**Introduction and Motivation**

The goal of my project was to improve the state of artificial intelligence in Massively Multiplayer Online games (MMOs). While MMOs have grown increasingly popular, the AI has been criticized for being simplistic and unchallenging. To see more on this, please refer to my project proposal document.

**Research Vehicle**

I chose EverQuest (EQ) as the MMO that I would specifically focus on primarily because of the open source server emulator EQEmu. This project has been in development for over a decade, and now consists of over 400,000 lines of mostly C++ code. This allowed me to compile and run my own EverQuest server on my computer, which I could connect to and play on using the EverQuest game client, and gave me the ability to modify only the aspects of the game that I was interested in improving. Although connecting to a server not operated by Sony is technically a violation of the game's EULA, in practice Sony has never raised legal issues with the developers of EQEmu, nor with any of the owners or users of EQEmu servers. Additionally, since players (myself included) must purchase the game client, Sony does receive monetary compensation.

The sheer size of the codebase proved to be one of the most difficult aspects of the project to deal with, along with the lack of documentation. My first attempt involved deleting massive quantities of code in order to remove the aspects of the game that were extraneous to my project, in the belief that this would make the remaining code easier to understand and extend, and would free up computation cycles and memory to be used for improved AI algorithms. This proved to be a costly waste of time, since in the end I was unable to eliminate all of the dependencies that existed, and was forced to abandon this approach. My second attempt was to identify the top-level functions, and then insert my custom code at specific points to divert the flow of control. This provided the additional benefit of allowing me to enclose all of my code with the “#ifdef TUPES\_AI” preprocessor command, so that my contributions can be easily located and can even be removed by simply commenting out the “#define TUPES\_AI” line in features.h, which then compiles a standard EQEmu server.

**EverQuest Finite State Machine**

Although not explicitly mentioned or structured in this way in the source code, the AI for EverQuest can best be viewed as a Finite State Machine (FSM). Below, I will list the three possible states that an NPC can be in, along with a brief description of the behaviour and transition conditions of each one.

**Default State**

Behaviour: stand still or move along a grid of waypoints, choosing the next waypoint randomly. Ignore other NPCs.

Transition: whenever a Client comes within a certain range, check its value against your “faction” from the database (for example, there's a faction called the 'Splitpaw Gnolls'. Every time a Client kills an NPC on this faction, its value against this faction declines. If it's low enough, the NPCs on that faction will want to attack that Client). If this “faction check” determines that the NPC wants to attack, then it will check its level relative to the Client. If the NPC is not significantly weaker than the Client, then it will transition to the Aggro State.

**Aggro State**

Behaviour: Move towards the Client, and attack once within range. Besides the normal melee combat, attacking can also involve casting detrimental spells on the Client, or beneficial spells on itself or its allies, but these details are irrelevant to the overall FSM.

Transition: if the Client dies or leaves the 'zone', return to the Default State (the game world is divided into zones, and NPCs cannot move between zones). If the NPC's health points drop below a certain percentage, potentially transition to the Flee State.

**Flee State**

Behaviour: Move away from the Client, ignoring all other Clients.

Transition: If the health points recover above the certain percentage, transition back to the Aggro State.

**Criticism of the EverQuest FSM**

There are many aspects of this FSM that result in unrealistic AI. I will briefly mention a few of the most egregious examples in point form.

* NPCs ignore each other. This causes bizarre scenes such as a wolf and a rat walking right past each other. Not only is this unrealistic, it leads to boring and static environments. For a richer and more immersive experience, Clients and NPCs should be considered equivalent to the NPC AI.

* If an NPC doesn't transition from Default to Aggro, then it will completely ignore the Client unless the Client attacks first. For example, an elephant will ignore Clients because they will pass the faction check, and a Splitpaw Gnoll will ignore a Client that is a much higher level than it, even if they are supposedly bitter enemies. Of course, no real animal or enemy soldier would allow someone to run right up to it without reacting. Not only is this highly unrealistic, it makes gameplay far too easy. Killing NPCs is simply a matter of randomly finding them and then running up to them.
* NPCs will always fight to the death (or until the Client changes zones, which is another unrealistic component of EQ which is unrelated to the AI and lies outside the scope of this project). In nature, many times an attacker will retreat if it is injured by its prey, and animals of the same species will often fight for resources or dominance (including humans) without killing each other. This can often make the gameplay too difficult, since if a Client is attacked by a more powerful NPC, the Client's death is almost guaranteed.
* NPCs transition from Flee to Aggro. Normally, if an animal is nearly killed by another animal but manages to escape, it doesn't return soon after to continue fighting.

