Automaton

A tool for designing intelligence

An implementation of cognition(? not really)

By TBD

Glossary

Because several words are used naming structures, and the structures resemble organic structures.

Synapse = Ganglion
Gate = Synapse
Brain = a collection of neurons, synapses and other components.

Principle operating components.

Neuron
Synapse
Gate
- Source
- Target

General predefined transformations.

Neuron

The default behavior of a neuron is the sum of all inputs compared vs a static threshold

Gate(Target)

The default gate is a scalar times the output from the related source neuron.

Principle interoperation components

Input

This is really a external algorithm for a neuron; or it is a subclass of neuron. Input serves to provide information from the environment.

Output

This is a specific algorithm for a neuron; and serves to provide output to the environment.

Shared (Input type and Output type)
A class of neuron should exist that can provide a source of communication between nodes(modules). Synapses connected to this result in a value that is seen in the current node and any other node that references this shared neuron.

Principle structure

Node
A node is a collection of neurons; it serves a specific function, and is used for building higher level structures. A node may also contain other nodes (or at least a reference) but a reference to a node must provide values for it as an instance. *too specific*

Brain
A brain is a special case Node that contains all other nodes, and is the source of neurons and synapses. Nodes technically do not allocate the neurons, but rather they come from a higher level root structure called the brain. A brain is also what contains the processing loop; nodes are processed, while the brain ‘node’ is what does the processing.

The Neuron

This is the simplest component. It accepts connections, and can source connections. Connections are made with gates and synapses. A neuron implements an algorithm to generate output(s) given inputs.

The Synapse

This is a connection between neurons. It is unidirectional. It interfaces with neurons with gates. A synapse is a double link reference between two neurons. It performs no translation.

The Gate

This is an interface between the synapse and the neuron.   It may be implemented as separate types of gates for the source connection and the target connection.

Source Gate

In essence, there's really only one transformation point, so this is generally a null operator.

Target Gate

This is the active part of the connection between neurons. The gate to the target neuron implements an algorithm to which transforms the output from the source neuron.

Default analog neuron algorithm

N_{out} = Neuron output level. All synapses receive this value.
$N_{in\#}$ = Input from other neurons. The neuron receives no indication of the purpose or identity of the input.
T = neuron threshold level; Essentially if the input is over this level, it is a positive output, or 0 or a negative level.
G() = translation function of the gate.

Neuron min and Neuron max set operating range limits; default value is -1 and 1.

N_{out} = Neuron Min < (Sum(Integral symbol) (N_{in1}+N_{in2}+$_{...}$) - T) < Neuron Max;

$N_{in\#}$ = G(SN_{out}); // bad representation

Default digital neuron algorithm

N_{out} = Neuron output level. All synapses receive this value.
$N_{in\#}$ = Input from other neurons. The neuron receives no indication of the purpose or identity of the input.
T = neuron threshold level; Essentially if the input is over this level, it is a positive output, or 0 or a negative level.
G() = translation function of the gate.

Neuron min and Neuron max set operating range limits; default value is 0 and 1.

N_{out} = 0 if Sum(Integral symbol) (N_{in1}+N_{in2}+$_{...}$) < T
1 if Sum(Integral symbol) (N_{in1}+N_{in2}+$_{...}$) > T

$N_{in\#}$ = G(SN_{out}); // bad representation

Default scalar gate algorithm

SN_{out} = Source Neuron output level

G_{gain} = a scalar constant from -1.0 to 1.0.

G(n) = G_{gain} * n;

Summary

What has been defined until now is sufficient to implement and demonstrate feedback systems. They are static structures, which themselves really do nothing but provide values and relations. Neurons may source connections to themselves; as an immediate loopback, connections may form loops such that eventually the neuron is connected back to itself via other neurons. Simple logic operations such as AND, OR, XOR, et al. can be created using digital or analog neurons and the scalar on the gate. Simple oscillators with variable time can be constructed also; more complex things like flip-flops and latches.

Processing Algorithm

There are several ways that this may be implemented. I will describe generally what was previously implemented, and additional refinements.

Each node has a process sequence identifier. Each neuron also as a process sequence reference. This sequence allows recursive loops to be resolved into a linear process.   When processed, the neuron's sequence is updated to the processing node's sequence; then subsequent requests to process this neuron based on recursion will return the node's prior value for use to compute the next value.

For brevity; PSeq will be used to reference the process sequence identifier.

The process... (pseudocode)

```
Increment brain.Sequence
For each neuron in a brain
    ProcessNeuron( brain.Sequence, neuron );

ProcessNeuron( fromSeq, neuron )
    Increment neuron.PSequence.
    If neuron.PSeq  is not equal  fromSequence
        // update the sequence now.  Further requests to process
        // this neuron because of a recursion will return the prior output value
        update neuron.PSequence == fromSequence
        input total = 0
        for each connected synapse
            // recurse
            ProcessNeuron( neuron, synapse.Source )
            neuron.inputTotal +=
                synapse.targetGate( synapse.Source.output );
        //Apply neuron algorithm to input total, and save output value.
        neuron.Output = Neuron.algorithm( neuron.inputTotal );
    return neuron.Output;
```

Generally the above will work. But, because most neurons in a node are all connected, after processing the first node, all other nodes will have stepped to the current sequence, and iterating through the remaining neurons in the main loop will be useless work. If the ProcessNeuron results whether it processed the neuron (ended up incrementing its PSeq), then the island root can be saved in a list.

```
for each neuron in island list
    if( ProcessNeuron( neuron ) did not process )
        Remove neuron from island list;
            it is not an island anymore,
            and a prior process iterated it's PSeq.

//then, to catch new islands… This process can be run at a lower priority, or
// for instance 1 in 10 times to process; if the node is in a non editable state, this
// process does not have to be done; unless nodes are triggered that create other nodes.

For each neuron in a node
    if( neuron.PSeq != node.PSeq ) // quick check to skip the call
        Add neuron to island list.
        ProcessNeuron( neuron );
```

Alternative optimizations may start by processing output nodes as the root node; recursive handling will handle gathering all inputs, and triggering algorithms on input nodes to sample the environment. Some input nodes may work asynchronously to the neural network, and when queried just return their prior state of asynchronous activity.

Input and Output Examples

Some examples of output might be simple abstract ideas like Move Forward/Backward,Turn Left/Right. These are values sent to other potentially complex systems to perform a function.

A simple linear radar might have an input that is the distance to the object the radar hit. A radar would also have outputs to control direction (mounted on a motor), or frequency (shift what sort of density it was looking for).

