Jefferson TDR

Jamie Doornbos
10/28/97
Revision 0.

"Find Best Action" Primitive.

The Find Best Action primitive is intended to encapsulate the process of finding the most productive interaction that a person can run at any given moment. The algorithm takes a comprehensive look at the current set of available interactions in the world, and scores them according to "advertisement values" and "contribution functions", as well as distance. This document describes the components of the first pass at this algorithm.

For each object…

On the macro level, the primitive is looking at objects. As each object is considered, it is culled based on the following criteria:

- The person calling the primitive is not considered. TBD: reflexive interactions will probably be a part of game play, but are not currently designed as such.
- Objects with the "occupied" flag set are not considered. This flag is set voluntarily by object interactions that can only be run by one person at a time. The flag is also set for objects being placed.
- Objects with a non-zero "use count" field are skipped. This value is voluntarily set by interaction trees when exiting, and is decremented automatically by the simulator each tick. It is essentially an optional, cheap way to help keep objects from getting crowded.
- Objects without a tree table are skipped.

TBD: the initial use count could be a per-interaction tuning value.

TBD: objects may also be culled by type, when "affector" types are implemented. (see person chapter).

TBD: A subset of objects may be chosen in order to optimize. That is, only a certain subset of objects will be analyzed each time the primitive is called. This may be designed such that the subset is random, and all objects are eventually picked after a number of calls proportional to the number of objects. Or it could be simply sequential, with a contiguous fracton being considered each time.

For Each Interaction…

Once an valid object has been found, each interaction in the object's tree table is examined and scored. Each interaction consists of a set of motive advertisements, a check tree, an action tree, and an attenuation value.

Motive Advertisements

Each interaction has a set of advertisement values, one value for each motive. The advertisements are supposed to roughly indicate how many much the motive can increase if that interaction is run. Zero indicates that the interaction has no effect on that motive. A negative value indicates a negative effect on the motive. The advertisement values are recorded statically for the object's class and are loaded into a tree table structure when the object is initialized.

Check Trees

Each interaction may also have an associated check tree. The purpose of the check tree is two-fold. One purpose is to determine if the interaction is available for that particular person. A check tree returns false if

the interaction is not available, and true if it is. For example, a “place on table” interaction check tree may return false if a person has nothing to place. The other purpose is to modify the motive advertisements to reflect how much the interaction can affect the particular person considering it. The check tree is provided with a copy of the static advertisements, which it can modify as necessary, based on the personality of the person, or on the relationship of the person with the object. If an interaction does not have a check tree, it is assumed available and the static advertisements are used.

TBD: interactions may also need to be culled by previous failure, requiring some record of the failure of an interaction. Otherwise the scores of an interaction will remain the same and the person would be likely to keep choosing the same interaction over and over.

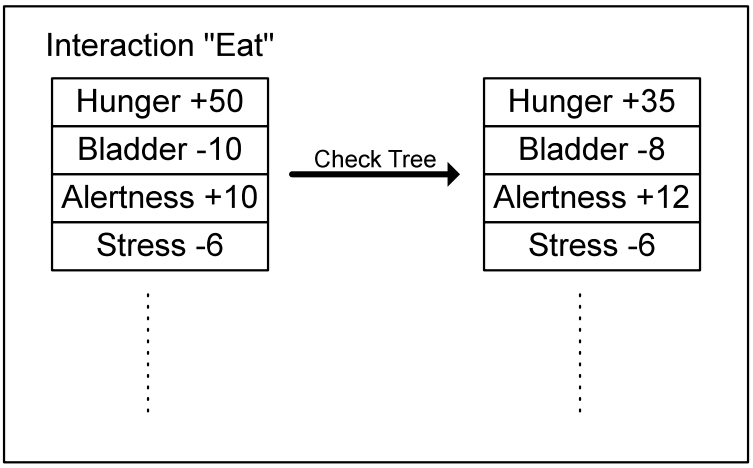

Figure: how the eat interaction advertisement might look before and after a check tree is run.

Scoring an interaction

Once the final set of motive advertisements has been found, their effect on the person is measured through a happiness contribution function.

Measuring happiness

A simple measure of happiness might be the average of all a person’s motives. “Find Best Action” takes this basic approach, but also uses a “cutoff point” for each motive above which additional motive value contributes nothing. So rather than a straight sum, the average is over the sum of the contributions for each motive. Important note: the cutoff points are tuned to the Motive Simulation, described in the person chapter. For instance, the cutoff for stress is at 0 because the motive simulation pushes stress toward zero.

TBD: Currently the motive contributions are wired into the algorithm, but will probably take on values within a certain range and be customized for each person.

TBD: cutoff values are probably part of a person’s personality.

