LEARNING PARADIGMS FOR GAME ARTIFICIAL INTELLIGENCE

Chukwuchekwa Ulumma Joy
Department of Mathematics
Federal University of Technology, Owerri
NIGERIA.
rejoice2k7@yahoo.com

Chukwuchekwa Nkwachukwu
Electrical & Electronics Engineering Department,
Federal University of Technology, Owerri
NIGERIA.

ABSTRACT

One of the main challenges facing Computer games is creating agents that are driven by artificial intelligence (AI) that interact with the player in reliable and entertaining ways. In the game world, it is being accepted that careful and considered use of learning makes it possible to come out with smarter and more robust AIs without the need to appropriate and counter every strategy that a player may adopt. It follows therefore, that rather than having all non-player character behaviours being determined by the time a game is produced, the game should instead evolve, learn, and adapt throughout the period the game is being played. The outcome of this is that the game grows with the player and is very difficult for the player to predict the next action, thus extending the play-life of the game. These learning techniques normally change the way that games are played, by forcing the player to continually search for new strategies to defeat the AI. This paper tries to highlight some of the learning paradigms for Game AI and the great potential they offer to the game world. It was discovered that each of the learning paradigm is suited to a different type of problem, and so the game developer has to be careful in the choice of a particular paradigm.

Key words: Machine Learning, Game Artificial Intelligence, Computer games, Artificial Neural Networks

INTRODUCTION

"It is anticipated that the widespread adoption of learning in games will be one of the most important advances ever to be made in game AI.... In addition, the careful and considered use of learning makes it possible to produce smarter and more robust AIs without the need to pre-empt and counter every strategy that the player might adopt (Manslow, 2002)". Artificial Intelligence (AI) is playing an increasingly important role in the success or otherwise of computer games and the quality and intricacy of the AI techniques used in games is also steadily increasing. The AI techniques used in computer games can be divided into those that are rule-based (deterministic) and those that make attempt at learning or adapting to the player's behaviour (nondeterministic) (Bourg, 2004). The rule-based techniques include the Finite state machines, scripting and Fuzzy logic. The nondeterministic techniques include the Neural Networks, Evolutionary Algorithms (e.g. Genetic Algorithms and Genetic programming), Bayesian Networks (Naive Bayes Classifier), Decision trees, K-Nearest Neighbours, and Reinforcement learning.

The deterministic act or performance is usually explicit and predictable. An example of a deterministic behaviour is a chasing algorithm where developers can clearly code a nonplayer character to move towards some target point by advancing along the x and y coordinate axes until the character's x and y coordinates coincide with the target location (Bourg, 2004)). Nondeterministic behaviour is uncertain and is unpredictable and the degree of uncertainty depends on the AI method used and how well that method is comprehended. An example of nondeterministic performance is a nonplayer character learning to adapt to the fighting tactics of a player (Bourg, 2004). Nondeterministic techniques aid learning and unpredictable gameplay and such learning could use a neural network, a Bayesian technique, decision trees or a genetic algorithm. Developers don't have to clearly code all behaviour in anticipation of all possible scenarios. These techniques also learn and extrapolate on their own and behaviour can emerge without explicit instructions. Deterministic techniques have advantages of being predictable, fast, and easy to implement, understand, test, and debug. The disadvantages are that it places the burden of anticipating all scenarios and coding all behaviour solely on the developers' shoulders, discourages learning or evolving, and after a little game play, tends to become predictable, thus often limiting the game's play-life.

Learning and the family of algorithms based on the principles of learning can be utilised by game developers and players in a variety of ways. For instance, solutions to problems that are very difficult to solve can be solved by learning algorithms, with little or no human supervision. Additionally, in-game learning can be used to adapt to conditions that cannot be anticipated prior to the game's release, such as the particular tastes, dispositions and styles of individual players (Manslow, 2002).

