situations. Besides for associating the situations with specific action sequence, planning is also supposed to be interacting with the self-awareness and goals and communicating with working memory in a real time manner. There are a lot of works contributing to planning problem for single BDI agents or multi-agents [64] [128] [80], whereas we suggest to focus on incorporating the planning ability in a cognitive-affective architecture to improve the believable and realistic NPCs. If planning could be realized as a module in CRAA, by communicating with the Cognitive Network and the Affective Network, it can contribute to adjusting the behavior performance and generating various emotions.

7.2.4 Refined Affective Modeling

We also aim to enhance the capability of the cognitive-affective architecture by expanding the set of emotions and taking into consideration the intricate relations and interaction among emotions. In Chapter 6, we have presented a solution to modulate the generation of emotions based on the Ekman's six basic emotions [37]. However, others have expanded the set of basic emotions to longer lists which may include secondary emotions or compound emotions [29]. Some also propose the notion of compound emotions are obtained by merging particular emotions [108, 34]. The development of high level cognitive functions is helpful to improve the emotion model from basic emotions to compound emotions as commonly the compound emotions are arisen from higher cognitive process such as shame, anxiety etc [29]. For example, the OCC theory includes self-awareness as one of the aspects in emotion appraisal. In some situations, by focusing on self agent instead of others, the emotions like pride or shame could be generated [108]. Moreover, the maintaining of goal has been widely included by multiple theories for the evaluation of non-basic emotions by adjudging the measurements which are highly correlated with the goals such as expectation [126] [120], desirability [126] [108] [120], praiseworthiness (viewed as being praiseworthy or blameworthy) [108] and so on. Planning is also considered in some emotion theories such as in Smith and Ellsworth's theory to evaluate the anticipated effort which is needed to deal with a situation [136], and in Frijda's theory to measure the certainty of happenings of an event [46]. These high level cognitive function modules such as self-awareness, goal maintenance, and planning are useful in evaluating extensive sets of emotions, and completing the functionality of the cognitive-affective architecture.

Appendix A

List of Publications

- S. Feng and A.-H. Tan. Self-Organizing Neural Networks for Behavior Modeling in Games. *In Proceedings of the International Joint Conference on Neural Networks*, Bercelona, Spain, 2010.

- S. Feng and A.-H. Tan. Learning Human Emotion Patterns for Modeling Virtual Humans. *In Proceedings of the Conference on Technologies and Applications of Artificial Intelligence (TAAI 2012)*, Taoyuan, Taiwan, November 11-13, 2011.

- S. Feng and A.-H. Tan. Biologically-Inspired Affective Model Based on Cognitive Situational Appraisal. *In Proceedings of the International Joint Conference on Neural Networks*, pp. 728-735, Brisbane, Australia, June 10-15, 2012.

- S. Feng and A.-H. Tan. Cognitive Regulated Affective Architecture: A Biologically-Inspired Approach. *Submitted to Neurocomputing.*

- S. Feng and A.-H. Tan. Autonomous Behavior Learning of Non-Player Characters in Games. *Submitted to Expert Systems with Applications.*

References

[1] http://www.csse.uwa.edu.au/cig08/.

[2] http://changingminds.org/explanations/theories/a_alphabetic.htm.

[3] D.H. Ackley and M.L. Littman. Generalization and scaling in reinforcement learning. *Advances in Neural Information Processing Systems 2*, pages 550–557, 1990.

[4] R. Agarwal and E. Karahanna. Time flies when you're having fun: cognitive absorption and beliefs about information technology usage1. *MIS Quarterly*, 24(4):665–694, 2000.

[5] J. Champandard Alex. *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors*. New Riders Games, 2003.

[6] A. Azadeh, M. Saberi, and Z. Jiryaei. An intelligent decision support system for forecasting and optimization of complex personnel attributes in a large bank. *Expert Systems with Applications*, 39(16):12358–12370, 2012.

[7] B. P. Baillie-De. *Programming Believable Characters for Computer Games(Game Development Series)*. Charles River Media, Inc., Rockland, MA, USA, 2004.

