NovaScript Cheat Sheet

4 July, 2014

The following NovaScript functions and statements are useful for enhancing graphic models. For more information, see http://www.novamodeler.com. Please send questions, suggestions, and errors to support@novamodeler.com.

Capsules Embedded as Cells in a CellMatrix with Square Cells

Location

`coords` A coordinates object[1]. Use **`coords.row`** and **`coords.col`** to get the calling cell's row and column number within the CellMatrix

`rows, cols` The total number of rows and columns in the enclosing CellMatrix

`WRAP(coords), WRAP(row, col)` Performs a "wraparound" of the coordinates if they exceed the dimensions of the CellMatrix or are negative. WRAP(coords) returns a coordinates object[1] containing the new row and column. WRAP(row, col) returns an array containing the new row and column.

Identifying neighbors

`CELLBLOCK(n), CELLWBLOCK(n)` Returns an array of cell state objects[2] of a square block ≤ *n* units away from the calling cell (including the center cell). CELLWBLOCK is the "wrapped" version, which treats the surface as a torus.

`BLOCK(n), WBLOCK(n)` Same as *CELLBLOCK(n)* but returns an array of coordinates[1] objects

`CELLRING(n), CELLWRING(n)` An array of cell state objects[2] of a square *exactly n* units away from the calling cell. CELLWRING is the "wrapped" version, which treats the surface as a torus.

`RING(n), WRING(n)` Same as *CELLRING(n)* but returns an array of coordinates objects

Getting values of cell components

`CELL(coords)` The state object[2] for the cell at coordinates *coords*[1].

`CELL_VALUE(coords, comp)` Returns the current value of *comp* in the cell at *coords*[1]

`CELLS()` Returns a 2-dimensional array of state objects[2] for the CellMatrix. E.g., CELLS()[row][col]

Summary Functions

`COUNT_CELLS(lst, comp, value)` *lst* is a list of cell state objects, *comp* is the name of a component in those cells, and *value* is a number or string. Returns the number of cells in *lst* for which the current value of *comp* equals *value*.

`ALL_CELLS(lst, comp, value)` Arguments same as above. Returns True if the current value of *comp* in **all cells** in *lst* equals *value*, else False.

`NO_CELL(lst, comp, value)` Arguments same as above. Returns True if the current value of *comp* in **none of the cells** in *lst* equals *value*, else False.

`SOME_CELL(lst, comp, value)` Arguments same as above. Returns True if the current value of *comp* in **at least one cell** in *lst* equals *value*, else False.

Capsules Embedded as Cells in a CellMatrix with Hexagonal Cells

`coords, rows, cols, CELL(coords)`, and **`CELLS()`** same as square CellMatrix

`HEXBLOCK(n)` A list (array) of coordinates objects[1] comprising the hexagonal block of cells ≤ *n* units away from the calling cell

`HEXRING(n)` A list (array) of coordinate objects for all cells comprising the hexagon *exactly n* units away from the caller

`HEXPATH(dir, dist)` Returns a list of coordinates objects comprising a path of length *dist* in the direction *dir* denoted by compass directions[3].

Capsules Embedded as Agents in an AgentVector

Referencing agents

`myId` The calling agent's id

`AGENTS_AT(coords)` List of agents located at *coords*[1]

`AGENT_IDS()` An array of ids for currently living agents.

`AGENTS()` An array of agent state[2] objects

`AGENT_COUNT()` Total number of agents

Grabbing values of agent components

`AGENT(id)` A state[2] object for agent *id*

`AGENT_VALUE(id, comp)` The current value of component *comp* in agent *id*

Location and movement

`rows, cols` The total number of rows and columns in the AgentVector

`CELL_COORDS(id)` Returns a coordinates object[1] for agent *id*, or of the calling agent if *id* is omitted

`LOCATION(id)` Returns an object with properties *x*, *y*, and *theta* of of agent *id* or the caller if *id* is omitted.

`MOVE(x, y)` Moves the calling agent to *x*, *y* (usually placed inside a Command component)

`SET_HEADING(theta)` Sets the directional heading (in radians)

`CWRAP(coords)`[1] Same as CellMatrix

`RANDOM_MOVE(loc), WRANDOM_MOVE(loc)` Returns a location object[6] representing a random move (non-wrapping and wrapping, respectively) of one unit from location object *loc*. If *loc* is omitted it defaults to the location of the calling agent.

Special movement components

init_x, init_y The name (not expression) of a term or pin that holds the initial *x* and *y* coordinates of the agent in the AgentVector

init_heading The name (not expression) of a term or pin that holds the initial direction (in radians) of the agent in the AgentVector

Birth, death and age

`birth` The time when the calling agent was created

`AGE(id), MYAGE()` The time since birth of agent *id* or the caller

`CREATE([init], [n])` Schedules the creation of *n* new agents (1 if *n* omitted) at the end of the time step. *init* is an initializer object containing values for properties in the new agent; if omitted the new agent is a clone of the caller

`KILL(id)` Schedules the elimination of agent *id* at the end of the time step

Summary Functions

`COUNT_AGENTS(lst, comp, value)` *lst* is a list of agent state objects, *comp* is the name of a component in those agents, *value* is a number, string, or other data type. Returns the number of objects in *lst* for which the current value of *comp* is *value*.