LEARNING PARADIGMS FOR GAME ARTIFICIAL INTELLIGENCE

2.1 LEARNING BY NEURAL NETWORKS

2.1.1 Overview Of Neural Networks

Artificial Neural Networks (ANNs) (Caudil, 1991; Basheer, 2000; Haykin, 1996) also known as Neural Networks (NNs), are constructed according to the model of the human brain and thus have outstanding ability to derive meaning from complex or fuzzy data and can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. Computers usually do well at repetitive tasks but they lack human-like capabilities for pattern recognition and decision-making. An ANN is simply a structure for information processing and pattern recognition that is constructed based on biological neural networks. An ANN is comprised of a series of neurons (units or nodes), interconnected by links (weights) with various characteristics. The characteristics of these weights can change during a training process for the ANN.

The human brain usually learns by adjusting synaptic connections (weights) between individual neurons. A typical artificial neuron is shown in Figure 1.1. ANNs learn by exposing the system to a set of input and output data (datasets) to allow the system to adjust and create connection weights relevant to the specific system it is learning. The aim of a learning (training) process is to establish connection weights between neurons to solve specific problems. ANNs training allows the system to discover and predict patterns or to solve problems that are very complicated and not linearly separable or for which more traditional computational techniques would not work. ANNs are good in pattern recognition and are robust classifiers with the ability of making generalization and also possess ability of making decisions from large and fuzzy input data.

ANNs have been utilized in many domains such as speech recognition, playing chess, fingerprints identification or facial characteristics and for solving diagnostic problems in biology and medicine. ANNs have proofs of capabilities in many domains including financial prediction (such as shares and currency), Control (such as aircraft, industrial processes and space), medical (such as diagnosis and prognosis), marketing (such as data mining), among others. ANNs with ability of learning by example makes them highly flexible and effective in many different domains.

2.1.2 Classification of Anns

ANNs may be categorised in many different ways according to one or more of their relevant features. Generally, classification of ANNs may be based on (Basheer, 2000) "(i) the function that the ANN is designed to serve (e.g., pattern association, clustering), (ii) the degree (partial/full) of connectivity of the neurons in the network, (iii) the direction of flow of information within the network (recurrent and non-recurrent), with recurrent networks being dynamic systems in which the state at any given time is dependent on previous states, (iv) the type of learning algorithm, which represents a set of systematic equations that utilize the outputs obtained from the network along with an arbitrary performance measure to update the internal structure of the ANN, (v) the learning rule (the driving engine of the learning algorithm), and (vi) the degree of learning supervision needed for ANN training. Supervised learning involves training of an ANN with the correct answers (i.e., target outputs) being given for every example, and using the deviation (error) of the ANN solution from corresponding target values to determine the required amount by which each weight should be adjusted. Reinforcement learning is supervised, however the ANN is provided with a critique on correctness of output rather than the correct answer itself, that is, the network is not explicitly given target outputs in the form of a training set, but is rewarded when then it does the right thing. Unsupervised learning does not require a correct answer for the training examples, however the network, through exploring the underlying structure in the data and the correlation between the various examples, organizes the examples into clusters (categories) based on their similarity or differences (e.g., Kohonen networks). Finally, the hybrid learning procedure combines supervised and unsupervised learning".

Once a network has been customised for a particular application, that network is ready to be trained. There are two approaches to training, supervised and unsupervised (Basheer 2000). The most often used ANN is the fully connected, supervised network (the multilayer perceptron) with backpropagation learning rule. This type of ANN is mostly useful at classification and prediction tasks. Another is the Kohonen or Self Organizing Map with unsupervised learning algorithm, which is very useful at finding relationships among complex sets of data.

2.1.3 Structure of Multi Layer Perception Neural Networks

Many ANNs are based on the principle of biological neural networks and contain layers of nodes (input, hidden, output) (Figure 2.1). These nodes are richly interconnected by weighted connection lines. Every input data point is normally associated with a weight and can increase or decrease the activation of the node (neuron or unit) depending on whether it is negative or positive. A typical multi layer perceptron (MLP) is shown in Figure 2.2.