Although it might seem ironic that I simultaneously complain of the gameplay being both too easy and too difficult, what this means is that the gameplay alternates between the two extremes, which leads to a frustrating experience. Gameplay ends up consisting of long periods of mindless killing of weak enemies, punctuated by unavoidable deaths caused by powerful enemies. There should be a decent chance of unsuccessfully hunting any weaker NPC a player comes across, and a decent chance of successfully escaping any stronger NPC a player is attacked by.

**Criticism of Finite State Machines for Modelling Behaviour**

Although I believe that the EQ FSM is poorly designed, I don't believe that improving the FSM will solve the fundamental problem. While FSM's are undoubtedly an invaluable programming tool for a broad variety of problems, I don't believe that they're an acceptable model of behaviour. Animals simply don't have a set number of discrete well-defined states that they transition between. Therefore, I have created a new AI model that I call a 'Finite Emotion Machine', which I believe more accurately reflects real animal behaviour.

**Finite Emotion Machine**

While a proper analysis of emotions would require discussing the incredibly difficult (and some would say intractable) issues raised in philosophy regarding the mind, consciousness, and free will, for my purposes I will rely on the common-sense “folk psychology” view of emotions. That is, animals can experience, at varying levels of intensity, from a set of distinct feelings. These emotions each have a unique character, but they can be broadly classified as either inherently pleasant or unpleasant to experience, which drives animals to engage in behaviour that either increases or decreases these experiences, respectively. Although these emotions can operate concurrently, in general only the most intense emotion will control behaviour at a specific point in time (e.g. A rabbit that is extremely hungry, but is being chased in terror by a predator, won't alternate between running away and stopping to feed).

While I believe that something similar lies at the core of human behaviour, this new AI system was only applied to animal NPCs. All of the NPCs that belong to the more “intelligent” races, such as humans, elves, gnolls, goblins, etc, had their AI code remain unchanged.

For the purpose of creating an FEM, to define an emotion one must simply implement the two functions that are the necessary criteria for being considered an emotion:

1. Elicit()

This function receives information regarding the internal or external state of the NPC, and alters the 'intensity' level of the emotion accordingly.

1. AffectBehaviour()

This function will be called for the emotion that has the highest intensity. It defines how the probability of particular behaviours are affected, possibly depending on the intensity of the emotion. As an implicit third rule of defining emotions, this change in behaviour, if successfully performed, should result in the Elicit function receiving information that will return a lower intensity for the emotion. This creates a negative feedback cycle, and ensures that NPCs won't continue to engage in a behaviour indefinitely.

**Specific Examples of Emotions**

**Fear**

The first emotion that I defined was fear, since I believe that it dominates much of the interaction between animals. Its Elicit function is called for every NPC or Client (aka Mob) in the NPC's observable vicinity, and it considers the Mob's relative size and distance. The larger the Mob is relative to the NPC, and the closer it is, the more fear that is elicited. There are additional factors that I would like to include as well, such as the direction and speed that the Mob is moving, but I didn't have enough time to implement them. The fear elicited by each Mob is summed to give the final fear value of the NPC, but the Mob that elicits the greatest fear is stored as the aversiveStimulus.

The AffectBehaviour function classifies the intensity of the fear into weak, moderate, and strong. A weak level simply causes an orienting and freezing response towards the aversiveStimulus, a medium level causes the NPC to maintain its distance from the aversiveStimulus by moving away, and a strong level causes the NPC to run in the opposite direction of the aversiveStimulus.

**Aggression**

The Elicit function for aggression is very similar to fear's Elicit function, except that the *smaller* the Mob is relative to the NPC, the more aggression that is elicited. The AffectBehaviour function is also very similar, except of course it causes the NPC to move towards the Mob at higher levels, and will cause the NPC to attack if the level is high enough and the Mob is close enough. However, unlike fear, the aggression caused by multiple Mobs is not cumulative: the total aggression level is equal to the amount caused by the Mob that elicits the most.

**Pain**

Unfortunately, I didn't have enough time to fully implement the pain emotion, which was intended to address the issue of NPCs always fighting to the death as described above. The Elicit function would only require the current health points of the NPC, and the AffectBehaviour function would cause the NPC to move away from other Mobs to rest and regenerate health points. This and other emotions will be implemented in future work.