Some other Neural Algorithms

Sigmoid function

The sigmoid function has an additional constant T that controls the shape of the curve, the output characteristic is -1 to 1 for most of the range outside a small target with a short range near 0 resulting which is an exponential signal. Biologically based neurons behave more like this. The original threshold parameter can be used to bias the 0 point.

Signal and Noise generation function

A neuron algorithm doesn’t have to respect any inputs, and it could just generate random values. It could generate a sine output based on time or other external parameters. …

Designing Digital Logics

As an experiment, I implemented operator overloads in C# on the Neuron class. This allowed structures to be built with expressions….

```
Neuron n1 = brain.GetNeuron("n1");
Neuron n2 = brain.GetNeuron("n2");
Neuron n3 = n1 & n2;
Neuron n4 = n1 ^ n2;
```

N3 and N4 are neurons that are created and connected to neurons N1 and N2.

Example implementation of operator XOR

```
Neuron operator ^( Neuron n1, Neuron n2 )
                Synapse s1 = n1.brain.GetSynapse( "^S1" );
                s1.gain = 0.25F;
                Synapse s2 = n1.brain.GetSynapse( "^S2" );
                s2.gain = 0.25F;
                Synapse s3 = n1.brain.GetSynapse( "^S3" );
                s3.gain = -1.0F;
                Neuron output = n1.brain.GetNeuron( "^Output");
                output.Logic = new NeuronLogic( NeuronLogic.Algorithm.digital );
                Neuron extra = n1.brain.GetNeuron( "^Extra" );
                s1 += n1;
                s2 += n2;
                output += s1;
                output += s2;
                output += s3;
```

synapses have the += operator set to be Link Synapse To/From

(think that’s incomplete…. was more of an idea)

Graphical Interactions

The brain itself is a logical construct, and should be implemented entirely separately from methods to manipulate it.

A infinite scape 2d surface called a board may be used to construct neural structures. There are 2 basic types of objects that can then be subclassed to specific purposes. Pieces should provide feedback for the user to observe their current state. A color intensity or shading on a portion may indicate current input levels, and current output. If digital, these changes will be quickly apparent, but in analog values close to 0 may not be able to be seen, so maybe a seperate indicator indicating sign. Text labels may be used to convey information about a neuron. Probe points on synapses may convey the values they carry.

Piece

A piece is a specific location. It can cover a large area. It may cover a different amount of area depending on the specific subclass of piece.

Via

A via is a pathway that connects pieces.

Piece Basic Subclasses

Background

The background piece is a root piece. It has methods such that you can interact with it and add pieces to it. This determines the sort of board it is. Implementation of right click to popup a menu to select a piece to be added to the background.

Neuron

A general neuron. Clicking on this piece should show some sort of properties to be able to set internal parameters. Example parameters might be the threshold level of the nuron.

Synapse

A synapse is actually a specific piece type that is a VIA. a synapse is started by clicking on a neuron which allows the neuron to confirm or deny the beginning of the connection. When the synapse is attached at the end, the neuron it is connected to also has the ability to confirm or deny the connection.

Synapse Gate Interface

The Synapse is a pathway, so it must have a place that it starts and a place that it ends, and these are two different places. If the ending point property of the synapse representation is interacted with it should control the gate transformation function; for example to set the gain.

Input

Adding an input to the board, should bring in a neuron type that has a algorthm that associates it to a value external to the brain.

Output

Adding an output to the board allows it to set a value external to the brain.

Some Outstanding Issues...

How to handle the sum of the inputs. If a neuron has MANY inputs, then the sum of the input will clamp the output at a maximum level. In reality this could be a destructive overflow event; in an electric circuit, this would resemble a current or voltage overflow, and if the part is a behavior like a resistor, will result in heat potentially destroying the part.

Piece Interface

At a minimum the piece must have the following methods available

```
          object  Create ( object psv_userdata );
                    A method to instantiate a new piece.

          void  Destroy ( object obj );
          // return 0 to disallow beginning of a connection, current path never really exists...
            int  ConnectBegin ( object psv_to_instance, Int32 x, Int32 y

          , IPiece piece_from, object psv_from_instance );
          // return 0 to disallow connection, current path dissappears.
            int  ConnectEnd ( object psv_to_instance, Int32 x, Int32 y

, IPiece piece_from, object psv_from_instance );
          // can return 0 to disallow disconnect...
            int  Disconnect ( object psv_to_instance );

//, PIPEICE piece_to_disconnect, object psv_to_disconnect_instance );
           void  OnMove ( object psvInstance );
           int  OnClick ( object psvInstance, Int32 x, Int32 y );
            int  OnRightClick ( object psvInstance, Int32 x, Int32 y );
            int  OnDoubleClick ( object psvInstance, Int32 x, Int32 y );

          void  Update ( object psvInstance, UInt32 cycle );
          void Draw( Graphics surface, long x, long y, int s );

          // result with the instance ID in the database
          void  SaveBegin ( DsnConnection odbc, object psvInstance );
          uint Save( DsnConnection odbc, uint iParent, object psvInstance );
          object Load( DsnConnection odbc, uint iInstance );
```

Further Neural Abstractions

All methods described so far have been described in a general manner, but are somewhat implied to work with floating point numbers from -1.0 to 1.0 or some sort of fixed float integer format that goes from -128 to 127 for instance. All of the calculations within a node should work in the same precision. Originally implemented with a abstract variant type that could be assigned a value of some type and read as any other type; much like casting integers to floating point types etc. Text types were included, but left used. Additional types might be structures or complex types with alternative processing rules.

The processes of a neuron could also be complex that are responsible for creating other nodes and neurons.   How to reference where this is created? How to define what is created at that point? Simple sum of inputs greater than a threshold to trigger an enable to peform the creation process.

Initially, numeric processes can be tracked very easily with this methodology and scripted rather quickly given proficiency with the UI and given that the UI enables quick operation. It could be scripted; as earlier mentioned, so that code creates the structure of the brain for other static processes instead of using a WSYWIG sort of design.

Outputs such as indicator lights or text messages may be valuable for communicating with other nodes; but then the type of the value should be encode in such a way that when connecting a synapse to a neuron, it gives what type of value it will be feeding, and then when attaching the synapse to a target neuron that neuron can check the value type for compatibility. Direct Show technology for assembling AV streams works similarly.

Nodes, when used on a higher level are a way of consolidating complex processes to simple input/output modules. Much like integrated circuits do for lumped passive components. Nodes may then have a representation that presents input and output points. So a simple thing like an oscillator at a specific frequency could be a node with a single output instead of the complex network of neurons that form it.

