

FUZZY Q-LEARNING FOR FIRST PERSON SHOOTERS

Youssef S. G. Nashed
Department of Information Engineering
University of Parma
43124, Parma - Italy
nashed@ce.unipr.it

Darryl N. Davis
Department of Computer Science
University of Hull
HU6 7RX, UK
d.n.davis@hull.ac.uk

KEYWORDS

Reinforcement Learning, Q-Learning, Fuzzy Systems, First Person Shooters, Quake-II.

ABSTRACT

Here machine learning techniques in the context of their application within computer games is examined. The scope of the study was that of reinforcement learning [RL] algorithms as applied to the control of enemies in a video game dynamic environment thus providing interesting new experiences for different game players.

The project proved reinforcement learning algorithms are suitable and useful for simulating intelligence of agents within a game. The implemented learning bot exhibited interesting capabilities to adapt to a human player playing style in addition to outperforming other implemented and downloaded bots. Test results showed that incorporating learning into a game can reduce the extensive scripting and tuning phase, while retaining the ability to guide the NPCs not to exhibit totally unrealistic or unexpected behaviors. This gives the designers the scope to explore new strategies. The effect of the exploration-exploitation policy on the learning convergence was studied and tested in depth. The combination of the two most popular reinforcement algorithms [Q-Learning and TD(λ)] resulted in faster learning rates and realistic behavior through the exploration period.

INTRODUCTION

Video games or electronic games can be defined as computer games that involve handling a user interface to provide visual feedback. Recently the video game industry has been growing exponentially and AI is playing an important role in the success of a game.

Nowadays video games are required to deliver stunning graphics, real world physics, impressive sound, believable AI, and gripping storylines to be able to survive in this fierce market. The lifetime of a game title on retailers' shelves depend on how much fun the game is, which in turn depends on many aspects including the repeatability of in-game situations; the less predictable the more fun the game.

First person shooters [FPS] games can be grouped under the Action genre which generally involves the player guiding a character through a set of virtual environments encountering several types of enemies. These games depend on quick responses and eye-hand coordination (Johnson and Wiles 2001).

FPS games are characterized by the player viewing the world from the eyes or the perspective of the main playing character. These games usually provide the player with various weapons to defeat increasingly challenging opponents.

This study requires aspects of Champandard's AI extensions to Quake-II (Champandard 2003) to be built into the Quake-II demonstrator code and then extended to allow learning in the game. This will enable studying the effect of applying machine learning algorithms to Non-player characters [NPCs] in terms of emerging behaviors, diversity, optimality, and believability.

ARTIFICIAL INTELLIGENCE IN GAMES

Board games [chess, checkers, go, etc...] have always been studied by AI researchers. These games act as a problem generator to be solved by AI algorithms especially when playing against a human professional player (Campbell 1999).

Graph machines describe the family of search techniques like breadth first, depth first, iterative deepening, or A* algorithms. Graph machines are particularly effective for implementing AI for board games. The game discrete state space can be represented as a tree or graph and any of these search mechanisms can be used to traverse the tree to find the optimal solution for a certain situation. The use of heuristics about the game or the opponent moves allows the tree branches and the searching time to be considerably decreased thus making for efficient search algorithms.

Modern video games often comprise of complex dynamic environments. The attempt to store the continuous state space of such a game using a graph is computationally impossible. Hence other world and agent models were adopted for complex video games, and other machines are used to work on these models like state machines and reactive machines (Brooks 1991). Graph machines are still used for path finding [A*] and planning (Orkin 2006).

There has always been a gap between AI research in academia and AI applied in video games. In the past game AI programmers often complained about being given insufficient CPU resources compared to graphics which often constrained them from employing any significant AI techniques. With the recent breakthroughs in computer and console hardware, game AI programmers can borrow well established academic research like neural networks and genetic algorithms (Zanetti and El Rhalibi 2004). Furthermore, academic researchers have started to take video games seriously and use it as a test-bed for novel AI research as games provide an increasingly realistic and complex

environments bundled in a reliable, cheap, and extendible package. There are also commercial games used and developed by the US army national guard to test new weapons and surveillance technologies [America's Army, AirForce Delta Storm, Microsoft Flight Simulator, etc...].

Game AI can be categorized into two broad groups:

Deterministic Agents: Not necessarily meaning that the NPCs are driven by designer scripts but that their behavior is deterministic i.e. when facing the same situation agents will always perform the same action or behavior. Scripted agents are easier to implement, debug, and extend but struggle to display complex behaviors.

Learning Agents: In contrast to scripted or rule based agents, learning agents can adapt to human players and game environments and can even learn new behaviors. Learning can speed up the development time and provide extra levels of intelligence for NPCs that are increasingly challenging for the player.

AI in First Person Shooters

Game AI is all about believability. If the NPCs in a game can convince the player that they are not artificially controlled characters then the AI is said to be successful. FPS games have been around for many years and their AI systems are well established. However, players still prefer the multiplayer and online versions of these games which means that the AI in FPS is still falling short from mimicking the skills of human players.

This subsection contains an overview of the AI methods that are widely used in FPS games giving examples from commercial games.

