Ecosystem Simulation Two
Next version ecosystem simulator

I have designed a four trophic level ecosystem using C. S4L5 Ecology helped me learn C++, and explore the use of genetic algorithms. I thought object oriented programming better fit the ecosystem paradigm. However, it required numerous links to keep the objects organized. I have found this ordering is unnecessary, I now access each organism as a single node from a large buffer. Indexing depends on the problem space. I can access them in order because each buffer is created as a large number of plant, herbivore, or carnivore nodes. The ecosystem indexes the lists onto a 2D grid through their x,y location. I need to keep track of the population counters to maintain the lists in contiguous order. Otherwise you could read outdated data, or step off into oblivion.

I Discuss new data structures, algorithms, or language features

This simulation is based on a 2D nutrient layer filled with energy. There are three lists which populate the ecosystem with plants, herbivores, and carnivores. Plants draw energy from the nutrient layer. Herbivores gain energy by eating plants. Carnivores gain energy by eating herbivores. The nutrient layer gains energy through herbivores and carnivores eating and reproducing. Both processes are messy, some energy is lost to the nutrient layer. It is also replenished each cycle by the amount lost by all the organisms due to their metabolism.

Each trophic layer is represented by its own node type: enode, pnode, hnode, and cnode. The ecosystem is indexed as a 2D grid of enodes. Each cell in the grid holds nutrient energy, and pointers to current, or future, resident plants, herbivores, or carnivores. Each organism contains its energy level, age, sensor network, various chromosomes filled with genetic information, and its location. The latter is how it finds its cell location in the ecosystem.

Each node type lives in its own buffer. Plants live in PL, herbivores in HL, and carnivores in CL. The ecosystem lives in E. I access each list linearly, one organism at a time. I access the ecosystem using a 2D addressing scheme. (E +Y*SZ +X) selects a single node inside the ecosystem located at X, Y, where SZ is the size of the grid. The grid is 5000 units square at present.

Eating and death delete organisms from their respective lists. Reproduction adds new organisms to their respective lists. A single buffer would require many pointers to clear the now open cells. So I added a trait to each organism->alive. It is a Boolean flag telling me whether this organism is still alive. I use this trait in the function cull(). It scans through each organism list, PL, HL, and CL, copying those which are alive to a second buffer. Yes, there is a second buffer for each of the three organism lists. The function die() sets some organisms ->alive == false. Culling finds all those organisms which are ->alive == true and copies them into their respective 'other' buffer. Once culling is complete it is time to run addNew(). This function copies the new organisms into their respective 'other' buffers after those already there due to cull(). Once the 'other' buffers are full, swap the buffers to use the new populations.

Thus we have a double buffered system for garbage collection. I never deallocate the memory, I just swap control to the newly filled buffers and overwrite the previous ones with new data.

II Describe functions and modules

moveHerbivore() Use DR chromosome to find next move, then take it.
moveCarnivore() Use DR chromosome to determine next move, then do so.
replenish(metabolize()) Lose energy through metabolism, feed that amount back to nutrient layer.
eat() Gain energy for metabolic processes.
reproduce() Exchange genetic material with mate and fill new organism buffers.
die() Check energy levels, if zero ->alive = false.
cull() Copy living organisms to their 'other' lists.
addNew() Add new organisms to end of 'other' lists.
bufferSwap() Swap current lists for 'other' lists.

There are eleven main functions in the simulation. There are many other functions, but they are ancillary to the task. Data logging, data display, sampling techniques, genetic marking, and time stamps are all useful, but not necessary to the simulation.

There are a number of frequently used tools: fRan(), vector(), sum(), scale(), bound(), initDir(), move(), crossover(), and mutation(). fRan() is quite useful - it gives me a random floating point number between 0.0 and 1.0 which allows me to easily scale to whatever range I like. Vector functions: vector(), sum(), and scale() are used throughout the simulation. They make thinking about the problem, and writing its solution, more simple. bound() is used to wrap the 2D ecosystem grid into a toroidal space. This lets the organisms keep walking in any direction only to return to where they were. It eliminates edges in the system. initDir() and move() use the DR chromosome to determine the organisms' next move.

Two genetic operators help the system breed better offspring. While crossover() refines the most fit organisms, mutation() allows the system to explore alternate behaviors. I use a 5% mutation rate for all offspring chromosomes. Without mutation the system runs well for a few thousand cycles, then develops larger and larger population swings. Finally, either the herbivore or carnivore population would drop to zero, ending the simulation. Adding a little mutation allows the system to run much longer, moderating large population swings. It seems the most fit organisms are not always the best fit. Less fit organisms help tame the simulation. However, adding too much mutation rapidly destroys the system.

But wait, there's more:

```
void herbScan( void );                      // Local sensor scan for plants and carnivores
void carnScan( void );                      // Local sensor scan for herbivores

   // Data Logging functions  /////////////////
void storeData( void );                     // Backup ecosystem to file
void loadData( void );                      // Restore ecosystem from file
void storeSamples( void );                  // Collect 1000 P,H,C samples
void loadSamples( void );                   // Read the samples into their own buffers
void loadSampTwo( void );                   // Read second sample into secondary buffers
void useSamples( void );                    // Build an ecosystem from those samples
void showNew( void );                       // Display data for last ten plants in PL
void showSamples( void );                   // Display chromosomes of 1000 samples
void stamp( void );                         // Add a time stamp output
void fStamp( FILE *fp );                    // Date stamp the filed data too
```

III Debug and Profile the code

- Debugging with gdb
- Profiling with gprof

g++ -g -o en ecoNode.c for debugging
gdb en

List debug commands

run, step, break point, info??

Profile script: g++ -o en -pg ecoNode.c
./en creates gmon.out
Once you have gmon.out type gprof en to read the results.
List profile commands. There are a few flags to change what you see and its verbosity

IV Describe code line by line

The system is built from three lists and one framework. Each list is contained in a buffer of organism specific nodes. Such as :

```
struct cnode      // Carnivore node
   {
   struct vt X;                          // 2D location vector
   float energy;                         // Chi
   bool alive;                           // Energy above zero.
   int age;                              // Used in reproduce()
   chromosome EN, CN;           // Energy and carnivore chromosomes
   chromosome SN, DR;           // Sensor and direction chromosomes
   };

struct enode      // Ecology node
   {
   float nutrient;                        // The nutrient energy of this cell
   struct pnode *P;                       // Point to plant,
   struct hnode *H;                       // herbivore, and/or
   struct cnode *C;                       // carnivore living here.
   };

   // Ecosystem and organism buffers
struct enode *E = (struct enode *) malloc( SZ * SZ * sizeof( struct enode ));
struct pnode *PL1 = (struct pnode *) malloc( SZ * SZ * sizeof( struct pnode ));
```

This scheme lets me work with the ecosystem grid, or any of the lists separately. Each organism node contains its own location vector. That is how I hook the 1D organism lists into the 2D ecosystem framework. I scan the plant list during senescence, flagging the dead. The function cull() scans each organism list, passing survivors to their alternate buffers. The location of each dead organism is used as an index into the ecosystem(x, y). Then that pointer is set to NULL, an indication to the system of an open spot. It is easier to show this than it is to describe it :

```
struct vt T;                              // 2D working vector
for(int i=0; i<Ppop; i++)
   if( (PL+i)->alive )
      {
      *(OPL+k) = *(PL+i);                 // Copy survivor node to alternate buffer
      k++;                                // Count the survivors.
      }
   else
      {
      T = (PL+i)->X;                      // Index Ecosystem plant location
      (E +T.y*SZ +T.x)->P = NULL;         // NULL plant pointer
      }
```

When an organism dies, through lack of energy or being too old, its ->alive flag is set to false. The function cull() creates a list of pointers in the alternate buffers OPL, OHL, or OCL from 0 to the number of survivors. When an organism reproduces it places its offspring into the NP, NH, or NC lists. After reproduction and culling the offspring are read from their new lists and added to the alternate buffer lists. When all of that is done you have a set of rebuilt lists in their alternate buffers. Each organism remains attached to its location in the ecosystem(x,y) grid. Lastly, we swap buffer pointers to use the alternate buffers as primary ones, while swapping those which were primary to their alternate roles.

Double buffering makes indexing kinder and quicker. There is no need to erase anything because you are overwriting memory with new data, while setting new end points. I allocated memory as a 2D grid so each cell in the ecosystem could hold a plant, a herbivore, and a carnivore simultaneously, albeit briefly :) If SZ is 4000, your grid has 16,000,000 cells which means there can be 16 million plants, 16 million herbivores, and 16 million carnivores at the same time. However, I have never seen it happen :) This is where I get segmentation faults. During addNew() the current population of plants plus the new offspring exceed the space in the PL buffer. Or newPpop+i > SZ*SZ In other words, it is not really an error in the programming sense. I simply need to make the plants less prolific, or the herbivores more efficient at eating, or both.

The 2D ecosystem(x,y) framework hosts the nutrient layer. Plants absorb their energy from the nutrient layer. Eating and reproduction are both messy operations. Each function adds a little energy to the nutrient layer. I rewrote metabolism() to sum all energy lost from all the organisms. It returns the value which nicely fits into the replenish(energy) function. I added 10.3 energy units more to each cell in ecosystem(x,y). I think it may be too rich. I have not tamed the overgrowth of plants. I have put in a population limiter. If more than ~50% of the area is filled don't breed. I am tweaking that number too. There are far too many PFA numbers at this time. But I am gaining some understanding. When the numbers make better sense I will give them variable names. Approximately 30 settings control the system. Very non-linear :) NOTE: October 17, 2024 - I have reduced the added energy to 1.3 units per cycle. I no longer need to limit the plant population. I also changed the initial energy in each cell of the ecosystem. That affected the most change to the simulation. kjr

I am able to use gdb, and gprof for this code. Compile for debugging, profiling, or both. Debugging lead me to the error/non-error segmentation fault. Profiling tells me eating takes most of the time. The next most used function is move(), which is referenced for moving, eating, and mating. This function translates the DR, direction gene into random direction probabilities and then to a direction vector. Use scale() to reach more distant cells. The next most used function is reproduce(). Any improvements to eat() or move() will affect processing greatly. It is good to avoid the Win32 graphics library at this time. Losing access to gdb and gprof is not worth the glitter. I will stick to command line so I can debug my bookkeeping. However, I am pretty close to moving to a graphical interface.

My simulation loops over these functions: moveHerbivores() and moveCarnivores(), replenish() the nutrient layer, eat(), metabolize(), reproduce(), and die(). The latter two functions cause a problem. Reproduction produces more organisms; how do I add them to a list? Dying creates gaps in the contiguous list. My solution is to add new organisms to a new plant, herbivore, or carnivore list. I added the ->alive flag so I could maintain an unbroken list by setting it to false. Then the cull() function moves the live organisms to their alternate buffers. Once that step is done add the new organisms to the end of each separate list. You now have three buffers with continuous lists. Call bufferSwap() to change each buffer's pointer to their alternate buffers. You are now ready to repeat the loop for the next cycle.