Computationally, neurons may be defined that perform complex functions themselves, but they lack the distinction of nodes that have multiple input points. A Node interface for a weapon might have a tracking input to indicate current direction, a output to turn the weapon left/right and up/down, and to trigger a firing sequence and possibly a reload trigger. This module could then be applied to a board and wired to as a module instead of a lot of separate inputs. Neurons could be made like the oscillator node that give a time and have just an output. An input might be a simple digital switch from the real world.

It may also be useful to be able to connect a synapse to a synapse so there can be dynamic feedback to set parameters of the gate to the next neuron. This would provide a dynamic value for the gain in the default gate to the next neuron.

Simulation Extension for biologic processes

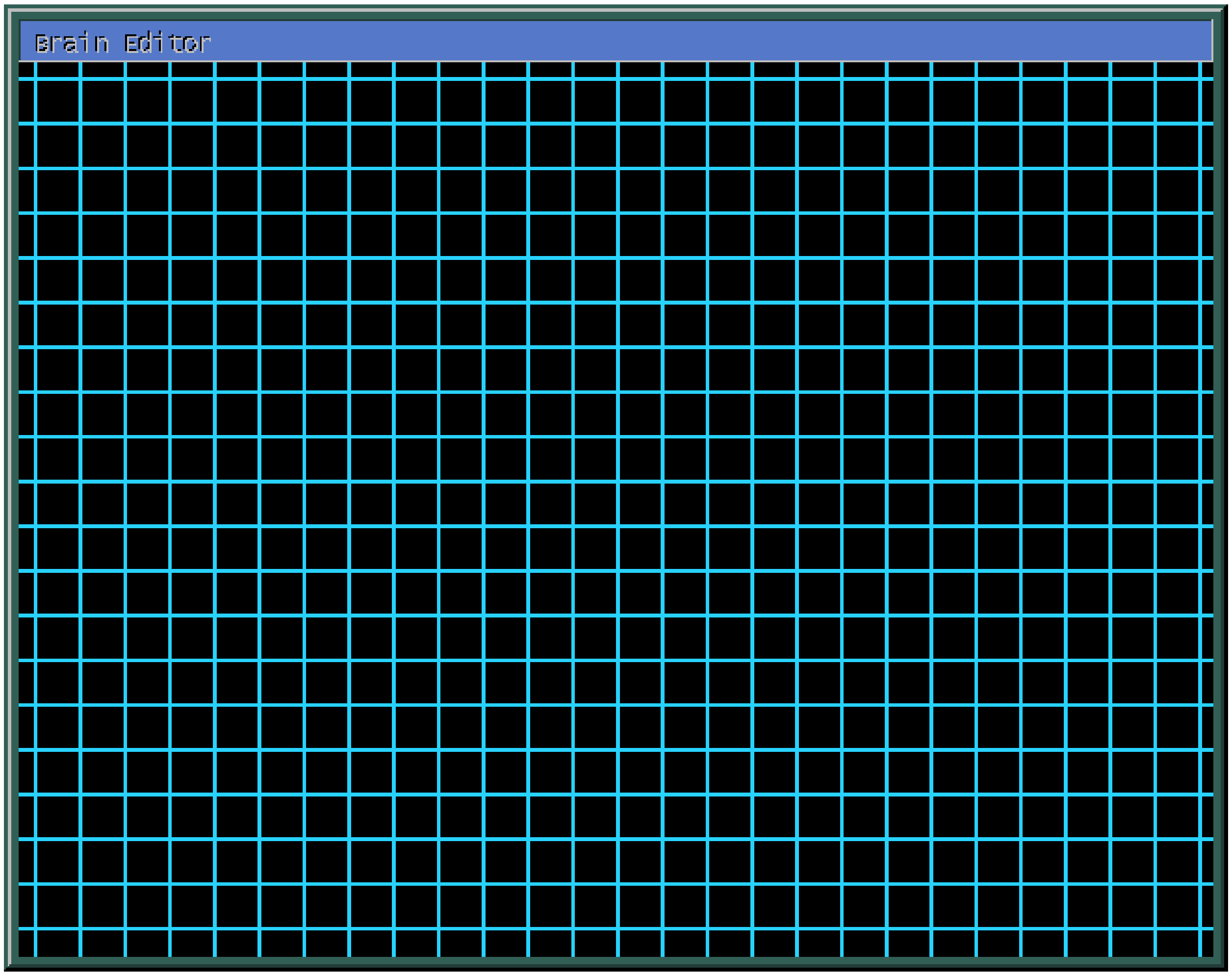
Instead of having a single type of value, maybe there are value types A,B, C, etc.   These could flow simultaneously but independently through the same channels. and cause complex interactions in neurons. This would be similar to different chemical substances which trigger separate responses…. or provide bidirectional behavior on a single network.

This extension would allow for a write-enable function; during (processing) if value L is present, then the threshold is adjusted toward the current input levels.   If a value K is present then the threshold is adjusted away from the current input levels… then if as the value of L and K diminish the threshold function remains the same; providing for back propagation training of networks.

User Interface Implementation

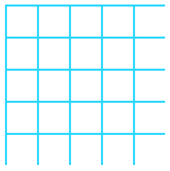
This is a detail that is really non brain specific, and any other sort of object (as a generic programming term) may be related with other objects. Their presentation may be piece class specific or may not require code in the object.

A grid with an origin and 2 (or 3, but for present display technology 2 should be sufficient; but only a few considerations need be made to provide for the extension).



Looks simple, but it’s actually more flexible….

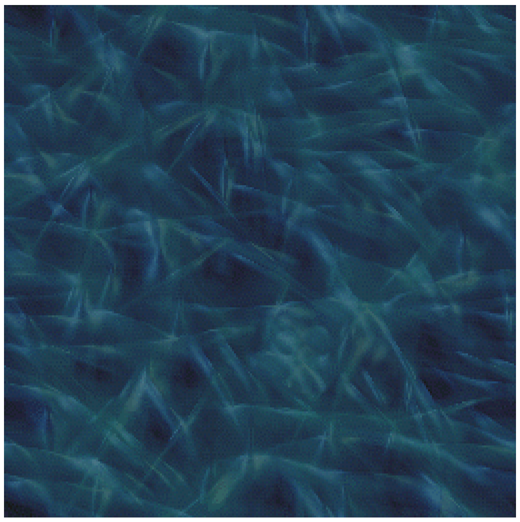
block background (10 by 10) %resources%/images/brainboard/background.gif



This image is actually transparent, but there’s nothing behind it so we don’t know.

this image is actually a single image that covers 10x10 patch (repeated in tiles) …

Another image might look like....



which might actually cover 25x25 or more relative to how you want it scaled verses the pieces you’re working with.

Neurons are 3x3 blocks. When a Piece receives a click event, it should receive a position for that event, to trigger different events. By simplifying to 3x3, any other scaling outside of the control won’t affect the basic logic.

