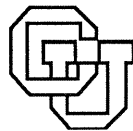


**AGENTSHEETS:
THE MANUAL**

Alex Repenning

CU-CS-699-94



University of Colorado at Boulder

DEPARTMENT OF COMPUTER SCIENCE

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR(S) AND DO NOT NECESSARILY REFLECT THE VIEWS OF THE AGENCIES NAMED IN THE ACKNOWLEDGMENTS SECTION.

**AGENTSHEETS:
THE MANUAL**

CU-CS-699-94 1994

**Department of Computer Science
University of Colorado at Boulder
Campus Box 430
Boulder, Colorado 80309-0430 USA**

AGENTSHEETS: THE MANUAL

Alex Repenning

Department of Computer Science and Institute of Cognitive Science
Campus Box 430
University of Colorado, Boulder CO 80309
(303) 492-1218, ralex@cs.colorado.edu
Fax: (303) 492-2844

This document is an early version of a manual for Agentsheets. Despite its history reaching back to 1990 and a large number of users Agentsheets has not featured a manual up to this point. This is about to change..

This document does not serve as a conceptual introduction to Agentsheets. It is assumed that the reader has some basic understanding of the Agentsheets paradigm and has used a Macintosh before.

This research is supported by the National Science Foundation under the grant MDR-9253425, Apple Computer Inc., and US WEST Advanced Technologies. The HCC group at the University of Colorado has provided invaluable and insightful help supporting this work.

INDEX

AgenTalk 24
Apply Tool To Agent 25
Click Mouse On Agent 25
Drag Mouse Onto Agent 25
Drag Tool Onto Agent 26
End-Users & Scenario Designers 1
Non-Spatial Communication (Verbal) 23
Press Key 26
Spatial Communication 23
Spatio-Temporal Metaphor Designer 1
Substrate Designer 1

Agentsheets: The Manual	1
1. Introduction	2
1.1. Philosophy of this Manual	2
1.2. Who uses Agentsheets How?.....	2
1.3. Getting Agentsheets.....	3
1.4. System Requirements.....	3
1.5. Further Reading	4
2. Your Turn	6
2.1. Using an Agentsheets Application	6
2.2. Creating an Agentsheets Application	11
3. The Agentsheets Environment	18
3.1. Important FRED Commands.....	18
3.2. Symbol Completion.....	18
4. Reference Manual	20
4.1. Menu Commands.....	20
4.2. AgenTalk.....	24
4.3. Dialog Boxes.....	28
4.4. Agents drawing and erasing Agents	29
4.5. Generic Depictions: Adding Flexibility to the task of drawing.....	30
4.6. Browsing Agent Classes	30
5. Power Users	32
5.1. Specializing the Demo Navigator	32
5.2. Adding your own sounds.....	32
5.3. Adding your own Tools.....	32
5.4. Modifying Depictions with 3rd Party Tools	32
5.5. Creating your own Worksheet classes and file types.....	32
5.6. New Version.....	32
6. Appendix A: Agent Classes	33
6.1. Class: AGENT.....	34
6.2. Class: SELECTABLE-AGENT	37
6.3. Class: COLOR-PALETTE-SHEET-AGENT	38
6.4. Class: VALUE-AGENT.....	39
6.5. Class: TEXT-NODE-AGENT.....	40
6.6. Class: CHARACTER-AGENT	41
6.7. Class: LINKABLE-AGENT	42
6.8. Class: COLOR-GALLERY-AGENT.....	43
6.9. Class: GROUPABLE-AGENT.....	44
6.10. Class: HYPER-AGENT	45
6.11. Class: BACKGROUND-AGENT	46
6.12. Class: SELECTABLE-BACKGROUND-AGENT	47
6.13. Class: COLOR-PIXEL-AGENT	48
6.14. Class: ICON-EDITOR-COLOR-PIXEL-AGENT	49
6.15. Class: LOCATION-AWARE-AGENT.....	50
6.16. Class: ACTIVE-AGENT.....	51

7. Appendix B: Agentsheet Classes	52
7.1. Class: AGENTSHEET.....	53
7.2. Class: LINK-AGENTSHEET.....	56
7.3. Class: ICON-EDITOR-SHEET	58
7.4. Class: COLOR-PALETTE-SHEET	59
7.5. Class: GRAPHER-AGENTSHEET.....	60
7.6. Class: COLOR-GALLERY-AGENTSHEET.....	61
7.7. Class: HYPER-AGENTSHEET	64
Index	1

1. INTRODUCTION

1.1. PHILOSOPHY OF THIS MANUAL

For the sake of brevity this manual focuses on the essential points of operation. That is, it makes assumptions regarding the familiarity of the reader's knowledge about things such as how to use a Macintosh. The hope is to make this manual not quite as boring as manuals sometimes seem to be. On the other hand, however, it is quite likely that I have left out crucial steps in my descriptions. This can happen if one is the author of a system. The philosophy of this manual is that I do not write about things that I would not like to read in somebody else's manual. Please feel free to inform me about your struggles so that I will be able to improve the quality of this manual for you and others. Thank you.

1.2. WHO USES AGENTSHEETS HOW?

The philosophy of Agentsheets is to facilitate the design of domain-oriented dynamic, visual environments through supporting three different types of users with a layered architecture:

- *End-Users & Scenario Designers*: use applications that have been tailored to their application domain by Spatio-Temporal Metaphor Designers. Often, they are unaware of the notion of agents. Typical activities of end-users include drawing, playing "what-if" games, and adjusting parameters.
- *Spatio-Temporal Metaphor Designer*: Design spatial representations and metaphors typically based on previous paper and pencil diagrams used by end-users and scenario designers. The notion of agents organized in a grid serves the designers as a construction paradigm to create dynamic, visual representations.
- *Substrate Designer*: Designs a substrate that enables high-level designers to do their jobs. The interaction between the spatio-temporal metaphor designer and the substrate designer leads to a gradual extension of the tool.

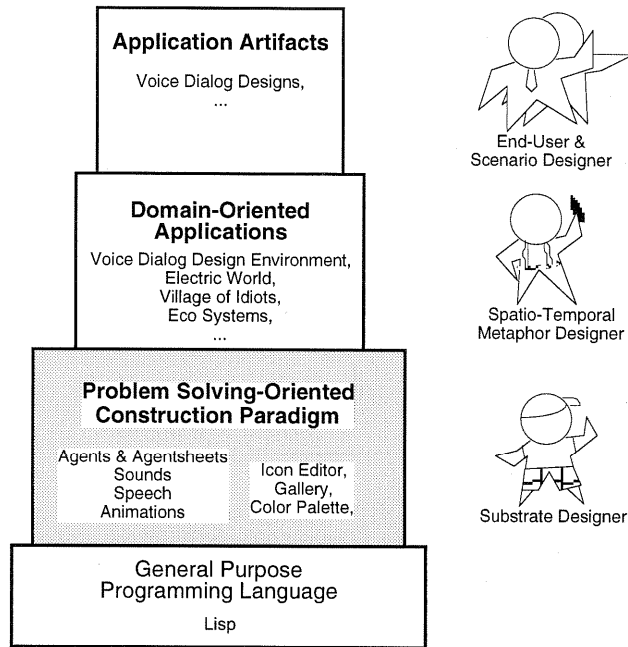


Figure 1. Layers and People

Up to this point the manual was only concerned with end-user activities. The following sections discuss additional issues relevant for designers of Agentsheets applications.

1.3. GETTING AGENTSHEETS

You can get Agentsheets by FTP. Connect via anonymous FTP to the Agentsheets server

```
ftp 128.138.204.87
```

At the Name: prompt type the word anonymous. Please use your Email address as password. Then cd to the agentsheets FTP folder

```
cd "mt evans"/"agentsheets FTP folder"
```

The quotes are crucial and need to be typed in order to specify Macintosh file names that contain blanks. Get the file Agentsheets.sea.hqx

```
get Agentsheets.sea.hqx
```

This will take a while since the file is about 6 MB. After de-binhexing the file you receive, which is a self extracting archive, uncompress the file simply by double clicking .

1.4. SYSTEM REQUIREMENTS

System software: System 7

RAM: >= 8 MB (you can get away with less if you enable virtual memory)

Hard Disk: > 10 MB hard disk space left

CPU: > 68020 (68030, 68040, 68LC040)

other software: you need a license for Macintosh Common Lisp (MCL 2.0). You do not actually need the software as Agentsheets includes the crucial parts of MCL but you WILL need the license in order to legally run Agentsheets. All restrictions of MCL 2.0 as a product apply.

1.5. FURTHER READING

This manual is only about **HOW** to do things but only superficially covers the philosophy of Agentsheets, i.e., the **WHY**. The following documents are relevant for the understanding of Agentsheets:

1. Repenning, A., "Agentsheets: A Tool for Building Domain-Oriented Dynamic, Visual Environments," University of Colorado at Boulder, Ph.D. dissertation, Dept. of Computer Science, 171 Pages, 1993.

The dissertation describes the Agentsheets philosophy at dept. Additionally, it provides a survey over some of the Agentsheets applications.

2. Repenning, A., "The OPUS User Manual," *Technical Report*, CU-CS-556-91, Department of Computer Science, University of Colorado at Boulder, Boulder, Colorado, 1991.

This is for power users that need to know more about the underlying object system.

3. Repenning, A., "Agentsheets: A Tool for Building Domain-Oriented Visual Programming Environments," *INTERCHI '93, Conference on Human Factors in Computing Systems*, Amsterdam, NL, 1993, pp. 142-143.

A brief overview on what Agentsheets is and what kind of things have been done with it.

4. Repenning, A. and W. Citrin, "Agentsheets: Applying Grid-Based Spatial Reasoning to Human-Computer Interaction," *1993 IEEE Workshop on Visual Languages*, Bergen, Norway, 1993, pp. 77-82.

A paper describing the implications of organizing agents in a grid.

5. Repenning, A., "Domain-Tailored Spatial Metaphors," *INTERCHI '93: Workshop on Spatial Metaphors*, Amsterdam, the Netherlands, 1993.

A survey of the Boulder visual programming tools directed by Clayton Lewis covering NoPumpG, ChemTrains and Agentsheets.

6. Repenning, A. and T. Sumner, "Using Agentsheets to Create a Voice Dialog Design Environment," *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, Kansas City, 1992, pp. 1199-1207.

This paper describes Agentsheets from two perspectives. Agentsheets as a system to create domain-oriented visual environment and the application of Agentsheets by US WEST to build a voice dialog design environment.

7. Repenning, A., "Creating User Interfaces with Agentsheets," *1991 Symposium on Applied Computing*, Kansas City, MO, 1991, pp. 190-196.

- E a r l y D r a f t -

An application-oriented paper that illustrates the use of Agentsheets to create a visual programming front-end to a commercial expert system.

2. YOUR TURN

2.1. USING AN AGENTSHEETS APPLICATION

The following procedure shows how to open an Agentsheets application. It is meant as a scenario and does, therefore, only describe one way to do things. More sophisticated techniques will follow in other chapters.

