# Agent Smith: An Application of Neural Networks to Directing Intelligent Agents in a Game Environment

Jonathan Wolf
Tyler Haugen
Dr. Antonette Logar
South Dakota School of Mines and Technology
Math and Computer Science Department
Rapid City, SD 57701
jonathan.wolf@gmail.com
tyler.haugen@mines.sdsmt.edu
antonette.logar@sdsmt.edu

## Abstract

An Intelligent Agent, or software robot, is a program that is capable of acting in place or in behalf of a user. Intelligent agents, those capable of perceiving and reacting to their environment, have become an important element of game programs. Many games use a finite state machine (FSM) approach to reacting to changes in the environment – changes in the environment generate changes in state. The unique approach taken in this research is to use an artificial neural network as the decision-making process. The agent perceives its environment, presents that information to the network as a feature vector, and receives it next action as an output from the network.

## Introduction

Agent Smith is project developed to examine the application of Artificial Neural Network in the video game industry. Intelligent Agents, or software robots, are often used to implement players in a video game. This project uses an Artificial Neural Network in place of a Finite State Machine to make an agent to behave intelligently.

## ANN vs. FSM

Finite State Machines (FSM), one of the direct competitors to Artificial Neural Networks (ANN), have seen much use in the video game industry in much part due to their general simplicity and ability to simulate required basic behaviors needed for a general illusion of intelligence. The FSM is nothing more than a cyclic graph in which its nodes, also known as states, represent a particular behavior simulated by the system. Connections from one node to another define a set of conditions that when satisfied allows a state change [Buckland05]. While similar in intent compared to ANNs in terms of its classification ability (i.e. ability to classify a set of sensory input to a particular response), it does have a few drawbacks in comparison.

## Intelligent Agents

Intelligent Agents are computer programs that carry out a task in place or on behalf of a user. Intelligent Agents themselves by themselves are not necessarily Artificial Intelligence but a mechanism for the delivery of Artificial Intelligence. This definition for an Agent is rather vague so it is important to look at some common characteristics that are assorted with Intelligent Agents. One the most important characteristic is autonomy. Agents are often used in situations where user feedback is not an option. For example, a video game that needs to ask the user its next move would not be very entertaining.

Communication and Cooperation can be important features of Intelligent Agents when multiple Agents are working toward the same goal. If agents were to be designed to guard an area of its environment utilizing communication and cooperation between them could greatly improve their effectiveness. Personality is also another feature that can be associated with an Intelligent Agent and in video games personality can be an important feature of the Artificial Intelligence. For example, one category of agents may aggressively attack the user regardless of situation where as a different agents may be more likely to regroup with other agents and attack in numbers. Agents are often implemented to be adaptive, learning from experience, its environment, or other Agents. Intelligent Agents are often given mobility in their environment. These features can make Intelligent Agents a good programming model for computer players in video games.

Agents can be design to handle the interaction and communication with its environment, other agents, and the user. There is still one question however, what form of intelligence should drive an Intelligent Agent.

## Artificial Neural Networks

ANNs are a form of artificial intelligence based on the working of biological Neural Networks. Though many different form of an artificial network exist the prevalent are most likely Multi-Layered Perceptron (MLP) networks trained with Backpropagation. As name suggest this form of ANNs consist of multiple layers. The first layer to the network is the input layer and simply pushes the network inputs into the hidden (processing) layer. In a biological model this corresponds to sensory inputs. The hidden layer applies a weight to each of its inputs and sums their value. The summation is run thought an activation function often the sigmoid (this is critical for the training of the network). The result of the activation function is that nodes output and pushed on to the next layer. The next layer is either another hidden layer or the output layer. Though there is no set number of hidden layers that can be used, mathematically having more than two hidden layers is overkill. The output layer also has weights associated with its input from the hidden layer. It will sum the weighted inputs but this time its output will be associated with a value or action that represents the network output.

This stage is often called the feed-foward stage and is fairly simple to implement but leaves out one important detail. How are those weights set? This is purpose of the Backpropagation algorithm. Given a set of input vector along with their expected output the Backpropagation algorithm will train the weights of a network. Once the weights are set network will be able to classify all inputs vectors back to their expected results.