Each organism has at least two chromosomes which I represent with unsigned integers. I shortened the nomenclature a little by using a typedef statement:

```
typedef unsigned int chromosome;
```

Then I built a few tools to manipulate each chromosome. Mutation picks a random point on the 32 bit chromosome. Then it performs a bit flip at that point. Mutation introduces bit manipulation operators, such as <<, >>, <<=, >>=, &, and |, for manipulating the 32 bit chromosome. Use these operators to convert the binary language of the chromosome into a trait expressed as an integer, a floating point number, or some factor between 0.0 and 1.0. >> and << are used to shift the mask anywhere we like on the 32 bit long chromosome. A typical mask is 3, 7, 15, 31, ... A binary 'decade' minus one. Imagine 3 in binary; it creates a 11 mask. Compare that mask to the chromosome at a certain point to obtain the raw genetic material. If your mask is 3 then your gene can hold {0, 1, 2, 3} in it. Most traits abhor a zero so I normally add one. Then my range will be {1, 2, 3, 4}. If my gene is 4 bits wide I use 15 as my mask. But imagine this case: take that 4 bits of information to give a range {1/8, 2/8, ... , 7/8, 8/8} Or you want to limit the top end of the range. {1/32 16/32} 0.03125 to 0.5 or whatever range you like. Use the mask size as your range, offset by one or more, then divide by the mask size or larger to achieve a limited factor. Examples in the code will clarify your understanding of these techniques.

The next genetic operator is crossover() which mimics sexual reproduction. This function combines the upper bits of the mother chromosome with the lower bits of the father chromosome. Crossover admixes the genes of the survivors, hopefully hybridizing stronger offspring. It provides enough genetic variation to partiallyexplore the problem space. {Note: I have added mutation into the crossover() operator to stabilize the system while searching for different answers. kjr Nov 11, 2024}

The only other genetic operator in this simulation is survival. That should be considered my fitness function. Instead of sampling a number of the most "fit" organisms I let the ecosystem determine which of them survive. When the energy level of any organism is too low it is culled from the system. This did not limit excess growth so I added a new gene: senescence. I wanted to eliminate any prospective immortal carnivores. {Note: An opportunity to create a new species - immortal but sterile. kjr November 11, 2024}

```
chromosome mutation( chromosome C )            // Mutate DNA in this chromosome
   {
   int i = rand() % 32;
   return C ^= (1 << i);                       // Flip a single bit at i
   }

chromosome crossover( chromosome CM, chromosome CP )   // Chromosome crossover
   {
   int i = rand() % 32;                        // Determine crossover point

   CM >>= i;                                   // Clear upper bits of mama chromosome
   CM <<= i;
   CP <<= 32 - i;                              // Clear lower bits of papa chromosome
   CP >>= 32 - i;

   return CM | CP;                             // Merge lower and upper parts
   }

void moveHerbivores( void )
   {
   struct vt A, B, C;                          // Working vectors
   int min, ring, ch, speed;

   for(int i=0; i<Hpop; i++)                   // Scan herbivore list
      {
      ch = 0;                                  // Limit choice count
      min = 1;                                 // { min <= speed <= 8 }
      speed = (int) (((HL+i)->HB & (7 << 28)) >> 28 ) + min;

      A = (HL+i)->X;                           // Current location
      do {                                     // Find an open spot within range
         ring = min + rand()%speed;            // {min <= ring < speed}
         B = scale( ring, move( (HL+i)->DR ) );   // Express direction genes
         C = bound( sum( A, B ) );             // Prospective new location
         if( ch > 8 )  break;  else  ch++;     // Limit search count to 8 choices
         }    while( (E +C.y*SZ +C.x)->H );    // While location is occupied

      (E +A.y*SZ +A.x)->H = NULL;              // Empty old location
      (HL+i)->X = C;                           // Remember where I live
      (E +C.y*SZ +C.x)->H = (HL+i);            // Fill new location
      }
   }

void moveCarnivores( void )
   {
   struct vt A, B, C;                        // Current position, move, prospective
   int min, ring, ch, speed;

   for(int i=0; i<Cpop; i++)                 // Scan carnivore list
      {
      ch = 0;                                // Choice counter
      min = 3;                               // { min <= speed <= 10 }
      speed = (int) (((CL+i)->CN & (7 << 28)) >> 28) + min;

      A = (CL+i)->X;                         // Current location
      do   {                                 // Find an open spot within range
         ring = min + rand()%speed;          // {min <= ring < speed}
         B = scale( ring, move( (CL+i)->DR ) );   // Motion vector from DR chromosome
         C = bound( sum( A, B ) );           // Search location

         if( ch > 8 ) break;  else  ch++;    // All your choice are us
         }  while( (E +C.y*SZ +C.x)->C );    // Carnivore already lives here

      (E +A.y*SZ +A.x)->C = NULL;            // Blank old spot
```

```
      (CL+i)->X = C;                          // Remember where I live
      (E +C.y*SZ +C.x)->C = (CL+i);           // Inhabit new spot
      }
   }
```

void metabolize(void);

Scan each organism list while incrementing the age and energy level. Each organism references its EN energy chromosome. Specifically the first gene EN::G1 which defines how much energy is used per time step. It is a nine bit gene so it can range from 0 to 511. I divide by 511 and cast to a float to obtain a number from 0.0 to 1.0. I am tweaking the result by adding a PFA number. I need to rethink this.

void replenish(void); // replenish nutrient layer in the E buffer

```
float metabolize( void )                      // Sum energy lost by all organisms
   {
   float energy = 0;                          // Balance global energy loss and gain

   for(int i=0; i<Ppop; i++)                  // EN::G5 energy use per cycle
      {
      (PL+i)->age += 1;                       //   EN::G5  6.1 - 13.1
      (PL+i)->energy -= (float) ((PL+i)->EN & 7) + 6.1;
      energy += (float) ((PL+i)->EN & 7) + 6.1;
      }
   for(int i=0; i<Hpop; i++)
      {
      (HL+i)->age += 1;                       //   EN::G5  2.2 - 9.2
      (HL+i)->energy -= (float) ((HL+i)->EN & 7) + 4.2;
      energy += (float) ((HL+i)->EN & 7) + 4.2;
      }
   for(int i=0; i<Cpop; i++)
      {
      (CL+i)->age += 1;                       //   EN::G5  5.3 - 36.3
      (CL+i)->energy -= (float) ((CL+i)->EN & 15) + 5.3;
      energy += (float) ((CL+i)->EN & 15) + 5.3;
      }
   return energy;                             // Pass energy use to replenish()
   }

void replenish( float energy )                // Balance energy lost during metabolism
   {
   for(int i=0; i<SZ*SZ; i++)
      (E+i)->nutrient += energy / (SZ*SZ) + 10.3;   // Spread energy evenly across ecosystem
   }

int senescence = (int) ((EN & (1023 << 22) >> 22) + 1;   EN::G1  1 - 1024 days
                ((EN & (511 << 22) >> 22) + 1;   for herbivores  1 to 512 days
                ((EN & (63 << 22) >> 22) + 1;    for plants       1 to 64 days

void die( void )
   {
   int senescence;
   for(int i=0; i<Ppop; i++)
      {
      senescence = (int) (((PL+i)->EN & (63 << 22)) >> 22) + 1;   EN::G1  1 - 64
      if( (PL+i)->energy < 0 || (PL+i)->age > senescence )
      (PL+i)->alive = false;           // Why didn't I take the blue pill?
      }
   for(int i=0; i<Hpop; i++)
```

```
      {
      senescence = (int) (((HL+i)->EN & (511 << 22)) >> 22) + 1;   1 - 512
      if( (HL+i)->energy < 0 || (HL+i)->age > senescence )
      (HL+i)->alive = false;
      }
   for(int i=0; i<Cpop; i++)
      {                        // How long is MY telomere?
      senescence = (int) (((CL+i)->EN & (1023 << 22)) >> 22) + 1;   1 - 1024
      if( (CL+i)->energy < 0 || (CL+i)->age > senescence )
      (CL+i)->alive = false;
      }
   }
```

void eat(void); // Absorb nutrients, eat plants, or eat herbivores

Organisms eat in order: plants, then herbivores, finally carnivores. Scan the plant list. Extract the x,y location to index into the ecosystem buffer E. Calculate satiety from its gene on the EN chromosome. If the plant's energy is greater than satiety the plant does not need to eat. If the plant is hungry calculate how much it can absorb by extracting the meal gene from the EN chromosome. I use three bits of the meal gene then divide it using four bits. This limits the maximum absorption to 50%. I added one before division for a lower limit on absorption. I did not want a plant which cannot absorb any nutrients. I added 1 to many (or all) of the gene expressions. So determine whether each plant is hungry. Then the plant gains energy at its absorption rate from the nutrient layer. Make sure to reduce the nutrient level by the same amount.

Next scan the herbivore list. This routine follows the outline used during plant eating. However, herbivores (and carnivores) eat over a wider area. First, they check the current x,y location for food. Then they scan the ring around x,y , checking each cell for plants (or herbivores) to eat. I index the rings with step. The current location would be step = 0, but I don't use that nomenclature. I set j instead. It is the index I use with dir[], the direction vector. If j=0 then we are pointing at the center of dir[]. Iterate j to walk around the first ring, which is referenced by step=1. Once you've visited each location on the step=1 ring, iterate step++. But you also set j=1 so it does not visit the center location again. It is easier to write than it is to explain :) The reason I did this becomes more clear as step increases and you visit more and more distant dir[] vectors. Notice you do not visit all of the neighboring points when step is greater than one. There are gaps. I don't think this matters but you should be aware of them. Herbivores scan the cells from x,y to the rings around them up to the fifth ring. nota bene { ecoDraw shows me the eating pattern. It causes mosaics and allows neighbors to eat closely to one another. kjr December 14 }

If there is a plant in a cell, the herbivore is 'asked' whether it is hungry. Or:

```
satiety = (float) ((HL+i)->EN & (31 << 10) >> 10) + 20.0;
if( (HL+i)->energy < satiety )
```

If this herbivore is hungry, scan the plant energy chromosome. Extract the 'meal' gene with

```
meal = (float) (((PL+i)->EN & (7 << 7) >> 7) + 1.0) / 16.0;
```

This gene ranges from 6.25 to 50.0% absorption. Add that amount of the plant's energy to the herbivore. Next, subtract that amount from the plant's energy. Add the remaining plant energy to the nutrient layer at E +Y*SZ +X. Set the ecosystem(x,y) plant pointer to NULL and mark the plant as no longer living. Once again it is easier to write the code than to explain it.

```
P = (E +C.y*SZ +C.x)->P;                    // Point to plant at E(x,y)
(HL+i)->energy += (PL+i)->energy * meal;    // Add energy to herbivore
(PL+i)->energy -= (PL+i)->energy * meal;    // Remove energy from plant
(E +Y*SZ +X)->nutrient += (PL+i)->energy;   // Remainder to nutrient layer
(E +Y*SZ +X)->P = NULL;                     // Plant no longer exists here
P->alive = false;                           // Plant is deceased
```