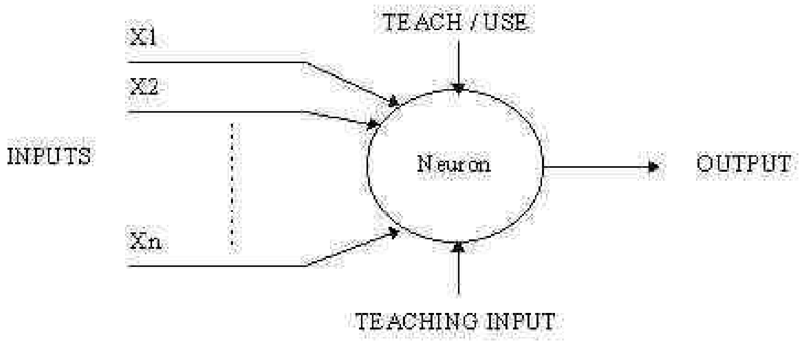

Figure 2.1: A Simple Neuron (extracted from Stergiou, C. & Sigamos, D., 1996)

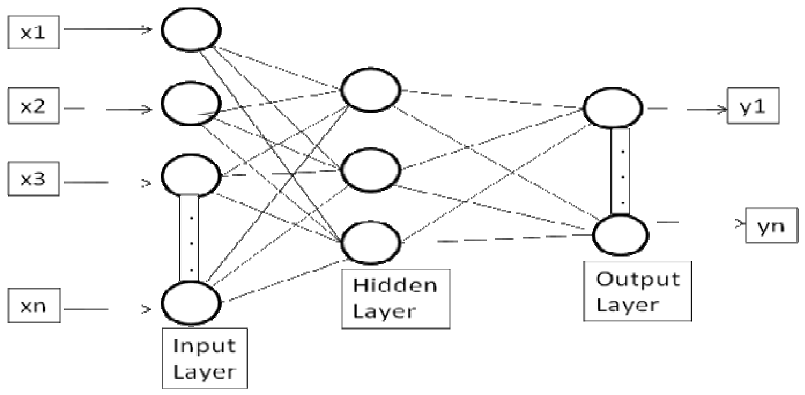

Figure 2.2: A typical multilayer perceptron neural network with one hidden layer

Neural networks are normally arranged in layers. Each layer in a multi layered network is an array of neurons. Information passes through each element in an input-output manner, that is, each neuron receives an input signal, manipulates it and forwards an output signal to the other connected neurons in the adjacent layer. MLP networks normally have three layers of neurons with only one hidden layer, but there is no restriction on the number of hidden layers. The only work that the input layer does is to receive the external information and propagate it to the next hidden layer. The hidden layer receives the weighted sum of incoming signals sent by the input units, and processes it by means of an activation function. The activation functions most commonly used are the sigmoid and hyperbolic tangent functions. The hidden units in turn send an output signal towards the neurons in the next layer. This adjacent layer could be either another hidden layer of arranged neurons or the output layer.

2.1.4 Training of Anns

Neural networks learn in two ways - supervised or unsupervised. In supervised learning, a set of input-output examples are presented to the network. At the start of the training process, the network is repeatedly presented with a set of input vectors along with the desired output vectors for each of them. As training progresses, the network changes itself internally until it reaches a stable stage at which the outputs given by the Neural Network are approximately the same as the actual value. Learning is an adaptive process during which the weights associated to all the interconnected neurons change in order to provide the best possible response to all the observed patterns (Adrian, 2001; Stergiou, C. & Sigamos, D., 1996).

In reality, the acquired data set is divided into training and test set. After training with the training data set, the performance of the ANN is evaluated by presenting the unknown test data set to the ANN. The application phase is related to the application of the net in reality, where usually no information about the desired output is presented to the neural network. “An ANN based system is said to have learnt if it can: (i) handle imprecise, fuzzy, noisy, and probalistic information without noticeable adverse effect on response quality, and (ii) generalize from the tasks it has learnt to unknown ones (Basheer, 2000).