[8] O.F. Baker and S. Abdul-Kareem. Using genetic algorithm to evolves algebraic rule-based classifiers for NPC prognosis. In *Proceedings of International Conference on Intelligent and Advanced Systems*, pages 71–74, Piscataway, NJ, USA, 2008.

[9] R. J. S. Baker and P. I. Cowling. Bayesian opponent modeling in a simple poker environment. In *the 2007 IEEE Symposium on Computational Intelligence and Games (CIG 2007)*, Hawaii, 2007.

[10] J.F. Baldwin, T.P. Martin, and B.W. Pilsworth. *Fril-Fuzzy and Evidential Reasoning in Artificial intelligence*. John Wiley & Sons, Inc., 1995.

[11] P. Bard. Emotion: I. the neuro-humoral basis of emotional reactions. 1934.

[12] D. Barrios-Aranibar and P.J. Alsina. Recognizing behaviors patterns in a micro robot soccer game. In *Proceedings of Hybrid Intelligent Systems, Fifth International Conference on*, page 6, 2005.

[13] J. Bates. The role of emotion in believable agents. *Communications of the ACM*, 37(7):122–125, 1994.

[14] C. Bauckhage, C. Thurau, and G. Sagerer. Learning human-like opponent behavior for interactive computer games. In *Lecture notes in computer science*, pages 148–155, 2003.

[15] C. Becker-Asano and I. Wachsmuth. Affective computing with primary and secondary emotions in a virtual human. *Autonomous Agents and Multi-Agent Systems, AAMAS,*, 20:32–49, 2010.

[16] D. C. Berry, L. T. Butler, and F. de Rosis. Evaluating a realistic agent in an advice-giving task. *International Journal of Human-Computer Studies*, 65:304–327, 2005.

[17] S. J. Blakemore and C. Frith. Self-awareness and action. *Current Opinion in Neurobiology*, 13:219–224, 2003.

[18] B.M. Blumberg and T.A. Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. In *Computer Graphics Proceedings. SIGGRAPH 95*, pages 47–54, New York, NY, USA, 1995.

[19] A. Brodal. *Neurological anatomy.* New York: Oxford University Press, 1982.

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

[21] W. B. Cannon. The james-lange theory of emotions: A critical examination and an alternative theory. *The American Journal of Psychology*, 39(1/4):106–124, 1927.

[22] G. A. Carpenter and S. Grossberg. ART 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930, July 1987.

[23] G. A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, June 1987.

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

[25] G. A. Carpenter and A.-H. Tan. Rule extraction: From neural architecture to symbolic representation. *Connection Science*, 7(1):3–27, 1995.

[26] G.A. Carpenter and S. Grossberg, editors. *Pattern Recognition by Self-Organizing Neural Networks.* Cambridge, MA: MIT Press, 1991.

[27] G.A. Carpenter and S. Grossberg. Adaptive Rresonance Theory. In M.A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 87–90. Cambridge, MA: MIT Press, 2003.

[28] Se-Hyung Cho, Ye-Hoon Kim, In-Won Park, and Jong-Hwan Kim. Behavior selection and memory-based learning for artificial creature using two-layered confabulation. In *Proceedings of The 16th IEEE International Symposium on Robot and Human interactive Communication*, pages 992–997, Aug. 2007.

[29] A. Damasio. *Descartes's Error: Emotion, Reason and the Human Brain*. NY: Avon Books., New York, 1994.

[30] F. Dan and Inc. Ryan Houlette-Stottler Henke Associates. The ultimate guide to FSMs in games. *AI Game Programming Wisdom2,Charles River Media*, pages 283–302, 2004.

[31] M. B. David and Glenn S. *AI for Game Developers*. O'Reilly Media, 2004.

[32] A. De Lucia, R. Francese, I. Passero, and G. Tortora. Development and evaluation of a virtual campus on second life: The case of secondDMI. *Computers and Education*, 52(1):220–233, 2009.