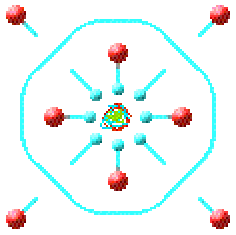
A more complex piece like a Sonar fixed direction input with basically a linear input resembling nearest object, from 0 to 1 at max; or rather 1 should mean the object is on the sensor, and taper off to 0 for things further away. Anyhow, this might be a pretty block with a speaker and only accept creating connections.

So to go back to the background piece, it just ignores position input, and on right click provides a menu to select options. This could be built to have a toolbox in parallel with the board so a selected type of tool could just be applied on left click anywhere on the grid; but that background piece is also given a absolute position… other pieces will receive a position relative to themselves. Sometimes, pieces will want to know an absolute position for disrelated functions, but this is often not the case.

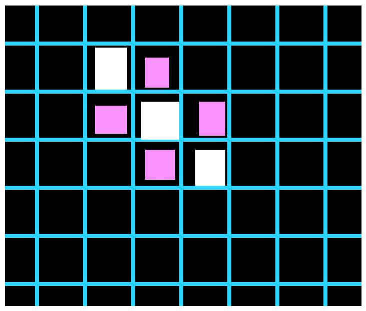
The board can be considered and implemented with layers. Each cell on the background is a stack of 0 or more layers. If no other layer is in the cell, the background is invoked with a absolute position because it’s relative to the absolute origin. Otherwise a example neuron would occupy 3x3 of these grid spaces with a layer.

When the neuron is left clicked on its edge, it results in the creation of a synapse that is attached to itself. The neuron may or may not know the position that the synapse is being attached. The position should be an attribute of the layer that represents the neuron; or it could know the relative position of the connection to the neuron, assuming that the center of that 3x3 cell is 0; It may even know the absolute position, but really, the brain objects themselves should not know or care; They should be position independant as their operation does not require it. So the description of the board and how it works is only attributes applied around the neuron, and with a reference to the neuron.

So to resume, the cursor should then on left click and drag begin a path, which claims a progressive chain of cells in a layer. This chain can be represented with a 5x5 image for all connection points and routings….



the center has no use. When drawing the diagonal, it must claim the two offset squares also… the following example, the diagonal connection is in white and its required diagonal fills are in purple. The diagonal fills may be taken from several of the cells represented in the 5x5. There are several cells that are in fact duplicate and therefore unused.