Finite State Machines

Finite state Machines [FSMs] are a natural way of thinking about synthetic creatures, consisting of a set of states, inputs, and outputs. An enemy is always in a given state [e.g. patrol]; when this state receives a specific input [e.g. player sighted] the state changes [e.g. attack] and an action is performed [e.g. fire]. Games like Doom, Descent, and Quake use FSMs to model the enemies (Bourg and Seeman 2004).

Fuzzy Logic

Fuzzy logic replaces the standard Boolean logic concept of absolute True or False with degrees of membership to a specified fuzzy set (Zadeh 1965). Fuzzy logic maps perceived input [crisp input] to a real number describing how close this input to a certain fuzzy variable [e.g. dangerous].

The game Unreal by Epic Games uses fuzzy logic to implement Fuzzy State Machines [FuSM]. These fuzzy machines provide smoother state transition than normal FSMs. 'S.W.A.T. 2' uses fuzzy logic to give each enemy a personality by parameterizing different fuzzy sets like aggression, courage, intelligence and cooperation (Johnson and Wiles 2001).

Planning

In planning actions and goals are decoupled which allows more modularity in the design and enables designers to produce more complex behaviors that would require a considerable amount of effort and computational memory to be expressed using FSMs.

The AI in the game F.E.A.R by Monolith Productions uses a method very similar to STRIPS (Nilsson 1998) to describe agent goals and actions.

AI in Quake

A brief overview of the bots developed in the Quake game series or for research reasons is given in this section.

The Omicron Bot: This bot plays the multiplayer version of Quake. It was created to emulate a human player in a multiplayer or online game. Omicron starts the game with no prior knowledge of the level structure then it drops waypoints till it knows its way around a level (Van Waveren 2001). The bot uses fuzzy logic for action preferences and weapon selection.

The Gladiator Bot: Like Omicron, Gladiator is a multiplayer bot for Quake-II. The only difference between both bots is that Gladiator has a whole range of characters each having its own personality by varying the fuzzy logic parameters (Van Waveren 2001).

Quake-III Arena Bot: The Quake-III bot uses a layered steering behavior approach which employs the type of subsumption architecture proposed by (Brooks 1986).

Quake-II is used as a research environment for developing autonomous agents. Anticipation bot (Laird 2001) uses the SOAR engine (Laird et al. 1987) with Quake-II to demonstrate dynamical hierarchical task decomposition and prediction. Emobot (Hooley et al. 2004) was developed to prove that simple emotions like fear and anger can have a great effect on agents' actions and believability

REINFORCEMENT LEARNING

The standard RL model is depicted in Figure 1. At each time step $t+1$ the agent receives s as the state of the environment which can be separated to i as input to the agent and r as the scalar reward signal for the action a chosen at the previous time step t . The agent's behavior B should choose the action that is expected to increase the long-run sum of rewards. The input state s is perceived by the agent as input according to the input function I , which is the way the agent views the environment state. Typically I is the identity function [that is the agent perceives the exact state of the environment], however this function can change in the case of partially observable environments (Kaelbling et al. 1996).

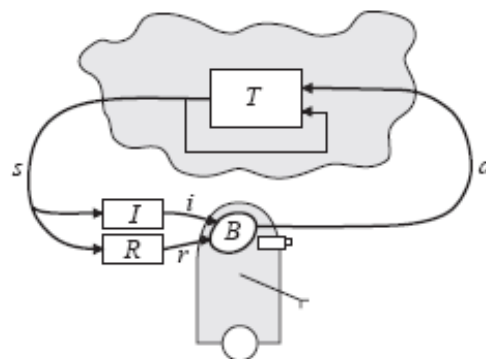


Figure 1: The Standard Reinforcement Learning model [taken from (Kaelbling et al. 1996)]

(Sutton and Barto 1998) identify four main components of a RL system:

A policy: often denoted as π , this is what determines what action the agent should choose in a given state.

A reward function: this is an external function to the agent providing the reward signals [usually a floating point number between -1 and 1].

A value function: also known as the expected reward E , which is the long term value of a state when operating under the current policy. The value function calculation depends on the model of optimality used.

A model for the environment: some RL algorithms start with a complete model of the environment; other algorithms learn this model; while some algorithms do not need such a model at all.

Exploration versus Exploitation

The exploration-exploitation dilemma is very popular in the field of RL algorithms. This problem is studied in depth within statistical decision theory and the k-armed bandit problems (Berry and Fristedt 1985).

In the k-armed bandit problems the agent is presented with k number of gambling machines, each having an arm that when pulled pays off 1 or 0, based on a certain probability. The agent is allowed a limited number of pulls. What should be the strategy undertaken to maximize the payoff in this situation?

The agent should try out every option in the first few tries [exploration], then - after forming a model that is close to being correct - be greedy and make use of highest paying off machines [exploitation]. The balance between the exploration period and the exploitation period is called the exploration policy. It is extremely difficult to find an optimal exploration policy, nevertheless various researches proposed reasonable policies that produce good results.