[33] J. Demiris and G. Hayes. Imitative learning mechanisms in robots and humans. In *proceedings, of the 5th European Workshop on Learning Robots*, pages 9–16, Bari, Italy, 1996.

[34] J. Dias and A. Paiva. Feeling and reasoning: A computational model for emotional characters. In *proceedings, 12th Portuguese Conference on Artificial Intelligence, EPIA 2005*, Covilha, Portugal, 2005.

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

[36] S. Duff, J. Thangarajah, and J. Harland. Maintenance goals in intelligent agents. *Masters Thesis, Department of Computer Science, RMIT*, August, 2009.

[37] P. Ekman. An argument for basic emotions. *Cogn Emotion*, 6:169–200, 1992.

[38] M.S. El-Nasr, J. Yen, and T.R. Ioerger. FLAME - fuzzy logic adaptive model of emotions. *Autonomous Agents and Multi-Agent Systems*, 3(3):219–257, 2000.

[39] C. Elliott, J. Rickel, and J. Lester. Integrating affective computing into animated tutoring agents. In *Fifteenth International Joint Conference on Artificial Intelligence -Animated Interface Agents Workshop*, pages 113–121, 1997.

[40] Y. Eric. Finite machne scripting language for designers. *AI Game Programming Wisdom2*, pages 319–325, 2004.

[41] A. Etkin, T. Egner, and R. Kalisch. Emotional processing in anterior cingulate and medial prefrontal cortex. *Trends in cognitive sciences*, 15(2):85–93, 2011.

[42] R. Evans. AI in games: A personal view. *available online at www.gameai.com/blaceandwhite.html*, 2001.

[43] S. Feng and A.-H. Tan. Self-organizing neural networks for behavior modeling in games. In *proceedings, International Joint Conference on Neural Networks*, pages 3649–3656, Barcelona, Spain, July 2010.

[44] A. Fink, J. Denzinger, and J. Aycock. Extracting NPC behavior from computer games using computer vision and machine learning techniques. In *Proceedings of 3rd IEEE Symposium on Computational Intelligence and Game*, page 8, Piscataway, NJ, USA, 2007.

[45] K. Framling. Guiding exploration by pre-existing knowledge without modifying reward. *Neural Networks*, 20:736–747, 2007.

[46] N. Frijda. Emotion, cognitive structure, and action tendency. *Cognition and Emotion*, 1:115–143, 1987.

[47] P. Gaussier, S. Moga, J.P. Banquet, and M. Quoy. From perception-action loops to imitation precesses: A bottom-up approach of learning by imitation. *Applied Artificial Intelligence*, 7-8(12):701–727, 1998.

[48] P Gebhard. Alma - a layered model of affect. In *proceedings, the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 29–36, Utrecht, 2005.

[49] R. Gilberto. Implementing a data-driven finite-state machine. *AI Game Programming Wisdom2*, pages 307–317, 2004.

[50] B. Gorman, C. Bauckhage, and M. Humphrys. Bayesian imitation of human behavior in interactive computer games. In *proceedings, 18th International Conference on Pattern Recognition*, volume 1, pages 1244–1247, 2006.

[51] J. Gratch and S Marsella. A domain independent framework for modeling emotion. *Jouranl of Cognitive Systens Research*, 5(4):269–306, 2004.

[52] J. Gratch and S. Marsella. Evaluating the modeling and use of emotion in virtual humans. In *Proceedings of the Third International Joint Conference on, Autonomous Agents and Multiagent Systems*, 2004.

[53] Robert Hecht-Nielsen. The mechanism of thought. In *Proceedings of International Joint Conference on Neural Networks*, pages 419–426. IEEE, 2006.

[54] K. V. Hindriks and M. B. Van Riemsdijk. Satisfying maintenance goals. *Declarative Agent Languages and Technologies*, pages 86–103, 2008.

[55] K.H.L. Ho. A model of fuzzy emotion and behaviour selection for an autonomous mobile robot. In *Proceedings. 6th IEEE International Workshop on Robot and Human Communication. RO-MAN '97 Sendai*, pages 332–337, New York, NY, USA, 1997.