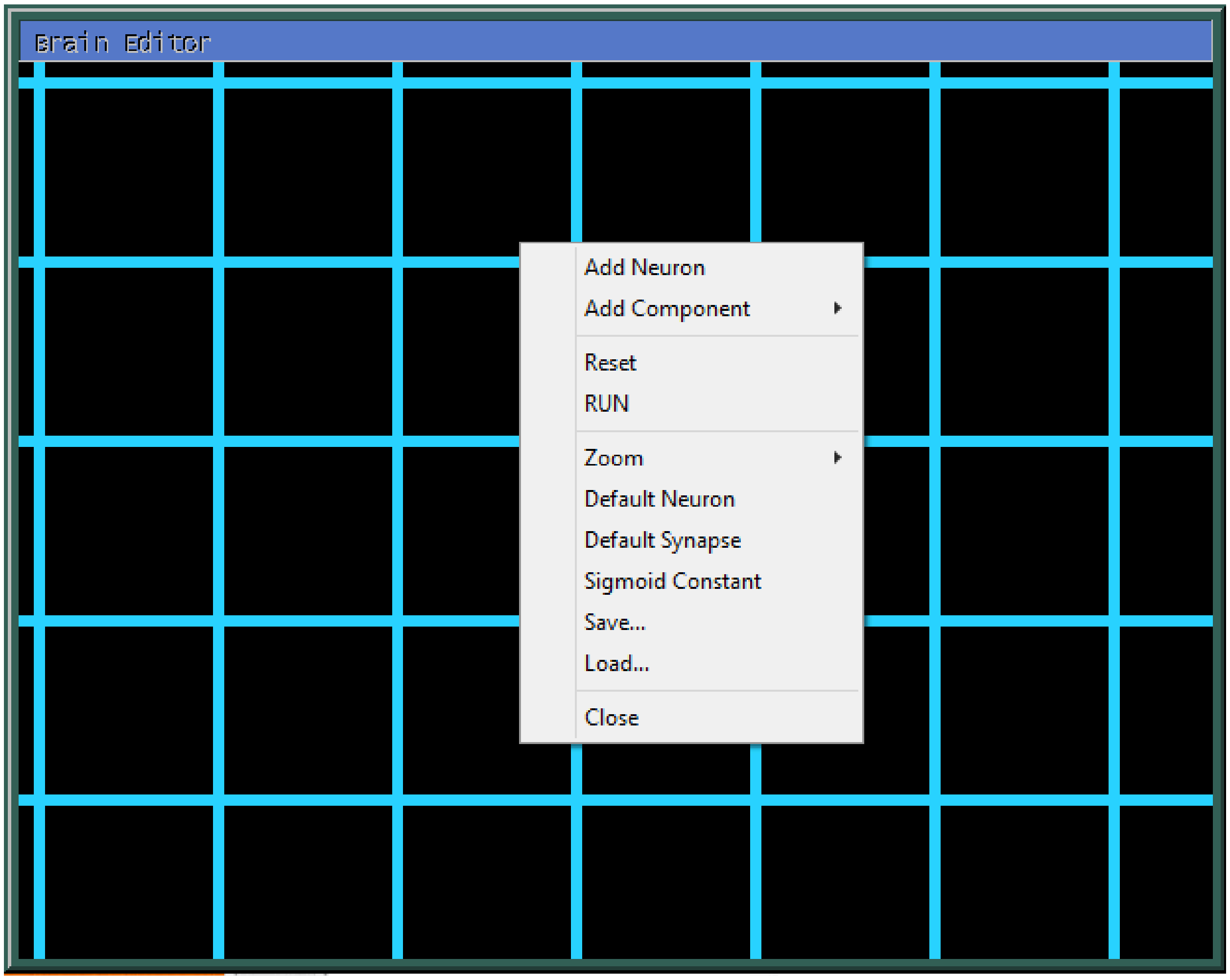


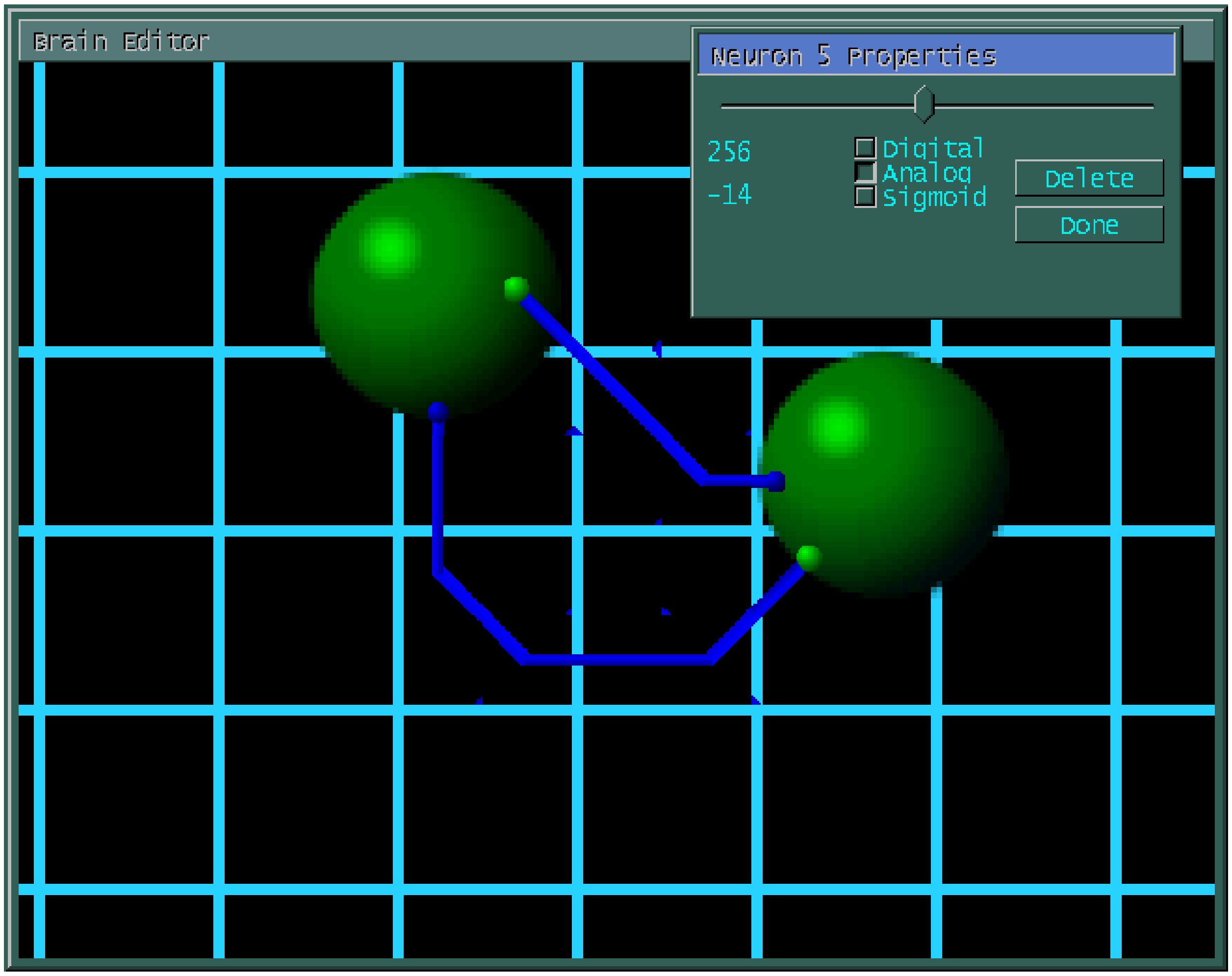
Example Implementation

Right click on the background brings up a menu of options… I’m not going to say any more than they already say….

broken image embedding; have to fix that right click on a piece (neuron) center will bring up its properties, left click will select it, and then dragging will move it, and all connections to it update appropriately… in the closest straight path.

Left click on an edge begins a connection to that area; each piece is really at least 3x3 regions. The outside 8 are beginnings of connections, the central one will trigger properties for the neuron (piece, generically from the UI standpoint); or allows for moving the piece. If there is an attachment already at the outer connection, then the selection is on that 1x1 mode of the chain that is a via. at the end,on the mouse release the piece should receive an on-cnonect at a position; allowing for rejection if already connected to that spot, or rejection because it's an input and doesn't receive connections…

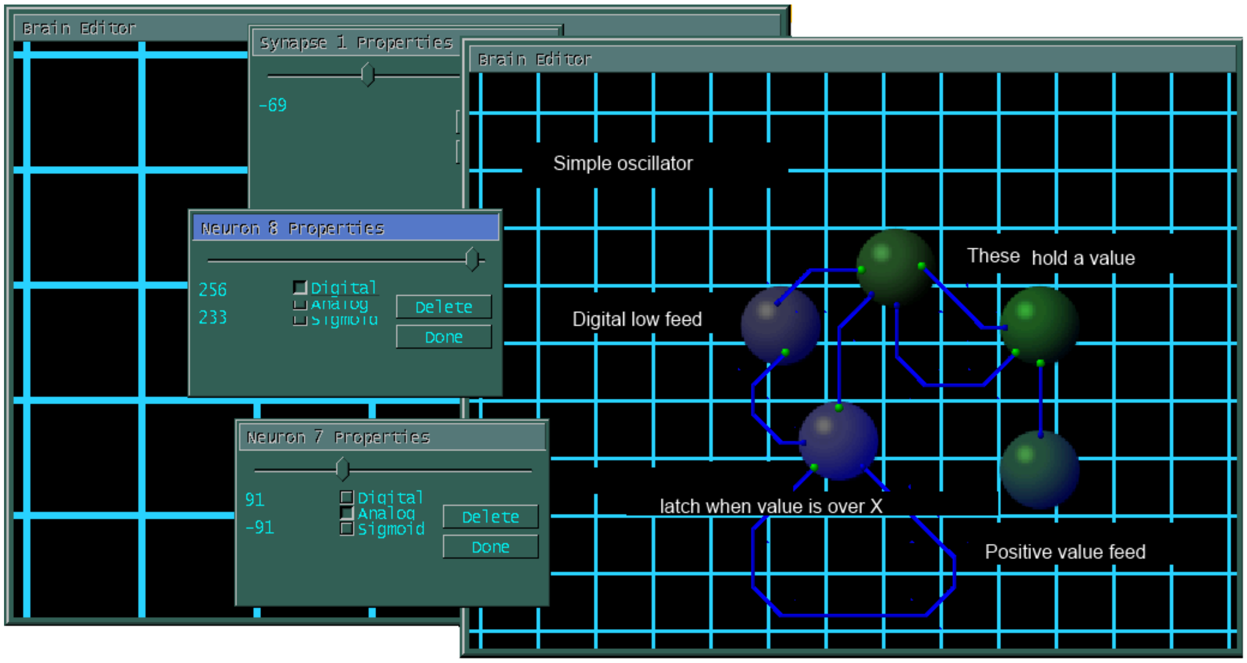




Right click on a neuron central, brings up a property dialog to adjust things like its mode and threshold. (more advanced interfaces include a dynamic list of registered algorithms or processes instead of the radio list)