Carnivores use the same algorithm to eat with a few differences. First, the satiety gene uses a 5 bit mask instead of using an 8 bit mask. This means the carnivore can spend more time resting than the herbivore. Eight times more to be exact :) I may need to cut this back to four times as much. That would help the carnivore population curve track the herbivore curve better. Second, the carnivore has a larger eating range than do herbivores. While the herbivore eats from step=0 to step=5, a carnivore eats from 0 to 7. I had had it set for 15 rings but thought that excessive. I am tweaking the satiety gene calculation. I started with the mask at 255. Now it is at 63, then I add 50.0 after shifting. Instead of the range 1 to 256, it is 50.0 to 113.0. A higher lower limit and a lower upper limit. I am tweaking for effect. I have achieved a control which is over aggressive. There are no large swings in population levels. Herbivores track plants, carnivores track herbivores. I did add another limit: replenish() is called only when the nutrient level falls too low. nota bene { I now track how much energy is used during metabolism. Then I spread that amount around the nutrient layer when I replenish it. I am adding a tiny bit more, 0.7 energy units, but it may not be necessary to maintain stability. kjr December 14 }

```
void eat( void )
   {
   struct pnode *P;                         // Point at plants
   struct hnode *H;                         // Point at herbivores
   struct vt A, B, C;                       // Working vectors
   int j, ring;                             // Direction, ring#
   float satiety, absorb;                   // Genetic expression buffers
   int ch, speed;                           // Choice counter, express gene HB::G1
   float spine, eat;                        // Plant vs Herbivore spine war
   float strength, withstand;               // Plant vs Herbivore poison war
   float camo, see;                         // Herbivore vs Carnivore camo war

   Peat = Heat = 0;                         // Count the eaten
                                            // Ceat is calculated in die()
      // Plants eat nutrients at E(x,y) only
   for(int i=0; i<Ppop; i++)                // Express two genes
      {
      C = (PL+i)->X;
                                            // EN::G3  19.1 - 82.1
      satiety = (float) (((PL+i)->EN & (63 << 10)) >> 10) + 19.1;
      if( (PL+i)->energy < satiety )        // Am I hungry?
         {                                  // EN::G4  6.25 - 50.0%  1/16 to 8/16
         absorb = (float) ((((PL+i)->EN & (7 << 7)) >> 7) + 1.0) / 15.0;
         (PL+i)->energy += (E +C.y*SZ +C.x)->nutrient * absorb; // Add energy to plant
         (E +C.y*SZ +C.x)->nutrient -= (E +C.y*SZ +C.x)->nutrient * absorb;// Remove energy
         }
      }

      // Herbivores eat at E(x,y) then the rings around it
   for(int i=0; i<Hpop; i++)
      {                                 // Spine wars PL::G2 vs HB::G4  PL::G3 vs HB::G3
      ch = 0;                                 // Limit choice count
      speed = (int) (((HL+i)->HB & (15 << 27)) >> 27) + 12;   // HB::G1
      eat = (float) (((HL+i)->HB & (31 << 15)) >> 15) / 31.0; // HB::G4  ability eat spines
      withstand = (float) (((HL+i)->HB & (31 << 15)) >> 15) / 31.0; // HB::G3 poison safe
      satiety = (float) (((HL+i)->EN & (63 << 10)) >> 10) + 56.1;   // EN::G3 56.1 - 119.1
```

```
   A = (HL+i)->X;                          // Current location
   while( true )                           // Forever loop requires break out
      {           // Should this be eating radius instead?  Express another Gene!!!
      ring = rand()%speed;                 // {min <= ring <= speed}   DR::G1..G8
      B = scale( ring, move( (HL+i)->DR ) );   // Express direction genes
      C = bound( sum( A, B ) );            // Prospective new location

      if( (E +C.y*SZ +C.x)->P && ((HL+i)->energy < satiety ))
         {                                 // PFA number   63+31 = 94  63/94 = 0.67
         P = (E +C.y*SZ +C.x)->P;          // Point to plant at E(x,y)  EN::G4  0.1 - 0.8
         spine = (float) ((P->PL & (63 << 18)) >> 18) / 94.0;  // PL::G2 % spines
         strength = (float) ((P->PL & (63 << 12)) >> 12) / 94.0; // PL::G3 % poison

         if( (eat >= spine) && (withstand >= strength ))
            {             // HL::HB::G4 > PL::PL::G2   &&   HL::HB::G3 >= PL::PL::G3
            absorb = (float) ((((HL+i)->EN & (7 << 7)) >> 7) + 1.0) / 10.0;
            (HL+i)->energy += P->energy * absorb;  // Add energy to herbivore
            P->energy -= P->energy * absorb;       // Remove energy from plant
            (E +C.y*SZ +C.x)->nutrient += P->energy;  // Remainder to nutrient layer
            (E +C.y*SZ +C.x)->P = NULL;    // Plant no longer exists here
            P->alive = false;              // Plant is deceased
            Peat++;                        // Count the fodder
            }
         }

      if( ch > 29 )  break;  else ch++;   // Limit search count
      }
   }

   // Carnivores eat in rings around E(x,y) along the 8 primary directions.
for(int i=0; i<Cpop; i++)                  // Express thirteen genes
   {
   ch = 0;                                 // Choice counter
   speed = (int) (((CL+i)->CN & (15 << 27)) >> 27) + 12;   // CN::G1   12 - 27
   see = (float) ((CL+i)->SN & 255) / 255.0;    // SN::G5  sight ability
   satiety = (float) (((CL+i)->EN & (7 << 10)) >> 10) + 8.0; // 8.0 - 15.0

   A = (CL+i)->X;                          // Remember location.

   while( true )                           // Eat until break
      {
      ring = rand()%speed;                 // {min <= ring <= speed}   DR::G1..G8
      B = scale( ring, move( (CL+i)->DR ) ); // Motion vector from DR chromosome
      C = bound( sum( A, B ) );            // Prospective new location

      if( (E +C.y*SZ +C.x)->H && ((CL+i)->energy < satiety) )
         {                                 // HB::G5  camouflage effectiveness
         H = (E +C.y*SZ +C.x)->H;          // Point to herbivore at E(x,y)
         camo = (float) ((H->HB & (31 << 10)) >> 10) / 50.0;   // Herbivore camouflage
         if( see >= camo )                 // CL::SN::G5 >= HL::HB::G5
            {                              // 0.125 - 1.0
            absorb = (float) ((((CL+i)->EN & (7 << 7)) >> 7) + 1.0) / 8.0;
            (CL+i)->energy += H->energy * absorb;  // Add energy to carnivore
            H->energy -= H->energy * absorb;    // Remove energy from herbivore
            (E +C.y*SZ +C.x)->nutrient += H->energy; // Remainder to nutrient layer
            (E +C.y*SZ +C.x)->H = NULL;    // Herbivore no longer exists here
            H->alive = false;              // Herbivore is deceased
            Heat++;                        // Count the fallen
            }
         }
               // Is 57 excessive?
      if( ch > 57 )  break;  else  ch++;   // All your choice are us.
      }
   }
```

```
   }                                          // End of eat()
```

reproduce()

Herbivore and Carnivore births affect the nutrient layer of the ecosystem:

```
maturity = (PL+i)->EN & 3;           // EN::G4, ignore 2 high order bits
if( ( (PL+i)->age > maturity ) && ((PL+i)->energy > 9.0 ) )      // PFA number

(NP+nw)->energy = (PL+i)->energy * 0.25;   // Natal energy
(NP+nw)->energy += (PL+r)->energy * 0.25;  // 1/8 from each parent
(PL+i)->energy -= (PL+i)->energy * 0.125;  // Parent loses 1/8 of its energy
(PL+r)->energy -= (PL+r)->energy * 0.125;  // Other parent loses energy


maturity = (HL+r)->EN & 7 + 3;             // At least one day old
if( ( (HL+r)->age > maturity ) && ((HL+r)->energy > 40.0 ) )

(NH+nw)->energy = (HL+i)->energy * 0.125;  // Natal energy
(NH+nw)->energy += (HL+r)->energy * 0.125; // 1/8 from each parent
(HL+i)->energy -= (HL+i)->energy * 0.125;  // Parent loses 1/4 of its energy
(HL+r)->energy -= (HL+r)->energy * 0.125;  // Other parent loses energy


maturity = (CL+i)->EN & 7 + 2;             // EN::G4, ignore high order bit
if( ( (CL+i)->age > maturity ) && ((CL+i)->energy > 172.5 ) )      // PFA number

(NC+nw)->energy = (CL+i)->energy * 0.25;   // Natal energy
(NC+nw)->energy += (CL+r)->energy * 0.25;  // 1/4 from each parent
(CL+i)->energy -= (CL+i)->energy * 0.25;   // Parent loses 1/4 of its energy
(CL+r)->energy -= (CL+r)->energy * 0.25;   // Other parent loses energy
```

This function is the most complex of the simulation. Basically you scan through the plant list looking for plants which have enough energy and are old enough to mate. I used the satiety gene here too. If the organism energy is above the satiety level it can mate. Maturity is determined with a gene on the EN chromosome. If the plant passes both criteria a random prospective mate is selected. If that plant is old enough, and has enough energy the two plants exchange genetic material and expend energy. Energy is depleted from both parent plants. Some of that energy is passed to the offspring plant. The remainder is added to the nutrient layer at that x,y location. Crossover() is called twice: once for the EN chromosome, and again for the PL chromosome.

Once you find a suitable mate, exchanged genetic material, and expended energy, you need to find a new home for the offspring plant. I use the same routine as in eat(); walk around the dir[] ring. If there is an open spot drop out of the loop. If there are no unoccupied cells in that ring step to the next one outward and keep searching. If you can't find an open spot within xx rings then abort.

I wanted to randomly choose points around the ring. The present algorithm makes this difficult. I need to break out of the loop if no open spot is found. You'll start an infinite loop if you cannot find a spot and there is no way to break out. This lead me to scan each ring in order while counting rings. It is mechanistic but should not be a problem. Each ring is of the same dimension; there are only eight locations for each ring, no matter how large the ring. This scheme allows neighbors to fit their offspring in among the others. It was a quick way to span space to find an open cell.
Plants start their search at the fifth ring. The outer limit is at the twentieth ring. This mimics seed dispersal and covers the barren landscape quickly. I walk around the ring in a do loop. If there is a plant (or herbivore or carnivore) in the cell, then increment dir[] to the next location on the ring. I use two notations to index a cell location. Directly with X and Y, and with the vectors A and B. A is the

location of the mother, while B is the current point in the current ring. Add the vectors to get the prospective location. Convert it an X,Y coordinate pair to index into the ecosystem buffer E.

I mentioned the ecosystem cells are referenced as a 2D grid indexed by X and Y coordinates. This is true, except the edges are connected. If X is less than 0 you are pointing at the right side of the grid. If Y is greater than the grid you are pointing at the top of the grid; connecting the X edges form the space into a tube. Joining the Y edges gives you a toroidal space. Any herbivore walking off one edge magically appears on the opposite edge. This gives you an 'infinite' space on a finite grid.