`ALL_AGENTS(lst, comp, value)` Arguments as above. Returns True if component *comp* in **all** agents in *lst* equals *value*

`NO_AGENT(lst, comp, value)` Arguments as above. Returns True if the current value of *comp* in **none** of the agents in *lst* is equal to *value*, else False

`SOME_AGENT(lst, comp, value)` Arguments as above. Returns True if the current value of *comp* in **at least one** of the agents in *lst* is equal to *value*, else False.

Capsules Embedded as Cells in a SimWorld

`AGENTS_AT, AGENT_COUNT, AGENT_IDS, AGENT_VALUE, AGENTS, CREATE, KILL, CELLBLOCK(n), CELLWBLOCK(n), CELLRING(n), CELLWRING(n)` Same as CellMatrix or AgentVector.

`MYAGENTS()` List of agents currently contained in the calling cell

`MYAGENT_COUNT()` Number of agents currently contained in the calling cell

`AGENTBLOCK(n, ["sort"]), AGENTWBLOCK(n, ["sort"]), AGENTRING(n, ["sort"]), AGENTWRING(n, ["sort"])` An array of state objects[2] of all agents contained in the cell block or ring specified by *n*. If "sort" is included, the list is sorted in increasing distance from the calling cell.

Capsules Embedded as Agents in a SimWorld

`MYCELL()` State object[2] of the cell containing the calling agent
`HEXMOVE(dist, dir)` (SimWorlds with hexagonal cells only) moves the calling agent distance *dist* in the direction *dir*[3].
`CELL, CELLS, CELL_VALUE` Same as CellMatrix

Capsules Embedded as Nodes in a NodeNetwork

`myId` The calling node id
`count` The number of nodes in the NodeNetwork
`CONNECTIONS_IN(id)` Returns the array of connections[5] **into** node *id* (if *id* is omitted assumed to be the caller)
`CONNECTIONS_OUT(id)` Returns the array of connections[5] **from** node *id* (if *id* is omitted assumed to be the caller)
`NODE(id)` Returns a state object[2] for node *id*
`NODE_COUNT()` Returns the total number of nodes
`NODE_VALUE(id, comp)` Returns the current value of component *comp* in node *id*
`NODES()` Returns the array of node state objects[2]
`INFLOW(id)` Returns the total strength of connections **into** node *id* (if *id* is omitted assumed to be the caller)
`OUTFLOW(id)` Returns the total strength of connections **from** node *id* (if *id* is omitted assumed to be the caller)

Capsules Embedded as Agents in a NetWorld

Coming soon…

Time

`TIME()` Current simulation time
`STEP(x, y)` Returns x if the current time is y or greater; 0 otherwise
`DT()` Returns current delta value (dt)
`SIMSTART()` Simulation start time
`SIMEND()` Simulation end time
`SIMMETHOD()` Integration method
`CLOCK()` Returns the current clock as an object

Input/Output

`BASEDIR()` Returns the current model directory
`LOAD(lst)` *lst* is a list of JavaScript or NovaScript filenames contained in the current model directory. Each is loaded into the runtime system (use in simulation initialization).
`OPENREAD(file)` Opens text filename *file*[4] for reading and returns a Java BufferedReader object (use methods *read* and *readLine* to perform input)
`OPENWRITE(file)` Opens text filename *file*[4] for writing and returns a Java PrintWriter object (use methods *print* and *println* to perform output)
`READFILE(file)` Returns the content of the filename *file*[4] as a string.

Generic Summary Functions

`COUNT(fn, lst)` *fn* is a function that takes one argument and returns a Boolean; *lst* is an array. Applies *fn* to each element of *lst* and returns the number of times the result is TRUE.
`TOTAL(fn, lst)` *fn* is a function that takes one argument and returns a number; *lst* is a list. Applies *fn* to each element of *lst* and returns the sum of the results.
`_.map(arr, fctn)` Applies function *fnct* to each element of array *arr*, and returns an array of the results.

Probability and Math Functions

Probability
`SEED(x)` Sets the seed of the random number generator and returns nothing; should be part of simulation initialization
`RANDOM()` Returns a uniformly distributed random number 0..1
`NORMAL(x, y)` Returns a random number from the normal distribution with mean *x* and standard deviation *y*
`POISSON(lambda)` Returns a random number from the Poisson distribution with density *lambda*
`FLIP(p)` Returns true with probability *p* and false with probability *1-p* (simulates a Bernoulli trial)
`UNIFORM(x, y)` Returns: a uniformly distributed random variable between x and y

Trigonometry
`Math.PI` Value of pi
`SIN(x), COS(x)` Returns the sin and cos of *x* (in radians)
`SINWAVE(x,y)` Returns x*sin(2π t/y), where *t* is the current time
`COSWAVE(x,y)` Returns: x*cos(2π t/y), where *t* is the current time

Math
`DERIVN(fn, n)` Returns the value of the nth derivative of *fn* at the current time, with precision based on the value of dt
`DISTANCE(x0, y0, x1, y1)` Returns Euclidean distance between points (x0, y0) and (x1, y1)
`Math.pow(x,y)` *x^y*
`Math.xxx` Any method xxx from the JavaScript Math library

Matrix Operations

A matrix in JavaScript is a two-dimensional array.