[56] Pogamut homepage is available online. http://artemis.ms.mff.cuni.ca/pogamut/.

[57] H. Inoue and H. Narihisa. Effect of parallel ensembles to self-generating neural networks for chaotic time series prediction. In *Proceedings of the IEEE Signal Processing Society Workshop*, volume 2, pages 896–905, Piscataway, NJ, USA, 2000.

[58] H. Inoue and H. Narihisa. Efficiency of self-generating neural networks applied to pattern recognition. *Mathematical and Computer Modelling*, 38(11-13):1225–1232, 2003.

[59] H. Inoue and H. Narihisa. Efficient pruning method for ensemble self-generating nerual networks. *Journal of Systemic, Cybernetics and Informatics*, 1(6):72–77, 2003.

[60] H. Inoue and H. Narihisa. Effective online pruning method for ensemble self-generating neural networks. In *Proceedings of Midwest Symposium on Circuits and Systems*, volume 3, pages 85–88, Hiroshima, Japan, 2004.

[61] H. Inoue and H. Narihisa. Self-organizing neural grove and its applications. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 1205–1210, Montreal, QC, Canada, 2005.

[62] W. James. Ii. what is an emotion? *Mind*, (34):188–205, 1884.

[63] N. Jennings and C. Collins. Virtual or virtually u: Educational institutions in second life. *International Journal of Social Sciences*, 2(3):180–186, 2007.

[64] T. Jonathan and P. Julie. Efficient intent-based narrative generation using multiple planning agents. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, May, 2013.

[65] C. U. Joy and C. Nkwachukwu. Learning paradigms for game artificial intelligence. *Academic Research*, 1, 2011.

[66] G. A. Kaminka, A. Yakir, D. Erusalimchik, and N. Cohen-Nov. Towards collaborative task and team maintenance. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007.

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

[68] In-Cheol Kim. Utbot: a virtual agent platform for teaching agent system design. *Journal of Multimedia*, 2(1):48–53, 2007.

[69] J.-H. Kim, S.-H. Cho, Y.-H. Kim, and I.-W. Park. Two-layered confabulation architecture for an artificial creature's behavior selection. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(6):834–840, Nov. 2008.

[70] J.-H. Kim, W.-R. Ko, J.-H. Han, and S. A. Zaheer. The degree of consideration-based mechanism of thought and its application to artificial creatures for behavior selection. *Computational Intelligence Magazine, IEEE*, 7(1):49–63, 2012.

[71] J.H. Kim, Y.D.and Kim and Y.J. Kim. Behavior selection and leaning for synthetic character. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 898–903, Portland, USA, Jun. 2004.

[72] Y.D. Kim, Y.J. Kim, J.H. Kim, and J.R. Lim. Implementation of artificial creature based on interactive learning. In *proceedings of 2002 FIRA Robot World Congress*, pages 369–373, 2002.

[73] W.-R. Ko, H.-S. Hyun, H.-J. Kim, S.-H. Choi, and J.-H. Kim. Behavior selection method for intelligent artificial creatures using the degree of consideration-based mechanism of thought. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 2461–2467, 2011.

[74] R. Kötter and N. Meyer. The limbic system: a review of its empirical foundation. *Behavioural brain research*, 52(2):105–127, 1992.

[75] Y. Kuniyoshi and H. Inoue. Qualitative recognition of ongoing human action sequences. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1600–1609, 1993.

[76] K.D. Kwon and K.R. Park. Behavior control of intelligent multi NPCs using vickrey auction system and hierarchical finite state machine. In *Proceedings of International Joint Conference SICE-ICASE*, pages 3821–3826, Oct 2006.

[77] J.E. Laird. It knows what you're going to do: adding anticipation to a quakebot. In *Proceedings of the fifth international conference on Autonomous agents*, pages 385–392. ACM, 2001.

[78] J.E. Laird and M. van Lent. Human level AI's killer application: interactive computer games. *AI Magazine*, 22(2):15–25, 2001.