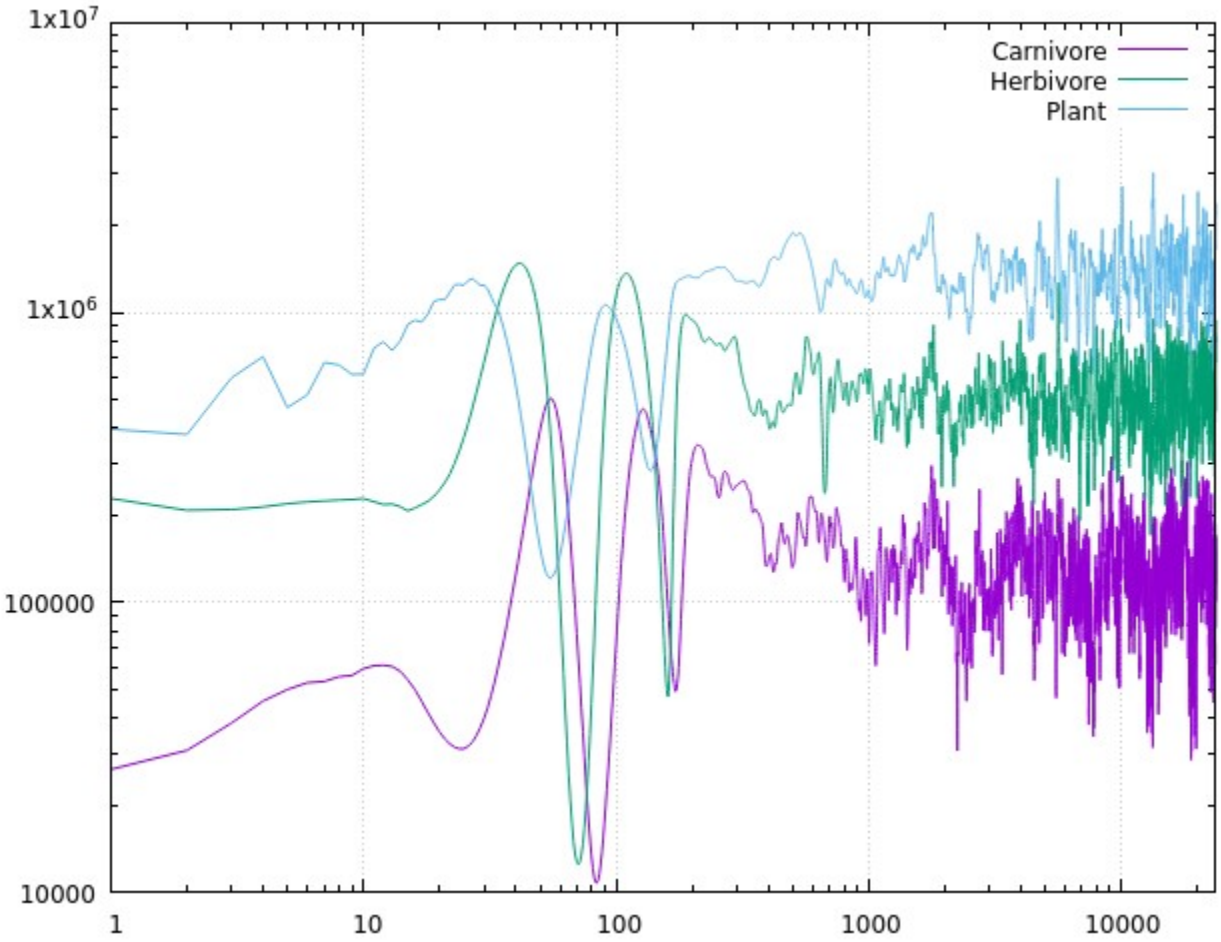
Once you have found an open cell it is time to form the new organism. I created a second buffer to hold the new organisms. This makes adding new organisms easier. I merge the two lists at the end of each cycle. The organisms eat(), metabolize(), reproduce(), die(), cull(), then add the new organisms at the end of the culled lists. Reproduction adds new organisms each to their own new list: NP, NH, or NC. Death marks the organisms which are too old, or have lost their energy, as deceased. Culling walks through the organism lists copying those which are alive to their alternate lists; the 'other' lists OPL, OHL, and OCL.

```
void reproduce( void )
   {
   int maturity, nw;                        // Age, new organism counter
   float satiety;                           // Am I hungry?
   struct vt A, B, C;                       // Working vectors

   newPpop = newHpop = newCpop = 0;         // Clear population counters here
      // 4000 * 4000 = 16,000,000   9/16 = 0.5625  9.5 Million seg faulted
   if( Ppop < SZ*SZ*0.5 )                   // HARD limit proportional to grid size
      {
      nw = 0;
      for(int i=0; i<Ppop; i++)
         {
         if( ((PL+i)->age > 3) && ((PL+i)->energy > 47.5) )
            {                               // && pollen > 0.25???
            A = (PL+i)->X;                  // Location of parent plant.
            int r = rand()%Ppop;            // Select random mate

            if( ((PL+r)->age > 3) && ((PL+r)->energy > 47.5) )
               {
                  // Need to find an empty spot in E
               int j = 1;                   // Skip center spot
               int ring = 5;                // Pollination inner boundary

               do {
                  B = scale( ring, dir[j] ); // Motion vector
                  C = bound( sum( A, B ) );  // Prospective new location

                  j++;                      // Walk around direction vector
                  if( j > 8 )               // End of dir[]
                     {
                     j = 1;                  // Search next
                     ring++;                 // outer ring
                     }
                  if( ring > 20 )  break;     // The Outer Limit!!
                  }    while( (E +C.y*SZ +C.x)->P );   // Occupied

                  // New plant needs to be filled
               (NP+nw)->X = C;                // Put plant at new location
               (NP+nw)->energy = (PL+i)->energy * 0.125;  // Natal energy
               (NP+nw)->energy += (PL+r)->energy * 0.125; // 1/8 from each parent
```

```
            (PL+i)->energy -= (PL+i)->energy * 0.125;  // Parent loses 1/8 of its energy
            (PL+r)->energy -= (PL+r)->energy * 0.125;  // Other parent loses energy

            (NP+nw)->EN = crossover( (PL+i)->EN, (PL+r)->EN ); // Crossover plant's
            (NP+nw)->PL = crossover( (PL+i)->PL, (PL+r)->PL ); // Chromosomes
            (NP+nw)->GM = (PL+i)->GM;     // Pass Mom's genetic marker on intact.
                                          // My name is 905
            (NP+nw)->alive = true;        // and I've just become alive
            (NP+nw)->age = 1;             // I'm the newest populator
                                          // of the planet called Earth.
            (E +C.y*SZ +C.x)->P = (NP+nw); // Point to plant data
            nw++;                         // Add new plant to count
            }
         }
      }
   newPpop = nw;                          // Set new plant list counter
   }                                      // End of plant population limiter

nw = 0;
for(int i=0; i<Hpop; i++)
   {
   maturity = (int) (HL+i)->EN & 3 + 1;   // EN::G4, ignore high order bit
   if( ((HL+i)->age > maturity) && ((HL+i)->energy > 105.0) )
      {
      A = (HL+i)->X;                      // Location of parent plant.
      int r = rand()%Hpop;                // Select random mate

         // Check whether (PL+r) is Old enough
      maturity = (int) (HL+r)->EN & 3 + 1; // At least one day old
      if( ((HL+r)->age > maturity) && ((HL+r)->energy > 105.0) )
         {   // Need to find an empty spot in E
         int j = 1;                       // Skip center spot
         int ring = 4;

         do {
            B = dir[ j ];                 // Walk around each direction
            B = scale( ring, B );         // Motion vector
            C = bound( sum( A, B ) );     // Prospective new location

            j++;                          // Walk around direction vector
            if( j > 8 )                   // End of dir[]
               {
               j = 1;                     // Search
               ring++;                    // next outer ring
               }
            if( ring > 14 )  break;
            }    while( (E +C.y*SZ +C.x)->H );  // Is this seat taken?

            // New herbivore needs to be prepared
         (NH+nw)->X = C;                  // Remember where I left my keys
         (NH+nw)->energy = (HL+i)->energy * 0.125;    // Offspring gains natal energy
         (NH+nw)->energy += (HL+r)->energy * 0.125;   // 1/8 from each parent

         (HL+i)->energy -= (HL+i)->energy * 0.250;     // Mother loses 1/4 of its energy
         (HL+r)->energy -= (HL+r)->energy * 0.125;      // Father loses 1/8 of its energy
         (E +C.y*SZ +C.x)->nutrient += (HL+i)->energy * 0.125;  // Afterbirth from mother

         (NH+nw)->EN = crossover( (HL+i)->EN, (HL+r)->EN );   // Crossover: Energy
         (NH+nw)->DR = crossover( (HL+i)->DR, (HL+r)->DR );   // Direction genes
         (NH+nw)->SN = crossover( (HL+i)->SN, (HL+r)->SN );   // Sensor genes
         (NH+nw)->HB = crossover( (HL+i)->HB, (HL+r)->HB );   // Herbivore genes
         // Genetic marker goes here
         (NH+nw)->alive = true;           // The Quickening
```

```
         (NH+nw)->age = 1;                // Day one of a new life
         (E +C.y*SZ +C.x)->H = (NH + nw); // Take up residence in my new home
         nw++;                            // Add new plant to count
         }
      }
   }
newHpop = nw;                            // Set new herbivore list counter

nw = 0;
for(int i=0; i<Cpop; i++)                // Reproduce carnivores
   {
   maturity = (int) (CL+i)->EN & 7 + 3;   // EN::G4
   if( ((CL+i)->age > maturity) && ((CL+i)->energy > 39.5) )
      {
      A = (CL+i)->X;                     // Location of parent herbivore.
      int r = rand()%Cpop;               // Select random mate

         // Check whether (PL+r) is Old enough
      maturity = (int) (CL+r)->EN & 7 + 3;  // EN::G4, ignore high order bit
      if( ((CL+r)->age > maturity) && ((CL+r)->energy > 39.5) )
         {
            // Need to find a empty spot in E and in OE
         int j = 1;                      // Skip the center spot
         int ring = 3;                   // Inner loop

         do {
            B = dir[ j ];                // Walk around
            B = scale( ring, B );        // the motion vector
            C = bound( sum( A, B ) );    // Prospective new location

            j++;                         // Walk around motion vector
            if( j > 8 )                  // Limit search count
               {
               j = 1;
               ring++;
               }
            if( ring > 9 )  break;
            }    while( (E +C.y*SZ +C.x)->C );

            // New carnivore needs to be filled
         (NC+nw)->X = C;                 // Put carnivore at new location
         (E +C.y*SZ +C.x)->C = (NC + nw); // Point to plant data

         (NC+nw)->energy = (CL+i)->energy * 0.125;  // Natal energy
         (NC+nw)->energy += (CL+r)->energy * 0.125; // 1/8 from each parent
         (CL+i)->energy -= (CL+i)->energy * 0.25;   // Mama loses 1/4 of its energy
         (CL+r)->energy -= (CL+r)->energy * 0.125;  // Papa loses 1/8 of its energy
         (E +C.y*SZ +C.x)->nutrient += (CL+i)->energy * 0.125;  // Afterbirth from mama

         (NC+nw)->EN = crossover( (CL+i)->EN, (CL+r)->EN );   // Crossover
         (NC+nw)->SN = crossover( (CL+i)->SN, (CL+r)->SN );   // Chromosomes
         (NC+nw)->CN = crossover( (CL+i)->CN, (CL+r)->CN );   // for offspring
         (NC+nw)->DR = crossover( (CL+i)->DR, (CL+r)->DR );   // genetics
         // Genetic marker goes here
         (NC+nw)->alive = true;
         (NC+nw)->age = 1;               // Newly born

         nw++;                           // Add new plant to count
         }
      }
   }
newCpop = nw;                            // Set new plant list counter   with offset
}                                        // End of reproduce()
```

VI Deeper



Once I created the sensor chromosome SN I had to express it. SN holds five sense genes: sight, smell, hearing, taste, and touch. Initially I implemented them as rudimentary tests: can a carnivore see well enough to sense a camouflaged herbivore? CN::SN::G5 > HB::G5 This works well enough but I can do better.