One interesting consequence of the FEM approach is that it allows new emotions to be defined easily. While I focused on the emotions that I believe are important for modeling animal behaviour in an MMO, any emotion can be implemented, even ones that don't exist in nature, simply by specifying the two functions that define the emotion.

The underlying approach behind the FEM model is that each emotion should be very simple, but that the interaction between the emotions, and between the emotions and the environment, and between the multiple agents in the environment, creates an incredibly complex system that is highly dynamic and unpredictable.

**Results and Evaluation**

The primary goal of the project, which was to create a more realistic AI for an MMO, has been successfully accomplished. Animals will orient towards you and each other. Large animals will chase smaller animals. Animals will charge at you, but if you can outrun them, they will eventually give up. All of these are dramatic improvements over the original AI.

While this FEM model resulted in an environment that was significantly more dynamic and interesting, it does have potential drawbacks and difficulties. Some of these issues could be easily addressed just by improving the functionality of the existing code. For example, since NPCs always run away in the exact opposite direction of the Mob that is eliciting the most fear, it is possible for NPCs to become trapped “bouncing” back and forth between two Mobs. This could be fixed simply by increasing the number of aversiveStimuli Mobs that the NPC will take into account in the fear's AffectBehaviour function. Other issues are much more fundamental, some of which I will list in point-form:

* the AI could become too challenging. For example, consider a player that is trying to attack a wolf, which then experiences fear and runs away. Unless the player is faster than the wolf, which is itself unrealistic, they will never be able to kill it. This caused me to consider how humans managed to hunt for thousands of years without guns, despite being much slower than almost every prey animal we would have encountered. It appears that there are a variety of non-exclusive successful strategies, some of which can be utilized in this game: players could cooperate to surround or corral an NPC so that they could get close enough to attack (in fact it is even possible to utilize other NPCs to help with this), and/or they could use ranged weapons such as bows and spears (and at least in EQ, spells). Also, while humans have a significant disadvantage in speed, our endurance is actually very good relative to other animals, enabling a technique still practiced in some parts of the world called 'persistence hunting', which involves chasing and tracking an animal literally to the point of exhaustion. To implement this, I've added an attribute to the NPC class called 'oxygen', which regenerates over time but is reduced when an NPC is running. If the oxygen level reaches 0, the NPC is no longer able to run, and would be easily killed. Unfortunately, I haven't yet implemented the tracking component, which could be accomplished by having the NPC periodically 'drop' items on the ground called '<type of NPC> tracks' (but due to the graphical models available, these wouldn't actually look like tracks).
* While unpredictable behaviour is good for the player, it makes the developers' job more difficult. Even as the creator of the system, I found it impossible to predict what an NPC would do once it was experiencing multiple emotions and there were multiple NPCs in one area reacting to each other.
* The fact that NPCs are evaluating every other NPC in their vicinity, as opposed to just the Clients as in the original AI, increases the computational workload. While I haven't encountered any issues with this as the only player on my server, it is possible that there would be scaling issues if my server were to try to handle the player population of a real EQ server, which could be in the thousands at any one time.
* There were many 'magic numbers' acting as parameters that I had to set. For example, if fear's Elicit function receives the size ratio and distance of the target being considered, a third number must be used to determine the relative weighting these two factors will have. As the number of emotions increases, so does the number of these parameters, and existing parameters that were already set could need to be adjusted. I set these numbers through trial and error, but this is a highly time consuming and unsatisfactory method.

**Future Work**

There are many additional features that could be added to this project. While the number of small changes and improvements that I intend to work on are innumerable, I will describe two long term projects that would significantly improve the AI in this system.

One follows naturally from the problem of setting parameters described above. A genetic algorithm could be used in order for these numbers to become optimized over time. Due to server shutdowns, these parameters would have to be stored in the database (or some other persistent storage), and then whenever an NPC is created, its NPC-specific parameters would be pulled from storage but then slightly randomized. Its performance would then be evaluated based on how long it survives, and depending on this performance, the stored parameters could be adjusted towards or away from the NPC's values. Over time, these parameters could evolve to become optimal or near optimal for each type of NPC. A difficulty with implementing this idea in a commercial game is that it would mean the NPCs would have to be allowed to evolve in a completed game world for a period of time before the game could be released.