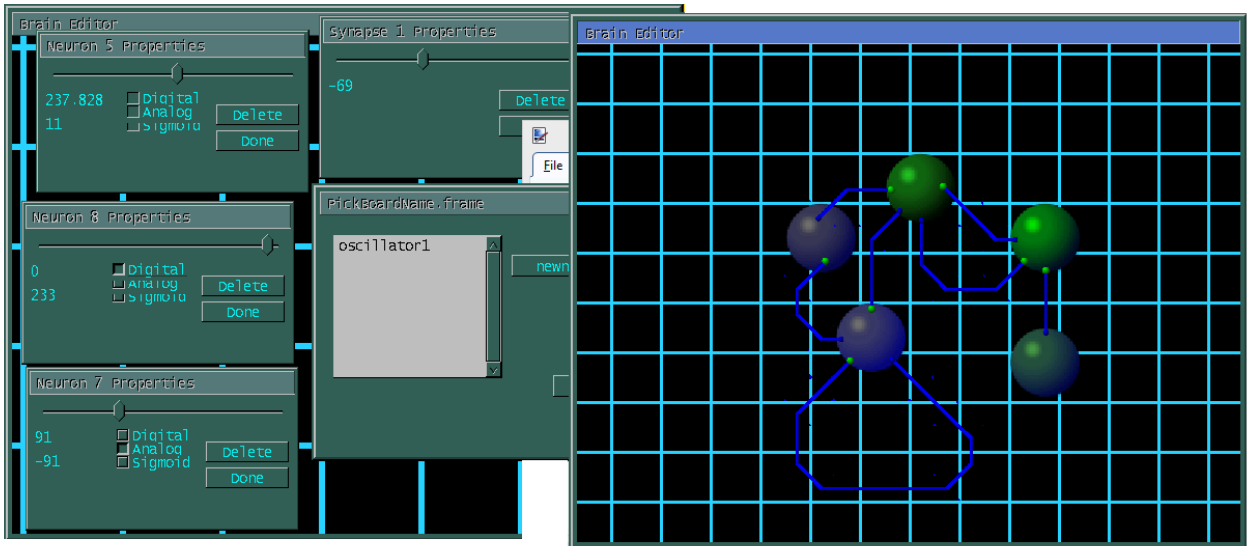
This is a simple 2 neuron system, it can't do much itself, but it will build to a level if one neuron is offset on its threshold below 0.

The intensity of green indicates the level of output it is generating. The synapse path may also have an intensity or shading to it to indicate the level it carries.



With a few more neurons, some redundant but present to illustrate other points, a up-down repetative cycle counter that builds and decays with time. From the first two self feedback, a latching neuron is added that is a digital output, and a high threshold. The blue shading around the piece indicates the threshold level. So when the first reach a high enough value, this neuron will turn on, and keep itself on, until its input goes under a certain level which it turns off… the positive value feed can be some external source that is a positive influence to the charging of the first pair. (And yes, I do realize that I'm mixing my venacular(?) all over)

Synapse gain properties have to be modified also….



Alternative Piece Implementation

A piece doesn’t have to be a blobular thing, this sort of mechanism works well for database schemas. So if each blob drew itself as a block of text that were the columns in a table, different cell points could be associated to the columns for building foreign key relationships.

A piece may represent a port on a router, and an allowed vlan it is connected to. Other high level network managment functions….

Simple Life

So a simple life form, simplified further for purposes of simulation, an insect. This example is implemented in BugBrain http://www.biologic.com.au/bugbrain/ . an insect can move forward, turn left, turn right, sense when its nose bumps something, detect differences in light, sense if there is a path left or right, and sense the thickness of what it is walking on. The bug will automatically eat edibles it finds.

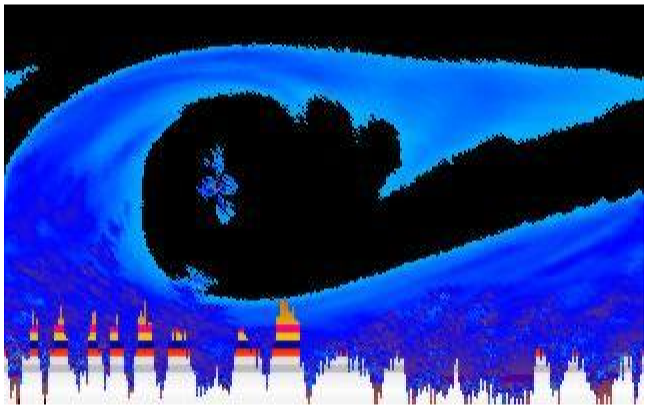
The goal is to eat all the food and not be eaten by the crow. The crow has a warning sign, and will only attack if you are moving. When the crow flies over, it casts a shadow, so the bug should stop and be still; this ends up being a pulldown override on the output to moving forward which is normally always on. The branch the bug is on has aphyds along the path, and branches to left and right, so the bug has to detect a turn, turn that way; if it turns early it will fall off the branch; go to the end, turn around when there’s no path forward, go back , continue down the branch, stopping when there is a crow, and the branch tapers off at the end so you have an analog adjust on just how far you can go before falling off.

Hive Processes

Taking several simple life objects with even simple inputs, a hive process can be developed, so slightly different behaviors are implemented in several different classes of bug. An additional input of knowledge of another of similar and/or enemy kind is near. This can build flocking and pack behaviors. Additional features may be added to the presence in an environment such as pheromone marking. If there is a food resource that depletes, until it does, maybe there isn’t a real idea of how much of a thing is left, so the workers continue to go out and gather, but failing to find enough to eat much less to bring back in excess, they die, and their corpse emits death pheromone to discourage repeated attempts to go in that area.

Agent communication becomes a requirement. ESP is the easiest to implement, to all brains share a common pool of signals they can use to communicate to all; or with established protocols to individuals.

Pheromone trails may be static; imfmediate marking on a 2d map representation. They may then spread and diminish to dilute from that point; a time based increasing circurcumferance that diminishes as the area increases. They could also be subject to wind…
http://www.afn.org/~cthugha/archives.html#dos



Pathfinding with autonomous agents

Path finding algorithms with many agents probing possible path. Representation of the intelligence gained from finding best paths is incoherent; but resembles there pheromone simulation of hive processes.