I based moveHerbivore(), moveCarnivore(), eat(), and reproduce() around a direction vector array, dir[]. The ecosystem is based on X,Y indexed cells. The direction vectors maintain that grid structure. If an organism is at cell (E +SZ*Y +X) it has 8 neighboring cells. The vector array mirrors that. But, the next layer out has 25-9 = 16 cells. This geometric expansion is expensive. It also means neighboring herbivores (or carnivores) are competing with one another. Whichever one feeds first starves the one who feeds next. So I limited motion, feeding, and placement of offspring to the 8 cardinal directions. At the first layer that means every cell. But with each subsequent layer outward this scheme has gaps. Those gaps allow neighboring herbivores access to the food which is in the gap of its neighbor, but directly accessible to it. Plants can place offspring compactly, while covering large areas with vegetation quickly. This scheme mimics the movement allowed a queen in chess.

Herbivores graze along those vectors while moving at their genetically determined speed. Each herbivore, and carnivore, has a direction chromosome DR. DR holds 8 genes, one for each of the

cardinal directions. Each gene represents the probability of moving in that direction. Each direction gene is 4 bits long. A carnivore moves along the direction randomly chosen from its highest probability genes. I am working on two speed settings for both the herbivore and carnivore population. When the herbivore is frightened, or the carnivore is chasing, they run. This requires more energy, that energy needs to be subtracted from the runner, then added to the energy total used to replenish() the system.

Then I thought of creating a sensor grid along the direction vectors. Each sensed point is along one of 8 major directions: N, NE, E, SE, etc. A carnivore could 'sense' the path which has the most prey for it to eat. That direction is passed to moveCarnivore() so it can feast. carnScan() forms a rudimentary sensor net around each animal, herbivore or carnivore. The function scans the local area along the cardinal vectors summing the number of herbivores found. Scan Hsense[] to find the highest prey density and return its index k. Use that as the index to dir[k] and apply scale, ring, and rand()%speed to move along that vector. Avoid leaping onto fellow carnivores. It is rude.

```
  // Create a sensor grid around the X,Y location of a carnivore.
void carnScan( void )
   {
   int maxRing = 15;                                        // Outer limit
   int X, Y;
   struct vt A, B, C;                                       // Working vectors

   for(int i=0; i<Cpop; i++)
      {
      A = (CL+i)->X;

      for(int k=1; k<9; k++)                  // We need to clear Hsense[] for each critter.
         (CL+i)->Hsense[ k ] = 0;

      for(int ring=1; ring<maxRing; ring++)                 // Scan direction vectors
         for(int k=1; k<9; k++)                             // Scan ring
            {
            B = scale( ring, dir[ k ] );
            C = bound( sum( A, B ) );                       // Search location
            if( (E +C.y*SZ +C.x)->H)  (CL+i)->Hsense[ k ]++;   // I found a herbivore :)
            }
      }
   }
```

In this case we are scanning outward for 15 'rings', a term I use to denote the distance from the start cell X,Y. Each ring has eight members so carnScan() examines 15 * 8 = 120 points. Each carnivore needs to be able to hold that data. I compressed it into ->Hsense[ ]++. The number of herbivores along each major direction vector is summed. Thus ->Hsense[3] could equal {0 to 15}. The next step is to find the direction with the most herbivores. This function should take an integer array, (CL+i)->Hsense[] as input. Then find its largest value. Return the index of that value to be used in dir[index] to point the direction vector. Its fingerprint is int maxHerb(int []).
Create the sensor array 10 rings outward for herbivores and carnivores. I want to use the sense genes to determine how far the organism can: see, hear, smell, taste, touch the plants, herbivores, and carnivores in that grid. PS[], HS[], and CS[] are density direction index arrays. How many plants in the PS[i] direction? How do I avoid the most carnivores by using CS[]? That sort of thing.

Here is the function using the CS[] carnivore sensor array. It finds the largest value and returns its index. This index is used with dir[index] to get the maximum population density direction vector. Use that direction and the sense chromosome{ touch, taste, see, smell, hear } spectrum. Each sense will have its own genetically determined value. If a carnivore's SN::see gene value is large, it can see even

well camouflaged herbivores. Or it can SN::smell herbivores better. A herbivore can SN::taste plants and notice poison. Or that herbivore can SN::smell if a plant has poison. Or the SN::touch gene so the herbivore will notice spines. I'm not sure about SN::hear yet. We shall see :)

```
int maxHerb( int CS[] )       // maxHerb( (CL+i)->Csense );
   {
   int Hmax;
   int max = -100;            // Guard
   for(int i=1; i<9; i++)
      if( CS[ i ] > max )
         {
         Hmax = i;
         max = CS[ Hmax ];
         }
   return Hmax;
   }
```

The minCarn() function is for finding a path between a bunch of hungry carnivores. The least dense population vector may be the way out. Or not.

I need to balance the amount of processor time it takes to create the sensor array against how useful it is for expressing the SN:: chromosome. Could we create a universal grid? Or can we get by with taking a sensor snapshot every second or third cycle? How useful is knowing your surroundings to your own survival? Is that value worth the added costs? You will be carrying a local sensor direction index array. Each one is a 9 integer array of values, so its not that high a cost. The cost comes in creating it each cycle. I've set the carnScan() to sum herbivore population along 8 primary directions for 15 steps. 8*15 = 120 cells are scanned. Each HS[] = {0 and 15}. Use maxHerb() to find the highest herbivore population density direction vector index. Aim the moveCarnivore() function in that direction for the best chance of a good meal.

If I limit herbScan() to 6 rings then I can limit the search to 6*8 = 48 cells. I need to check how fast a herbivore moves.

We need to rebuild the HB and CN chromosomes, especially grazing or pouncing distance. EN:: holds { metabolism, absorption, satiety, maturity, and senescence } PL:: will hold { pollination, poison, spines, scent, Seed dispersal range:5 to 12??, camouflage? } HB:: will hold { Speed of motion, grazing range, scent, camouflage, eat spines, eat poison } CN:: will hold { speed of motion, attack range, camouflage?, sleep } I am gradually working out how these genes can be expressed. It will take a few versions until I am satisfied.

```
void initDir( void )
   {
   int x = 1, y = 1;              // For unit vectors from D2Q9 LBM
   dir[0] = vector( 0,0 );                  //   Center
   dir[1] = vector( x,0 );                  //   East
   dir[2] = vector( -x,0 );                 //   West
   dir[3] = vector( 0,y   );                //   North
   dir[4] = vector( 0,-y );                 //   South
   dir[5] = vector( x,y );                  //   Northeast
   dir[6] = vector( x,-y );                 //   Southeast
   dir[7] = vector( -x,y );                 //   Northwest
   dir[8] = vector( -x,-y );                //   Southwest
   }
```

Do I stick with this order, the D2Q9 arrangement from Lattice Boltzmann theory? Or do I pick a more clock-like ordering? Say { N, NE, E, ... W, NW } or { E, SE, S, ... , N, NE } if you're a clockwise type person. If you are more mathematically inclined you would use the { E, NE, N, ... , S, SE } ordering which maps better to degrees around the origin. I chose the mathematical version.

```
   // Mathematical dir[]  :: { E, NE, N, NW, W, SW, S, SE }
void initDir( void )
   {
   int x = 1, y = 1;                          // For unit vectors
   dir[0] = vector( 0, 0 );                   // Center
   dir[1] = vector( x, 0 );                   // East
   dir[2] = vector( x, y );                   // Northeast
   dir[3] = vector( 0, y );                   // North
   dir[4] = vector( -x, y );                  // Northwest
   dir[5] = vector( -x, 0 );                  // West
   dir[6] = vector( -x, -y );                 // Southwest
   dir[7] = vector( 0, -y );                  // South
   dir[8] = vector( x, -y );                  // Southeast
   }

struct vt move( chromosome DR )               // Get direction from chromosome
   {
   float dirProb[9];                          // LBM version Math version
   dirProb[8] = (float)  (DR & 15);                 // SW   DR::G8  SE
   dirProb[7] = (float) ((DR & (15 << 4)) >> 4);    // NW           S
   dirProb[6] = (float) ((DR & (15 << 8)) >> 8);    // SE           SW
   dirProb[5] = (float) ((DR & (15 << 12)) >> 12); // NE           W
   dirProb[4] = (float) ((DR & (15 << 16)) >> 16); // S            NW
   dirProb[3] = (float) ((DR & (15 << 20)) >> 20); // N            N
   dirProb[2] = (float) ((DR & (15 << 24)) >> 24); // W            NE
   dirProb[1] = (float) ((DR & (15 << 28)) >> 28); // E    DR::G1  E
   dirProb[0] = 0.0;                          // 0 is at the center

   float tt = 0;                              // Eight directions, 4 bit probability range
   for(int i=1; i<9; i++)                     // Scan members of ring
      tt += dirProb[i];                       // Sum probabilities

   for(int i=1; i<9; i++)
      dirProb[i] /= tt;                       // Normalize direction probabilities

   float rn = fRan();                         // Gives us a number from 0.0 to 1.0 inclusive
   float sum = 0;                             // Sum probabilities until they exceed rn
   int dr = 0;                                // Direction buffer
   while (sum < rn)
      {
      dr++;                                   // Sum probabilities from 1 to 8 if necessary
      if (dr > 8) dr = 0;
      sum += dirProb[ dr ];                   // Probabilities must total 1.00
      }                                       // for this to work properly

   return dir[ dr ];                          // Prospective direction
   }
```

I modified crossover(). It now mutates a bit 5% of the time. I implemented it here so there is no favoritism. All bits are capable of being flipped to search the ecosystem more thoroughly.

```
chromosome crossover( chromosome CM, chromosome CP ) // Chromosome crossover
   {
   int i = rand() % 31;                         // Determine crossover point

   CM >>= i;                                    // Clear upper bits of Mama chromosome
   CM <<= i;
```

```
   CP <<= 31 - i;                              // Clear lower bits of Papa chromosome
   CP >>= 31 - i;

   if( fRan() > 0.95 )
      return mutation( CM | CP );              // Mutate chromosome 5.0% of the time
   else
      return ( CM | CP );                      // Merge upper and lower parts
   }
```

I now have implemented 21/37 of the genes. Plants have poison and spines. Herbivores can sense poison and spines, as well as enjoy a bit of camouflage. Carnivores can see, which lets them find the camouflaged herbivores. Keeps everything in balance. I want the herbivores to be able to sense spiny plants by touch and poisonous plants by smell. Carnivores should be able to sense herbivores by smell too. I also need to express pollination probability and hook it to the plant's age. Oh and poison should count against how much energy the herbivore can absorb from it. We could even have the herbivore keel over, though I would need to monitor its affect to the nutrient layer.