Some approaches to this problem use dynamic programming incorporating basic Bayesian reasoning (Berry and Fristedt 1985), or learning automata techniques (Kaelbling et al. 1996). In (Gu et al. 2003) a genetic algorithm was used for exploration, while Jouffe proposed a heuristic based method based on the frequencies and success of the choices (Jouffe 1998). However the most successful and used approaches are that of Ad-Hoc strategies.

Ad-Hoc strategies are simple undirected [randomized] algorithms. They are far from optimal but are proven reasonable, traceable, and computationally efficient for most applications. The two main exploration strategies that are commonly used are the ϵ -greedy exploration and the Boltzmann exploration.

In the ϵ -greedy exploration algorithm actions with the highest estimated payoff are always chosen. The downside of this approach is that the algorithm can get stuck in suboptimal actions while the optimal solution is starved and never found. A useful heuristic is optimism in the face of uncertainty in which actions are initialized with high optimistic estimations so that negative feedback is needed to remove the action from selection.

The Boltzmann exploration algorithm uses the Boltzmann distribution to assign action selection probabilities.

$$P(a) = \frac{e^{\frac{E(a)}{T}}}{\sum_{a' \in A} e^{\frac{E(a')}{T}}} \quad (1)$$

Where A is the set of all actions, $P(a)$ is the selection probability of action a , $E(a)$ is the estimated pay off of action a , and T is the temperature of the system.

The temperature parameter T is decreased over time to decrease exploration and allow exploitation of higher estimated actions. T values need careful tuning as they are used to control convergence.

Delayed Reward

The two main problems facing RL methods are temporal credit assignment and the curse of dimensionality.

In the standard RL model the agent performs an action then receives an immediate reward from the environment; this is not always applicable in the case of complex dynamic environments where the agent has to execute a series of actions to reach a state of the environment with a high reward signal [goal]. This kind of reward functions is called a sparse reward function.

Sparse reward functions are zero everywhere, except for a few places. The contrast with dense reward functions, which give non-zero rewards most of the time.

To cope with such environments the agent must learn from delayed rewards. When a reinforcement signal is received the value function of all the actions leading to the current state should be updated with a discounted version of the reward. Temporal credit assignment [delayed reward] problems are well modeled as Markov Decision Processes [MDPs].

MDPs consist of a set of actions A , a set of states S , a reward function R , and a state transition function T with which $T(s, a, s')$ means the probability of the environment being in state s' after executing action a in state s .

Some RL algorithms use a model of the environment. These use MDPs and learn the state transition function or construct the model while learning.

To form a model of a video game environment is extremely difficult; both sets of states and actions would be huge. Consequently this study is concerned with model free techniques like Q-learning and Temporal difference (TD) algorithms.

Temporal Difference TD(λ)

TD methods introduced in (Sutton 1988) do not wait until the end of an episode like Monte Carlo methods to update the value function or estimate. Instead it uses the temporal difference Δ_t in the expected return $V(s_t)$ value of a state s_t to the previous one(s).

The parameter λ controls how many steps or states to look back when adjusting the state expected return values, when $\lambda=0$ the algorithm only looks one step back. The update rule for a TD(0) algorithm is:

$$\Delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Where r_{t+1} is the reward signal received at state s_{t+1} and γ is the discount factor.

$$V(s_t) = V(s_t) + \alpha \Delta_t$$

Where α is the learning rate.

In addition to the state expected return value V , (Sutton 1988) defines **eligibility traces** of a state $e(s)$ as the degree to which a state has been visited in the recent past. These traces are used when updating the expected value function of

a state each according to its eligibility. The update rule for $e(s)$ is as follows:

$$e(s) = \begin{cases} \lambda \gamma e(s) + 1, & s = \text{current state} \\ \lambda \gamma e(s), & \text{otherwise} \end{cases} \quad (2)$$

Q-Learning

Q-Learning (Watkins and Dayan 1992) is probably the most used RL algorithm. Q-Learning can be considered a $TD(0)$ method but instead of estimating a value function for each state Q-learning maintains a value function $Q(s,a)$ for each state-action pair.

Let's say the agent has performed an action a , received a reward signal r , and moved from state s to s' . The Q value for the state-action pair (s, a) is updated using an equation very similar to the TD update equation, except that the highest Q value for the pair (s', a') is used. Where a' is a possible action that can be executed in state s' .

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \quad (3)$$

Where α is the learning rate and γ is the discount factor.

Generalization

In a simple discrete state space environment, the state and action spaces can be enumerated and the Q values stored in a lookup table. In the case of a large continuous state space environment the use of such a data structure is impractical. This problem is called the curse of dimensionality.

To address this problem some sort of input generalization should be employed. Generalization uses compact storage of learned experience and applies it to similar states and actions.

Input generalization is achieved by the use of function approximators such as Cerebellar Model Articulation Controller [CMAC] (Sutton 1996), neural networks (Lin 1992), and fuzzy inference systems (Berenji 1996) (Jouffe 1998).

DESIGN AND IMPLEMENTATION

Quake-II has been used as a test bed for AI researchers for some time because it is easily extendible, multiplatform, open source, and has a huge amount of community support [maps, levels, tools, etc...]; also the graphics and physics requirements are very modest compared to today's processing power leaving computational resources free for AI calculations. Being a FPS game, Quake-II players move in a 3D virtual world viewing it from the eyes of the main character. The main task of a player is staying alive and eliminating his/her opponents.