[79] P. Lankoski and S. Bjork. Gameplay design patterns for believable non-player characters. In *proceedings, 2007 Digital Games Research Association Conference*, pages 416–423, 2007.

[80] D. S. Lavindra, D. Anthony, and H. James. Planning with time limits in bdi agent programming languages. In *Proceedings of the thirteenth Australasian symposium on Theory of computing*, volume 65, January, 2007.

[81] J. LeDoux. The emotional brain, fear, and the amygdala. *Cellular and molecular neurobiology*, 23(4-5):727–738, 2003.

[82] J. E. LeDoux and E. A. Phelps. Emotional networks in the brain. *Handbook of emotions*, pages 109–118, 2010.

[83] S. C. Lee, E. Lee, W. Choi, and U. M. Kim. Extracting temporal behavior patterns of mobile user. In *Proceedings of Fourth International Conference on Networked Computing and Advanced Information Management*, pages 455–462, Sep. 2008.

[84] P. Leong and Miao Chunyan. Fuzzy cognitive agents in shared virtual worlds. In *Proceedings of International Conference on Cyberworlds*, pages 23–25, Los Alamitos, CA, USA, 2006.

[85] A. G. Li, Y. Huang, and Z. H. Li. Iteration learning sgnn. In *Proceedings of International Conference on Neural Networks and Brain Proceedings, ICNNB'05*, volume 3, pages 1912–1916, Beijing, China, 2005.

[86] A.G. Li. Fast photo time-stamp recognition based on sgnn. In *Proceedings of Third International Symposium on Neural Networks. (Lecture Notes in Computer Science Vol.3972)*, pages 316–321, Berlin, Germany, 2006.

[87] A.G. Li, D. Qiu, and Z. Li. Predicting time between software failures using isgnn. *(IJCSNS) International Journal of Computer Science and Network Security*, 6(6):116–118, June 2006.

[88] J. Liu, B. Kuipers, and S. Savarese. Recognizing human actions by attributes. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3337–3344, 2011.

[89] P. D. MacLean. Psychosomatic disease and the" visceral brain" recent developments bearing on the papez theory of emotion. *Psychosomatic Medicine*, 11(6):338–353, 1949.

[90] R. P. Marinier and J. E. Laird. A computational unification of cognitive behavior and emotion. *Cognitive Systems Research*, 10(1):48–69, 2009.

[91] Y. Marom, G.M. Maistros, and G. Hayes. Experiments with a social learning model. *Adaptive behavior*, 9(7):209–240, 2001.

[92] S. Marsella and J. Gratch. Ema: A computational model of appraisal dynamics. In *proceedings, European Meeting on Cybernetics and Systems Research*, 2006.

[93] S.C. Marsella and J. Gratch. EMA: A process model of appraisal dynamics. *Cognitive Systems Research*, 10(1):70–90, 2009.

[94] M. McPartland and M. Gallagher. Learning to be a bot: Reinforcement learning in shooter games. In *Artificial Intelligence in Interactive Digital Entertainment (AIIDE)*, Stanford, USA, 2008.

[95] H. Miyamoto and M. Kawato. A tennis serve and upswing learning robot based on bi-directional theory. *Neural Networks*, 11(7/8):1331–1344, 1998.

[96] J. Moren and C. Balkenius. A computational model of emotional learning in the amygdala: From animals to animals. In *procedings, 6th International conference on the simulation of adaptive behavior*, Cambridge, Mass, 2000. The MIT Press.

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

[98] L. Morgado and G. Gaspar. Emotion in intelligent virtual agents: The flow model of emotion. *Intelligent Virtual Agents*, pages 31–38, 2003.

[99] E. A. Murray. The amygdala, reward and emotion. *Trends Cogn Sci*, 11:489–497, 2007.

[100] F. Nah, B. Eschenbrenner, and D. DeWester. Enhancing brand equity through flow and telepresence: A comparison of 2D and 3D virtual world. *MIS Quarterly*, 35(3):731–748, 2011.