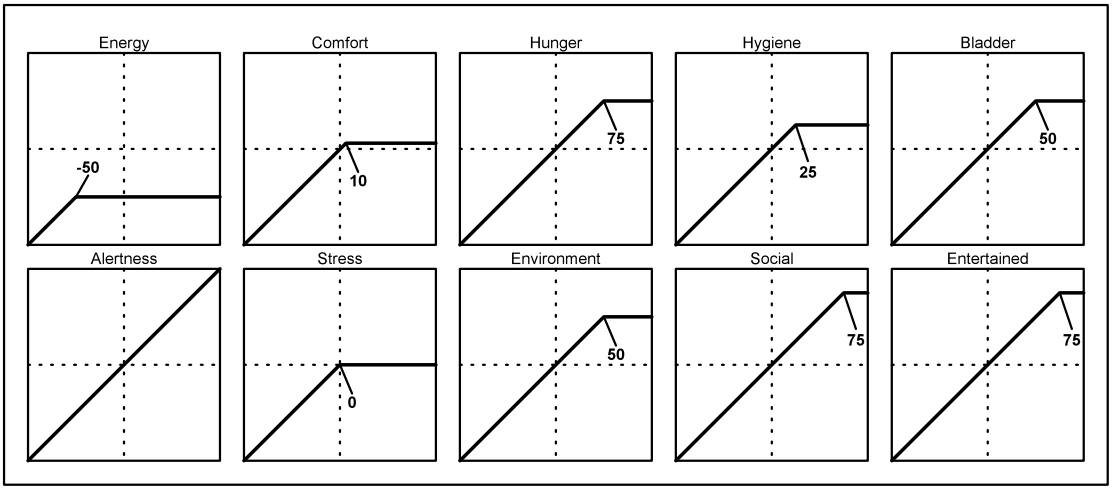

Figure: contribution functions for each motive.

Overall score

The overall score of a given interaction for a given person is obtained by scoring the person twice: once with their current motive values as they are, and again with their current motive values augmented by the motive advertisements of the interaction. The former is subtracted from the latter to find the overall score. Thus, an interaction that provides for only one motive will have an overall score of zero if that motive is already over the cutoff point.

Attenuation

Once the overall score of an interaction is known, the distance from the person to the object is used to dampen the score. The dampening factor is a floating point number from 0 to 1 and is provided by the interaction. An interaction may provide a value of 0, which means there is no dampening. The multiplier used to dampen the score is:

attenuated score = overall score / (attenuation * distance + 1)

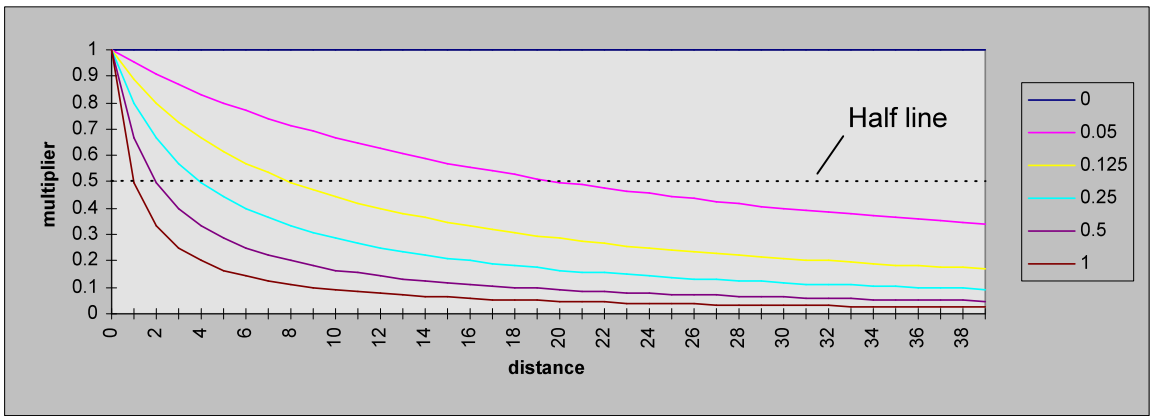

Figure: sample curves for several attenuation values.

Building the interaction list and sorting

Current implementation. TBD: this could change significantly.

Each instance of cXPerson currently keeps an “InteractionVector”. This structure subclasses from an STL vector of interaction records, which contain the object id, the interaction number, and the attenuated score. For debugging purposes, the interaction vector also stores the person’s base happy and the simulation tick count at the time of computation. Also for debugging, each interaction record contains some intermediate results, such as the original happy score, and the distance. Currently, 512 entries are reserved when the person is initialized.

When the primitive is called, the vector is cleared and rebuilt. Once it is built, its base pointer and size are passed to the standard lib routine, qsort, with a function that compares the attenuated scores of two interactions records. After that, the 0^{th} entry in the vector is taken as the return value of the primitive.