It is an interesting thought experiment. But I wrote it to model energy flow which is working. When the ecosystem is dynamically stable, the maximum energy in the organisms follows a pyramid shape. Plants most, herbivores in the middle, and carnivores have the least. But, then, the carnivores have the highest metabolism and the plants the lowest. I am hoping to make the scheme analytic so it is useful for more than just a diversion. I can test genetic ideas and see how they survive the competition.

VII Assignments

- Run ecoNode with the compile directive #define FIRST active.
- Rename one of the ecoXX.dat files to eco.dat
- Comment out #define FIRST and run ecoNode again to use the eco.dat snapshot.
- Uncomment some of the sampling functions to see how they work
- Yes, comment the loadData() function too :)
- Increase the mutation setting in crossover()
- Run the simulation to create new snapshots and the sample.dat file.
- Store them separately.
- Use loadSamples(), loadSampTwo(), and useSamples() to implement both sample pools.
- Monitor the GM chromosomes to see how the two populations interact.

VII Links

https://www.geeksforgeeks.org/genetic-algorithms/

https://medium.com/@byanalytixlabs/a-complete-guide-to-genetic-algorithm-advantages-limitations-more-738e87427dbb

https://cs.baylor.edu/~donahoo/tools/gdb/tutorial.htm

https://web.eecs.umich.edu/~sugih/pointers/summary.html

http://www.gnuplot.info/

http://www.gnuplot.info/download.html

VIII Appendices

Gene Expression techniques

Each chromosome has 32 bits of genetic information mapped as individual genes. I use shifted masks to extract individual genes. I implement the age of maturity gene by casting it as an integer. The compiler doesn't need the assist but it is nice to be explicit for the reader. The gene uses a mask of 3 (11 in binary) so the maximum age is 3 cycles. However, the minimum age could be zero which would cause problems. I limit the minimum age by adding 1.

```
int maturity = (int) (((PL+i)->EN & (7 << 19) >> 19) + 1;         PL::EN::G2  1 - 8
int maturity = (int) (((PL+i)->EN & (3 << 19) >> 19) + 1;
```

I implement the metabolism gene on the EN chromosome with mask equal to 15 (or 1111 in binary).

```
(PL+i)->energy -= (float) ((PL+i)->EN & 15) + 10.0;  Metabolism   PL::EN::G5
meal = (float) (((PL+i)->EN & (7 << 7) >> 7) + 1.0) / 16.0;       PL::EN::G4
```

Plants eat nutrients directly from the environment. Herbivores eat plants, and carnivores eat herbivores. I wanted a simple way to figure out the size of a meal, or how much energy can be absorbed by the organism from its energy source. I used a mask of 7 shifted 7 bits to the left for gene EN::G4. The value of the gene plus 1.0 gives a range from 1.0 to 8.0. I wanted to limit digestion efficiency to 50% or less so I divided those 8 choices by 16. That gives me a range from 1/16 to 8/16 or 6.25 to 50.0%. Divide by 12 to get 75.0% as the upper limit.

```
satiety = (float) ((HL+i)->EN & (31 << 10) >> 10) + 20.0;     //  HL::EN::G3
```

This gene tells me when an organism is hungry. I set it for a range of 20.0 to 51.0 energy units. I built the EN::G3 gene with a nine bit gene, offering a range of 0 to 511. I masked fewer bits (5) to limit the range to 0 to 31 with an offset of 20.0 energy units. I am tweaking the routines to better implement their genes. The metabolism of a plant is different from a herbivore which is different from a carnivore. I am finding their optimum settings. Offsets allow finer settings than simply changing the mask size.

For instance:

```
senescence = (int) (((HL+i)->EN & (63 << 22)) >> 22) + 1;     //  HL::EN::G1
```

In this case I chose 6 out of the 10 possible bits. 1024 is too old. It doesn't let the system follow environmental stresses rapidly enough. 63 or 31 are working now. Both plants and carnivores maximum age is 32, while herbivores can reach 64 cycles old.

```
int speed = (int) 3 + (((CL+i)->CN & (3 << 29)) >> 29);       // CL::CN::G1

float metab = (float) (EN & 127) + 1.0;                        EN::G5  1 - 128
float meal = (float) ((EN & (7 << 7) >> 7) + 1.0) / 16.0;      EN::G4  6.25 - 50.0%
float satiety = (float) ((EN & (511 << 10) >> 10) + 1.0 ;      EN::G3  1 - 512
int maturity = (int) ((EN & (7 << 19) >> 19) + 1;              EN::G2  1 - 8
int senescence = (int) ((EN & (1023 << 22) >> 22) + 1;         EN::G1  1 - 1024 days
```

```
(NP+nw)->EN = crossover( (PL+i)->EN, (PL+r)->EN );   // Crossover both plant's
(NP+nw)->PL = crossover( (PL+i)->PL, (PL+r)->PL );   // Chromosomes
```

I think this mimics nature more closely than one long chromosome. It also lets me craft more genes. 32 bits can be split many ways, it depends on the size of the mask you need. Remember, the gene can be used for the range of difference which is then offset how ever you wish. Or divide the result by the mask to get a number from 0.0 to 1.0, or 0 to 100%. Depends on what you want to do. Cast to integer or to float, also depending on what you want to do. Each chromosome is an unsigned int so it is good to cast it to an integer. Not strictly necessary but it is informative. It also depends on your compiler, is it strict or more laid back?

Organism energy modification.

initEcosystem() fill each cell with 100 units of energy.

initPlants() give each plant 750 units of energy

initHerbivores() give each herbivore 2500 units of energy

initCarnivores() give each carnivore 5000 units of energy

metabolize() Use the EN::G1 gene from each organism to determine how much energy is used per time cycle. I am currently tweaking how things work. The gene is 9 bits long but I am using fewer bits for plants and herbivores. I use 5 bits for plants and 6 bits for herbivores. So carnivores can use 511 units per time step, herbivores can use 63 units, while plants can use 31 units. I am adding an offset to each gene expression. 4.7 units for plants, 13.7 units for herbivores, and 257.1 units for carnivores. Highly PFA :)

```
(PL+i)->energy -= (float) (((PL+i)->EN & (31 << 22)) >> 22) + 4.7;      PL::EN::G1
(HL+i)->energy -= (float) (((HL+i)->EN & (63 << 22)) >> 22) + 13.7;     HL::EN::G1
(CL+i)->energy -= (float) (((CL+i)->EN & (511 << 22)) >> 22) + 257.1;   CL::EN::G1
```

eat() Plants draw energy from the nutrient layer in the Ecosystem. Herbivores eat plants to gain energy. Some of the plant energy is added to the herbivore while the rest of the energy lands in the nutrient layer. Carnivores eat herbivores. Some of the energy is absorbed by the carnivore while the rest winds up in the nutrient layer.

```
absorb = (float) (((PL+i)->EN & (255 << 13)) >> 13) + 5.0;              PL::EN::G2 absorb
avail = (float) (((PL+i)->EN & (255 << 4)) >> 4) + 3.0;                 PL::EN::G3 avail
avail = (float) (((HL+i)->EN & (63 << 4)) >> 4) + 1.0;                  HL::EN::G3

(HL+i)->energy += avail;     Energy in plant available to herbivore
(E +Y*SZ +X)->nutrient += P->energy - avail;
```

Energy from plant not absorbed by herbivore is added to the current cell of the Ecosystem.

replenish() Add 0.14 units of energy to each cell in the Ecosystem every time cycle. metabolize() now returns the total amount of energy lost by each organism. I use it as input to replenish(). I spread the energy evenly across E->nutrient plus a little extra; 2.7 right now. If it is too low the system collapses within a few thousand cycles. At higher settings it also collapses after population swings. Somewhere over 2.0 and less than 3.0 is working best. 100,000 cycles OK. {kjr November 11, 2024}

How do I test each module for accuracy?

initDir() -	void checkDir(void);	
initEcosystem() -	void check(void);	// Sanity checking code
initPlants() -	void printPlants(void);	
initHerbivores() -	void printHerbivores(void);	
initCarnivores() -	void printCarnivores(void);	
moveHerbivores() -		
moveCarnivores() -		
metabolize() -	void checkEnergy(void);	// Super sanity assured
replenish() -		
eat() -		
reproduce() -	void checkOffspring(void);	
die() -		
cull() -	void checkAlternate(void);	// Lucidity approaches
	void checkOthers(void);	// Nirvana achieved

bufferSwap() -	
addNew() -	displayNew() checkAlternate()
move() -	printPlants() printHerbivores() printCarnivores()
display(chromosome) -	geneLog() showGenes()

Metabolism
or a day in the life of a herbivore

Sleep at base metabolism, wake up and use more energy / cycle. Eat plants to gain energy. If chased by a carnivore run. This uses the highest rate of energy. A carnivore wakes up sensing its own hunger. It hunts, it chases, it feeds. If it is not hungry then it won't expend energy hunting. Is there an upper limit? For a herbivore or carnivore we need an "I'm full" gene determined level. How much energy does a herbivore gain for each plant it eats? Carnivore from a herbivore? When a herbivore or carnivore is sated it does not feed. It can move around. It can reproduce.

A plant uses a quantity of energy each day due to its metabolism. It contains a quantity of energy available to herbivores. A plant absorbs XX units of energy from the nutrient layer each day.

All organisms need a senescence gene.
How long should a plant live if it is not eaten? 127 days? 2^7 = 128
How long for a herbivore? 1023 days? 2^10 = 1024
How about a carnivore? 2047 days? 2^11 = 2048