The project involved the design and implementation of three agent types. Each agent [bot] was designed and implemented in an incremental fashion, its action was built and tested individually and every bot contributed in the implementation of its successor. The navigation module used a reactive architecture and is divided into two levels: *Low level actions* [short term goals] like obstacle avoidance and turning; and *high level actions* [long term goals] like taking cover, gathering, investigating, chasing, fleeing, and dodging. There are only two effectors for the navigation module: *Step*: which takes a 3D vector specifying the direction as input, where only the first two dimensions are used; *Turn*: which takes three floating point values as input, specifying the yaw,

pitch, and roll angles for the bot orientation. Other classes of steering behaviors are described for each agent in turn.

Rule Based FSM Bot "Tom"

Tom is the first and simplest bot, implemented using scripts. Tom uses a FSM to control his actions and decisions. Here FSMs are designed as a form of Rule Based Systems [RBS]. The implemented RBS is a *forward chaining* RBS with a *First Applicable* conflict resolution mechanism (Bourg et al. 2004). This system can be described in terms of three mutually exclusive sets namely: states, actions, and inputs, and a rule base.

States

They dictate the action that is chosen in a given point in time. Tom can be in one and only one of the following states:

"Searching" Looking around exploring as much possible from the level terrain.

"Wandering" Randomly moving around an area.

"Attacking" Firing the held weapon at an enemy.

"Hunting" Pursuing the enemy's current position.

"Gathering" Collecting Health, Armor, Weapons, or Ammo.

"Fleeing" Running away from the enemy to a safe position.

Actions

Actions are the outputs of an AI system. They are considered the bot's effectors to interact with the environment. Tom's effectors are outlined below:

"Turn" Keep yawing with a certain angle.

"Move" Step in a given direction.

"Shoot" Fire the held weapon.

"Seek" Go to a given location on the map.

"Avoid" Go away from a given location on the map.

Inputs

They represent the values and observations that are fed into the system through sensors. Tom has the following Boolean sensor inputs:

"Enemy Sighted" Another player or bot is in the field of view.

"Item Sighted" An item health, weapon, or ammo is in the field of view.

"Is Confident" Health is above 20%.

"Is Hit" A projectile or weapon damage has been inflicted.

Rules

The rule base consists of production rules. These rules are basically in the form of "*If ... then*" statements. They are crafted by an expert to transfer a human knowledge to the system. Table 1 illustrates the rules at each state.

Fuzzy Rule Based Agent "Yianni"

Yianni uses Tom's RBS redesigned as a Fuzzy Rule Based System (FRBS). Instead of Boolean sensors Yianni uses fuzzy logic to classify sensor crisp values as degrees of membership to a specific fuzzy set.

There are two types of FRBSs; The Mamdani FRBS and the Takagi-Sugeno-Kang (TSK) FRBS. The difference between them is the types of inputs [antecedents] and outputs [consequents] used in both of them (Alcalá et al. 2000).

Table 1: Rules for the FSM Scripted Bot

Rank	State	Input	New State	Action
1	Searching	Enemy Sighted	Attacking	Shoot
2	Searching	Is Hit	Searching	Turn
(Default)	Searching	Any Other	Wandering	Move
1	Wandering	Enemy Sighted	Attacking	Shoot
2	Wandering	Item Sighted & ¬Is Confident	Gathering	Seek
3	Wandering	Is Hit	Searching	Turn
(Default)	Wandering	Any Other	Wandering	Move
1	Attacking	Enemy Sighted & Is Confident	Hunting	Seek
2	Attacking	Enemy Sighted & ¬Is Confident	Fleeing	Avoid
(Default)	Attacking	Any Other	Wandering	Move
1	Hunting	Enemy Sighted	Attacking	Shoot
(Default)	Hunting	Any Other	Wandering	Move
1	Gathering	Enemy Sighted	Attacking	Shoot
2	Gathering	Item Sighted & ¬Is Confident	Gathering	Seek
3	Gathering	Is Hit	Searching	Turn
(Default)	Gathering	Any Other	Wandering	Move
1	Fleeing	Enemy Sighted	Attacking	Shoot
(Default)	Fleeing	Any Other	Wandering	Move

The Mamdani system rules are constructed in the following form:

IF X_1 is A_1 and ... and X_n is A_n THEN Y is B

Where $X = \{X_1, X_2, \dots, X_n\}$ is the set of real value input variables, $A = \{A_1, A_2, \dots, A_n\}$ is the set of fuzzy linguistic variables for the inputs, Y is an output variable, and B is also a fuzzy linguistic variable.

The TSK consequents are a function of the input variables, taking the form:

IF X_1 is A_1 and ... and X_n is A_n THEN Y is $(\mu_1 \wedge \dots \wedge \mu_n)$

Where μ_1 is the truth value for input variable X_1 in the fuzzy set defined by A_1 , and \wedge is the disjunction [min] operator.

This implementation uses the TSK FRBS which can be expressed in terms of its input and output variables along with the rule base.