2.1.1. Launching Agentsheets

Double click the file Agentsheets/COLOR. This will bring up the start up window. You can get rid of the window by clicking at it.

Hint: You can suppress the greeting sound by holding down the shift key.

The listener window will inform you about the progress of loading and will warn you if resources such as the sound file or the speech manager are missing.

2.1.2. Open a Gallery

To work with Agentsheets means either to use or to create Agentsheets applications. An application consists of a gallery describing the spatial characteristics of agents, and a text file containing class definitions specifying the behavior of agents.

To open a gallery use the **File > Open > Gallery...** menu command. This will give you a selection gallery files. For now, lets start with a simple application called Particle World.



Figure 2. Particle World Gallery

What you see here is the gallery containing two agents. The purpose of a gallery is to serve as a repository of agents that can be placed into worksheets. The particle world gallery features two types of agents that are used to simulate a very simplistic mechanical model of particles. Fixed particles remain in space wherever you put them. Movable particles, on the other hand, adhere to a naïve model of gravity that makes them want to drop unless if they are on top of an obstacle such as a fixed particle.

2.1.3. Creating a Worksheet and Operating Tools

Next you create a worksheet with the **File > New > Worksheet...** menu command. The worksheet is just another agentsheet serving as work area for the user.

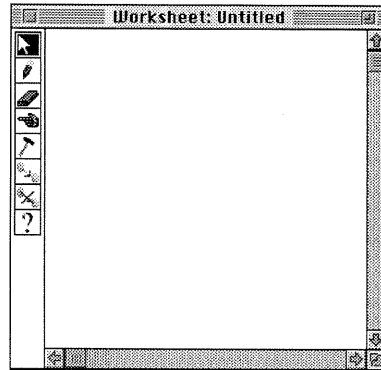










Figure 3. Empty Worksheet

On the left hand side of the worksheet you find a tool bar containing the tools described in the table below (from top to bottom). Tools are applied to agents by selecting the tool in the tool bar and clicking at the agent that the tool should be applied to.




Table 1. The Worksheet Tool Bar

Tool	Name	What it does
	Point	select and drag agents
	Draw	drop an agent based on the selection in the gallery
	Erase	remove an agent
	Operate-on	a generic tool that can be used for custom specific purposes
	Whack	a generic tool that can be used for custom specific purposes
	Link	create a link between a pair of agents
	Unlink	delete link between agents
	Get-Info	get information

Power Users: If you are in pointing mode (pointer cursor) you can use short cuts for frequent operations:

- command click -> draw
- shift command click -> erase

2.1.4. Drawing, Moving and Erasing Agents

To draw into the worksheet pick the type of agent by selecting it in the gallery. Do not try to drag the selected agent from the gallery into the worksheet. Instead, go to the worksheet and select the draw tool, , and draw by clicking and dragging. Select the "fixed" particle agent and draw something like in the figure below. At any time you can change to tool to move misplaced agents with the point tool, , or you can get rid of unwanted agents using the erase tool, .

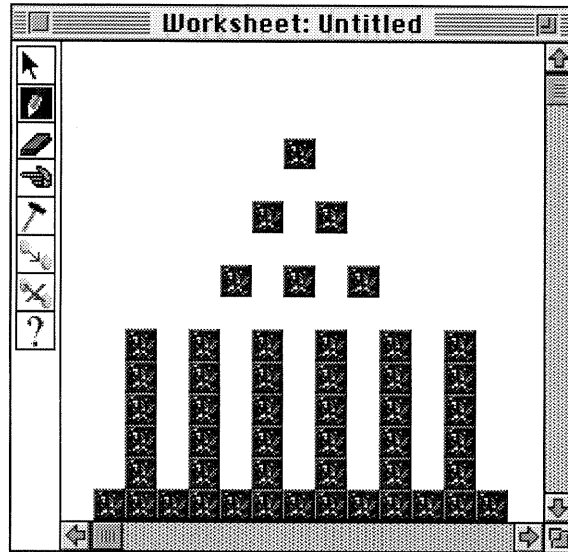


Figure 4. Fixed Particle Cascade

Introduce a movable particle agent by selecting it in the gallery and drawing it on top of the cascade:

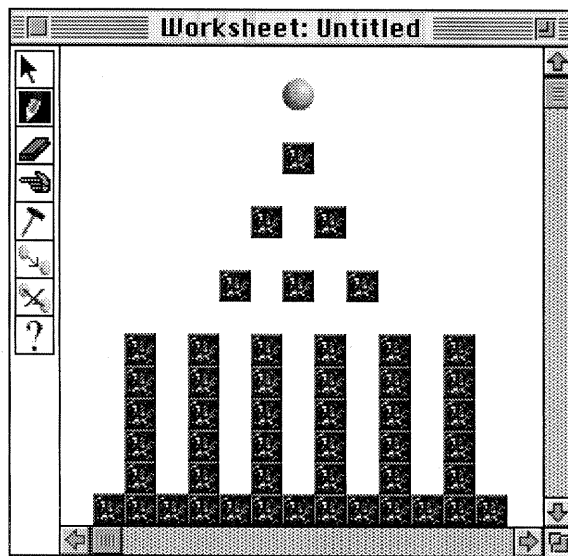


Figure 5. Movable Particle on top of Cascade

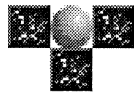
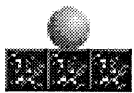
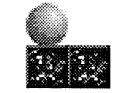
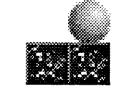
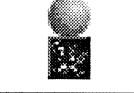
Nothing will happen so far. The movable particle agent are subclasses of active agents. They will remain suspended in mid air until you activate the worksheet.

2.1.5. Activating Worksheets

At any point in time agents can *react* to commands issued by users or other agents. However, in order to get the *autonomous* behavior of agents going you need to activate the worksheet. Note, only agent classes that are subclasses of active-agent can exhibit autonomous behavior. If your application does not feature such agents then there will be no point in activating the worksheet. On the contrary, activating the worksheet will just burn CPU cycles due to the overhead of the active-agent process scheduler.

Activate the worksheet with the **Worksheet > Run** menu command. The movable particle agents start to drop according to the following rules:

Table 2. Behavior of Movable Agents

Situation	behavior of movable particle agent
	stays
	stays
	drops to the left
	drops to the right
	drop randomly either left or right

Multiple movable particle agents can be dropped using the draw tool. After a while you will notice an approximation of a normal distribution manifesting itself in the columns filled differently with movable particles.

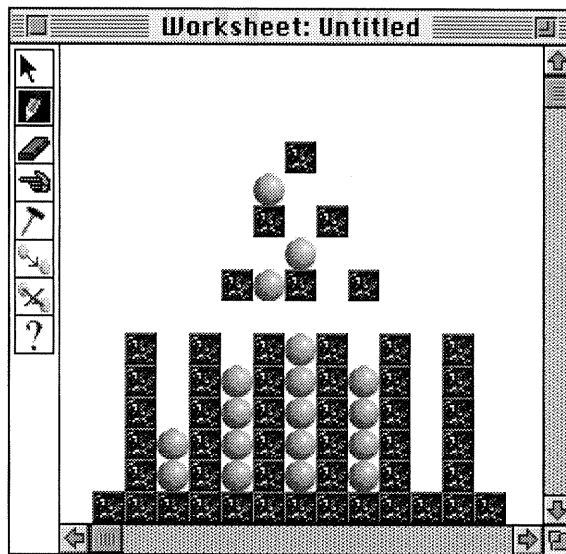



Figure 6. Particle World with Emerging Distribution

Even when the movable particle agents have settled they are still active. For instance, if you break out a piece of the wall consisting of fixed particles, using , then the movable agents will drop further.

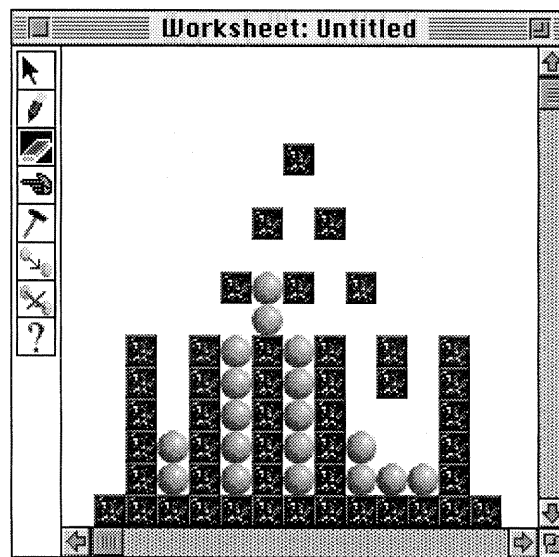


Figure 7. Turn down the Wall in the Particle World

2.1.6. Summary

The Particle World application, while being very simple, illustrates a number of essential principles and philosophies of the Agentsheets environment. First, an agentsheet can be viewed as a dynamic drawing media containing agents with autonomous behavior. The agents operate, conceptually, simultaneously. Interesting, possibly complex behavior typically emerges on a global level from the interaction of agents with simple individual behavior. Second, Agentsheets support the metaphor of *participatory theater* [??].

There is no rigid distinction between interacting with agents and running a simulation. Users can participate in the "play" excessively by direct manipulating agents, or they can let things happen by passively observing what the agents are doing based on the task defined. That is, they can change the play at any point in time by issuing commands to agents using mechanisms such as applying tools to agents.

2.2. CREATING AN AGENTSHEETS APPLICATION

A minimal Agentsheets application consist of a gallery specifying the *look of agents* and a class file containing source code describing the *behavior of agents*. Agentsheets supports an opportunistic design model. That is, look and behavior of agents do not have to be defined in any particular order.

2.2.1. Creating Galleries and Depictions

A typical start for an Agentsheets application is the creation of a gallery. Use the **File > New > Gallery** menu command to create a gallery. You will be asked about the depiction size. A size of 32 x 32 is the default agent size corresponding to the standard Macintosh icon size. Note that the selection of a size is quite a commitment since every agent in the gallery will be of this size and, furthermore, there is no simple way to later change that size.

Before you create new agent depictions set to depth of your screen to either 4, 16, or 256 color mode. The depth, i.e., the number of bits used to represent color information, of gallery depictions depends on the setting of your monitor. I recommend generally using 256 color mode except if you have very simple depictions that make only limited use of color.

Problem: Currently, Agentsheets does not support editing of 1 bit (black and white) depictions. However, they can be edited using third party icon editor applications such as ResEdit by Apple Computer Inc.

To add a new depiction to the gallery use the **Gallery > New...** menu command. It will place a new depiction into the gallery.

Shortcuts: Selected depictions in the gallery can be deleted using the <delete> key and renamed using the <return> key.

The following approaches, elaborated in the following sections, are used to edit depictions:

- Use the built-in icon editor
- Grab portions of the screen
- Use third-party icon editor applications such as ResEdit

Use the built-in icon editor

The most frequently used method to edit depictions is by using the built-in icon editor. Invoke the icon editor by double-clicking the depiction to be edited.