2.1.5 Training Mlp with the Back-Propagation Algorithm

There are numerous learning rules but the most often used is the Delta rule or Back-propagation rule. A neural network is trained to map a set of input data by iterative adjustment of the weights. Information from inputs is fed forward through the network to optimize the weights between neurons. Optimization of the weights is usually made by backward propagation of the error during training or learning phase. The ANNs read the input and output values in the training data set and change the value of the weighted connections (links) to reduce the difference between the predicted and target values. The error in prediction is reduced across many training cycles (epochs) until network reaches specified level of accuracy. This kind of training algorithm is known as the backpropagation algorithm and details can be found in (Stergiou & Sigamos, 1996).

2.1.6 Game Examples of Learning by Neural Networks

Neural networks, as AI techniques, have been applied in a wide variety of problems, and the computer games industry is no different from the numerous industries in which the NNs can be applied. This is because an NN can be used to make decisions or interpret data based on previous input and output examples that it has been given. This training set can be composed of many different types of data that represent many different types of events, characters, or environments (Sweeter, 2004). In a computer game, the input can be a set of variables from the game world, which usually represent the attributes of the game world, game event, or game character. The output from the neural network can be seen as a decision, a classification, or a prediction. For example (Bourg, 2004), the input to the NN could represent the attributes that describe other characters that the AI has encountered in the game world, consisting of variables like health, hitpoints, strength, stamina, attack, etc. The outputs could be a set of possible actions that the AI can take, such as talk, run away, attack, or avoid. Alternatively, the output could be a classification of how the AI feels about this character, such as lothe, dislike, neutral, like, or love. This feeling could then contribute to the AI's decisions about how to react to this character in different situations.

In the game world, Neural networks can be used as neural controllers for robotic applications, which implies that you can have a computer-controlled, half-track mechanized unit in your game or you may want to use a neural network to handle the flight controls for a spaceshift or aircraft. Neural network can also be useful in threat assessment game. They are also applied in the Attack and Flee game. Details of these applications are found in (Bourg, 2004). In the game world, NNs have also been applied in the following tasks Battle Cruiser: 3000AD, Black &White, Creatures, Dirt Track Racing, and Heavy Gear (Sweetser, 2004a).

2.2 LEARNING BY OPTIMISATION (GENETIC ALGORITHM)

2.2.1 OVERVIEW OF GENETIC ALGORITHMS

Genetic algorithms (GAs) (Mitchell, 1996) are a family of computational models based on evolution. Genetic algorithms are inspired by Darwin's theory of evolution. Thus, solution to a problem solved by genetic algorithms is evolved. Genetic algorithms (GAs) was originally conceived, introduced and investigated by Holland in 1975 and by the students of Holland, example De Jong (Whitley, 1994). Genetic algorithm can also be seen as any population based model that uses selection and recombination operators to generate new sample points in a search space. "Genetic algorithms (GAs) have been shown to be a useful alternative to traditional search and optimization methods, especially for problems with highly complex and ill-behaved objective functions. A key element in a genetic algorithm is that it maintains a population of candidate solutions that evolves over time. The population allows the genetic algorithm to continue to explore a number of regions of the search space that appear to be associated with high performance solutions (Grefenstette, 1992)."

"Almost every practical heuristic search algorithm is controlled by some set of parameters ... No matter which variation operator you choose, there are a number of possible parameterizations that you must decide. Each decision is important. If you make a poor choice for your parameters you can generate truly disastrous results (Michalewicz, 2004)." When using genetic algorithm, factors to be considered include the representation, the evaluation function, the variation operators, the population size, the termination criterion among others.

A genetic algorithm operates according to the following steps (Timothy, 1993): "

- **Initialization** – Randomly generate a population.
- **Evaluation** – Test each individual, using the objective function. Compute a fitness value, which is a measure of how well the individual optimizes the function.
- **Parent Selection** – Choose pairs of individuals from the population in such a way that those with higher fitness will be chosen more often.
- **Reproduction** – Generate (usually two) children from each pair of parents. Each parent contributes half of its genetic makeup to each child.
- **Mutation** - Randomly change a tiny amount of the genetic information in each child".