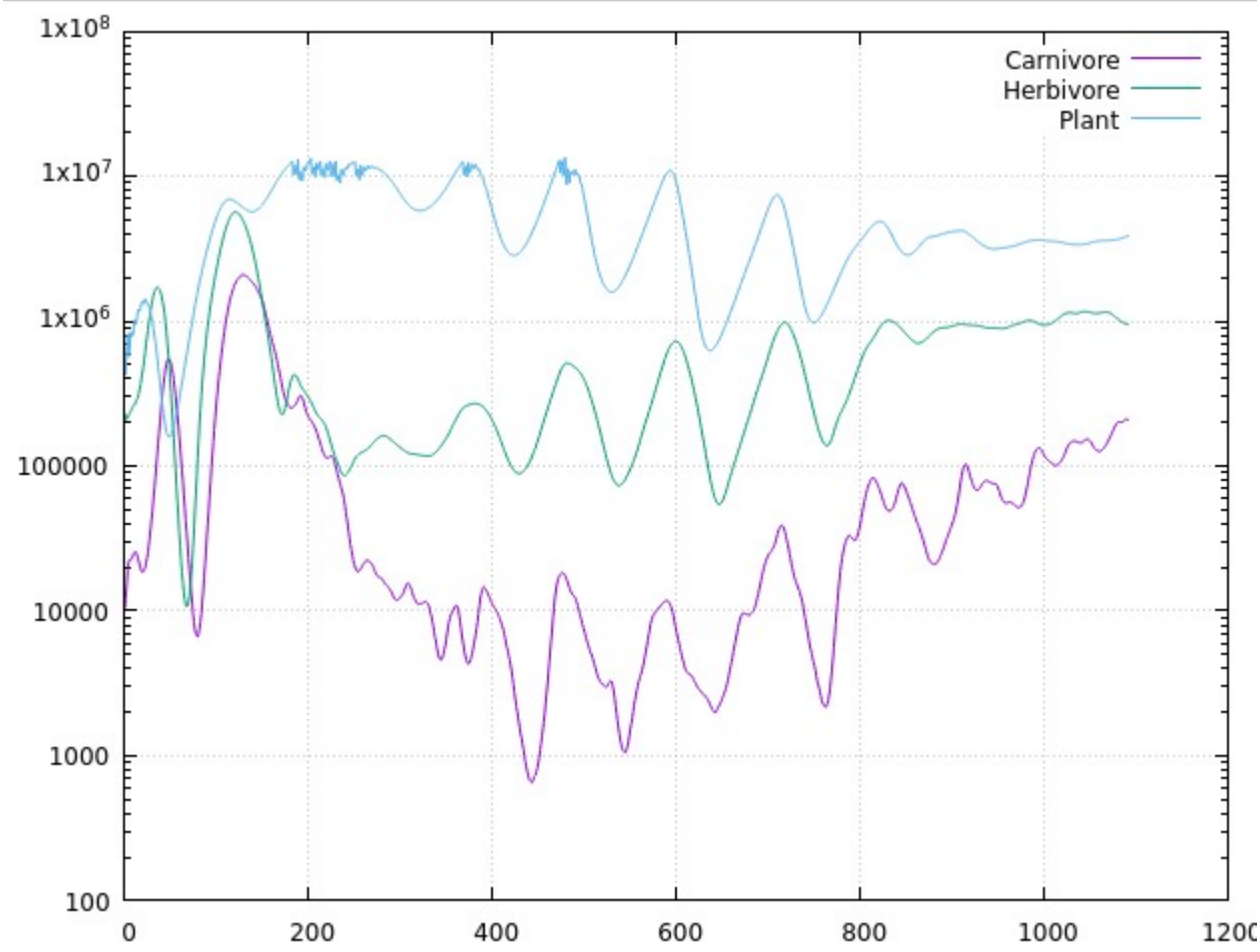
```
Then H->SN::See > P->PL::Camo   eat()    SN::G5
and  H->SN::Smell > P->PL::Scent  eat()  SN::G3   PL::G1
if P->PL::Pollen and P->EN::Age fit criteria  reproduce()
Plant can breed from age 3 to age 5 if I am not mistaken
or at age == 4 and 5 to be more accurate then senescence
This means it can pollinate twice and it can be pollinated twice
Or we could extend the range downward
   If age == 3 then reproduce() if pollen/2 > xx
   at age == 4 reproduce() if pollen > xx
   at age == 5 reproduce()
```

Gene Mapping

```
see = (float) ((HL+i)->SN & 255) / 255.0;                    SN::G5  sight ability
hear =  (float) (((CL+i)->SN & (255 << 8)) >> 8) / 255.0;    SN::G4  hearing ability
smell = (float) (((HL+i)->SN & (255 << 16)) >> 16) / 255.0;  SN::G3  scent capability
taste   = (float) (((HL+i)->SN & (15 << 24)) >> 24) / 15.0;  SN::G2  taste sense
touch = (float) (((HL+i)->SN & (15 << 28)) >> 28) / 15.0;    SN::G1  touch sensitivity

Hard wire plant senescence at 5 cycles
Then set its maturity at 3 or 4 cycles.
Thus each could create two or three offspring.
They can get eaten at any age from 1 to 5 cycles.

pollen = (float) ((PL+i)->PL & 63) / 63;                     PL::G5  pollination probability

Then H->SN::See > P->PL::Camo   eat()
and  H->SN::Smell > P->PL::Scent  eat()
if P->PL::Pollen and P->EN::Age fit criteria  reproduce()
Plant can breed from age 3 to age 5 if I am not mistaken
or at age == 4 and 5 to be more accurate then senescence
This means it can pollinate twice and it can be pollinated twice
Or we could extend the range downward
   If age == 3 then reproduce() if pollen/2 > xx
   at age == 4 reproduce() if pollen > xx
   at age == 5 reproduce()

      32 bits      ->EN
Metabolism gene - How much energy do I use each day?
Absorb gene - How much energy can I gain per plant (or herb)?
Satiety gene - When am I full?
Maturity gene - When am I old enough to reproduce?
Senescence gene - How long can I live (if not eaten)?

Energy genes:      ->EN      in all organisms
   7 b metabolism                 2^7 = 128
   3 b absorption                 2^3 = 8
   9 b satiety                    2^9 = 512
   3 b maturity                   2^3 = 8
   10 b senescence                2^10 = 1024

Plant genes:       ->PL
   6 bits pollination probability  2^6 = 64
   6 bits poison probability
   6 bits poison strength
   6 bits spine probability
   6 bits spine length
   2 bits free

Herbivore genes:   ->HB
   5 bits probability of running when carnivore sensed
   5 bits probability of hiding
   5 bits percent camouflage effectiveness
   5 bits ability to eat spiny plants
   5 bits ability to withstand poison (should affect energy absorbed by strength of poison)
   5 bits escape range  leap?
   2 bits speed                   2^2 = 4

Carnivore genes:   ->CN
   6 bits attack probability when herbivore sensed
   2 bits attack range            2^2 4 is more accurate 0 to 3
   6 bits sleep probability  affected by energy level hungry or sated?
   6 bits hide probability        2^6 64  bit value/64 = % value
   6 bits percent camouflage effectiveness
```

```
   3 bits pursuit max              2^3 8 steps is about right
   2 bits speed      speed gene  It would scale the dir[] vector
   1 free

    Five senses { sight, hearing, smell, taste, touch }
Sensor genes:      ->SN     in herbivores and carnivores
   8 bits sight            camouflage effectiveness would be tested here
   8 bits hearing            hear predator or prey
   8 bits smell     32-24 = 8      smell poison or prey
   4 bits taste            taste poison, touch spines
   4 bits touch      2^4 = 16  pursue vibration of ground?? or hide if sensed

Direction genes:  ->DR  in herbivores and carnivores
   4 bits East probability
   4 bits West probability
   4 bits North probability
   4 bits South probability
   4 bits Northeast probability
   4 bits Southeast probability
   4 bits Northwest probability
   4 bits Southwest probability

(PL+i)->EN for energy genes { metabolism, absorption, satiety, maturity, senescence }
(PL+i)->PL for plant genes { pollination%, poison%, poison strength, spine%, spine length }
(HL+i)->HB for herbivore genes { run%, hide%, camo%, eat spine%, eat poison% }
(CL+i)->CN for carnivore genes { attack%, attack range, sleep%, hide%, camo%, pursuit time }
(HL+i)->SN for sensor genes { touch range, hearing range, seeing range, smelling range }
(CL+i)->DR for direction genes { E, W, N, S, NE, SE, NW, SW }

(PL+i)->EN for energy genes { energy/step, absorb%, available%, maturity }
(PL+i)->PL for plant genes { pollination%, poison%, poison strength, spine%, spine length }
(HL+i)->SN for sensor genes { touch range, hearing range, seeing range, smelling range }
(HL+i)->HB for herbivore genes { run%, hide%, camo%, eat spine%, eat poison% }
(CL+i)->CN for carnivore genes { attack%, attack range, sleep%, hide%, camo%, pursuit time }
(CL+i)->DR for direction genes { E, W, N, S, NE, SE, NW, SW }

   // Genetic information for each gene in each chromosome
Direction genes:  ->DR  direction probability chromosome

dirProb[8] = (float)  (DR & 15);                            DR::G8 move SW
dirProb[7] = (float) ((DR & (15 << 4)) >> 4);               DR::G7 move NW
dirProb[6] = (float) ((DR & (15 << 8)) >> 8);               DR::G6 move SE
dirProb[5] = (float) ((DR & (15 << 12)) >> 12);             DR::G5 move NE
dirProb[4] = (float) ((DR & (15 << 16)) >> 16);             DR::G4 move S
dirProb[3] = (float) ((DR & (15 << 20)) >> 20);             DR::G3 move N
dirProb[2] = (float) ((DR & (15 << 24)) >> 24);             DR::G2 move W
dirProb[1] = (float) ((DR & (15 << 28)) >> 28);             DR::G1 move E

float metab = (float) (EN & 127) + 1.0;                     EN::G5  1 - 128
float absorb = (float) ((EN & (7 << 7) >> 7) + 1.0) / 16.0; EN::G4  6.25 - 50.0%
float satiety = (float) ((EN & (511 << 10) >> 10) + 1.0 ;   EN::G3  1 - 512
int maturity = (int) ((EN & (7 << 19) >> 19) + 1;           EN::G2  1 - 8
int senescence = (int) ((EN & (1023 << 22) >> 22) + 1;      EN::G1  1 - 1024 days

pollen = (float) ((PL+i)->PL & 63) / 63;                    PL::G5  pollination probability
poison = (float) (((PL+i)->PL & (63 << 6)) >> 6) / 63;      PL::G4  poison probability
strength = (float) (((PL+i)->PL & (63 << 12)) >> 12) / 63;  PL::G3  poison strength
spine = (float) (((PL+i)->PL & (63 << 18)) >> 18) / 63;     PL::G2  spine probability
length = (float) (((PL+i)->PL & (63 << 24)) >> 24) / 63;    PL::G1  spine length

touch = (int) ((HL+i)->SN & 63;                             SN::G4 touch range
hear = (int) ((HL+i)->SN & (63 << 6)) >> 6;                 SN::G3 hearing range
see = (int) ((HL+i)->SN & (63 << 12)) >> 12;                SN::G2 seeing range
smell = (int) ((CL+i)->SN & (63 << 18)) >> 18;              SN::G1 smelling range
```

```
run = (float) ((HL+i)->HB & 31) / 31.0;                      HB::G7 run probability
hide = (float) (((HL+i)->HB & (31 << 5)) >> 5) / 31.0;       HB::G6 hide probability
camo = (float) (((HL+i)->HB & (31 << 10)) >> 10) / 31.0;     HB::G5 camouflage effectiveness
eat = (float) (((HL+i)->HB & (31 << 15)) >> 15) / 31.0;      HB::G4 ability to eat spines
withstand = (float) (((HL+i)->HB & (31 << 15)) >> 15) / 31.0; HB::G3 withstand poison
escape = (int) ((HL+i)->HB & (31 << 25)) >> 25;              HB::G2 escape range
speed = (int) ((HL+i)->HB & (3 << 30)) >> 30;                HB::G1 speed gene

attack = (float) ((CL+i)->CN & 63) / 63.0;                   CN::G7 attack probability
range = (float) (((CL+i)->CN & (3 << 6)) >> 6) / 4.0;        CN::G6 attack range
sleep = (float) (((CL+i)->CN & (63 << 8)) >> 8) / 63.0;      CN::G5 sleep probability
hide = (float) (((CL+i)->CN & (63 << 14)) >> 14) / 63.0;     CN::G4 hide probability
camo = (float) (((CL+i)->CN & (63 << 20)) >> 20) / 63.0;     CN::G3 camouflage effectiveness
time = (int) ((CL+i)->CN & (7 << 26)) >> 26;                 CN::G2 maximum pursuit time
speed = (int) ((CL+i)->CN & (3 << 29)) >> 29;                CN::G1 speed gene
```