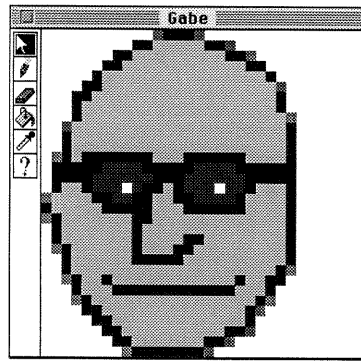








Figure 8. Icon Editor

The combination of icon editor and color palette are used to define color depictions. Use the **Applications > Color** Palette menu command to get a color palette. Both, the icon editor and the color palette are agentsheets. The icon editor features the following tools:

Table 3. The Worksheet Tool Bar

Tool	Name	What it does
	Point	move pixels (swap colors)
	Draw	draw a pixel in the color most recently selected in the color palette or defined by the color sampler tool
	Erase	change color of pixel to white
	Fill	fill area of identical color
	Sample	define drawing color by sampling the color
	Get-Info	print coordinate and color value of pixel

Grab portions of the screen

The **Gallery > Grab Screen 1:1** and the **Gallery > Grab Screen Any Size** menu commands are used to grab portions of the screen. This is helpful if you already have pictures that you like to convert into agent depictions. For instance, if you have 32 x 32 depictions use **Gallery > Grab Screen 1:1** to copy any Macintosh icon you are interested in into your gallery. Be aware of copyright issues.

The **Gallery > Grab Screen Any Size** menu command is very powerful if you use it in 256 color mode to create high quality miniatures through the use of color anti aliasing. In effect, it will improve screen resolution by compensating size reduction by adding color or gray scales. For example the picture of the lowercase character "a" in a 110 point size font reduced to a depiction of size 16 x 16 without anti aliasing would result in a hard to recognize blob.

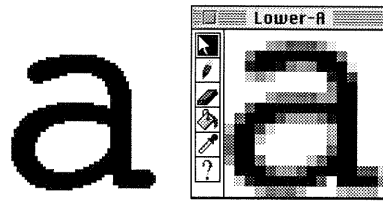


Figure 9. Dithering improves quality of shrinking pictures

A technique frequently used to create high-quality depictions is to draw them in a large format in a drawing package such as Canvas first and then use **Gallery > Grab Screen Any Size** to shrink them to the depiction size.

Use third-party icon editor applications

The open architecture of Agentsheets allows users to use their favorite icon editor tools. Agent depictions are stored as "cicn" resources. For instance, ResEdit by Apple has more powerful mechanisms to create icon masks. On the other hand, unlike the Agentsheets icon editor ResEdit cannot edit icons with less than 8 or more than 64 rows and columns. ResEdit and similar tools are very powerful but, at the same time, can be very dangerous unless you know exactly what you are doing.

2.2.2. Cloning Depictions

Galleries also support the incremental creation of the looks of agents. Cloning in Agentsheets is a visual inheritance mechanism allowing the creation of new looks based on old ones. The attributes of visual inheritance that can be added or overwritten are the color pixels of a depiction. Additionally, a cloning operation defines a spatial transformation from the source of the clone to the clone. Spatial transformations include simple operations such as rotations but also more sophisticated operations such as bending a depiction.

Start creating a simple "car drives on road" Agentsheets application by creating a gallery containing two depictions called road and car-red:

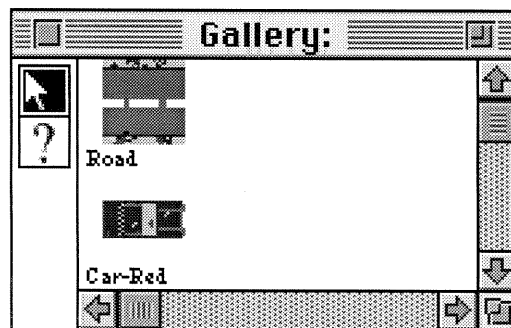


Figure 10. Road and Car Depiction

Instead of redrawing road depictions in order to reflect the diversity of road pieces necessary to draw a road system you can make use of cloning. Select the road depiction, and use the **Gallery > Clone...**

menu command. Type in the name of a new road piece, Road-n-s, into the cloning menu and select the Rotate 90 Orientation transformation.

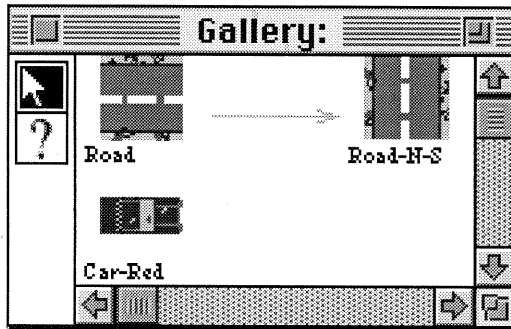


Figure 11. Gallery after adding Road Clone

The arrow between the Road and the Road-N-S depiction represents the rotation cloning relationship. Surprisingly, it is often necessary to apply more complex cloning operations such as bending. The Bend Down Right cloning operation will transform a straight piece of road into a curve segment:

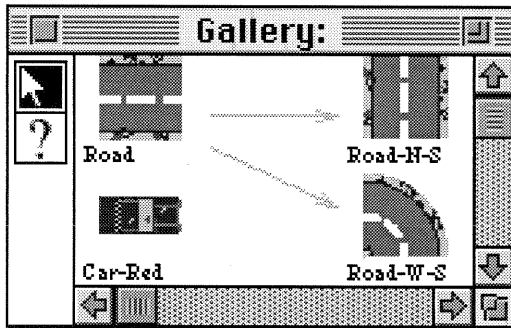


Figure 12. Bending a straight piece of road

Arrange the road pieces into something like in the figure below:

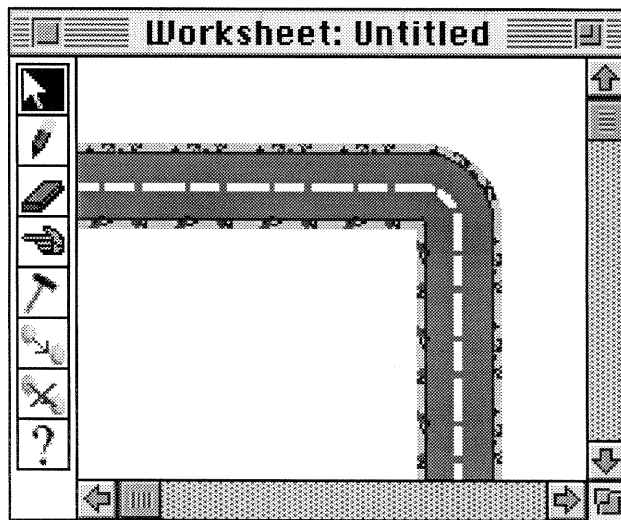


Figure 13. Simple Road System

You can always get some information about the agents by applying the get-info tool. If you do you will get information about the stack of agents located at one position in the agentsheet. For instance clicking at a road piece will return the following information representing the stack of agents located at cell position 1,2.

```
row: 1 column: 2 layer#: 0
a AGENT depiction: ROAD
a SELECTABLE-BACKGROUND-AGENT depiction: NIL
```


The first line indicates the position of the click, the second line describes the top most agent. It is an instance of the Agent class with a Road depiction. Below the agent looking like a road segment is an agent of type SELECTABLE-BACKGROUND-AGENT without a depiction. Worksheets are filled with SELECTABLE-BACKGROUND-AGENTS in order to provide basic interaction with an agentsheet cell. SELECTABLE-BACKGROUND-AGENTS cannot be erased.


2.2.3. Defining Behavior of Agents

Now you need to define the behavior of a car. Create a text file using the **File > New > Text File** menu command. Type in the following text (the numbers are only for reference, i.e., they are not part of the code).

```
1. (in-package :agents)
2. (d->c Car-Red CAR-AGENT)
3. (create-class CAR-AGENT
4.   "simple example of movable active agent"
5.   (sub-class-of ACTIVE-AGENT)
6.   (instance-methods
7.     (OPERATE-ON () "move car on road"
8.       (case (effect (0 1) 'depiction)
9.         (road (self 'move 0 1))
10.        (t (play-sound 'honk))))))
```

All the AgenTalk code should reside in the :agent package <1>.

Depictions have to linked to class names. Statements like <2> are declarations of links between depictions and class names. To draw into a worksheet means to create agent instances. According to <2> whenever a depiction called Car-Red is selected in the gallery, an instance of class CAR-AGENT is created with a  depiction.

Agent classes have to be created <3> - <10>. Documentation strings provided <4> can be accessed through latter without having to lookup the source file. Line <5> defines CAR-AGENT to be a subclass of ACTIVE-AGENT. For now we have a single method defined in <7> - <10> called OPERATE-ON. This is the name of the  tool. Consequently, when you apply the OPERATE-ON tool to a CAR-AGENT it will react according to the code provided in <8> - <10>; depending on the outcome of querying the depiction of the agent to the right <8> it will either move to the right if that depiction happens to be a road <9> or it will play a sound <10>.

After evaluating the file and minimizing the mask of the car depiction (**Gallery > Minimize Mask** menu command) you can draw a car on top of the road. Minimizing the mask will make the background of the car depiction transparent.

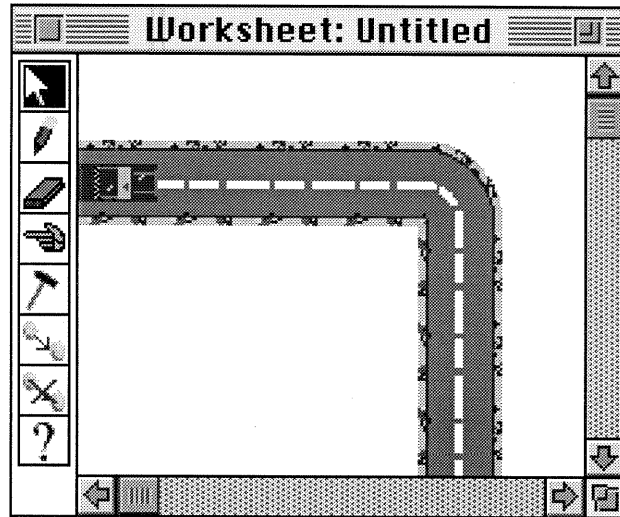





Figure 14. Car on Road

Applying the  tool will make the car move on position to the right as long it is on a straight piece of road. The too can be applied repetitively either by clicking at the car with the  too or by dragging the  tool into the car.

In a next step you can turn the car into a more autonomous unit by making use of the task scheduling mechanism. Change the class definition to:

```
(create-class CAR-AGENT
  "simple example of movable active agent"
  (sub-class-of ACTIVE-AGENT)
  (instance-methods
    (OPERATE-ON () "move car on road"
      (case (effect (0 1) 'depiction)
        (road (self 'move 0 1))
        (t (play-sound 'honk))))
    (FOREGROUND-TASKS () (self 'operate-on))))
```

Activating a worksheet will force the agent scheduler to send periodic process messages to all instances of active-agents. In the case that the receiving agent is the front most agent it will receive FOREGROUND-TASKS messages. Otherwise, it will receive BACKGROUND-TASKS messages. In the example the FOREGROUND-TASKS message will invoke the operate-on message. If you active the worksheet the car will move autonomously to the end of the straight segment of road pieces.

