Logical and sub-logical analysis

Unedited posts from archives of CSG-L (see INTROCSG.NET):

Date: Thu Oct 14, 1993 9:19 am PST
Subject: Logical and sub-logical analysis

[From Bill Powers (931014.0910)] Avery (931014.1530)

> The control system responsible for direction of movement produces an error signal proportional to the angle between the current direction of locomotion (the direction the CROWD person is headed) and the direction of the goal. But there could be another system that ascertains whether motion in the current direction is blocked for the goal ...

In CROWD there is such a second system. However, it ascertains something much simpler which has the same effect: whether something is too close. Instead of analyzing the scene and reasoning about it, the person in CROWD simply avoids, at all times, too much proximity to anything, divided into two classes: total proximity in the left-forward region, and ditto in the right-forward region. Too much proximity to the left results in a veer to the right, etc, the radius of curvature automatically being inversely related to total proximity. As designed, this system simply overpowers the destination-seeking system, but it could have modified the destination-seeking system's reference angle to avoid conflict.

In your diagram, a CROWD person would simply head toward the goal until it got close to the wall, and then veer left or right to avoid collision. It would move along the wall to an opening, and if the sides were far enough apart, through the opening and on toward the goal. The effect is to pick an opening that is reasonably close to the direction of travel. Not, of course, by the shortest path. The person doesn't actually seek the opening; in effect it just keeps moving until there is decrease in proximity in a direction toward the destination, and that reduces the tendency to veer away from that direction.

In my original stab at the CROWD program, each person had a 360-degree map in 6-degree bins, with total proximity recorded in each bin. The idea was to scan the map and look for minima in total proximity near the direction of the goal, and set the position of that minimum as the immediate destination. When the person got to that gap, another scan would take place and a new gap would be sought. This would have resulted in more intelligent behavior in some circumstances, but because what lay beyond any given gap was invisible, the result wouldn't have been a lot better. And because there were other active people on the field, a gap wouldn't necessarily stay put or continue to exist while you were traveling toward it; in fact the scan would have to be redone on every step, or at least very frequently, and a wrong guess about which gap would remain open might result in a very unintelligent-looking path. Also, as you can guess, by the time I had 10 or 15 active people, all making maps, the program was simply CRAWLING along.

I soon saw that I was getting into a complicated logic-level model, and that I still had to solve the collision-avoidance problem, so I looked for a simpler way. The simpler way proved to be just to avoid collisions and forget about the map. This method has all sorts of intelligent-looking effects that would require very complicated searching and reasoning to accomplish by a brute-force logical system -- for example, getting back out of a cul-de-sac, or deciding that there are no gaps of a large enough size and turning away to go completely around the crowd.

In fact the simpler program does many things that I never would have thought to provide for at the logic level and that would have had to be added to make the program actually work. Just think of the logical problems when one group of five people heading toward one destination crosses the path of another group of five heading toward a different destination, all in the midst of a scattering of stationary obstacles!

One of the most interesting things about CROWD is the way people rationalize it after seeing it operate, but without knowing how it works. It's almost certain that the "explanation" will be at too high a level. "Look," the observer says, "he's trying to get out of that pocket, looking for the opening." Of course he isn't; he doesn't even know he's in a pocket or that there is an opening. All he knows is proximity to left and right, and the direction and proximity of the destination.

Your proposed rule

> Direction D is blocked for goal G if, proceeding in D, you will arrive at an obstacle, such that, when you then head toward D, the obstacle is still in the way.

is probably an unnecessary complication, a rationalization of something that can be done far more simply at a sub-rational level. While the rule can be stated relatively briefly, getting the information needed to apply the logic could be extremely complex. For example, to determine whether the direction of travel will intersect an obstacle, you must project the direction of travel to see whether the resulting line crosses the boundary of some outline; this means testing every point along the line to see if its coordinates are the same as some coordinate in the boundary of some object. Then you must change the direction of the line infinitesimally and do the computation again, until finally a direction is found where the distance to the intersection is sufficiently large. This is the sort of burden of computation that results from characterizing the problem as one of logic and just trying to compute your way through it.

If you think in terms of analog perception, it's obvious that if you're going to collide with an object, that object will be seen in the middle of the perceptual field, where "straight ahead" always is. In CROWD I assumed some kind of visual perception, with proximity being indicated roughly by the area of each image, which I could compute as proportional to the inverse square of the distance. I could then, knowing the position and orientation of the person, and the positions of all obstacles, easily compute the proximity that would be seen to left and right for all objects -- in effect, simulating the environment and the visual equipment.

The result fits your proposed rule, but without applying any logical rule. And the ensuing action doesn't require reasoning, either: you don't need to reason that if you keep going in the same direction, you'll hit the obstacle, and therefore you should institute a turn. You just hook up the perception to the turning apparatus via a comparator.

I am always amused by "fuzzy logic" programs, which try to solve simple analog problems by means of logic-type reasoning. If the inverted pendulum is medium-left and moving left, apply maximum rightward force; if medium-left and moving right, apply medium leftward force. And so forth for dozens of rules. Also, grade the amount of each variable by some distribution around the exact condition, in a "fuzzy" way. So you end up doing something with a high-speed RISC processor that could be accomplished at a sub- rational level with one operational amplifier and a couple of resistors and capacitors.

If you solve these problems at the lowest level possible, then when the time comes to consider more complex aspects of the situation, all you need is to decide which analog solution to bring into play. You don't need to carry it out at the logic level: it's already been solved sub-logically. This means that the logic actually required will be quite simple. Let the logic level pick the destination (I'm hungry and it's 12:00, so pick the coordinates of the hot-dog stand as a destination), and the analog systems will get you there without bumping into anyone.

Best, Bill P.