Fuzzy Input Variables [Antecedents]

Sensor Input fuzzification provides smooth transition between states and emulates the way humans think. Crisp floating point values are fed into the fuzzification component to be mapped to a degree of membership of a linguistic variable (e.g. how close an enemy is).

Yianni has the following list of sensor inputs some with an accompanying fuzzy membership function and others are Boolean sensors.

"Projectile Distance" The length of the vector from the bot to the closest projectile observed.

"Enemy Distance" The length of the vector from the bot to the closest enemy observed.

"Item Distance" The length of the vector from the bot to the closest item observed.

"Confidence" The crisp value for this fuzzy variable is gathered from the agent's personal status and calculated as follows:

$$\text{Confidence (out of 400)} = \text{Health (\%)} + \text{Armor (\%)} + (\text{Ammo}/2) + (\text{Weapon Rank} * 10)$$

Four different game variables form the crisp value for the fuzzy Confidence input variable. Each variable is clamped to 100 then added together to give the final value out of 400.

In Quake-II, maximum ammo for a weapon is 200, hence the division by 2. Weapons rank from 1-10 with increasing

powers, so the multiplication by 10 gives the weapon strength out of 100 [a bot switches automatically to a higher rank weapon when acquired].

"Got Shot" A Boolean variable indicating that the bot has been hit by a projectile or any enemy weapon.

"Enemy Present" A Boolean variable indicating that the bot can see an enemy.

"Enemy Disappeared" A Boolean variable indicating that the bot was seeing an enemy but then the enemy ran away or took cover.

Fuzzy Output Variables [Consequents]

TSK consequents are assigned the value of the minimum [sometimes the product] of the truth values computed for the antecedents in a fuzzy rule. If more than one rule has the same consequent the output variable will have multiple confidence values to choose from. This can be solved in various ways; the most popular techniques are bounded sum [sum and bound to 1], and maximum value [equivalent to OR-ing the confidences together].

The maximum value method is used here to resolve the multiple confidences per output conflict. Yianni has the following output variables each representing a tendency to a certain action:

"Dodge" Try to avoid incoming visible projectiles [bullets, rockets, etc...]

"Flee" Moving away [opposite direction] from an attacking enemy.

"Attack" Aiming and shooting the held weapon at a visible enemy.

"Chase" Moving towards or pursuing an enemy.

"Gather" Moving towards a visible item [Health Kit, Armor, Weapon, or Ammo]

"Search" Turning around looking for enemies.

"Investigate" Moving towards a location in the level and searching the area.

"Wander" Random movement and turning while avoiding obstacles.

Actions

The agent's overall action is a weighted vector for a movement direction vector and a weighted orientation angle for turning.

Each output fuzzy variable contributes to the action with the value calculated after firing the fuzzy rule where this variable is a consequent. This provides smooth control implementing the architecture proposed by Saffiotti for controlling autonomous robots (Saffiotti et al. 1993).

Fuzzy Rules

TSK rules are used to control Yianni's behavior and can be seen in Table 2. It is noticeable the amount of rules required for Yianni is far less than that for Tom. This proves to be more efficient for the inference system as fewer rules have to be matched at every time step.

Fuzzy Q-Learning Agent "Babis"

Babis is a learning agent and is built upon Yianni's fuzzy system. The learning algorithm implemented is a Fuzzy Q-Learning method (Glennec and Jouffe 1997) used by many researchers in the robotics field (Deng et al. 2003)

Table 2: Fuzzy Agent Behavior Rules

Output	Condition
Dodge	Projectile Distance is <i>near</i>
Flee	(Enemy Distance is <i>near</i> OR Enemy Distance is <i>close</i>) AND Confidence is <i>weak</i>
Attack	Enemy Present
Chase	(Enemy Distance is <i>near</i> OR Enemy Distance is <i>close</i>) AND Confidence is <i>strong</i>
Gather	(Item Distance is <i>near</i> OR Item Distance is <i>close</i>) AND Confidence is <i>weak</i>
Search	Got Shot AND \neg Enemy Present
Investigate	Enemy Disappear
Wander	Always [Default Behavior if none of the above Rules Fire]

Using Fuzzy logic for input state generalization in reinforcement learning methods has been used for a while now (Berenji 1996). While most fuzzy reinforcement learning algorithms used Actor-Critic architectures (Berenji et al. 1992) (Wang et al. 2007), the method implemented here has the advantage of being exploration insensitive because of the use of eligibility traces of (Sutton 1988).

Fuzzy Q-learning is specifically designed to work with TSK FRBSs which makes it suitable for adding learning to the existing system built for Yianni. Firstly the TSK rule base has to be amended to allow J competing actions defined for every combination of the state [input] vector. Then a q value is associated with each of the competing action as follows:

$$\begin{aligned} \text{IF } x \text{ is } S_i \text{ then } & a[i, 1] \text{ with } q[i, 1] \\ & \text{OR } a[i, 2] \text{ with } q[i, 2] \\ & \dots \\ & \text{OR } a[i, J] \text{ with } q[i, J] \end{aligned}$$

Where x is the vector of input variables $\{x_1, x_2, \dots, x_n\}$, S_i is the state vector defined " x_1 is $S_{i,1}$ AND x_2 is $S_{i,2}$ AND ... AND x_n is $S_{i,n}$ ", $(S_{i,j})_{j=1}^n$ are fuzzy sets, $a_i \in A_i$, and A_i is the set of possible actions that can be chosen in state S_i .