At this point you have defined four different looks of agents. Already you can select the depictions and use them to draw agents into a worksheet. All the agents created will by of class Agent, the most generic agent class. The Agent class is the used as default class. That is everything you draw into the worksheet will be of that class unless you specify otherwise.

Probably the easiest way to illustrate how behavior of agents is defined is by looking at a simple existing application; the particle world. The listing below is the complete class file of the particle world application:

1. (in-package :agents)
2. (d->c movable MOVABLE-PARTICLE-AGENT)


- E a r l y D r a f t -

```
3. (create-class MOVABLE-PARTICLE-AGENT
4.   (sub-class-of ACTIVE-AGENT)
5.   (instance-methods
6.     (MOVE (DR DC)
7.       (super 'move DR DC)
8.       (unless (effect (1 0) 'instance-of 'background-agent) (effect (1 0) 'impact)))
9.     (IMPACT C) (play-sound 'bit))
10.  (BACKGROUND-TASKS C)
11.  (if (effect (1 0) 'instance-of 'background-agent)
12.      (self 'move 1 0)
13.      (let ((Direction (nth (random 2) '(1 -1))))
14.          (and (effect (0 Direction) 'instance-of 'background-agent)
15.               (effect (1 Direction) 'instance-of 'background-agent)
16.               (self 'move 1 Direction))))))

17. (d->c fixed FIXED-PARTICLE-AGENT)

18. (create-class FIXED-PARTICLE-AGENT
19.   (sub-class-of AGENT)
20.   (instance-methods
21.     (IMPACT C) (play-sound 'bit))))
```

Lisp code lives in packages. Class files have to be in the :agents package <1>.

Depictions have to be linked to class names. Statements like <2> and <17> are declarations of links between depictions and class names. To draw into a worksheet means to create agent instances. According to <2>, for example, whenever a depiction called `movable` is selected in the gallery, an instance of class `MOVABLE-PARTICLE-AGENT` is created with a  depiction.

Agent classes have to be created. <3> and <18> are the headers of class definitions. <3>-<16> defines the `MOVABLE-PARTICLE-AGENT` class. `MOVABLE-PARTICLE-AGENT` is a subclass of `ACTIVE-AGENT` <4>. Activating a worksheet will force the agent scheduler to send periodic process messages to all instances of active-agents. In case that the receiving agent is the front most agent it will receive `BACKGROUND-TASKS` messages. Otherwise, it will receive `BACKGROUND-TASKS` messages.

3. THE AGENTSHEETS ENVIRONMENT

3.1. IMPORTANT FRED COMMANDS

Enter	ED-EVAL-OR-COMPILE-CURRENT-SEXP
Help	ED-INSPECT-CURRENT-SEXP
Tab	ED-INDENT-FOR-LISP
c-Space	COMPLETE-SYMBOL
m-.	ED-EDIT-DEFINITION
c-x c-d	ED-GET-DOCUMENTATION (for functions and CLOS methods)
c-x c-o	FRED-GET-METHOD-DOCUMENTATION (for OPUS methods)
c-x c-m	ED-MACROEXPAND-CURRENT-SEXP
c-x c-t	TIME-OF-SEXP
c-ForwardArrow	ED-FORWARD-SEXP
c-BackArrow	ED-BACKWARD-SEXP
c-m-a	ED-START-TOP-LEVEL-SEXP
c-m-e	ED-END-TOP-LEVEL-SEXP

3.2. SYMBOL COMPLETION

Symbol completion is a helpful programming aid to complete symbols that have only been partially been typed. The symbol completer can search for feasible completion of the string of characters that has already been typed by a user into an existing symbol in the Lisp system. The completion mechanism features:

- *Simple one key operation:* Since most users do not like to remember many different completion functions, this completion package combines several completion strategies into one function (using a cascading filter scheme).
 - 1) prefix search in window package
 - 2) prefix search in all packages
 - 3) substring search in window package
 - 4) substring search in all packages

- E a r l y D r a f t -

If no symbol is found the next strategy is employed. If only one symbol is found it will be used as completion. If multiple symbols are found then either the symbol is completed as far as possible or a menu is offered.

- *Partial Completion* (sounds like a contradiction): If at any stage in the search there is more than one interned symbol matching what you typed so far and if the common prefix of these symbols is longer than what is typed so far you will get this common prefix.
- *Preserves Case*: The completion will assume the same case of the string typed so far (lower case, upper case, or capitalized), e.g., "*Apple-" gets completed to "*Apple-Menu*"
- *Works also in Dialog Boxes*: the completer work with any kind of text entry not just in FRED buffers.
- *It's Small*: more comments than code ;-)

Examples:

sense-<symbol complete> will return the list of all agent sensors

-selection<symbol complete> will return the list of functions dealing with selections

4. REFERENCE MANUAL

4.1. MENU COMMANDS

The File, Edit, Eval, Tools and Windows menus are extensions of the original MCL menus. The Gallery, Worksheet, and Applications menus are completely Agentsheets specific.

4.1.1. The File Menu

New ▶

Worksheet	Create a new worksheet
Gallery	Create a new gallery
Text File	MCL: Create a new text file (FRED buffer)

Open ▶

Worksheet	Open existing worksheet
Gallery	Open existing gallery
Text File	Open existing text file

Open Selection Use current selection as filename to open text file

Close Close window (worksheet, gallery, text file)

Save Save window (worksheet, gallery, text file)

Save As... Like Save but name file first (worksheet, gallery, text file)

Save Copy As... Save copy of text file

Revert... Revert edit operation

Page Setup... Define printing parameters (page size, orientation, ...)

Print... Print window

Quit... Quit Agentsheets

4.1.2. The Edit Menu

Undo <operation>	Undo <Operation> (text file)
Undo More	
Cut	Delete selection and copy selection to clipboard
Copy	Copy selection to clipboard
Paste	Copy clipboard to insertion point
Clear	Delete selection without side effect to clipboard
Select All	Select all selectable objects in window
Search	Search for substring in window

4.1.3. The Eval Menu

Eval Selection	Evaluate selection
Eval Buffer	Evaluate entire FRED buffer contents
Load...	Load Lisp source file
Compile File...	Compile Lisp source file
Abort	Interrupt evaluation with no continuation possible
Break	Interrupt evaluation with continuation possible
Continue	Continue evaluation from break or error
Restarts	Get restart menu

4.1.4. The Tools Menu

Apropos	Search for related symbols
Documentation...	Access documentation strings
Edit Definition...	Find the definition of functions, macros, or variables
Inspect	Pop up inspector central
List Definitions	Create Fred function navigation window
Search File	Search a directory with text file for a substring
Search Systems...	Search all files of a system for a substring

Backtrace	Pop up stack navigation window
Fred Commands	List all Fred commands
Listener Commands	List all Listener commands
Print Options...	Dialog box to edit Lisp print parameters
Environment...	Dialog box to edit Lisp environment parameters

4.1.5. The Windows Menu

This menu holds the list of all open windows.

4.1.6. The Gallery Menu

New...	Create a new depiction
Clone...	Clone depiction
Reclone	Reclone depiction
Minimize Mask	Create icon mask ; assume pixel in upper left corner is the background color. All pixel with color other than background color become part of the mask.
Rename	Change name of depiction. Shortcut: <return> key
Delete	Delete depiction. Shortcut <delete> key
Grab Screen 1:1	Grab screen region of identical size to depiction and copy content into depiction
Grab Screen Any Size	Grab screen region of arbitrary size. Uses anti aliasing to create high quality color miniatures

Views

End User View	In end user view depictions can only be selected for drawing. If you are in designer view this command will switch to end-user view. If you are already in end-user view then this will clean up the depictions.
Designer View	Designer view enables additional depiction editing operations. In designer view all depictions and cloning relationships are shown.

Define End-User Depictions

The current selection of depictions will become the set of end-user depictions. Only these depictions will be visible in the end-user view.

Select End-User Depictions

Select all depictions that are visible in end-user view. This is typically used to change the set of end-user depictions; select end-user depictions, adjust selection (add/remove depictions) and define end-user depictions again.

Edit Behavior

Find definition of behavior of selected depiction

Class File...

Create a link to the class file, the Lisp source file defining the behavior of the agents. This file, or a compiled version of that file, will be loaded when the gallery gets loaded.

4.1.7. The Worksheet Menu

Run

Activate current worksheet; start agent scheduler

Stop

Deactivate current worksheet; stop agent scheduler

Link

Link two selected agents with child link. The color of the link is defined by the `*Default-Link-Color*` variable.

Link (Palette Color)

Link two selected agents with child link. The color of the link is defined by the `*Color-Palette-Value*` variable which, in turn, can be defined by selecting a color with the color palette.

Both, `*Default-Link-Color*` and `*Color-Palette-Value*` can be redefined by the user with custom color values. MCL/Agentsheets also includes a set of variables with predefined color values: `*Black-Color*` `*Blue-Color*` `*Brown-Color*` `*Dark-Gray-Color*` `*Dark-Green-Color*` `*Default-Link-Color*` `*Gray-Color*` `*Green-Color*` `*Light-Blue-Color*` `*Light-Gray-Color*` `*Orange-Color*` `*Pink-Color*` `*Purple-Color*` `*Red-Color*` `*Tan-Color*` `*White-Color*` `*Yellow-Color*`.

4.1.8. The Applications Menu

Demo Navigator

Load a demo file that is used to present a list of Agentsheets applications in quick succession.

Color Palette

Pop up a color palette to select a color. The color defined by the color palette is used by the depiction editor and the **Worksheet > Link (Palette Color)** menu command. The selected color can be accessed through the `*Color-Palette-Value*` variable.

Class Grapher

Pop up the class grapher. Classes can be inspected more closely by double clicking nodes in the graph.

Documenter

Create a documentation of the entire class system by listing all classes currently loaded, describing their methods and variable including documentation strings provided by the creators of the classes.

Othello

The Othello game.

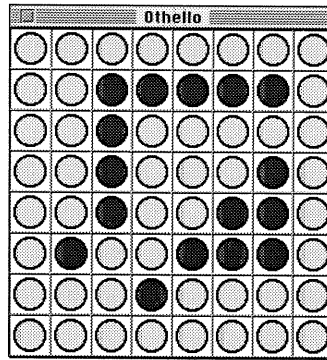


Figure 15. Othello Game

Tension Releaser

A game that was crucial during the last period of writing my dissertation. It also illustrates the use of Agentsheets screen animation not confined to windows.

4.2. AGENTALK

AgentTalk is the language and mechanism facilitating the communication between agents.

4.2.1. Interaction between Users and Agents

A system consisting of Agentsheets and a user can be viewed as distributed cognition [??] in which a typically small number of users interact with a typically very large number of agents. The combination of user and agents become a new entity. Two kinds of communications can be distinguished: agent-agent communication and user-agent communication. Because communication is essential to the construction paradigm, it is worthwhile to elaborate the different means of communication.