A complete pass through the above processes is referred to as a generation. After each generation is completed, a new one starts with the evaluation of each of the children. Genetic algorithms (GAs) work in the following way. Firstly, a population of random organisms are created (initialized). The organisms are then tested on the problem that is being solved and then they are ranked in order of fitness. If the best organisms have reached our performance goal, we stop otherwise we take the best organisms and repeat the process. In the basic genetic algorithm, solutions are encoded as fixed length vectors (chromosomes). The initial population of solutions is chosen randomly. These solutions are called chromosomes and are allowed to evolve over a number of generations. At each generation, a measure of how well the chromosome optimizes the objective function is calculated. Subsequent generations are created through a process of selection, recombination (crossover), and mutation. Chromosome fitness is useful in selecting which individuals will recombine. Recombination (crossover) operators merge the information contained within pairs of selected parents' chromosomes by placing random subsets of the information from both parents into their respective positions in a member of the subsequent generation. Nevertheless, because of the selective pressure applied through a number of generations, the overall trend is towards higher fitness chromosomes. Mutations are used to help preserve diversity in the population. Mutations introduce random changes into the chromosomes (Shumeet, 1995).

2.2.2 GAME EXAMPLES OF LEARNING BY GENETIC ALGORITHM

Once the behaviour of an agent has been parameterized and a performance measure developed, it can then be improved by using an optimization algorithm (such as genetic algorithm) to search for sets of parameters that make the agent perform well in game (Manslow, 2002).

The possible applications of Genetic Algorithms are numerous. This is because any problem that has a large enough search domain could be suitable for genetic algorithm applicability. Genetic algorithms offer opportunities for developing interesting game strategies in areas where traditional game AI is weak, particularly where traditional methods of search and optimization are too slow in finding a solution in a very complex search space, in that genetic algorithm is a robust search method requiring little information to search effectively in a large, complex, or poorly understood search space, or in nonlinear problems. Genetic algorithms have been used for problem solving, modelling, and applied to many scientific, engineering, business, and entertainment problems. Also, GAs have been successfully used in problems in machine and robot learning, such as classification and prediction, designing neural networks, evolving rules for learning classifier systems, and the control of robots (Sweetser, 2004b). "Genetic algorithms are slowly but surely gaining popularity with game developers. They are currently used mostly as in-house tweaking tools, but they are also beginning to be used in-game, either as an integral part of the gameplay or as an aid for the user (Buckland, 2004)"

> *There are many ways in which genetic algorithms could be used in computer games. For example, genetic algorithms can be used in a real-time strategy (RTS) game to tune the AI's strategy to target the human player's weaknesses. This could simply involve tuning a set of parameters that define the AI's personality, in terms of its preference for types of unit, its weighting on offensive and defensive and defensive, preferences for scientific advances, and so on. Alternatively, genetic algorithms could be used to tune the behaviour of individual or groups of units in an RTS. Additionally, genetic algorithms could be used in a role playing game (RPG) or first person shooter (FPS) to evolve behaviours of characters and events. For example, genetic algorithms could the creatures in the game that have survived the longest and evolve them to produce future generations. This would only need to be done when a new creature is needed. Furthermore, genetic algorithms could be used in games for pathfinding,.... This genetic algorithm could be extended to include obstacle avoidance, factoring for different types of terrain and possibly using waypoints instead of vectors....Some computer games in which genetic algorithms have been applied successfully include Cloak, Dagger, and DNA, the Creatures series, Return Fire 11, and Sigma (Buckland, 2004).*