`CSVTOMAT(csv)` *csv* is a string where line is a comma separated sequence of values. Returns the matrix in which each row corresponds to a line in *csv*.
`COLUMNSPLIT(tab)` *tab* is a 2-dimensional array derived from a table, where the first row contains column headers. Returns an object in which each property name is a column header with property value an array comprising the corresponding column.
`ROWSTOOBJS(tab)` *tab* is a 2-dimensional array derived from a table, where the first row contains column headers. Returns an array of objects, one for each non-header row. In each object properties are column headers bound to the entry for that column in the corresponding row.
`TRANSPOSE(mat)` Returns the transpose of mat, where *mat* is a matrix (i.e. 2-dimensional array)

Debugging a Model

Closing a non-responsive Nova window
Windows: ctrl+shift+esc to open Task Manager, select 'Java Platform', then 'EndTask'
Mac:

Simulation Feedback
`ALERT(msg)` Displays *msg* in an alert box
`PRINT(msg)` Prints *msg* to the console
You may also use **Table component** or **Spy plugin** to display the value of components as the model runs.

Console commands
command+p (Mac) or **ctrl+p** (PC) Repeat last command at console
`_.keys(x)` display the properties of x
`main` Top level capsule

If you step through a simulation, you can type commands at the console to get the current value of objects.

Given an AgentVector named *myav* at the top level:

`main.myav.AGENT_COUNT` The number of agents in *myav*
`main.myav.AGENTS[0].Self.dx` The value of a component named *dx* in the first (0th) agent embedded in *myav*
`main.myav.LOCATION(0).x` The *x* coordinate of the first agent in *myav*

Given a CellMatrix named *Life_Matrix* at the top level:

`main.Life_Matrix.rows` The number of rows
`main.Life_Matrix.CELL(15,15).state` The value of a component called 'state' in cell 25,25

`var lst = main.Life_Matrix.CELL(15,15).CELLBLOCK(2)` An array of 25 cell state objects surrounding cell (15, 15), including the center.

`COUNT_CELLS(lst, "state", 1)` The number of cells in *lst* whose component 'state' is currently equal to 1.

Given a SimWorld component named *world* at the top level:

`main.world.AGENT(0).AGENT_IDS()` An array of the ids of all alive agents in *world*

`main.world.AGENT(0).MYCELL()` The cell that contains agent 0 (1st agent)

`main.world.CELL(0,0).MYAGENT_COUNT()` The number of agents that fall in cell (0,0).

JavaScript General

Note that JavaScript and NovaScript are **case sensitive**.

Defining constants

Global constants are usually defined in the program window in the top-most level of the model.

```
const unburned = 0, burning = 1;
```

Declaring variables

```
var x, y = 17, z = "hello";
```

Arrays – one dimensional

```
var myCars=new Array("Saab","Volvo","BMW");
var a = new Array();
a[0]="red";
a[1]="blue";
var b = [1,2,3,4];
print(b.length);
var x = b[0] + b[1];
foo = [];
foo.push("hi");
```

Arrays – two dimensional (i.e., matrices)

```
var array2d = [[1,2],[3,4],[5,6]];
var x = array2d[0][0];
```

Loops

```
a = [11,22,33]
for (var i in a) {
    print("Item " + i + "=" + a[i]);
}
for (var i = 0; i < 10; i++) {
    x = x + i;
}
foo = [];
for (var i = 1; i != 4; ++i) foo.push(i)
```

Comparison operators

```
x == y  // True if x and y equal
x != y  // True if x and y inequal
```

Conditional Statements

```
if (x > y) {
    z = x;
} else {
    z = y;
}
z = (x > y) ? x : y;
```

Custom Functions

```
function triple(y) {
    return y * y * y;
}
```

Commenting Code

Most components have a comment field for comments (recommended). You can also put comments in code:

```
/* This is a code comment
which can span multiple lines */

// Single-line comment (doesn't work in terms)
```

Notes

[1] A *coordinates* object has two properties, *row* and *col*. Any function that takes **`coords`** as an argument can accept either a coordinates object or two integers (row, col).

[2] a *state object* is a type of object where you can get the current value of an individual component contained in the object simply by referencing it by name, e.g., *CELLS(2,3).mystock*

[3] Directions are denoted by compass directions; i.e., “N”, “NE”, “SE”, “S”, “SW”, “NW”.

[4] If a filename begins with “/” it is treated as an absolute pathname; otherwise it is treated as relative to the current model directory.

[5] A connection object has 3 properties: *id* (the node id of the source), *strength* (the raw strength of the connection), and *n_strength* (the normalized strength of the connection, where the total strength of all connections into the caller is 1).

[6] A location object is an object that has two properties *x* and *y*