Agent-Agent Interaction

Agents communicate with each other by sending messages:

- *Non-Spatial Communication (Verbal)*: this is the traditional object-oriented way; a message is sent by some sender object executing a message-sending expression denoting a receiver object, a message name, and possibly some message arguments. The important point here is that the act of message sending is not based on any special features of either sender or receiver object.
- *Spatial Communication*: The act of sending a message makes use of the spatial organization of the container containing the objects involved in the communication. In the case of Agentsheets, the spatial communication makes use of the grid structure of agentsheets or uses the linking facility of agents.

The rest of this section describes the spatial communication mechanisms in Agentsheets in some detail. Agents can communicate with each other through space using different approaches of addressing each

other. The most typical means of referring to some other agent is by using relative grid coordinates (Figure 2-10).

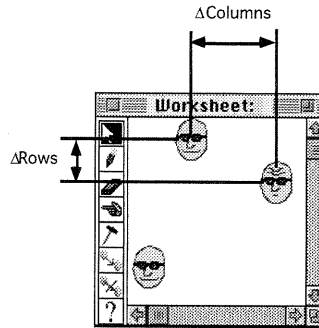


Figure 16: Relative References

In Figure 2-11 the uppermost agent refers to the rightmost agent through a relative coordinate (DRows=1, DColumns=2). The corresponding AgenTalk primitive is:

(effect (DRows DColumns) Message)

Agents can also refer to other agents using absolute coordinates (Figure 2-11).

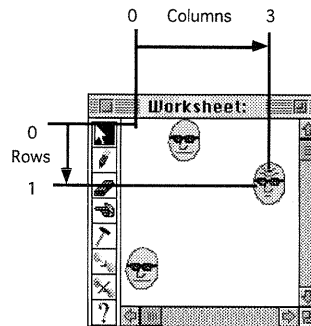


Figure 2-11: Absolute Reference

The corresponding AgenTalk primitive is:

(effect-absolute (Row Column) Message)

For pseudo-spatial relations, links can be used (Figure 2-12).

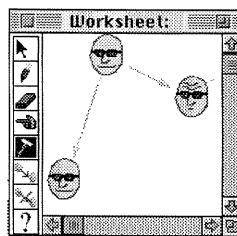


Figure 2-12: Link Reference

Links are used in situations in which a communication channel should be preserved if the message sending agent or the receiving agent get moved. Moving either agent will also update the position of the link.

Links are typed. The type of a link is used to select links through which messages are sent. If multiple links are of the same type, then the message gets broadcast through all these links. The result returned to the sender of the message is the concatenation of all the results returned by message receivers.

Messages can be sent in either direction, i.e., forward or reverse, through links:

(effect-link type message)
(effect-reverse-link type message)

All the previous communication mechanisms are used for agent-agent communication. In some cases it is necessary for agents to communicate with their containers, the agentsheets.

(effect-sheet message)

sends a message from an agent to the agentsheet containing it.

More sophisticated mechanisms allow agents that are located in detached agentsheets to communicate with each other. The agentsheet may even reside on separate computers that are connected over a network.

User-Agent Interaction

Users communicate with agents through messages. The messages need to be translated by input and output devices. A user invokes messages by operating an input device such as the mouse or the keyboard. Similar to QUICK [??], a user-interface design kit, Agentsheets supports a variety of method invocation techniques:



<user poll: does this need elaboration?: yes no don't care>

- ***Click Mouse On Agent:*** Single and double mouse clicks on an agent can invoke messages. The dispatching of messages can depend on modifier keys (shift, control, option, command) to distinguish different intentions of a user. Example: double clicking an agent in the agent gallery will expand the agent into its bitmap that can be edited.


AgentTalk messages: SENSE-CLICK, SENSE-COMMAND-CLICK, SENSE-CONTROL-CLICK, SENSE-DOUBLE-CLICK, SENSE-OPTION-CLICK, SENSE-SHIFT-CLICK, SENSE-SHIFT-COMMAND-CLICK, SENSE-SHIFT-CONTROL-CLICK, SENSE-SHIFT-OPTION-CLICK

- ***Drag Mouse Onto Agent:*** Dragging the mouse cursor onto an agent can invoke messages. The dispatching of messages depends on modifier keys (shift, control, option, command). Example: Dragging the mouse with no modifier key is used to move agents from one place to another in an agentsheet.

AgentTalk messages: SENSE-DRAG-INTO, SENSE-COMMAND-DRAG-INTO, SENSE-CONTROL-DRAG-INTO, SENSE-OPTION-DRAG-INTO, SENSE-SHIFT-COMMAND-DRAG-INTO, SENSE-SHIFT-CONTROL-DRAG-INTO, SENSE-SHIFT-DRAG-INTO, SENSE-SHIFT-OPTION-DRAG-INTO, SENSE-FINISH-DRAGGING

- ***Apply Tool To Agent:*** Tools from the tool bar can be applied to agents. Agents, in turn, invoke messages. Designers can introduce new tools to the tool bar. Example: Applying the draw tool, , will create a new agent, whereas applying the eraser tool, , will delete an agent.

AgentTalk messages: The name of the tool is the message being sent to the agent when applying the tool to the agent.

- **Drag Tool Onto Agent:** Dragging a tool onto an agent can invoke messages. By default these messages are identical to the messages invoked when applying a tool to an agent. However, designers can overwrite the default causing a different message to be invoked when applying a tool to an agent than when dragging the same tool onto the agent. Example: Dragging the draw tool, , leaves a trace of agents in the agentsheet.

AgenTalk messages: DRAG-TOOL-INTO

- **Press Key:** An agent can become the keyboard focus that receives all the key press events. Pressing keys can invoke messages. Example: In the Agentsheets gallery pressing the <Delete> key will delete all selected agents.

AgenTalk message: SENSE-KEY <Char>

Agents, in turn, send messages to users via output devices making use of different interaction modalities. Beside the visual interaction modalities - for instance through dialog boxes or animation - Agentsheets includes acoustic interaction modalities. Depending on the nature of the interaction, the human-computer communication bandwidth can be increased with the careful addition of sound and speech. While in many Agentsheets applications sound and speech have been included by designers mainly for entertainment purposes, in other applications, such as the Voice Dialog Environment, the use of sound and speech is essential.

Equal Rights for Users and Agents

A simple but very important principle that reduces the accidental complexity of programming Agentsheets applications is the "equal rights for users and agents" policy with respect to communication with agents. Every message that can be sent to an agent from the user can also be sent by any agent including the receiving agent. That is, messages sent to an agent by a user, for instance, also be sent by some other agent. The advantage of this equal rights policy is that agents can mimic user actions such as clicking at some agent or applying a tool to some agent, which greatly simplifies the creation of higher-level interaction mechanisms such as a "programming by example" extension.

4.2.2. The Agent Environment

Agent methods are invoked in a special environment. Unless if you are a power user in need of low-level control you will never deal with the environment. The environment consists of a set of dynamic but not global variables:

<code>!\$agents\$!</code>	The Agents data structure: Array of Array of Array of List of Agent
<code>!\$window\$!</code>	MCL window
<code>!\$row\$! !\$column\$! !\$layer\$!</code>	Agent cube coordinate
<code>!\$x\$! !\$y\$!</code>	Reference point: upper left coordinate of cell boundary (in pixel)
<code>!\$cell-width\$! !\$cell-height\$!</code>	Cell size (in pixel)
<code>!\$galleries\$!</code>	Every Agentsheet layer is associated with a gallery: Array of Color-Gallery-Agentsheet

4.2.3. Broadcasting messages to Active Agents

```
;;-*- Mode: Lisp; Package: AGENTS -*-
;; E X P E R I M E N T A L
;; V 1.0 8/22/93 by Alex Repenning
;; last update: 1/24/94

(defmethod BROADCAST-ACTIVE-AGENTS (Class-Name Selector &rest Arguments)
  (instance-method-of AGENTSHEET)
  (with-environment (0 0)
    (setq !$layer$! 0)
    (catch :exit-broadcast-active-agents
      (dolist (Agent Active-Agents)
        (when (or (and Class-Name (aim Agent 'instance-of Class-Name))
                  (null Class-Name))
          ;; modified expansion of EFFECT-ACTIVE-AGENT
          (locally
            (declare (special !$agents$! !$row$! !$column$! !$x$! !$y$! !$cell-width$! !$cell-height$!))
            (let ((Row (aim Agent 'row)) (Column (aim Agent 'column)))
              (let ((!$x$! (+ !$x$! (* (- Column !$column$!) !$cell-width$!))
                    (!$y$! (+ !$y$! (* (- Row !$row$!) !$cell-height$!))
                          (!$row$! Row)
                          (!$column$! Column))
                    (declare (special !$x$! !$y$! !$row$! !$column$!))
                    (apply-instance-message Agent Selector Arguments))))))))))

#! Examples

(defvar AS *Active-Window*)

(aim AS 'broadcast-active-agents nil 'flash) ; all active agents flash
(aim AS 'broadcast-active-agents 'vehicle-agent 'flash) ; all vehicles
(aim AS 'broadcast-active-agents 'red-car-class 'flash) ; all red cars
!#
```

4.3. DIALOG BOXES

A large number of dialog boxes are covered with the `user-query` function that allows you to create dialog boxes on the fly. That is, with `user-query` you do not have to design dialog boxes in advance. Instead, you can *compute* what should go into your dialog boxes.

```
USER-QUERY Query-Title Queries &key Window-Width [function]
```

```
in: Query-Title {string},
    Queries {list of list: <Name> [[<Default-Value>] <Type>]}.
out: Values {list of: {t}.
Query the user for an input.
<Type> can be nil -> editable-text,
      :boolean -> checkbox,
      :menu-set <list-of-values> -> pop-up-menu,
      :set <list-of-values> -> pop-up-menu (same as :menu-set for backward compatability),
      or :button-set <list-of-values> -> radio-button-cluster.
```

Example: the following call to `user-query`

```
(user-query "Bla"  
'(("number of years for PhD:" 6)  
  ("how many internships:" 2)  
  ("in CS" t :boolean)  
  ("with CogSci certificate" t :boolean)  
  ("residency:" "out state" :menu-set ("in state" "out state"))  
  ("foobar:" blee :menu-set (baz blee))  
  (" nil :line)  
  ("thesis option:" "Plan B" :button-set ("Plan A" "Plan B"))  
  ("buttons:" button2 :button-set (button1 button2 button3)))  
:window-width 300)
```

will create a dilalog box:

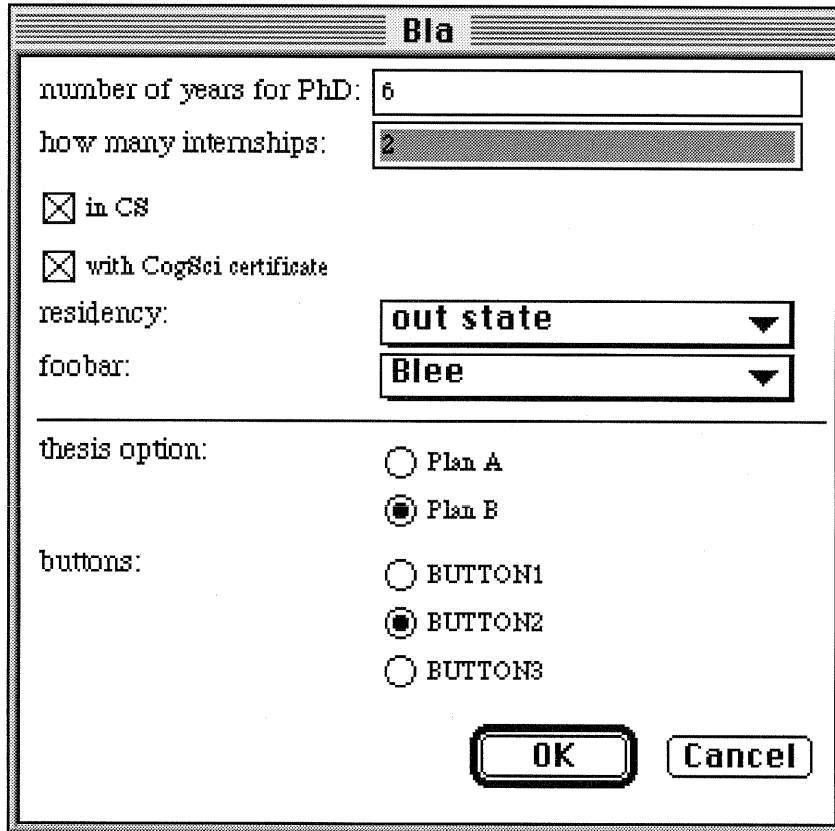


Figure 17. Generated Dialog Box

4.4. AGENTS DRAWING AND ERASING AGENTS

While it is typically up to the user to draw and erase agents in a worksheet there also can be cases when agents have to draw and erase other agents.

Erasing is the simplest operation. All you have to do is to send that agent an ERASE message. In other words you send the same message as it would be send if you apply the ERASE tool.

To draw is slightly more complicated; just sending a DRAW message is probably not what you need to do. DRAW would just draw an agent based on the current selection in the galley by invoking the following fragment of code:

```
(defmethod DRAW ()
  (instance-method-of AGENT) "
  Push agent. Every depiction in the gallery has an associated class.
  An instance is created of this class and pushed onto the location of
  the click."
  (let ((Selected-Depiction (self 'sense-selected-gallery-depiction)))
    (when Selected-Depiction
      (let ((Agent (create-anonymous-instance
                    (depiction->class-name Selected-Depiction)))
            (effect (0 0) 'push-agent Agent Selected-Depiction)
            (effect (0 0) 'init))))))
```

If you like to have control over the class and the depiction of the agent to be drawn use something like:

```
(defmethod AGENT-CREATING-METHOD ()
  (instance-method-of MY-AGENT)
  (let ((Agent (create-anonymous-instance <what-class?>)))
    (effect (<dx> <dy>) 'push-agent Agent <what-depiction?>)
    (effect (<dx> <dy>) 'init)))
```

4.5. GENERIC DEPICTIONS: ADDING FLEXIBILITY TO THE TASK OF DRAWING

<connect this section with the road example to illustrate how one single end user view can expand into a large set of designer views>

4.6. BROWSING AGENT CLASSES

To locate useful functionality use the describe method that is featured by each Agent (and object).

```
DESCRIBE (&Key Stream (Variable *Describe-Instance-Variable*) (Method *Describe-Instance-Method*) (Instance *Describe-Instance-Instance*) (Class *Describe-Instance-Class*) (Opus::Own *Describe-Instance-Own*) (Opus::Inherited *Describe-Instance-Inherited*) (Opus::Generic *Describe-Instance-Generic*) (Opus::Specific *Describe-Instance-Specific*) (Opus::Unique *Describe-Instance-Unique*) (Instance-Of *Describe-Instance-Instance-Of*))
```

in: &key ...

Print out a description of the object.

```
(value-agent 'describe :instance t :inherited nil)
```

Class Object: VALUE-AGENT

Can be used as a spreadsheet cell carrying a value.

Represents this value textually.

Super Class of: TEXT-NODE-AGENT

Sub Class of: SELECTABLE-AGENT

Instance Methods:

Own:

From Class: VALUE-AGENT

Generic Methods:

SENSE-OPTION-CLICK ()

Interactively modify (inspect) the value of the agent.

VALUE (&Optional New-Value)

in: &optional New-Value {t}.

Access value associated with agent.

Specific Methods:

SENSE-DRAW-REQUEST ()

Instance Variables:

Own:

From Class: VALUE-AGENT

PRINT-NAME initform: 0

VALUE initform: 0 Value of agent

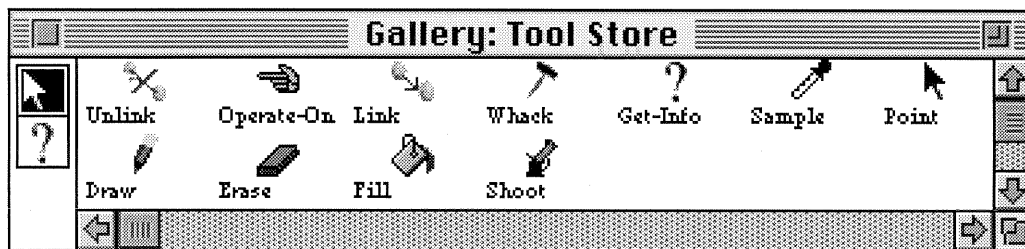
5. POWER USERS

5.1. SPECIALIZING THE DEMO NAVIGATOR

5.2. ADDING YOUR OWN SOUNDS

5.3. ADDING YOUR OWN TOOLS

There is one special gallery that gets loaded at startup time. The set of tools can be extended by designers to reflect domain-oriented semantics.



5.4. MODIFYING DEPICTIONS WITH 3RD PARTY TOOLS

5.5. CREATING YOUR OWN WORKSHEET CLASSES AND FILE TYPES

5.6. NEW VERSION

Auto Methods

Depiction Selection by Keyboard

6. APPENDIX A: AGENT CLASSES

A current version of the following list can be created with the

(agent 'doc :inherited nil :instance t)

lisp command

```
OBJECTLIBRARY
```

```
=====
```

```
date: 24.1 1994
```

```
time: 19:9
```

```
AGENT
```

```
SELECTABLE-AGENT
```

```
  COLOR-PALETTE-SHEET-AGENT inherits from: ICON-EDITOR-COLOR-PIXEL-AGENT, SELECTABLE-AGENT
```

```
  VALUE-AGENT
```

```
    TEXT-NODE-AGENT inherits from: VALUE-AGENT, LINKABLE-AGENT
```

```
  CHARACTER-AGENT
```

```
  LINKABLE-AGENT
```

```
    COLOR-GALLERY-AGENT inherits from: LINKABLE-AGENT, LOCATION-AWARE-AGENT
```

```
  GROUPABLE-AGENT
```

```
  HYPER-AGENT
```

```
  BACKGROUND-AGENT
```

```
    SELECTABLE-BACKGROUND-AGENT
```

```
  COLOR-PIXEL-AGENT
```

```
    ICON-EDITOR-COLOR-PIXEL-AGENT
```

```
  LOCATION-AWARE-AGENT
```

```
  ACTIVE-AGENT
```

6.1. CLASS: AGENT

Class Object: AGENT

An agent has sensors and effectors to interact with the world he lives in.

Super Class of: SELECTABLE-AGENT, HYPER-AGENT, BACKGROUND-AGENT, COLOR-PIXEL-AGENT, LOCATION-AWARE-AGENT

Sub Class of: OBJECT

Class Methods:

Own:

From Class: AGENT

Generic Methods:

DISABLE-SOUND (Enable)

Instance Methods:

Own:

From Class: AGENT

Generic Methods:

AGENT-BELOW ()

out: Agent {Agent}.

Return the second agent on the stack.

ALL-AGENTS ()

out: Agents {list of: {Agent-Instance}}.

Return the list of agents at this cell.

BRING-AGENT-TO-FRONT (Depiction &Optional (Update Nil))

in: Depiction {symbol}, &optional Update {boolean} default nil.

Bring the agent matching <Depiction> to the top of the agent stack.

The FIRST agent found by searching from top to bottom is chosen.

CIRCLE ()

Circling an agent may be used for selecting a set of agents

COVER (Agent-On-Top His-Depiction)

in: Agent-on-Top {Agent-Instance}, His-Depiction {symbol}.

The agent being covered receives this message containing the covering agent with his depiction.

DEPICTION (&Optional New-Depiction &Key No-Window-Update)

in: &optional Depiction {symbol},
&key No-Window-Update {boolean}.

out: Depiction {symbol}.

Access to depiction of agent.

Unless <No-Window-Update> is set to a non-nil value the agentsheet containing the agent will be updated.

DEPICTION-BELOW ()

out: Depiction {symbol}.

Return the depiction of the second agent on the stack.

DRAG-TOOL-INTO (Tool From-Delta-Row From-Delta-Column)

in: Tool {symbol},

From-Delta-Row From-Delta-Column {fixnum}.

<Tool> gets dragged into here from <From-Delta-Row> <From-Delta-Column>

DRAW ()

DRAW-A-DEPICTION (A-Depiction &Optional (Mode :Srcor))

DRAW-CELL ()

Draw all agents of a cell without regarding the layer-dispatch function.

DRAW-LAYER ()

Draw all agents within the receiving layer without regarding the layer-dispatch function.

ERASE ()

FLASH ()

Flash agent by inverting it several times.

GET-INFO ()

Print information about this agent

MAKE-ME-THE-KEYBOARD-FOCUS-CELL ()

The cell containing the sender of this message will receive all

- E a r l y D r a f t -

future SENSE-KEY messages.

NO-MATCHING-METHOD (Selector &Rest Arguments)
in: Selector {symbol}, &rest Arguments {t}.
Default agent error handling: flash agent and notify user
with message selector and arguments.

OPERATE-ON ()
Operate on the agent

POP-AGENT ()
out: Agent {Agent-Instance}.
Pop an agent from the agent stack and return it.

PRINT ()
Print itself to printer view

PUSH-AGENT (Agent &Optional Depiction &Key Ignore-Covering No-Window-Update)
in: Agent {Agent-Instance} &optional Depiction {symbol},
&key Ignore-Covering {boolean}, No-Window-Update {boolean}.
Push the agent on top of the agent stack.
If <Ignore-Covering> is not set to a non-nil value then send a cover message
to the agent being covered.

RELEASE-MOUSE-BUTTON ()
Issued after the release of the mouse botton. Is also issued in
case of a finish-dragging event (precedes it).

REMOVE-AGENT (Agent &Optional (Update T))
in: Agent {Agent-Instance},
&optional Update {boolean} default t.
Remove <Agent> from stack.