## Creation

ANNs and FSMs have a few shared traits that don't differentiate much between them. For instance, both require a known set of responses, both require a known of set of sensory input (bonus if sensory input is easily expressed as a numerical quantity), and both require that a developer determines what particular conditions invoke a particular kind of response.

However, ANNs have a distinct advantage over FSMs in that when defining responses to conditions, it is more natural for the developer, given a set of sensory input vectors, to simply sit down and judging each sensory input vector presented that the response in that particular case should be X instead of Y, etc.. Over a course of several different variations of such input vectors, one develops the needed training vectors to train an appropriate network. On the other hand, FSMs do not posses that same simplicity, but rather requires the developer to build a state diagram and determine change conditions on every state to every other state that has a transition connection. While still not necessarily too complex, it still generally requires a bit more effort to build and maintain such a diagram.

The advantage in ANNs is even raised slightly more when considering creating “multiple personalities”, so to say. When designing multiple sets of such systems, the creation time and management time in ANNs, after factored over a few such sets, especially if needing extended later on, will drastically cut down hours spent developing and maintaining such “multiple personalities.”

## Implementation

Using FSMs it is fairly easy to develop an FSM with even just basic programming knowledge. The math that deals with FSMs is fairly trivial and straight-forward, and does not require much code to actually get a minimal FSM system up and running. ANNs require quite a bit more coding work, and do require quite a bit more mathematical knowledge in how things work and why certain things don't work (e.g. over-training a set of training vectors, having the first two training vectors be at the value extremes, etc.). In this regard, FSMs are easier at a coding level.

Overall, the implementation work between the two is not necessarily a huge issue considering that these systems are well defined in literature, have free code samples for them, etc. Either way the ANN will take slightly longer to code, and usually requires a separate training tool to be built with it for the developers. However, once built, code maintenance time is minimal with both.

## Serialization

ANNs, due to their very simplistic numeric storage requirements (e.g. an array of numbers), it is very easy (and quite natural) to externalize the object to an external form, be it plain text, XML, or binary. Across a set of ANNs, their output behaviors can be shared along with unique network values that can simply come from the process of training a network. Serialization of an ANN is thus quite straight-forward in that regard.

FSMs, on the other hand, can still be externalized, but do carry with them some extra baggage. Although some aspects of the design are still shared from one node definition to another (e.g. sensory inputs), there are aspects that become more complicated to describe, such as each particular transition line and performing such transition conditions related thereof. In particular, referencing of other nodes via pointer leads to the classical pointer-identifier issues with serialization of items.

## Extensibility

ANNs have a natural ability, simply from the way they are built, to adapt to new changes in the system, in some variations dynamically on their own at run-time (something FSMs would be very hard pressed to emulate). As a result of this more dynamic nature, modifying and adding to an ANN based system is fairly straight-forward and requires a few extra training vectors along with a retraining and general testing. An FSM may be more complicated in that node connections might change drastically to incorporate new states and other input leading to time wasted rewiring the FSM. From a maintenance standpoint this causes FSMs to be generally less extensible than ANNs, circumstances depending of course. In general, any complicated interactions with FSMs, if not well defined and known ahead of time (i.e. a general rarity), will cause for such increased development time.

## The Game

To test the effectiveness of a neural network as a decision-making engine in a game, a simple single-player game was implemented. The purpose of the game is for the player to capture a location represented by a treasure chest. The agents are tasked with defending the treasure. Thus, from the perspective of the agents, the person playing the game represents a threat and the other agents represent allies. The player can disable agents by touching them on the back – the player has no other weapons. The agents can shoot fireballs at the player and when the player is struck by a fireball, its health is diminished. The fatigue of the agent is determined by its current action. The possible actions are to flee, rest, attack, wander, and return to the treasure. Resting will decrease the agent's fatigue while all other actions will increase it by differing amounts. The player wins by reaching the treasure, and loses if his health is decreased to zero by absorbing repeated attacks from the agents.

