Swarm Space Library

The Swarm Space library is the beginnings of a library to assist in building environments for interacting agents. In general, environments can be just as varied as the agents themselves (in one view, the environment itself is simply another agent). However, many simulations have similar types of environments that can be helpfully supported by generic code.

> The current space library only addresses simple kinds of discretized 2d space. Improvement is planned in the future: see the todo list (../todo.html) for ideas. Briefly, coordinates need to be elevated to the status of objects, which should hopefully allow spaces of different scales and boundary conditions to interact through a common reference system. In addition, other types of spaces are desired: continuous coordinates, other dimensions, arbitrary graphs, etc.

Discrete2d

Root class of all 2d discrete spaces. A Discrete2d is basically a 2d array of ids. Subclasses add particular space semantics onto this. Currently Discrete2d grids are accessed by integer pairs of X and Y coordinates.

Creation

- setSizeX: (int)x Y: (int)y
  Set the world size.
- createEnd
  Create the lattice, precompute the offsets based on Y coordinate.

Querying Discrete2d state

- (int)getSizeX and - (int)getSizeY
  Get the size of the lattice.

Global Operations

- fastFillWithValue: (int)aValue
- fastFillWithObject: anObject
- fillWithValue: (int)aValue
- fillWithObject: anObject

- `makeOffsets`
  Given an array size, compute the offsets array that caches the multiplication by ysize. See the `discrete2dSiteAt` macro.
- `(id *)allocLattice`
  Allocate memory for the lattice.

Grid2d

Grid2d is a generic container class to represent agent position on a 2d lattice. It gets most of its behaviour from Discrete2d, adding extra code to check that you don't overwrite things by accident. Grid2d is pretty primitive: only one object can be stored at a site, no boundary conditions are implied, etc. A fancier Grid2d is in the works.

New Methods

- `setOverwriteWarnings: (BOOL)b`
  If set to true, then if you try to store something at a site that doesn't have 0x0 there, a warning will be generated.

Overridden methods

- `putObject: anObject atX: (int)x Y: (int)y`
  Replaces the Discrete2d method. First check to see if it should do overwrite warnings, and if so if you're going to overwrite: if both conditions are true, print out a warning message. Regardless of the check, it writes the new object in.
+ `createBegin: aZone`
  Make overwrite warnings be on by default.

DblBuffer2d

DblBuffer2d augments Discrete2d to provide a form of double buffered space. Two lattices are maintained: lattice (the current state), and newLattice (the future state). All reads take place from lattice, all writes take place to newLattice. newLattice is copied to lattice when `- updateLattice` is called. DblBuffer2d can be used to implement one model of concurrent action, like in Ca2ds. `NOTE`: be very careful if you're using low-level macro access to the world, in particular be sure that you preserve the write semantics on the newLattice.

Overridden Methods

```
+createBegin:  aZone
      Set number of states to 2.

- initializeLattice
      Initialize lattice to random 1/3 in state 1.

- stepRule
      Run Conway's Life rule (simpleminded version).
```

Diffuse2d

Discrete 2nd order approximation to 2d diffusion with evaporation. Math is done in integers on the range [0,0x7fff].

New Methods

```
- setDiffusionConstant:  (double)d
      Set the diffusion constant.  Values over 1.0 might not be valid.

- setEvaporationRate:  (double)d
      Set the evaporation rate.  Values over 1.0 don't make much sense.
```

Overridden Methods

```
+createBegin:  aZone
      Set diffusion constant and evaporation rate to 1.0, numStates to 0x7fff.

- initializeLattice
      Initialize world to 0.

- stepRule
      Run discrete approximation to diffusion.  Roughly, it's
      newHeat = evapRate * (self + diffuseConstant*(nbdavg - self))
      where nbdavg is the weighted average of the 8 neighbours.
```

Object2dDisplay

Object2dDisplay helps display 2d arrays of objects. Create a Object2dDisplay, give it a Raster widget to draw on, a Discrete2d, a message to call on each object, and (optionally) a collection of objects and it will dispatch the message to all objects with the Raster widget as an argument. In addition, Object2dDisplay can help you make probees.

color = state / m + c
If not set, assume m == 1 and c == 0.

Use

- display
  Draw the array on the given widget. Note that you still have to tell the widget to draw itself afterwards. The code for display uses the fast macro access in Discrete2d on the cached return value from getLattice. It also caches the drawPointX:Y: method lookup on the display widget - this is a nice trick that you might want to look at.

Int2dFiler

The Int2dFiler class is used to save the state of any Discrete2d object (or a subclass thereof) to a specified file.

A Note on Output Formats

The issue of what format(s) the class should know about is still open. For the purposes that I have encountered, simply storing the values in space-delimited line format:

```
value0_0 value0_1 ...
value1_0 value1_1 ...
...
...
```

has been sufficient. In particular, the output of this class has been used, in conjunction with the fragstats (ftp://ftp.fsl.orst.edu/pub/fragstats.2.0/) package, to apply landscape metrics to the output of a Swarm simulation. Nevertheless, I (Manor Askenazi (mailto: manor@santafe.edu)) would be very interested in hearing from end-users about any other (presumably more complex formats), as the need arises.

- setDiscrete2dToFile: (Discrete2d *)aSpace
  Set the target space to be filled. This message can be used more than once, but often it is useful to keep one Int2dFiler per space (e.g. when the space is saved multiple times).
- setValueMessage: (SEL)aSelector
  This message is optional. It is used when the target Discrete2d contains objects. By sending each object the message specified by the selector, the Int2dFiler is able to get from the object an integer representing its state, which it then writes to the file.
- setBackground: (int)aValue