SENSE-CLICK ()
React to the click of the mouse.

SENSE-COMMAND-CLICK ()
Push agent. Every depiction in the gallery has an associated class.
An instance is created of this class and pushed onto the location of
the click.

SENSE-COMMAND-DRAG-INTO (From-Delta-Row From-Delta-Column)
Push agent. Same as SENSE-COMMAND-CLICK

SENSE-CONTROL-CLICK ()
Typical use: Debugging
Default: show the classes and the depictions in a cell.

SENSE-CONTROL-DRAG-INTO (From-Delta-Row From-Delta-Column)

SENSE-DOUBLE-CLICK ()
If an agent gets clicked twice in a short time frame then
ADDITIONALLY to any sense-..-click message the agent receives this message
Typical use: elaboration, agent expansion, hyper agents.

SENSE-DRAG-INTO (From-Delta-Row From-Delta-Column)
Move the agent from the cell exited to the cell entered.

SENSE-FINISH-DRAGGING (From-Delta-Row From-Delta-Column)
Issued at time of releasing the mouse key(s) after dragging.

SENSE-KEY (Char)
in: Char {Char}.
The cell being the current keyboard focus receives this message
every time a key is typed.

SENSE-OPTION-CLICK ()
Typical use: interact with agent parameters, e.g., pop up a dialog box.

SENSE-OPTION-DRAG-INTO (From-Delta-Row From-Delta-Column)

SENSE-POSITION ()
out: Row-Column {list: <Row> <Column>}}.
Return the position of the agent.

SENSE-SELECTED-GALLERY-DEPICTION ()
out: Depiction {symbol}.
Return the name of the currently selected depiction in the gallery
associated with the receiving layer.
In case that multiple depictions are selected return the one selected first.
If no depiction is selected return nil.

SENSE-SHIFT-CLICK ()

SENSE-SHIFT-COMMAND-CLICK ()
Pop agent. Remove the agent on top of the stack.

- E a r l y D r a f t -

SENSE-SHIFT-COMMAND-DRAG-INTO (From-Delta-Row From-Delta-Column)
Pop agent. Same as SENSE-SHIFT-COMMAND-CLICK.

SENSE-SHIFT-CONTROL-CLICK ()

SENSE-SHIFT-CONTROL-DRAG-INTO (From-Delta-Row From-Delta-Column)

SENSE-SHIFT-DRAG-INTO (From-Delta-Row From-Delta-Column)

SENSE-SHIFT-OPTION-CLICK ()

SENSE-SHIFT-OPTION-DRAG-INTO (From-Delta-Row From-Delta-Column)

SENSE-UPDATE-REQUEST ()
Check dependent agents and react to them.

TOP-AGENT ()
out: Agent {Agent-Instance}.
Return agent from the agent stack without modifying the stack.

Specific Methods:

ERASE-CELL ()
Erase the rectangular area of the cell.

FILL-CELL ()
Erase the rectangular area of the cell.

INVERT-CELL ()
Invert the rectangular area of the cell.

MOVE (Delta-Row Delta-Column &Key Wrap-Row Wrap-Column (Update-Source T) (Update-Destination T))
in: Delta-Row Delta-Column {fixnum},
&key Wrap-Row Wrap-Column {fixnum},
Update-Source Update-Destination {boolean} default t.
Pop the agent and push it to the stack of agents at the
new place denoted by <Delta-Row> <Delta-Column>.
Specifying <Wrap-Row> and/or <Wrap-Column> will confine the agent
into the range 0..Wrap - 1

PLAY-SOUND (Name)
in: Name {symbol}.
Play a sound called <Name>
The function list-of-sounds returns the names of all loaded sounds

SENSE-DRAW-REQUEST (&Optional (Mode :Srcor))
in: &optional Mode {keyword} default :srcor
Causes the agent to draw himself

Unique Methods:

ADAPT-TO-NEIGHBOURS ()

DETERMINE-SPECIFIC-DEPICTION (My-Depiction)

GET-LINKS (&Optional Name)

SAVE (&Optional File)
in: &optional File {symbol}. Standard Output if omitted.
Saves agent type, depiction, and sheet position (row, column, layer).
Also saves all internal state information - instance and class variables.

SAVE-CLASS (&Optional File)
in: &optional File {symbol}. Standard Output if omitted.
Saves the class variables.

SAVE-INSTANCE (&Optional File)
in: &optional File {symbol}. Standard Output if omitted.
Saves agent type, depiction, and sheet position (row, column, layer).
Also saves all instance variables.

Class Variables:
Own:
From Class: AGENT
SOUND-ENABLED value: T {boolean} if nil don't play any sound

Instance Variables:
Own:
From Class: AGENT
DEPICTION initform: NIL symbol refering to gallery pictures

6.2. CLASS: SELECTABLE-AGENT

Class Object: SELECTABLE-AGENT

A SELECTABLE-AGENT acts similar to objects on the Macintosh desktop; SELECTABLE-AGENTS can be selected by clicking at them or the current selection can be adjusted with shift clicks. Multiple SELECTABLE-AGENTS are selected by circling them. Operations like cut, copy and paste are supported.

Super Class of: COLOR-PALETTE-SHEET-AGENT, VALUE-AGENT, CHARACTER-AGENT, LINKABLE-AGENT, GROUPABLE-AGENT

Sub Class of: AGENT

Instance Methods:

Own:

From Class: SELECTABLE-AGENT

Generic Methods:

CIRCLE () circling is identical to shift clicking

MOVE (Delta-Row Delta-Column &Key Wrap-Row Wrap-Column (Update-Source T) (Update-Destination T)) Extensio
Update selection.

SELECT () switch to selected mode

SENSE-CLICK ()

Select the agent, i.e., draw reversed.

SENSE-FINISH-DRAGGING (From-Delta-Row From-Delta-Column)

Update all the links leading to and leaving from agent.

SENSE-SHIFT-CLICK ()

Adjust selection. Toggle reversed/non-reversed.

UNSELECT () switch to unselected mode

Unique Methods:

SENSE-DRAW-REQUEST ()

Instance Variables:

Own:

From Class: SELECTABLE-AGENT

SELECTED initform: NIL {boolean}

6.3. CLASS: COLOR-PALETTE-SHEET-AGENT

Class Object: COLOR-PALETTE-SHEET-AGENT

Determine the value of *Color-Palette-Value*.

Only one COLOR-PALETTE-SHEET-AGENT can be selected at a time.

Super Class of:

Sub Class of: ICON-EDITOR-COLOR-PIXEL-AGENT, SELECTABLE-AGENT

Instance Methods:

Own:

From Class: COLOR-PALETTE-SHEET-AGENT

Generic Methods:

SELECT ()

Set *Color-Palette-Value* to RGB color value of selected agent

SENSE-CLICK ()

Assure that only receiver is marked as selected.

SENSE-COMMAND-CLICK ()

SENSE-DOUBLE-CLICK ()

Change color using color picker.

SENSE-DRAG-INTO (From-Delta-Row From-Delta-Column)

Same as clicking.

SENSE-SHIFT-COMMAND-CLICK ()

Specific Methods:

DRAW-SELECTION-FRAME ()

Draw an overlaying frame to indicate selection.

6.4. CLASS: VALUE-AGENT

Class Object: VALUE-AGENT

Can be used as a spreadsheet cell carrying a value.

Represents this value textually.

Super Class of: TEXT-NODE-AGENT

Sub Class of: SELECTABLE-AGENT

Instance Methods:

Own:

From Class: VALUE-AGENT

Generic Methods:

SENSE-OPTION-CLICK ()

Interactively modify (inspect) the value of the agent.

VALUE (&Optional New-Value)

in: &optional New-Value {t}.

Access value associated with agent.

Specific Methods:

SENSE-DRAW-REQUEST ()

Instance Variables:

Own:

From Class: VALUE-AGENT

PRINT-NAME initform: 0

VALUE initform: 0 Value of agent

6.5. CLASS: TEXT-NODE-AGENT

Class Object: TEXT-NODE-AGENT

Agent used in class browser.

Super Class of:

Sub Class of: VALUE-AGENT, LINKABLE-AGENT

Instance Methods:

Own:

From Class: TEXT-NODE-AGENT

Generic Methods:

 COLUMN (&Optional New-Column)

 LEVEL (&Optional New-Level)

 ROW (&Optional New-Row)

 SENSE-DOUBLE-CLICK ()

 This methods is missplaced as it it OPUS specific.

Instance Variables:

Own:

From Class: TEXT-NODE-AGENT

 LEVEL initform: NIL level in graph; level of root is 0

 ROW initform: NIL logical row in sheet

6.6. CLASS: CHARACTER-AGENT

Class Object: CHARACTER-AGENT
Super Class of:
Sub Class of: SELECTABLE-AGENT
Instance Methods:
Own:
From Class: CHARACTER-AGENT
Generic Methods:
CHARACTER (&Optional Char)
in: &optional Char {char}.
out: Char {char}.
Access the character represented.
SENSE-CLICK ()
SENSE-KEY (Char)
Unique Methods:
SENSE-DRAW-REQUEST (&Optional Mode)
Instance Variables:
Own:
From Class: CHARACTER-AGENT
CHARACTER initform:

6.7. CLASS: LINKABLE-AGENT

Class Object: LINKABLE-AGENT

Super Class of: COLOR-GALLERY-AGENT, TEXT-NODE-AGENT

Sub Class of: SELECTABLE-AGENT

Instance Methods:

Own:

From Class: LINKABLE-AGENT

Generic Methods:

GET-LINKS (&Optional Name)

in: &optional Name {Symbol}.

out: Links {list of: {Links}}.

If <Name> is supplied then return links with that name.

Otherwise return ALL links.

GET-REVERSE-LINKS (&Optional Name)

in: &optional Name {Symbol}.

out: Links {list of: {Links}}.

If <Name> is supplied then return links with that name.

Otherwise return ALL reverse links.

LINK () smart link

LINK-DESTINATION (Name Link)

in: Name {symbol}, Link {Link}.

Establish a new link. Receiver is the link ``destination''

LINK-SOURCE (Name Link)

in: Name {symbol}, Link {Link}.

Establish a new link. Receiver is the link ``source''

REMOVE-LINKS (&Optional Link)

in: &optional Link {Link}.

If <Link> is supplied then remove it otherwise remove ALL links.

REMOVE-REVERSE-LINKS (&Optional Link)

in: &optional Link {Link}.

If <Link> is supplied then remove it otherwise remove ALL reverse links.

SENSE-FINISH-DRAGGING (From-Delta-Row From-Delta-Column)

Update all the links leading to and leaving from agent.

UNLINK () smart unlink

Unique Methods:

SAVE-INSTANCE (&Optional File)

in: &optional File {symbol}. Standard Output if omitted.

Saves agent type, depiction, and sheet position (row, column, layer).

Also saves all instance variables.

