Technical Corner:

Fuzzy Logic

You can use Fuzzy Logic NOW and improve almost any application.

By Gary L. Alston

OK, so a lot us have heard of fuzzy logic, but most of us don't have a real handle on it. More to the point, most of you don't really know what good it is. To start with, Fuzzy Logic (from here on abbreviated as FL) is a misleading name. What it does is allow a method for a program to make decisions based on highly variable data, as in **real world** data. That's all. More importantly, FL is simple. Many of you have read articles on FL in Dr. Dobbs and in other publications, and the subject matter looks pretty imposing. Scary even. Unfortunately, most of the articles I've seen apply FL technique to exotic problems; these are problems we're not likely to see in our normal consumer software work. The name and the exotic application of the technique in articles is enough to scare normal people away from it, or at the least make them feel that FL is not for them.

Not necessarily true! FL has a wide variety of uses. To illustrate this, there is a little game in this STARgazer (VB source included) that shows how FL can be used in consumer software. (In this case it's a game but it's relatively simple to create examples for other specific applications. The game was provided more for the purpose of juxtaposition of ideas than any other reason.) Later on we'll discuss how the game works.

There's also a F.L. program included.

This program is a WIn3 app called FuzzGen, and it's a little CASE tool. What it does is to allow you to create a visual representation of how a decision **looks** in order to make understanding it easier. It then generates source code for you in one of three standard high level languages. It was used to create the Fuzzy routines for the demo game.

Writer J. Scott Edwards says that *"this is a great roll up your sleeves and jump right in"* type of product. He's right. Bottom line? FL isn't for the exotic, it isn't hard to understand, it isn't something that requires a math major, and it works. The accompanying disk-based FuzzGen manual contains a small tutorial on FL, so we don't need to rehash that here. Suffice to say that with the right tools FL is quite simple.

On to the game.

The game (FUZZGAME.EXE) is simplistic, and is a cheesy knock off of the the basic shoot-at-'em arcade games. Essentially, a plane flies from right to left and you need to use the crosshair to show the point where the missile you launch will detonate. Like in the real game, a "hit" is rewarded with an explosion... and that's where any similarity ends. What happens is that the difficulty of hitting the target changes based on how close you were the last time. What we're trying to demonstrate is that the game is reacting in real-time based on the ability of the player. *It is adjusting play based on the **user's** ability rather than using a pre-set model and accumulating points.*

Why was this cheesy game used to demonstrate this? The primary reason is that my 7 year old, Lindsey, is barely capable of playing some of the arcade-style educational games around that purport to be designed for her age. Some of the damnable things are too hard for **me** to play! Hopefully this game, simple as it is, may get some of you thinking about ways to make software more friendly to people. Children who have poor hand-eye coordination needn't suffer because a game is beyond their ability, for instance. It **is** possible to create games that adjust themselves to the (limited or otherwise) capability of the user as opposed to forcing the user to the speed of the game. Obviously there are myriad ways to implement FL routines in games for a wide variety of reasons. In fact, I may be showing a poor example, because I'm not a game programmer. I take comfort in the fact that I am addressing a very clever audience. I'm likewise confident that you will be able to see past my lack of writing ability to see the greater point.

More importantly, once you look at the source code and the FL model used you will realize how simple it all is.

How it works.

Start of the game is by clicking the PLAY button. In the picture box you'll see a crude image (it's an icon) of an aircraft moving from right to left. Putting the cursor in the picture box will change its' shape to a crosshair. Clicking at any point will cause the "missile" to fire. (Another crude icon.)

Missiles and aircraft are updated at a constant rate of 10 times per second by using a timer with a 100 mSec interval. The distance is variable on both the missile and the aircraft.

Detonation of the missile occurs at the spot clicked. When the missile reaches this point we look at the position of the aircraft. The distance between the two is the data point fed to the FL routine. The FL routine is used to gauge the "speed" of the NEXT aircraft.

You have two pieces of information to study here -- one is the source for the

game itself, and the other is using FuzzGen to view the FL routine and how it is used as the model. By all means change the model and experiment with it. The great thing about FL is that you can change the behaviour of the game by simply using a different set of outputted variables!

Other considerations; other ideas.

Now for the $64 thought -- in most games, the player is awarded points based on the ability of the player to perform as expected by the program. The traditional missile-shooting games that this game knocks off certainly work this way; they tend to spit out random targets at random speeds and detonation proximity is the only difficulty factor rated. Our game is different in that the target speed is no longer a purely random event: we have a criteria (last proximity) to work with to make the next target action easier on the player. The key thought I had here is that it seems easy to make a game self-adjusting such that slower players have a built-in feedback mechanism, they can "practice" while playing, and the game experience may be better for them. Faster players would have the same level of enjoyment as slower players since the game can adjust itself to them...

Of course, this isn't applicable to all games for obvious reasons. However, this is something I think deserves serious consideration when developing games for the younger crowd. I'd like to impose another example, if I may... my daughter Lindsey is usually pretty hot for her new Game Gear games since they are supposed to be geared to her age level. However, she rarely gets past the first level without considerable practice. The games that she can actually get somewhere with are the ones she really plays. This is why I think games are a great candidate for feedback techniques, and I think FL is the fastest possible way to implement these.

Bottom Line.

FL isn't exotic, it isn't hard, and it's suitable for a wide variety of uses. In this short article I have demonstrated a method for using FL as a feedback mechanism in a game, and I have tried to make the case that some games are good candidates for such usage. FL is usable for a great deal more than simple feedback mechanisms, however. The uses you can find for it are staggering. I know of one use that seems so obvious you wonder why it wasn't done before now: a police dept. uses FL technique as a filter for physical descriptions of suspects in the police database. The victim claims that the assailant was "tall,"
but what is tall? The FL filter takes into account the age/height of the victim. To me, at 6 feet 1 inch, tall is, oh, 6'5" or better. To 7 year old Lindsey anything over 5 feet is tall... and FL takes "tall" and makes it mean something more useful.

So, you application developers can hopefully see uses in financial analysis software, banking (do we give these guys a loan?), and even games, making the computer opponent truly variable with Fuzzy Logic techniques.

The included ZIP's (FuzzGen.ZIP and FuzzGame.ZIP) contain the necessary files for you.