2.3 LEARNING BY DECISION TREES

Decision Trees (Evans, 2002; Hopgood, 2001) are a promising Learning Paradigm for game AI because they are easy to use and provide a high level of flexibility with less computational requirements. If learning is to occur only before a game is released, DTs are often attractive and are also commonly used if learning must occur during gameplay, because of their computational efficiency. A decision tree is a way of relating a series of inputs (usually measurements from the game world) to an output (usually representing something you want to predict) using a series of rules arranged in a tree structure. For example (Rabin, 2004, inputs representing the health and ammunition of a bot could be used to predict the probability of the bot surviving an engagement with the player. At the root node, the decision tree might test to see whether the bot's health is low, indicating that the bot will not survive if that is the case. If the bot's health is not low, the decision tree might then test to see how much ammunition the bot has, perhaps indicating that the bot will not survive if its ammunition is low, and will survive otherwise. Decision trees are particularly useful for applications like in-game learning because (in contrast to competing technologies like neural networks) extremely efficient algorithms exist for creating decision trees in near real-time.

One famous algorithm used for the decision tree learning is ID3 (Roach, 2009b), which uses the Training Set to decide which attribute is the most important in dividing the cases into the different outcomes. This attribute is then placed at the top of the Decision Tree, and the process repeats to find the next most important attribute along each branch. Choosing the best attribute uses a measurement from Coding and Information Theory, called *entropy*. The description of the ID3 algorithm is given by (Roach, 2009b).

2.3.1 Game Examples of Learning by Decision Trees

"The best known game-specific use of the decision trees is in the game Black & White where they are used to allow the creature to learn and form "opinions" (Evans, 2002). In Black & White, a creature will learn what objects in the world are likely to satisfy his desire to eat, based on feedback it gets from the player or world. For example, the player can provide positive or negative feedback by stroking or slapping the creature. A decision tree is then created that reflects what the creature has learned from its experiences. The creature can then use the decision tree to decide whether certain objects can be used to satisfy its hunger. While Black & White has demonstrated the power of decision trees to learn within games, they remain largely untapped by the rest of the game industry (Rabin, 2004)".

"Although DTs are highly flexible, there some things they cannot efficiently model. For example, a battle between opposing armies might be likely to result in a draw if they are of approximately equal size. A DT that was trying to learn to predict the outcome of a battle would therefore find the prediction of draws problematic unless the difference in size between the armies was explicitly represented in one of its inputs. This is because the DT has no capacity to derive the size difference itself, but without, the conditions under which a battle is likely to be drawn cannot be represented by a single rule or a simple tree (Baekkelund, 2006)"

2.4 LEARNING BY BAYESIAN NETWORKS TECHNIQUE

Bayesian networks (Bourg, 2004; Hopgood, 2001) allow an AI to perform complex humanlike reasoning when faced with uncertainty. Bayesian updating is a technique for handling the uncertainty that arises from statistical variations or randomness (Hopgood, 2001). Bayesian inference and networks enable non-player characters (NPCs) to make decisions when the states of the game world are uncertain. Bayesian networks are graphs that represent the relationship between random variables for a given problem (Bourg, 2004). These graphs help in performing reasoning or decision making in the face of uncertainty. Bayesian networks consist of nodes representing random variables and arcs representing the casual relationship between variables. In a Bayesian network, variables relating to particular states, features, or events in the game world are represented as nodes in a graph, and the casual relationships between them as arcs. Probabilistic inference can then be performed on the graph to infer the values of unknown variables, or conduct other forms of reasoning (Rabin, 2004).

2.4.1 Game Examples of Learning by Bayesian Network

"One particularly important application for Bayesian networks in games lies in modelling what an AI should believe about the human player based on the information it has available. For example, in a real- time strategy game, the AI can attempt to infer the existence or nonexistence of certain player-built units, like fighter planes or warships, based on what it has seen produced by the player so far. This keeps the AI from cheating and actually allows the human to deceive the AI by presenting misleading information, offering new gameplay possibilities and strategies for the player (Rabin, 2004)".