[101] W. S. Neal Reilly. Modeling what happends between emotioanl antecedents and emotional consequents. In *proceedings, Eighteenth European Meeting on Cybernetics and Systems Research*, pages 607–612, Vienna, Austria, 2006.

[102] I. Noda. Hierarchical hidden markov modeling for teamplay in multiple agents. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 38–45, Oct 2003.

[103] K. L. Nowak and F. Biocca. The effect of the agency and anthropomorphism of users' sense of telepresence, copresence, and social presence in virtual environments. *Presence*, 12(5):481–494, 2003.

[104] K. N. Ochsner and L.F. Barrett. A multi-process perspective on the neuroscience of emotion. *G. Bonnano and T.J. Mayne (Eds.) Emotion: Current issues and future directions*, pages 38–81, 2001.

[105] K. N. Ochsner and J. J. Griss. The cognitive control of emotion. *TRENDS in Cognitive Science*, 9(5):242–249, 2005.

[106] K.N. Ochsner, S.A. Bunge, J.J. Gross, and J.D.E. Gabrieli. Rethinking feelings: an FMRI study of the cognitive regulation of emotion. *Cogn. Neurosci.*, 14:1215–1229, 2002.

[107] J. Orkin. Symbolic representation of game world state: Toward real-time planning in games. In *proceedings, AAAI Challenges in Game AI Technical Report*, pages 26–30, 2004.

[108] A. Ortony, G. Clore, and A. Collins. *The Cognitive Structure of Emotions*. Cambridge Univ. Press, Cambridge, U.K., 1998.

[109] A. Ortony and T. Turner. What's the basic about basic emtoions? *Psychological Review*, 97(3):315–331, 1990.

[110] J. W. Papez. A proposed mechanism of emotion. *Archives of neurology and psychiatry*, 38(4):725, 1937.

[111] T. Paul. Stack-based finite-state machines. *AI Game Programming Wisdom2*, pages 303–304, 2004.

[112] L. Pessoa. On the relationship between emotion and cognition. *Nature Reviews Neuroscience*, 9(2):148–158, Feb 2008.

[113] R. W. Pew, A. S. Mavor, et al. *Modeling Human and Organizational Behavior: Applications to Military Simulations*. National Academy Press, 1998.

[114] R. Picard. *Affective Computing*. MIT Press, 1997.

[115] D. D. Price and J. J. Barrell. Some general laws of human emotion: Interrelationships between intensities of desire, expectation, and emotional feeling. *Journal of Personality*, 52(4):389–409, 1984.

[116] A. Prieditis. Applying model-based decision-making methods to games: applying the locust AI engine to Quake 3. *Game Programming Gems 6, Charles River Media*, 2006.

[117] S. Rabin. Implementing a state machine language. *AI Game Programming Wisdom, Charles River Media*, 2002.

[118] P. Rochat. Five levels of self-awareness as they unfold early in life. *Consciousness and Cognition*, 12:717–731, 2003.

[119] E.T. Rolls. Brain mechanisms of emotion and decisionmaking. *Int Congr Ser*, 1291:3–13, 2006.

[120] I. J. Roseman, P. E. Jose, and M. S. Spindel. Appraisals of emotion-eliciting events: testing a theory of discrete emotions. *J. Perso. Soc. Psychol*, 59(5):899–915, 1990.

[121] D. Rousseau. Personality in computer characters. In *In working Notes of the AAAI-96 Workshop on AI/Alife*, Menlo Park, CA, 1996. AAAI Press.

[122] G.A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, Cambridge University, 1994.

[123] S. J. Russel and P. Norvig. Prentice Hall, Englewood Cliffs. NJ. 1995.

[124] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

[125] S. Schachter and J. E. Singer. Cognitive, social and physiological determinants of emotional states. *Expert Systems with Applications*, 69:379–399, 1962.

[126] K. R. Scherer. Appraisal considered as a process of multilevel sequential checking. *Appraisal processes in emotion: Theory, method, research*, pages 92–120, 2001.