So pheromone marking can leave information like go to here and avoid here, and form an area or boundary. But that doesn't really imply that it's a good or bad way to go. It may be an exceedingly good way to go, worth the possible risk; maybe it's a spread indicator from choosing to go left or right after that point.

Representation of path finding for vehicles is based on a database that is a street map. http://www.openstreetmap.org The agents should be able to add information to such a database if given a foreign environment.

Pathfinding does require having a target to find find a path for...

Design Challenges

How to represent relation to other variable and fixed objects. Especially with inputs such as angles end up with wrap conditions.

Map systems with a 2D representation, probably in a flattened array…

Maybe the system is based more on regions and portals…. (flatland)

There are no limits; every neuron may produce signal indefinitely. Eventually limits on resources should be given; stock energy level just to maintain function for an organic. Organic inputs should also include synthesis to alternate outputs; which requires transmission of only certain types of values (chemical compounds).

Available capacitance is a factor for electronics… this will limit outputs of neurons naturally as it cannot possibly exceed the sum of its input; or it must also be sourced to a power source so it can provide a powered or amplified output. An oscillator for instance is a power source. So budgeting may be a factor in designing circuits.

OpenSimulator Interface

(Design in progress)

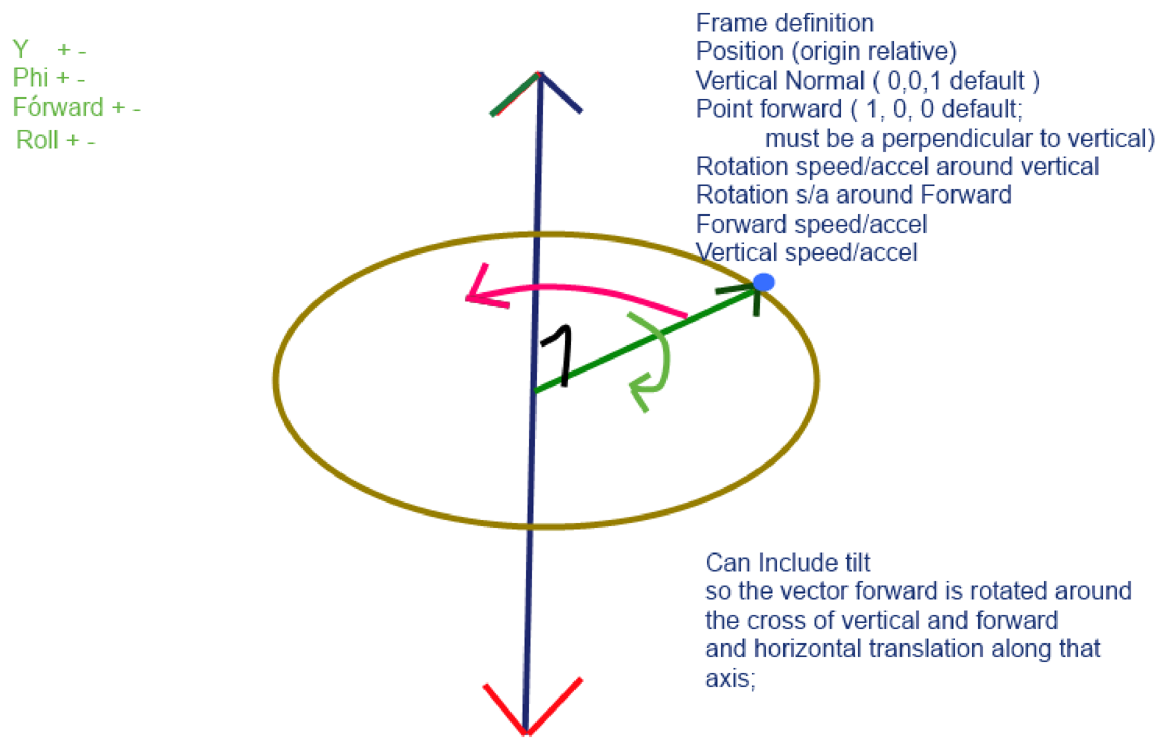
A single bot server will handle objects which are themselves handled with the physics of the platform server. A bot server is implemented as a login to a grid; only one login to a grid at any point is allowed at any time. Separate bots will require separate logins. But bot-accounts are really servers for other normally inanimate objects in the grid. The objects may have scripts on them to offload actions, and the objects may of course tell their creator (the bot-account server) statuses directly and without regard to what created them.

In this world, bot inputs; and abstractions for neural computations and modelling. For instance every object may immediately know where every other object is from the server; but the server doesn't compute angles and relative positions, all positions are relative to the world origin.

Terrain information may be known; as this will influence ability of server physics.

IGV - Inverse Gravity Vehicle

Persuming an inertialess drive, acceleration may be applied along one axis vertically, and then in any angle perpendicular to that axis. Tilt/Roll may be applied for any angle perpendicular to the axis, or around the vertical axis.


Y + -
Phi + -
Fórward + -
Roll + -
1
Frame definition
Position (origin relative)
Vertical Normal (0,0,1 default)
Point forward (1, 0, 0 default;
must be a perpendicular to vertical)
Rotation speed/accel around vertical
Rotation s/a around Forward
Forward speed/accel
Vertical speed/accel
Can Include tilt
so the vector forward is rotated around
the cross of vertical and forward
and horizontal translation along that
axis;

Interactions with objects are somewhat limited from the user of this service.

So we assume we give the agent server a simple avatar like a box. This box can somehow communicate to give you an object for a certain price (pay to play). With this new object, you can start an instance of simulation….

Simulation arena 1: RTS generic resource gather, build, destroy things built
simulation arena 2: player controlled token vs creature generators (gauntlet/tower defend/attack)
Simulation arena 3 : static turn based stategy like chess; the token given is a board, with peices in a stock position (life size for easy interaction)
Simulation arena bug/swarm : this has to summon a field; it should use the current simulator as the backdrop; again a simulation without a goal is a failed simulation; although the real idea is to make a thing that makes up it's own decision about what goal it has?

But really relative motion isn't required; so it actually has to be faked. That is an object can immediately move to any other location in a simulation (if allowed; I suppose teleport rights can be removed on the sim) but even then, I’m not sure there are speed limiters in the sim; so motion still has to be manually controlled.

To implement fake things like radar ends up just being a design challenge for the neurons; although it is a reasonable way to view the world and build relative sense of it; A radar can be started as a single point, but in a 2D sense, additional points to the left and right may be added; the quality of signal result may also be added which can represent color; and eventually be muxed to a matrix of vision input. Muxing can be done with frequencies, so results for water are different from iron or brass; or information that’s more abstract like friend/foe/environment can be communicated (maybe as 3 threshold inputs? so multi-point radar can composite into something like 0.1-friend 0.5-enemy 0.2-environment 0.1-food; so thing like good hit on food may override the enemy, or additional friend in scan can be used to move to flock and attack.