Bayesian networks are based on a mathematical theory known as Bayes' Theorem, which is used to calculate the probability of an event occurring given a known related piece of information. Bayes' theorem states that

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}.$$

This theorem can be used to calculate the statistical probability of events occurring even if we know next to nothing about the world, which provides some information for reasoning under uncertainty. It helps to determine a more realistic and probable assumption about the occurrence of A if we know that B has occurred. As a human player plays a game they make mistakes but as the player learns more about the game and the world he adapts to make better decisions. This is the same with Bayesian networks. Under uncertain conditions the network will calculate the probability of a certain variable, which may be the wrong decision, but as more information is uncovered about the world the network is able to update the probabilities of other variables, causing the calculated variable to be updated, thus producing machine learning and an evolving artificial intelligence. An example where the Bayesian technique can be used in AI game is illustrated (Roach, 2009a):

> *"In a futuristic warfare game, an NPC trooper is to make a decision about whether to jump out of hiding and make a run towards the location the PC was last seen. It bases its decision on whether the PC is thought to still have ammunition, and whether or not it thinks the PC has already teleported away from that position".*

Since the above situation concerns making decisions in uncertainty, the Bayesian Belief Network is best suitable for this problem. The Bayesian Network representation of the above scenario is shown below:

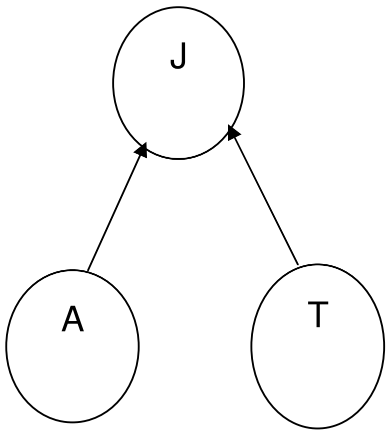

Fig. 2.3: The Bayesian Network Representation of the Warfare game

Where A is a belief in whether the PC is thought to still have ammunition, T is the belief that the PC has already teleported away from its previous position and J is the belief that NPC jump from hiding and running towards the PC will be successful. Details of the implementation of this game application can be found in (Roach, 2009a).

ISSN: 2223-9553
Volume 1, Issue 2, September 2011

2.5 LEARNING BY REINFORCEMENT

This involves learning the relationship between an action taken by the agent in a particular state of the game world and the performance of the agent. Once this has been done, the best action (that is that which yields the best average performance) can be selected in any state. This technique attempts to learn the optimal mapping from state to action by considering a representation of the state as an input vector and the probability of each action as an output vector. This idea forms the basis of reinforcement learning (RL). Reinforcement learning has been successfully applied to a wide variety of complex problems ranging from creating AI players of Othello and backgammon, to balancing a pole on a moving cart (Manslow, 2002; Champandard, 2002).

Reinforcement learning has enabled computers to teach themselves how to play many classic games such as Checkers, Othello, Backgammon, Go, Chess, and card games like Poker and Blackjack (Manslow, 2004). The game developers find the reinforcement learning very useful because of the following reasons (Manslow, 2004): "

- The same RL engine can be used to solve a wide variety of unrelated problems, ranging from producing a competent Chess player, controlling an aircraft to fly as low as possible to avoid radar without crashing, controlling AI vehicles to follow paths specified as a series of waypoints, controlling the movements of dogfighting aircraft, to producing a competent player of a real-time strategy game.
- Provided that the problem is set up correctly, RL is likely to find a close to optimal solution with minimal effort...
- It can find optimal behaviours even in situations where the effect of an action might not be immediately apparent....
- It learns as an AI interacts with the game world, making it suitable for use both during development and to facilitate in-game learning once a game is complete".

CONCLUSION

In this paper, some of the major learning paradigms in game AI and the great potential they offer to the game world have been investigated and illustrated. This report did not actually exhaust all the available learning paradigms but simply highlighted some of the common paradigms used in game AI. Also, because of space, this paper did not cover implementation details for each learning paradigm. The implementation details can be found in the sources cited in the paper. It is also worthy of note that each of the learning paradigm is suited to a different type of problem, and so the game developer has to be careful in the choice of a particular paradigm (see Baekelund, 2006).