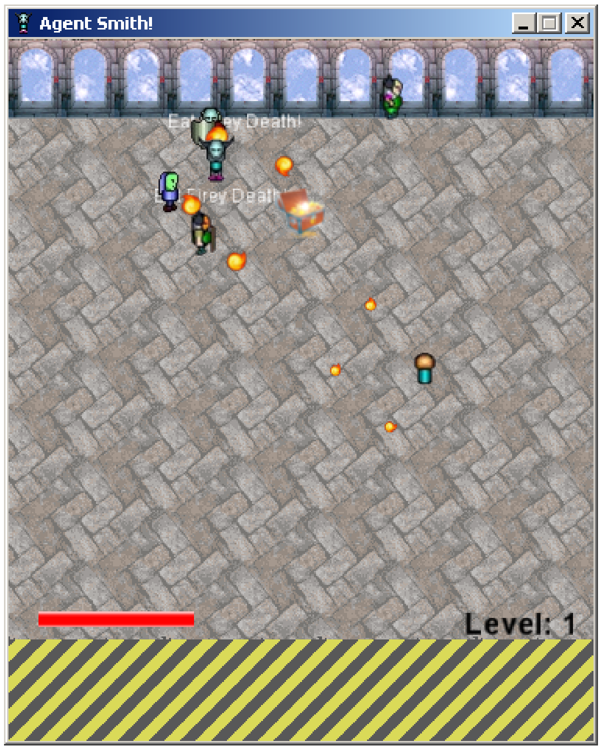

Figure 1: Game Screen Shot 1

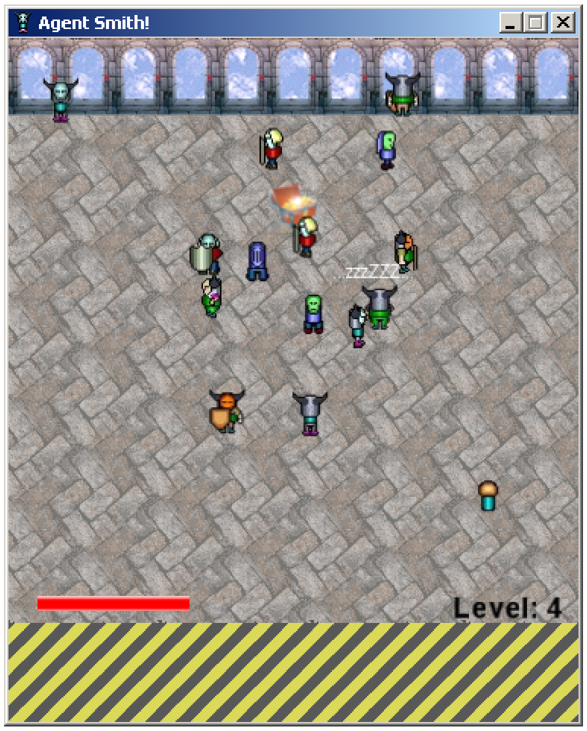

Figure 2: Game Screen Shot 2

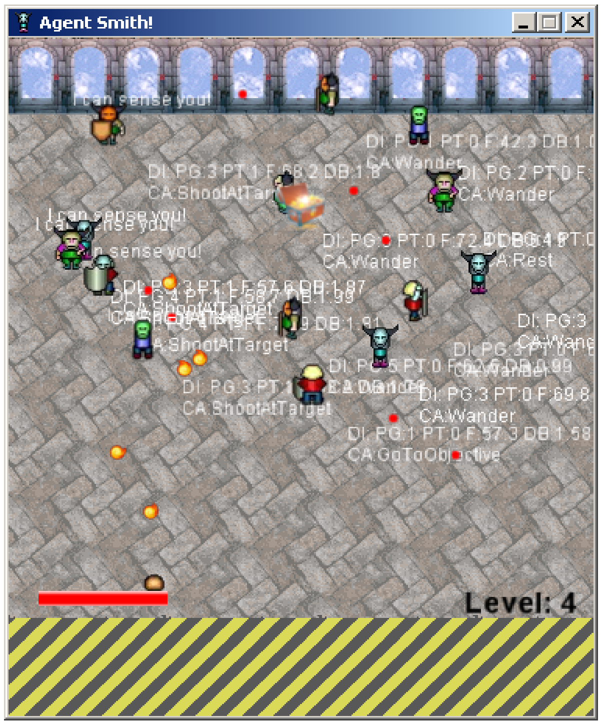

Figure 3: Game Screen Shot 3
(DebugMode On)

## Network Inputs

Each agent has five different inputs that it can act one. The fatigue described is one the inputs and was given a range from zero to one, being fully rested and zero a being fully fatigued. The agents are also aware of their distance from the objective and the number of allies in its vicinity. The ability to sense the user and the health of the user make up the two remaining inputs.