Sonar is only distance; though I would think it could know if the thing was soft or hard depending on how clear the echo was? Maybe smooth or rough?

RTS Implementation Considerations

For implementation as a companion intelligence in RTS simulations…
What the ‘view’ of the world might look like.

Hive Systems

Hive systems have a collaborative effort of autonomous agents. There is in theory a Queen entity. This is the shared consciousness… it is shared memory location essentially. A common repository of knowledge available. Sharing this knowledge may be abstracted at some point, but consider that everyone works within the same world framework, and instantly communicates information for all. This is more to note and not forget that there is more than ‘a’ entity here.

Map Handling

1) In the beginning there is the void, and the intelligence knows nothing. It knows where it starts; this becomes (0,0,0,...).

There are multiple types of information. There is information which is temporary and should decay in time. This can take the form of point-radius. A single event should in fact be a collection of information; such that if a unit is attacked, the location of the unit, and a virtual point where it thinks it might have been attacked from should also be marked in a different color. Some units should move to where the victim unit was, some will want to further push to where the attack came from. The type of attack such as weak/strong against this unit type should also be noted in the connection between the points.

There is information that is a boundary, such that nothing further may pass; this is best represented with a plane of point-normal form… further a collection of all planes which this must be within. every point within this will be within a total of 2 planes in every direction from the current. If there are multiple planes that intersect a line, there must be 1 plane the point is above and 1 point the point is below.. which is a net 0… +1 for planes above, -1 for planes it is below, if the total is 2 the point is within the boundary. This is a definition of a closed surface.

There is information that can be ‘etched’ in stone, and by this information about quick pathways or highways may be formed. This is a set of point-radius points.

There is an algorithm available for managing a closed, convex, set of points such that all points that are in near relation can be noted, and from any other point within the structure a limited number of searches will be performed, in somewhat of a ‘london tourist’ sort of resolution. (Spaceweb) This is stable, within closed parameters… if a ‘dip’ in the outside surface is formed, the nearness begins to fail; a generous connection behavior can be enabled; that is more points than those immediately near are compared for further nearness; Alternatively, if definitely convex, a more pessimistic algorithm can be used; but this is a O(K’th-root(n)) where K is the number of dimensions (2 dimensions is sqrt(n), 3 dimensions is cube-root(n), 4 is 4th root(n), etc) search for things near some other point from any other point. If the current location is also maintained as a 'cursor', the search becomes O(1)... and well location of nearness depends on the density and probably has a theoretical value of e^Kdim… I dunno)

Marker types

'known universe' a container called 'unknown boundary'. Assume we get an initial view of the world also which can be analyzed for merit. This will have to be addressed later; assume that within 1 radius there is nothing of note, other than we can extend the boundary of known universe.

- a general sphere of low resolution? 12 sided container of planes?
- Minimum must be a triangle with 3 planes, a square is easier to compute.
- A cube may be the minimum bounded space
- although not required, it may be assumed that the absolute limits of the universe that is knowable will itself be a simple shape (square map).

1. behavior 1)
    a. advance toward nearest unknown boundary.
2. behavior 2)
    a. advance toward buddy unit.
3. behavior 3)
    a. if an enemy unit is seen, shoot at it.
4. behavior 4)
    a. if damage is received, mark my location with a radius of possible max range of attack minus my range something (a larger area than just 'here' so a unit will have opportunity to modify behavior) It should also mark the location and range of the attacker… (plus a falloff? or an additional mark…)
5. behavior 5)
    a. avoid places where damage happened.
6. behavior 6)
    a. move towards places that caused damage.
        i. locate and identify type of unit; but on damage something like move away.
        ii. locate and attack.
7. avoid places of death.
8. gather team
    a. As new units are produced they will be assigned possibly to existing groups to grow existing forces as resources allow (instead of allowing deprication).
    b. This behavior should note the positions of 'friends' and move in such a way to bring these closer to a merge… (advance towards center for instance).
9. Mark impassible object.
    a. as obstacles that cannot be mounted a boundary should be marked. This boundary is 'obstruction boundary'.

        i. obstruction boundaries are also tied to marks for point-radius sorting.
        ii. obstruction boundary may form a closed island within the unknown boundary.
        iii. Obstruction boundary creation must consider what exists of the unknown boundary. Initially impassibility will be the plane and point in front of the impass; closing boundaries for the left and right edges may be considered if not unknown, and known to be 'passable'
    b. Impassible for some may not be impassible for all; a land cannot go across sea, but air can go across land obstacles and sea; so a class of motive mechanism should be applied as impassible… or seperate boundaries can be maintained.
10. Mark Object of interest
    a. If a 'metal' point is observed, mark its location; tie it in relation to the pathing cloud that happens.
11. Mark My Path
    a. As a unit travels, it should leave a trail of many smaller marks that indicate where it has been.
    b. (is a combat path)
    c. (is a resource path)
12. Mark where I attacked
13. mark where I killed
14. mark where I damaged
15. Follow path that was marked by (peer). maybe only 1… maybe a food drone leaves a separate path from a combat drone.
16. (permutation of elements above) while following path, find death at the end, retrace path, mark beginning with death to avoid further casualties.

Marks

1. Damage happened here, and depending on 'panic' will be a certain size. (A more damaging attack may be a factor, or a personality trait may over/under-react to the attack). This should be a time-decay mark; either effective radius decay, or general opacity)
2. Damage happened from here (may be a spread of points to cover an arc of possibility depending on cover and actual observation of the enemy). This should be a time-decay mark; either effective radius decay, or general opacity)
3. Death of a unit. this may be a rapid decay type… depending on the unit the type of the mark will vary. This should be a time decay; but may be a time-growth (expanding cloud of pheromone) with a general opacity decay.
4. Path to death; dead end (unless is a combat and damage is of a type unit is good against)

Mark Relations

1. mark relation 1) type of damage. (gradient scale(s)? weapon classifications? Club? Laser? Pistol? Artillery? Radiation?) This should be tied to the mark1 and mark2 and should not itself be a mark.
2. Death mark (3) and mark (1,2).
3. consolidation of path markers… reinforce ‘goodness’ of the path; such that if the reinforcing is greater than the diffusion, the areas can slide… so need a certain stack maybe… maybe the expand and fall off of the top; giving the stack of marks an elevation and adding a minor part to its own layer… or adding 1/N parts per peer marker radius….
4.
5.