The learning agent has to find the best consequent [action] for each rule using the quality value q . The q values are initialized to zeros at the beginning. At each time step an Exploration/Exploitation Policy [EEP] is used to choose an action for each rule using the Boltzmann Probability assignment from equation (1).

Let i^+ be the index of the selected action in rule i using the EEP, and i^* is the index of the action having the maximum q value in rule i . Therefore the overall quality Q of the inferred global action a in state x is given by:

$$Q(x, a) = \frac{\sum_{i=1}^N \alpha_i(x) * q[i, i^+]}{\sum_{i=1}^N \alpha_i(x)} \quad (4)$$

Where N is the total number of rules, and $\alpha_i(x)$ is the inferred truth value for input vector x in rule i .

The value V for state is given by:

$$V(x, a) = \frac{\sum_{i=1}^N \alpha_i(x) * q[i, i^*]}{\sum_{i=1}^N \alpha_i(x)} \quad (5)$$

To update the action q values, we have to calculate the error signal ΔQ . Let x be a state, a the action applied to the system, y the new state and r the reinforcement signal. By substituting in equation (3) we get:

$$\Delta Q = r + \gamma V(y) - Q(x, a) \quad (6)$$

By using an ordinary gradient descent we get:

$$\Delta q[i, i^+] = \alpha * \Delta Q * \frac{\alpha_i(x)}{\sum_{i=1}^N \alpha_i(x)} \quad (7)$$

Where Δq is the amount added to the q value of action i^+ , and α is the learning rate.

To speed up the learning process and make it exploration insensitive we use the eligibility traces from equation (2) in the form:

$$e(i, j) = \begin{cases} \lambda \gamma e(i, j) + \frac{\alpha_i(x)}{\sum_{i=1}^N \alpha_i(x)}, & j = i^+ \\ \lambda \gamma e(i, j), & \text{otherwise} \end{cases} \quad (8)$$

Where λ is the discount parameter of TD(λ)

After introducing eligibility traces equation (7) becomes:

$$\Delta q[i, j] = \alpha * \Delta Q * e(i, j) \quad (9)$$

Finally, the learning algorithm steps are:

1. Collect sensor information and fuzzify them to state x .
2. For each rule, choose the consequent action using Boltzmann EEP.
3. Apply the global action a from blending all rules actions and receive reinforcement.
4. Compute the overall quality of the system $Q(x, a)$ using equation (4).
5. Observe the new state y .
6. Compute the value of the new state $V(y)$ using (5).
7. Calculate the error signal ΔQ using formula (6).
8. Update the eligibility traces for all actions using (8).
9. Use the ΔQ and the eligibility traces to update all the action q values using (9).

It is worth noting that the convergence of this algorithm depends on many parameters:

The number of input variables forming the state vector and the number of linguistic variables describing each input decide how many rules are specified. The temperature parameter in the Boltzmann probability equation used for EEP decides how long is the exploration period and when to start choosing only actions with maximum q values. The discount and learning rate factors affect the speed of convergence of the learning algorithm.

Action q values can be initialized based on prior knowledge giving some sort of bias towards certain actions in specific state input configurations. This provides more human control on the learning process but might prevent the agent from exploring new solutions.

Babis has the same fuzzy input and output variables as Yianni. Additional rules were added to cover the remaining input space. Three input variables were considered in the learning process: Enemy Distance, Item Distance, and Confidence; Making Chase, Flee, and Gather the three competing actions for the bot to choose from.

TESTING AND RESULTS

The three bots were developed within the FEAR platform (Champanand 2003). The platform is developed to support AI research and teaching through building synthetic creatures in virtual environments. Theoretically it can interface with any game engine as a backend, yet its only implementation is with the Quake-II engine.

Bots are written in C++, compiled into Dynamic Link Libraries (DLLs) and are defined with XML files.

The DLL file is the implementation of the control processes, using FEAR components specified in the XML files through interfaces called hooks to interact with the game engine.

The knowledge in the rule base of the implemented bots is based upon experience in the game and trial and error. In the Fuzzy Q-Learning algorithm we use $\gamma = 0.9$ as the discount factor and $\alpha = 0.1$ as the learning rate.

The reward function is defined as follows:

On pain: $r = -0.1$, on item pickup: $r = +0.1$, on death: $r = -1.0$, hurting an enemy: $r = +0.1$, and killing an enemy: $r = +1.0$

To evaluate the performance of the implemented bots, a simple map was created comprising of one large room, scattered obstacles, and various power-ups. Performance evaluation is achieved through death matches between implemented bots, the difference between how many times a bot has killed the other is called the frag difference.

The frag difference is used as the main performance evaluation factor. It conveys superiority of a bot's actions over the other.

Tom VS Yianni

The winner from the first two implemented bots will have the chance to compete against Babis the learning bot to assess whether learning has added any power to the agent's behaviors or not.