REFERENCES

BAEKKELUND, C. 2006. A Brief Comparison of Machine Learning Methods, AI Programming Wisdom 3, Charles River Media, 2006. Edited by Steve Rabin.

BASHEER, I. A and HAJMEER, M., 2000. Artificial Neural Networks: Fundamental, Computing, Design, and Application. Journal of Microbiological Methods, Vol. 43, Issue 1, pp. 3 – 31.

BOURG, D. M. and SEEMANN, G., 2004. *AI for Game Developers.* O'Reilly: California.

BUCKLAND, M., 2004. Building Better Genetic Algorithms, AI Programming Wisdom 2 Charles River Media, 2004. Edited by Steve Rabin.

CAUDIL, M. AND CHARLES, B., 1991. *Naturally Intelligent Systems,* A Bradford Book – The MIT Press, Cambridge, Massachesetts.

CHAMPARD, A. J., 2002. The Dark Art of Neural Networks. AI Programming Wisdom, Charles River Media, 2002. Edited by Steve Rabin.

EVANS, R., 2002. Varieties of Learning. AI Programming Wisdom, Charles River Media, 2002. Edited by Steve Rabin.

GREFENSTETTE, J. J., 1992. Genetic Algorithms for Changing Environments. Parallel Problem Solving from Nature 2.

HAYKIN, S., 1999. *Neural Networks – A Comprehensive Foundation*, Second edition, Prentice-Hall, Inc.

HOPGOOD, A., 2001. *Intelligent Systems for Engineers and Scientists.* CRC Press: New York.

MANSLOW, J., 2002, Learning and Adaptation. AI Programming Wisdom, Charles River Media, 2002.Edited by Steve Rabin.

MANSLOW, J., 2004. Using Reinforcement Learning to Solve AI Control Problems, AI Programming Wisdom 2, Charles River Media, 2004. Edited by Steve Rabin.

MASTERS, T., 1993, *Practical Neural Network Recipes in C++*. Academic Press, Inc.

MICHALEWICZ, Z. and FOGEL, D. B., 2004. *How to Solve It: Modern Heuristics, Second Edition*. Springer-Verlag Berlin Heidelberg .

MITCHELLl, M., 1996. *An Introduction to Genetic Algorithms.* MIT Press.

MONTANA, D. J. AND DAVIS, L., 1989. Training Feedforward Neural Networks Using Genetic Algorithms, "Proceedings of the International Joint Conference on Artificial Intelligence, pp. 762-767.

RABIN, S., 2004. Promising Game AI Techniques, AI Programming Wisdom 2, Charles River Media, 2004. Edited by Steve Rabin.

ROACH, P., 2009a. CS3SO8 AI for Computer Games Developers Handouts. Faculty of Advanced technology, University of Glamorgan Wales, UK.

ROACH, P., 2009b. CS4S22 Expert Systems Handout Techniques. Faculty of Advanced technology, University of Glamorgan Wales, UK.

SHUMEET, B. and RICH, C., 1995. Removing the Genetics from the Standard Genetic Algorithm. Proceedings of the Twelfth International Conference on Machine Learning, Lake Tahoe, CA.

STERGIOU, C. and SIGANOS, D. 1996. Neural Networks. Available from: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html [Accessed 20 July 2010].

SWEETSER, P., 2004a. How to Build Neural Networks for Games, AI Programming Wisdom 2, Charles River Media, 2004. Edited by Steve Rabin.

SWEETSER, P., 2004b. How to Build Evolutionary Algorithms for Games, AI Programming Wisdom 2, Charles River Media, 2004. Edited by Steve Rabin.

WAINWRIGHT, R. L., 1993. Introduction to Genetic Algorithms, Theory and Applications. Proceedings of the first International Conference on Genetic Algorithms. Editor – John Grefenstette.

WHITLEY, D., 1994. A Genetic Algorithm Tutorial. Stat. Comput. Vol.4, pp 65 – 8