Chromosome exchange

```
(NP+nw)->EN = crossover( (PL+i)->EN, (PL+r)->EN );   // Crossover both plant's
(NP+nw)->PL = crossover( (PL+i)->PL, (PL+r)->PL );   // Chromosomes
(NP+nw)->GM = (PL+i)->GM;                            // Genetic marker unchanged

(NH+nw)->EN = crossover( (HL+i)->EN, (HL+r)->EN );   // Crossover both plant's
(NH+nw)->HB = crossover( (HL+i)->HB, (HL+r)->HB );   // Chromosomes
```

```
(NH+nw)->SN = crossover( (HL+i)->SN, (HL+r)->SN );   // Chromosomes
(NH+nw)->DR = crossover( (HL+i)->DR, (HL+r)->DR );

(NC+nw)->EN = crossover( (CL+i)->EN, (CL+r)->EN );   // Crossover
(NC+nw)->SN = crossover( (CL+i)->SN, (CL+r)->SN );   // Chromosomes
(NC+nw)->CN = crossover( (CL+i)->CN, (CL+r)->CN );
(NC+nw)->DR = crossover( (CL+i)->DR, (CL+r)->DR );
```

Rebuilt Energy Chromosome

Chromosome EN has only 31 bits filled by RAND_MAX so only they are useful

```
RAND_MAX = 2147483647;  This is not the full range of ~4 billion
```

Metabolism gene - how much energy do I use each day 2^7 = 128
Absorb gene - how much energy will I gain per plant (or herb) 2^3 = 8
Satiety gene - when am I full? 2^9 = 512
Maturity gene - old enough to reproduce? 2^3 = 8
Senescence gene - how long can I live (if not eaten) 2^10 = 1024

2^3 = 8 / 16 50% maximum absorption add 1 to numerator before division

```
float metab = (float) (EN & 127) + 1.0;                      EN::G5  1 - 128.0
float absorb = (float) ((EN & (7 << 7) >> 7) + 1.0) / 16.0;  EN::G4  6.25 - 50.0%
float satiety = (float) ((EN & (511 << 10) >> 10) + 1.0 ;    EN::G3  1 - 512.0
int maturity = (int) ((EN & (7 << 19) >> 19) + 1;            EN::G2  1 - 8
int senescence = (int) ((EN & (1023 << 22) >> 22) + 1;       EN::G1  1 - 1024 days
           ((EN & (511 << 22) >> 22) + 1;  for herbivores  1 to 512 days
           ((EN & (63 << 22) >> 22) + 1;   for plants      1 to 64 days

Replenish  E(x,y)->nutrient *= 1.01;  for a 1% increase in the nutrient level energy
```

Minimum energy would be satiety for both Ma and Pa organisms:

```
if( ((PL+r)->age > maturity ) && ((PL+r)->energy > satiety) )

int senescence = (int) ((EN & (1023 << 22) >> 22) + 1;            EN::G1  1 - 1024 days
((EN & (511 << 22) >> 22) + 1;   for herbivores  1 to 512 days
((EN & (63 << 22) >> 22) + 1;    for plants      1 to 64 days

void die( void )
   {
   int senescence;
   for(int i=0; i<Ppop; i++)
      {
      senescence = (int) (((PL+i)->EN & (63 << 22)) >> 22) + 1;   PL::EN::G1  1 - 64
      if( (PL+i)->energy < 0 || (PL+i)->age > senescence )
      (PL+i)->alive = false;
      }
   for(int i=0; i<Hpop; i++)
      {
      senescence = (int) (((HL+i)->EN & (511 << 22)) >> 22) + 1;  HL::EN::G1  1 - 512
      if( (HL+i)->energy < 0 || (HL+i)->age > senescence )
      (HL+i)->alive = false;           // Why didn't I take the blue pill?
      }
   for(int i=0; i<Cpop; i++)
      {                                // How long is MY telomere?
      senescence = (int) (((CL+i)->EN & (1023 << 22)) >> 22) + 1; CL::EN::G1  1 - 1024
```

```
    if( (CL+i)->energy < 0 || (CL+i)->age > senescence )
    (CL+i)->alive = false;
    }
}
```

Miscellaneous Notes

Diffuse deposited nutrients over time:
Nutrients are supplied through replenishment, feeding, or reproduction.
Need nutrient diffusion time step relationship
j = - D * dc/dx
D is the diffusivity constant
dc/dx is rate of change of concentration in the x direction (negative away)
And j is the amount flowing through a space

How about a little variation in nutrient replenishment?
Richer areas along watersheds, or in patches
Add along one or two edges with rhythmic pulses
Flow from one or two edges toward and out of the opposite side or sides
Flow could be toroidal or not, depending on your whims :)

Separate speed from eat radius moveCarnivore(), moveHerbivore() should use speed not eat(). It should have its own gene: eat radius.

Make both CN::Speed and HB::Speed larger 4 bits instead of just two
HB::{ camo, eat spine, withstand poison, speed } useful and expressed
CN::{ speed, ?? } useful. Now rework both HB and CN with more useable genes.
What do we need? Eat radius is good.
Move speed can be 4 bits for HB and 5 bits for CN with modifiers of course.

Herbivore run/hide choice implies the ability to run so maybe a high gear speed gene?
And run/hide is a continuum not a toggle switch so HB::run at 90% would trigger running
HB::run below a certain threshhold would trigger hiding.
HB::run would trigger moveHerbivore() to use HB::High speed gene
Then a limit on how many cycles to hold the frightened state

CN::attack/rest is also a continuum
CN::chase would be a high speed gear for chasing
CN::pursuit time would be number of cycles for chase
CN::attack would also trigger a higher metabolic rate, same for Herbivores
Thus CN::metab and HB::metab for the higher rates.

If HB->SN::see < CN::camo then HB gets eaten.
Do we need Boolean flags for keying higher metabolic rates in metabolism() or do we extract the energy from the running organisms directly? I say we modify metabolism() to notice the change. Because it also sums the energy used by all of the organisms throughout this cycle. Why keep track of that in two places which would require a global variable? We could check HB::run/hide and CN::attack/rest against each other in metabolism() just as we are checking genes inside other functions.

Or extract the normal metabolic() energy rate and extract more in moveHerb() moveCarn() at the higher metabolic rate.

moveHerbivore() & moveCarnivore() need to access the SN genes of each of them
That will fulfill the SN::taste, SN::touch, SN::smell, SN::see can be implemented.
Implement a move/do not move continuum with your sensing. For example: while you are moving you search for a cell which is NOT occupied by a similar organism. Along the way you can collect information about each cell you visit along the way. Sense the presence of a herbivore, plant, or carnivore.

```
if( (E +Y*SZ +X)->H)  C->Hsense[ i ]++;   // I found a herbivore :)
```

Sense along rings of 8 points at the Cardinal directions E,NE,N etc.

```
Sum P,H,C in dir[i]  put into sense[8];
```

Plants can have a scent or they could be camouflaged in some way
Two SN genes: smell, touch vs two PL genes: scent, camo
Herbivores can have a scent so HB::scent vs carnivore SN::smell
How do we compare run and chase? HB::CN
Attack vs defend?? CN::HB

```
if( SN::G1->smell > PL::G4->poison ) do not eat
if( SN::G1->taste > PL::G4->poison )
if( SN::G4->touch > PL::G2->spine ) do not eat
if( HB::G3->withstand > PL::G3->strength ) eat() but EN::G4->absorb * 0.4 or more
if( EN::G2->maturity && EN::G1->senescence > PL::G5->pollen ) mate
if( SN::G2->see > HB::G5->camo ) eat
if( SN::G3->hear > HB::G6->hide ) eat

Camouflage is susceptible to better vision.
Poison is sensed by better smell and/or taste.
Carnivores could be sensed by better sight, hearing, or smell
Herbivores could be sensed through the same mechanisms.
Carnivores need to sense herbivores: camo, sight, smell, vibration - touch or hearing

if( fRan() < PL::G5->pollen ) spread your seed
   Link this trait to the age of the plant.
   Older plants, nearing senescence, increase the probability of pollination.
   If PL::G5->pollen = 75% (0.75) then any chance less than 0.75 allows pollination.
Senescence:
   at 5 cycles PL::G5->pollen = 100%??  95%
   at 4 cycles or EN::G1->senescence - 1   PL::G5->pollen * 0.5
   at 3 cycles or EN::G1->senescence - 2   PL::G5->pollen * 0.25

How do I combine spine probability with spine length to get a reasonable metric
for comparison with the ability to eat spiny plants?

Ability to eat poisoned plants EP ranges from none to all or 0.0 to 1.0 probability
It competes with poison probability (concentration) PP combined with poison strength PS.
PP ranges from 0.0 to 1.0    PS from 0.0 to 1.0
Now, how do I combine PP and PS???

PP * PS  gives 0.0 to 1.0    if PP=0.5 * PS=0.8 = 0.4   0.9 * 0.9 = 0.81
if( EP > PP * PS ) eat();  Equality goes to the plant

if( PS > 95 && EP < 50 ) ->alive = false;  H->energy to E(x,y)->nutrient

Express the pollination gene once reproduction of plants is working OK.
```

```
        PL::G5, pollination probability
pollen = (float) (((PL+i)->PL & 63) + 1) / 64;    // in reproduce()
if( fRan() > pollen ) spread your seed
```

I have not written a graphic interface yet. I am still working out the bookkeeping aspect of the problem. However, when I do I will monitor how the direction vector concept works at placing offspring in the environment.

I think (HL+i)->Psense[] and (CL+i)->Hsense[] are sufficient.
Then it would be feeding and pouncing with no running.
That way there is no need a higher metabolic rate
Use maxPlant() and maxHerb()
How do I find the second highest density paths?

When do we use the DR direction genes and when do we sense plants or herbivores? DR for reproduction and eating I'm pretty sure. The SN and ->Psense[] or ->Hsense[] for movement. Once I am here I use the DR chromosome. When I am about to move I herbScan() and carnScan(). In that order. The herbivores will move toward the highest plant population density. Then the carnivores will sense the highest herbivore density and move after them. Feasting; the survivors get to reproduce. Yes, scan, move, eat, reproduce, die. *Vita brevis*.

Think about a genetic marker chromosome, GM. It can contain 31 bits of information. It is passed on in one chunk from the mother to her offspring. There is NO crossover with a matching paternal GM chromosome. We could use it to mark specific populations of organisms generated by the user and implemented as a second sample pool inside useSamples(). Since the chromosome is monolithic we can trace whether the user created viable organism samples or not. Run a count on GM of a certain type. Remember to initialize GM to be all zeros so any difference triggers a flag. We could mark the various sample pools with different GM settings. Then determine how they compete for resources with one another.

==========

Write script for installing and using gnuplot to view output from ecoNode (dt.dat file).

www.gnuplot.info
http://www.gnuplot.info/download.html

https://sourceforge.net/projects/gnuplot/files/gnuplot/6.0.1/

https://sourceforge.net/projects/gnuplot/files/gnuplot/ For any newer version.