[127] M. Scheutz, A. Sloman, and B. Logan. Emotional states and realistic agent behaviour. In *GAME-ON*, Nov, 2000.

[128] S. Sebastian, D. S. Lavindra, and P. Lin. Hierarchical planning in bdi agent programming languages: a formal approach. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, May, 2006.

[129] D. Shapiro, P. Langley, and R. Shachter. Using background knowledge to speed reinforcement learning in physical agents. In *the 5th International Conference on Autonomous Agents.*, Montreal, Quebec, Canada., 2001.

[130] M.A. Sharbafi, C. Lucas, O.A. Haghighat, A.T. Ghiasvand, and O. Aghazade. Using emotional learning in rescue simulation environment. *Transactions on Engineering, Computing and Technology*, 13(1):333–337, May, 2006.

[131] M. Si, S.C. Marsella, and D.V. Pynadath. Modeling appraisal in theory of mind reasoning. *Journal of Autonomous Agents and Multi-Agent Systems*, 20:14–31, 2010.

[132] D Silver. Ai game programming wisdom 3. 2006.

[133] P. J. Silvia. Looking past pleasure: Anger, confusion, disgust, pride, surprise, and other unusual aesthetic emotions. *Psychology of Aesthetics Creativity and the Arts*, 3(1):48–51, 2009.

[134] P. J. Silvia and E. M. Brown. Anger, disgust, and the negative aesthetic emotions: Expanding an appraisal model of aesthetic experience. *Psychology of Aesthetics, Creativity, and the Arts*, 1(2):100–107, 2007.

[135] A.J. Smith. Application of the self-organising map to reinforcement learning. *Neural Networks*, 15(8-9):1107–1124, 2002.

[136] C. A. Smith and P. C. Ellsworth. Patterns of cognitive appraisal in emotion. *Journal of Personality and Social Psychology*, 48:813–838, 1985.

[137] C.A. Smith and R. S. Lazarus. Emotion and adaptation. *Handbook of Personality: theory and research*, pages 609–637, 1990.

[138] M. Song, G. Gu, and R Zhang. Behavior control of multi-robot using the prior- knowledge based reinforcement learning. In *the 5m World Congress on Intelligent Control and Automation*, Hangzhou, P.R. China, June, 2004.

[139] A. W. Staats. Positive and negative reinforcers: How about the second and third functions? *Behavior Analyst*, 29(2):271–272, 2006.

[140] A. Staller and P. Petta. Introducing emotions into the computational study of social norms. In *Proceedings of 2000 Convention of the Society for the Study of Artificial Intelligence and the Simulation of Behaviour (AISB'00)*, pages 17–20, Birmingham, UK, 2000.

[141] R.S Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[142] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press, 1998.

[143] P. Sweetser. How to build neural networks for games. *AI Game Programming Wisdom2*, pages 615–625, 2004.

[144] P. Sweetser. Strategic decision-making with neural networks and influence maps. *AI Game Programming Wisdom2*, pages 439–446, 2004.

[145] A.-H. Tan. Adaptive Resonance Associative Map. *Neural Networks*, 8(3):437–446, 1995.

[146] A.-H. Tan. Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing. *IEEE Transaction on Neural Networks*, 8(2):237–250, 1997.

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

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

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

[150] A.-H. Tan and Y. Kang. Agent-augmented co-space: Toward merging of real world and cyberspace. In *B. Xie et al. (Eds.): ATC 2010, LNCS 6407*, pages 298–312. Springer, Heidelberg, 2010.

[151] A.-H. Tan, N. Lu, and D. Xiao. Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback. *IEEE Transactions on Neural Networks*, 9(2):230–244, 2008.

[152] C. Tang, C. Zhou, H. Hu, W. Pan, and L. Xie. A semi-supervised learning system for service robots to recognise human actions. *Advanced Robotics*, 28(13):907–918, 2014.

[153] B. Tastan and G. Sukthankar. Learning policies for first person shooter games using inverse reinforcement learning. In *Seventh Artificial Intelligence in Interactive Digital Entertainment (AIIDE)*, Stanford, USA, 2011.