Results from ten death matches [each has a time limit of 3 minutes in *accelerated mode*] were collected and presented in Table 3.

Table 3: Death Match Results for Tom VS Yianni

#	Tom's Kills	Yianni's Kills	Frag Difference
1	974	1028	54
2	900	1002	102
3	832	928	96
4	894	942	48
5	833	880	47
6	900	1003	103
7	918	1018	100
8	947	1050	103
9	815	853	38
10	839	890	51

Yianni VS Babis

Babis' performance depends on the learning algorithm and the time it takes to converge to an optimum policy, which in turn depends on many parameters.

During the exploration period the agent may behave unrealistically, so the temperature parameter was chosen to show the effect of the exploration period on Babis' performance through ten death matches [each has a time limit of 5 minutes in *accelerated mode*] against Yianni. Results are plotted in Figure 2. $T=0.5$ yields a good exploration policy but with one loss while a lower exploration time does not find the optimal behavior policy.

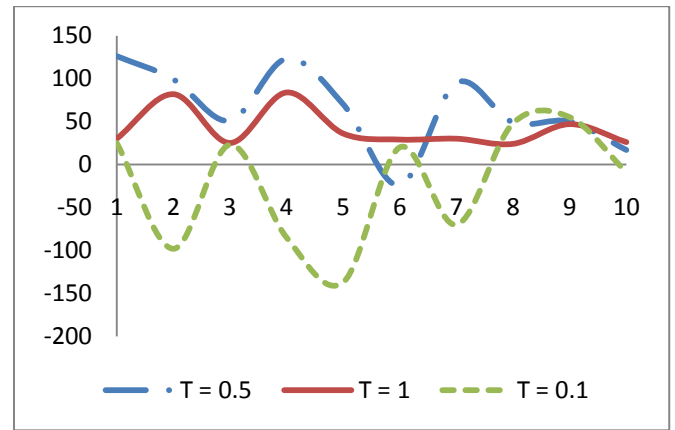


Figure 2: Temperature Effect On The Frag Difference

To speed up learning TD(λ) methods were combined with the Fuzzy Q-learning algorithm by using the eligibility traces for each rule action. Another ten death match simulations were run to show the effect of changing the λ value on Babis' performance. Results are plotted in Figure 3.

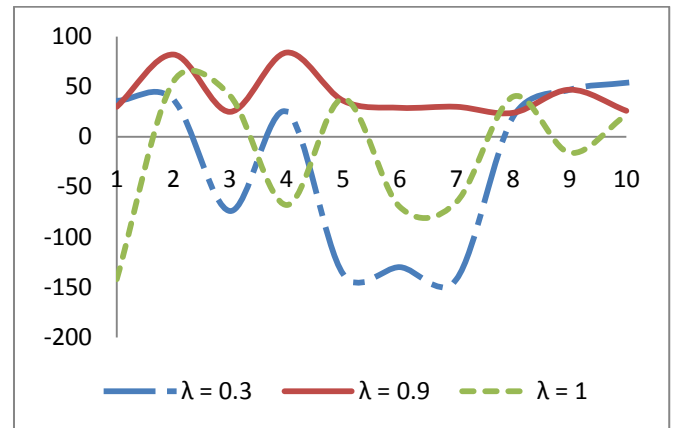


Figure 3: λ Effect On The Frag Difference

ANALYSIS AND FUTURE WORK

From the results presented and from observing the bot actions, the following points are noticed: Babis the learning bot is evidently superior to both the scripted bots [and Champanard's bots]. The learnt modified rule base presented surprising preferences. For instance Babis prefers chasing when confidence is weak and flees when confidence is strong. This may seem to be counterintuitive. However, after careful observation this strategy proved to be very useful. At the beginning of a confrontation Babis is usually cautious inflicting damage while avoiding the opponent, then after dealing some damage and getting hurt in the process, Babis becomes aggressive and finishes the badly wounded fleeing enemy.

An advantage of Fuzzy Q-learning is the amount of human control over the learning system. This is a desirable feature in the game AI field, as game designers do not usually wish for completely unexpected behaviors from the NPCs; they demand tools to script or at least guide the NPC in the course of the game. Though the proposed learning technique works well in adapting to the environment and the opponents, the basic fuzzy membership functions still have to be crafted by a designer or an experienced player. Fuzzy tuning addresses this problem where not only the consequents are learnt but

also the fuzzy membership function boundaries and even the functions themselves (Alcalá et al. 2000) (Er et al. 2004). The reward function was another problem with the implementation. The reward signal is general to the agent's global action, which might be deceiving especially in the case of multiple opponents. Hierarchical reinforcement methods (Ponsen et al. 2006) separate the reward signal for every action, so the agent receives accurate feedback. A combination of the subsumption architecture and hierarchical reinforcement learning can be an interesting experiment within game AI.

CONCLUSION

Reinforcement learning, while popular in the robotics field, has never found its way through to the video game AI applications. The project proved reinforcement learning algorithms are suitable and useful for simulating intelligence of NPCs and agents within a game. The implemented learning bot exhibited interesting capabilities to adapt to a human player playing style in addition to outperforming the other bots.