## The Agents

To test the effectiveness of the ANN driven agent two different agent types were developed, a passive agent and an aggressive agent. The goal was to be able to quickly make agents with distant personally both capable of making intelligent decisions. Agents are given the ability to communicate with other agents within a certain distance. If one agent see the user in its field of vision and its can alert the agents within a certain distance.

## Testing & Debugging

FSMs are slightly more natural to work with than ANNs given that an FSM will exhibit a static known behavior that is slightly more easily for the developer to test, especially at design time. ANNs, given that the training vectors may not always be the most well-defined, can sometimes exhibit strange behaviors that may seem quite contrary to what was intended while testing at run-time, resulting in potential time wasted toying with training vectors and retesting. As a result, depending on how well fit training vectors are used (while also being careful to not over-train, etc.), it can be slightly more work to debug and fix an ANN than a FSM, granted that the FSM was developed with all possible conditions in mind ahead of time (again, a general rarity). In other words, a poorly designed yet complex FSM, especially if not built right the first time and major edits are required later on, the time wasted performing those edits generally outweigh the time wasted toying with training vectors. However this isn't to say that an ANN will always produce chaotic results to the point they neglect their usefulness – in many cases the authors have seem them almost always out-perform most other classification systems.

## Results

In a short amount of time two sets of inputs vectors were developed. One set to describes the behavior of the passive agent and another to describe the aggressive agent. These agents acted intelligently showing signs of collaboration and personality.

### Observations of Network Behavior

The inexactness of the ANN can sometimes even be exploited in that a chaotic behavior actually turns out to be artistically relevant for the intended purpose of the system (e.g. for “Agent Smith”, the agents at one time decided to run away rather than fight – although not the intended result it turned out to be an artistically relevant behavior for an agent to decide to do). One of the major difficulties with working with ANNs is the finding training vectors that generalize well. Training sets that do not generalize well can lead to the described chaotic behavior

One way to make the ANN not as spontaneous (in terms of state change frequency), thus possibly avoiding sudden chaotic behavior switches, is to hybrid it with a transition mechanism of some kind. In a way this can be seen as somewhat of a hybrid between the ANN and FSM where the state control and transitioning control is decoupled between the two systems. Either way, the transition mechanism has the net effect of dampening the behavior result as to not switch from one behavior to another too rapidly.

## Conclusion

Depending on the required behavioral complexity for a particular purpose, it is common to see simple behaviors being modeled with an FSM simply because simple behavior doesn't require much complexity in design. For these simplistic systems, the FSM does not reach a level in which its faults cause any significant strife with the development effort. However, for more complex behaviors and other more complicated interactions, especially anything dealing with a dynamic nature, an ANN will almost certainly result in less development time, even after taking into account any potential time wasted toying with training vectors, which as described above is generally going to be shorter than the time it takes to rewire a complicated FSM. Overall, the ANN is a better choice than the FSM for anything more than a simplistic system.

[1] Alpaydin, Ethem. Introduction To Machine Learning. Combridge, Massachusetts: The MIT Press, 2004.

[2] Buckland, Mat. Programming Game AI by Example. Plano, Text: Wordware Publishing Inc, 2005.

[3] Coppin, Ben. Artificial Intelligence Illuminated. Sunbury, Massachusetts: Janes and Bartlett Publishers, 2004.

[4] Hagan, Martin, Howard Demuth, and Mark Beale. Neural Network Design. Bolder, Colorado: University of Colorado Bookstore.

[5] Jones, Tim. AI Application Programming. Hingham, Massachusetts: Charles River Media, 2005.