[154] T.-H. Teng, A.-H. Tan, W.-S. Ong, and K.-L. Lee. Adaptive cgf for pilots training in air combat simulation. In *Information Fusion (FUSION), 2012 15th International Conference on*, pages 2263–2270, 2012.

[155] T.-H. Teng, A.-H. Tan, and A. Yeo. Self-organizing neural networks for learning air combat maneuvers. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8, 2012.

[156] T. Thompson, J. Levine, and G. Hayes. Evotanks: Co-evolutionary development of game-playing agents. In *the 2007 IEEE Symposium on Computational Intelligence and Games (CIG 2007)*, pages 328–333, Hawaii, 2007.

[157] T. Unemi. Scaling up reinforcement learning with human knowledge as an intrinsic behavior. In *Scaling up reinforcement learning with human knowledge as an intrinsic behavior*, pages 511–518, 2000.

[158] D. Wang, B. Subagdja, A.-H. Tan, and G. W. Ng. 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*, pages 14–16, Pasadena,California, July 2009.

[159] S. Wanitchaikit, P. Tangamchit, and T. Maneewarn. Self-organizing approach for robot's behavior imitation. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 3350–3355, May 2006.

[160] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3/4):279–292, 1992.

[161] W.X. Wen, H. Liu, and A. Jennings. Self-generating neural networks. In *Proceedings of International Joint Conferencen on Neural Networks*, volume 4, pages 850–855, June 1992.

[162] W.X. Wen, V. Pang, and A. Jennings. Self-generating vs. self-organizing, what's different? In *Proceedings of IEEE International Conference on Neural Networks*, pages 1469–1473, New York, NY, USA, 1993.

[163] S. Wermter, M. Elshaw, C. Weber, C. Panchev, and H. Erwin. Towards integrating learning by demonstration and learning by instruction in a multimodal robot. In *Proceeding of Workshop on Robot Learning by Demonstration*, pages 72–79, October 2003.

[164] J. Westra and F. Dignum. Evolutionary neural networks for non-player characters in quake III. In *5th International Conference on Computational Intelligence and Games*, pages 302–309, 2009.

[165] J. Wortons, B. Happell, and H. Johnston. The role of the nurse practitioner in psychiatric/mental health nursing: exploring consumer satisfaction. *Journal of Psychiatric and Mental Health Nursing*, 13:78–84, 2006.

[166] D. Xiao and A.-H. Tan. Self-organizing neural architectures and cooperative learning in multi-agent environment. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 37(6):1567–1580, 2007.

[167] X.-H. Xu. Six-step reasoning model for robot-soccer. *Journal of Harbin Institute of Technology (New Series)*, 8(3):244–248, 2001.

[168] J. Yang, S. Min, CO. Wong, J. Kim, and K. Jung. Dynamic game level generation using on-line learning. In *Proceedings of Technologies for E-Learning and Digital Entertainment - Second International Conference, Edutainment*, pages 916–924, 2007.

[169] L. Yifan, P. Musilek, and L. Wyard-Scott. Fuzzy logic in agent-based game design. In *Proceedings of Annual Meeting of the North American Fuzzy Information Processing Society*, volume 2, pages 734–739, Piscataway, NJ, USA, 2004.

[170] H. Yuan and L. Zhen. A preliminary research on decision model based on bayesian techniques for a npc in computer games. In *2010 International Symposium on Computational Intelligence and Design*, Hangzhou, China, 2010.

[171] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[172] S. Zanetti and A. E. Rhalibi. Machine learning techniques for FPS in Q3. In *2004 International Conference on Advances in Computer Entertainment Technology*, 2004.

[173] S. Zhou, J. Xi, M. W. McDaniel, T. Nishihata, P. Salesses, and K. Iagnemma. Self-supervised learning to visually detect terrain surfaces for autonomous robots operating in forested terrain. *Journal of Field Robotics*, 29(2):277–297, 2012.