Testing results showed that incorporating learning into a game can reduce the extensive scripting and tuning phase usually following the AI development, while retaining the ability to guide the NPCs not to exhibit totally unrealistic or unexpected behaviors. This gives the game designers the prospect to explore new strategies.

The effect of the exploration-exploitation policy on the learning convergence was studied and tested in depth. The combination between the two most popular reinforcement algorithms (Q-Learning and TD (λ)) resulted in faster learning rates and realistic behavior through the exploration period.

REFERENCES

- Alcalá, R.; Casillas, J.; Cordón, O.; Herrera, F.; and Zwir, S. J. 2000. "Learning and tuning fuzzy rule-based systems for linguistic modeling". *Knowledge-based Systems: Computer techniques* 3 (29), 889–941.
- Berenji, H. R. 1996. "Fuzzy Q-Learning for Generalization of Reinforcement Learning". In *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems*, 2208-2214.
- Berenji, H. R.; and Khedkar, P. 1992. "Learning and Tuning Fuzzy Logic Controllers Through Reinforcements". *IEEE Transactions on Neural Networks*, 3, 724-740.
- Berry, D. A.; and Fristedt, B. 1985. *Bandit Problems: Sequential Allocation of Experiments*. Prentice-Hall, Englewood Cliffs, NJ.
- Bourg, D. M.; and Seeman, G. 2004. *AI for Game Developers*. O'Reilly, California.
- Brooks, R. A. 1986. "A robust layered control system for a mobile robot". *IEEE Journal of Robotics and Automation*, 14-23.
- Brooks, R. A. 1991. "Intelligence without representation". *Artificial Intelligence* 47, 139-159.
- Campbell, M. 1999. "Knowledge Discovery in Deep Blue". *Communication of the ACM*, 42 (11), 65-67.
- Champanand, A. J. (2003). *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors*. New Riders Publishing.
- Deng, C.; and Er, M. J. 2003. "Real-time dynamic fuzzy Q-learning and control of mobile robots". In *2nd WSEAS International Conference on Electronics, Control and Signal Processing*. Singapore.
- Glorennec, P. Y.; and Jouffe, L. 1997. "Fuzzy Q-Learning". In *Proceedings of the 6th IEEE International Conference on Fuzzy Systems*.
- Gu, D.; Hu, H.; and Spacek, L. 2003. "Learning Fuzzy Logic Controller for Reactive Robot Behaviours". In *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatron*. Kobe, Japan.
- Hooley, T.; Hunking, B.; Henry, M.; and Inoue, A. 2004. "Generation of Emotional Behavior for Non-Player Characters: Development of EmoBot for Quake II". In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI04)*, San Jose, California, 954-955.
- Johnson, D. and Wiles, J. 2001. "Computer Games with Intelligence". *10th IEEE International Conference on Fuzzy systems*. Melbourne, Australia.
- Jouffe, L. 1998. "Fuzzy Inference System Learning by Reinforcement Methods". *IEEE Transactions On Systems, Man, And Cybernetics*.
- Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research* 4, 237-285.
- Laird, J. E. 2001. "It Knows What You're Going To Do: Adding Anticipation to a Quakebot". *Agents 2001 conference*.
- Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. "Soar: An architecture for general intelligence". *Artificial Intelligence*, 33, 1-64.
- Lin, L.-J. 1992. "Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching". *Machine Learning*, 8, 293-321.
- Nilsson, N. J. 1998. "STRIPS Planning Systems". In *Artificial Intelligence: A New Synthesis*, 373-400
- Orkin, J. 2006. "3 States and a Plan: The AI of F.E.A.R.". *Game Developers Conference GDC*.
- Ponsen, M.; Spronck, P.; and Tuyls, K. 2006. "Hierarchical Reinforcement Learning in Computer Games". *ALAMAS'06 Adaptive Learning Agents and Multi-Agent Systems*, 49–60. Brussels: Vrije Universiteit.
- Saffiotti, A.; Ruspini, E. H.; and Konolige, K. 1993. "Blending Reactivity and Goal-Directedness in a Fuzzy Controller". In *Proceedings of the Second IEEE Conference on Fuzzy Systems*, San Francisco, CA, 134-139.
- Sutton, R. S. 1988. "Learning to Predict by the Methods of Temporal Differences". *Machine learning*, 3, 9-44.
- Sutton, R. S. 1996. "Generalization in reinforcement learning: Successful examples using sparse coarse coding". *Advances in Neural Information*, 8.
- Sutton, R. S. and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.
- Van Waveren, J. 2001. "The Quake III Arena Bot". *University of Technology Delft*.
- Wang, X.-S.; Cheng, Y.-H.; and Yi, J.-Q. 2007. "A fuzzy Actor-Critic reinforcement learning network". *Information Sciences*, 177, 3764–3781.
- Watkins, C. J. and Dayan, P. 1992. "Q-learning". *Machine Learning*, 8 (3), 279-292.
- Zadeh, L. A. 1965. "Fuzzy sets". *Information and Control*, 8, 338-353.
- Zanetti, S. and El Rhalibi, A. 2004. "Machine learning techniques for FPS in Q3". In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, Singapore, 239 - 244.