While a genetic algorithm could be used for setting the parameters, which is of primary importance in the Elicit function, a more ambitious project would be to use reinforcement learning algorithms for improving the AffectBehaviour function. One of the primary components of RL algorithms is a reward signal, and this model already provides that. Since emotions such as pain and fear would be considered as negative, the increase of these emotions could be used as a negative reward (punishment), and the decrease of these emotions as a positive reward. The difficult part would be encoding states and actions in a way that they could be assigned values, so that these values could be changed over time by the RL algorithms. For states, a feature based approach could be taken, so that the current state would be determined by the value of a finite number of features, such as the distance to a target. The difficulty with encoding the actions would be determining the level of granularity. For example, the set of actions to choose from could be {move left, move right, move forward, stand still}, with more complex behaviours being performed by chaining these simple actions together, or the set of actions could be more like {hunt, escape predator, find safe resting spot, etc}.

**README**

Because my installation and compilation processes are fairly involved, I have written the README here and copied it into the readme.txt file.

**Operating System**

For creating my initial server, I followed the guide at<http://proskeptic.com/wordpress/> almost verbatim. Since the only Ubuntu system I could get it to compile on was 11.10 32bit, and my desktop is 12.10 64bit, I created a virtual machine using VirtualBox and installed Ubuntu 11.10 32bit, as described in the guide. Using 12.10 will result in everything compiling except the EQEmuLoginServer. If you do not create a virtual machine using the User and Computer names described in the guide, several of the ini and xml files will need to be modified (see write\_scripts.sh below).

**Scripts**

I have included several bash scripts in my submission (taken from the guide), which expect there to be a /home directory, and will create everything in /home/eqemu. Please execute 'chmod ugo+x' for each of these scripts, and then execute them in the following order:

configure.sh: if you're not using a virtual machine, please read this one *very* carefully, since it appears to modify the kernel. Also installs dependencies using apt-get, including mysql-server

downloads.sh: downloads source code and moves files around as needed.

compile.sh: compiles code and creates links to executables. Assumes you are using Ubuntu 11.10, so if you're not, you may have to modify lines 21, 24, and 25.

create\_db.sh: creates MySQL database and inserts data.

write\_scripts.sh: creates three ini files and one xml file. If you are not using the User and Computer names described in the guide, either this script or the scripts it produces will have to be modified.

**Compilation**

If you've followed the steps above, there is already a /home/eqemu/source/EQEmuServer/zone directory with many source files and an executable called 'zone' that was compiled during the compile.sh script above. My submission contains a directory called zone, which is intended as a drop in replacement for this. Once you've replaced the old zone directory with my zone directory, you must move into it and run the make file in it. This is also where you would want to use a tool like grep to find all of my code, which, as described above, is always prefaced with “#ifdef TUPES\_AI”. The majority of my code is in npc.cpp (and the corresponding npc.h), with some in MobAI.cpp. Although the total amount of my code is not large, please keep in mind the amount of time and work that went in to reading the (mostly undocumented) existing source code, in order to figure out exactly where to insert my code and which functions to use.

**Running and Stopping**

Once the zone executable has been compiled, and the computer has been restarted, the script startup.sh located in /home/eqemu/server is used to start the server. Another script named killeq.sh can be used to shutdown the server.

**Game Client**

I'm not sure how you want to do this. Either I can lend you my client to install (must be on Windows), or we can arrange a meeting where I bring my laptop with EverQuest installed and you can use it to connect to the server that you've compiled with this code. If you plan on obtaining your own client somehow, please keep in mind that only certain versions of the game work, such as Titanium and Secrets of Faydwer (which is what I have).

**Evaluating**

If you end up playing the game to evaluate the AI, you may find it useful to update the Status attribute of your account in the accounts table in the database to a value of 250. This gives you access to the following commands (among many others) while you play:

#invulnerable on // so you don't have to worry about dying

#zone southkarana // takes you to a zone with lots of animals, other possibilites include // westkarana, northkarana, and eastkarana

#gmspeed on // for moving VERY fast (not realistic if testing normal user experience)

#time 12 // for when it's dark, this will make it noon

#kill // for killing the targeted NPC

#heal // for healing the targeted NPC

#size <some\_int> // for changing the size of target, normal values are around 3-10

**Optional**

One thing to keep in mind is that my new AI code is only for animal NPCs. This means

that if you keep the database as is, there will some animals controlled by my code

intermixed with a bunch of NPCs controlled by the (dumb) existing AI code.

This can be interesting, but if you want to just have my AI controlled NPCs, you can

execute the following SQL statement to the MySQL database:

DELETE FROM npc\_types WHERE bodytype <> 21