Instance Variables:

Own:

From Class: LINKABLE-AGENT

LINKS initform: NIL {list of {Links}}

REVERSE-LINKS initform: NIL {list of {Links}}

6.8. CLASS: COLOR-GALLERY-AGENT

Class Object: COLOR-GALLERY-AGENT

A gallery agent is a part of a gallery agentsheet. Every gallery agent is in charge of maintaining its cloning relationship.

Super Class of:

Sub Class of: LINKABLE-AGENT, LOCATION-AWARE-AGENT

Instance Methods:

Own:

From Class: COLOR-GALLERY-AGENT

Generic Methods:

BACKUP-VP-DESIGNER-LOCATION ()

CHILDREN ()

out: Children {list of: {symbol}}.

Return the names of all children depictions.

CLONING-OPERATION (&Optional New-Cloning-Operation)

GET-INFO () Extension: Add icons depth information

LEVEL (&Optional New-Level)

POP-AGENT ()

Gallery agent should only be movable in VP designer view

RESTORE-VP-DESIGNER-LOCATION ()

ROW (&Optional New-Row)

SENSE-COMMAND-CLICK ()

SENSE-DOUBLE-CLICK ()

Edit Icon

SENSE-FINISH-DRAGGING (From-Delta-Row From-Delta-Column)

In the grapher nodes should not be stacked up

SENSE-SHIFT-COMMAND-CLICK ()

VALUE (&Optional New-Value)

Specific Methods:

CREATE-ICON-MASK ()

Mimimize icon mask by removing all the pixels reachable by paint starting to leak from coordinate 0, 0.

GRAB-SCREEN ()

Copy content of screen one-to-one to depiction from user solicited screen position.

GRAB-SCREEN-ANY-SIZE ()

Copy arbitrary-sized content of screen to depiction.

Solicit size and position from user.

RECLONE ()

Use the the cloning links and the cloning operator to reclone the Icon.

Unique Methods:

SENSE-DRAW-REQUEST ()

Instance Variables:

Own:

From Class: COLOR-GALLERY-AGENT

CLONING-OPERATION initform: NIL {lambda ({Icon}) -> {Icon}}

LEVEL initform: NIL level in graph; level of root is 0

ROW initform: NIL logical row in sheet

VP-COLUMN initform: NIL column in VP designer view

VP-ROW initform: NIL row in VP designer view

6.9. CLASS: GROUPABLE-AGENT

Class Object: GROUPABLE-AGENT

Multiple GROUPABLE-AGENTS can be united into a group.
Selecting one agent in a group will automatically also
select the other members of the same group and it will
raise them to the top of their stacks.

Only one group can be selected at a time.

Super Class of:

Sub Class of: SELECTABLE-AGENT

Instance Methods:

Own:

From Class: GROUPABLE-AGENT

Generic Methods:

SENSE-SHIFT-CLICK ()

Specific Methods:

SENSE-CLICK () select all group members

Instance Variables:

Own:

From Class: GROUPABLE-AGENT

GROUP-MEMBERS initform: NIL all the agents in the group

6.10. CLASS: HYPER-AGENT

Class Object: HYPER-AGENT

A hyper agent contains a link to another agentsheet.

Super Class of:

Sub Class of: AGENT

Instance Methods:

Own:

From Class: HYPER-AGENT

Generic Methods:

SENSE-OPTION-CLICK ○

Instance Variables:

Own:

From Class: HYPER-AGENT

AGENTSHEET initform: NIL {Agent-Instance}

6.11. CLASS: BACKGROUND-AGENT

Class Object: BACKGROUND-AGENT

The background agent sits in the back of every agentsheet layer. Without it the sheet has no way to react to things like clicks. The background agent can be shared among one agentsheet in order to reduce the number of agents required.
Super Class of: SELECTABLE-BACKGROUND-AGENT

Sub Class of: AGENT

Instance Methods:

Own:

From Class: BACKGROUND-AGENT

Generic Methods:

POP-AGENT ()

out: Agent {Agent-Instance}.

Poping the background agent would desensitize the cell of the agentsheet. Therefore, pop-agent just returns the background agent without removing it from the stack.

SENSE-DRAW-REQUEST ()

Unique Methods:

DRAW ()

SAVE (&Optional Filename)

6.12. CLASS: SELECTABLE-BACKGROUND-AGENT

Class Object: SELECTABLE-BACKGROUND-AGENT

The background agent sits in the back of every agentsheet layer. Without it the sheet has no way to react to things like clicks. The background agent can be shared among one agentsheet in order to reduce the number of agents required.

Super Class of:

Sub Class of: BACKGROUND-AGENT

Instance Methods:

Own:

From Class: SELECTABLE-BACKGROUND-AGENT

Generic Methods:

LINK () like click

SENSE-CLICK () reset selections

UNLINK () like click

6.13. CLASS: COLOR-PIXEL-AGENT

Class Object: COLOR-PIXEL-AGENT

A color pixel agent can be of any size. Its color is determined by the value of the instance variable color which can be accessed from other agents by the COLOR message.

Super Class of: ICON-EDITOR-COLOR-PIXEL-AGENT

Sub Class of: AGENT

Instance Methods:

Own:

From Class: COLOR-PIXEL-AGENT

Generic Methods:

COLOR (&Optional New-Color &Key (Update T))

in: &optional New-Color {t}, &key Update {boolean} default t.

out: Color {t}.

Access the color of an agent. Changing the color will update the screen.

Specific Methods:

SENSE-DRAW-REQUEST ()

fill the cell with the color of the agent

Instance Variables:

Own:

From Class: COLOR-PIXEL-AGENT

COLOR initform: *RED-COLOR*

6.14. CLASS: ICON-EDITOR-COLOR-PIXEL-AGENT

Class Object: ICON-EDITOR-COLOR-PIXEL-AGENT

Modify icon: change color, and fill.

Is typically shared.

Super Class of: COLOR-PALETTE-SHEET-AGENT

Sub Class of: COLOR-PIXEL-AGENT

Instance Methods:

Own:

From Class: ICON-EDITOR-COLOR-PIXEL-AGENT

Generic Methods:

CIRCLE ()

Color pixel of icon using the RGB value of *Color-Palette-Value*.

DRAW ()

Color pixel of icon using the RGB value of *Color-Palette-Value*.

ERASE () Change pixel color to white

FILL () fill with palette color

FILL-VALUE (Value)

Fill receiving agent AND ALL 4-adjacent agents with color defined in *Color-Palette-Value*.

GET-INFO () also print color

SAMPLE () pick up color

SENSE-CLICK ()

Color pixel of icon using the RGB value of *Color-Palette-Value*.

SENSE-COMMAND-CLICK ()

Start color filling.

SENSE-DRAG-INTO (D-Row D-Column)

SENSE-FINISH-DRAGGING (U V)

SENSE-OPTION-CLICK ()

Start circling.

SENSE-SHIFT-OPTION-CLICK () power user feature: show CTable of icon

Unique Methods:

SENSE-DRAW-REQUEST ()

6.15. CLASS: LOCATION-AWARE-AGENT

Class Object: LOCATION-AWARE-AGENT

These agent know their location within the grid.

Super Class of: COLOR-GALLERY-AGENT, ACTIVE-AGENT

Sub Class of: AGENT

Instance Methods:

Own:

From Class: LOCATION-AWARE-AGENT

Generic Methods:

COLUMN (&Optional New-Column)

INIT ()

MOVE (Delta-Row Delta-Column &Key Wrap-Row Wrap-Column (Update-Source T) (Update-Destination T))

ROW (&Optional New-Row)

SENSE-FINISH-DRAGGING (From-Delta-Row From-Delta-Column)

Issued at time of releasing the mouse key(s) after dragging.

Instance Variables:

Own:

From Class: LOCATION-AWARE-AGENT

COLUMN initform: 0 grid coordinate

ROW initform: 0 grid coordinate

6.16. CLASS: ACTIVE-AGENT

Class Object: ACTIVE-AGENT

Active agents take the initiative, i.e., they don't wait for user events.

If the simulation is running then active-agents receive periodical TASKS messages. TASKS methods can be used to implement animation etc.

Active agents are aware of their location. EFFECT-ACTIVE-AGENT can be used to send messages to other active agents without keeping track of their position.

Super Class of:

Sub Class of: LOCATION-AWARE-AGENT

Instance Methods:

Own:

From Class: ACTIVE-AGENT

Generic Methods:

INIT ()

SENSE-OPTION-CLICK ()

Debugging: print list of active agents in Agentsheet.

SENSE-SHIFT-COMMAND-CLICK ()

SENSE-SHIFT-COMMAND-DRAG-INTO (From-Delta-Row From-Delta-Column)

TASKS ()

This method gets called periodically when the simulation is running

Specific Methods:

BACKGROUND-TASKS ()

This method gets called periodically when the simulation is running
AND the agent is NOT on top of the stack.

BACKGROUND-TASKS ()

This method gets called periodically when the simulation is running
AND the agent is on top of the stack.

7. APPENDIX B: AGENTSHEET CLASSES

A current version of the following list can be created with the

(agentsheet 'doc :inherited nil :instance t)

lisp command

```
OBJECTLIBRARY  
=====
```

```
date: 24.1 1994
```

```
time: 19:11
```

```
AGENTSHEET inherits from: DOCUMENT, TOOL-WINDOW
```

```
LINK-AGENTSHEET
```

```
ICON-EDITOR-SHEET
```

```
COLOR-PALETTE-SHEET
```

```
GRAPHER-AGENTSHEET
```

```
COLOR-GALLERY-AGENTSHEET inherits from: COLOR-DEPICTION-STORAGE, GRAPHER-AGENTSHEET
```

```
HYPER-AGENTSHEET
```

7.1. CLASS: AGENTSHEET

Class Object: AGENTSHEET

An agentsheet is a cube structured container of agents.

Agents = array of Rows (adjustable)

Rows = array of Cells (adjustable)

Cells = array of Agent-Instance-Stacks

Agent-Instance-Stacks = list of Agent-Instances.

Super Class of: LINK-AGENTSHEET, HYPER-AGENTSHEET

Sub Class of: DOCUMENT, TOOL-WINDOW

Class Methods:

Own:

From Class: AGENTSHEET

Generic Methods:

NEW (&Key (Window-Show T) (Title "Untitled") (Vertical-Scrollbar T) (Horizontal-Scrollbar T) Explode-From (Background-Agent-Class-Name 'Selectable-Background-Agent))

in: &key Window-Show {boolean} default t,
Title {string} default "Untitled",
Vertical-Scrollbar {boolean} default t,
Horizontal-Scrollbar {boolean} default t,
Explode-From {list: <x1> <y1> <x2> <y2>} default nil,
Background-Agent-Class-Name {symbol} default 'selectable-background-agent.

Create a window.

Instance Methods:

Own:

From Class: AGENTSHEET

Generic Methods:

ADD-ACTIVE-AGENT (Agent)

in: Agent {Agent}.

Add a new active agent.

CELL-ORIGIN (Row Column)

in: Row Column {fixnum}.

The cell <Row> <Column> is going to be the new origin of the sheet.

Also causes an window update.

CELL-SIZE (&Optional Width Height)

in: &optional Width Height {fixnum}.

out: Width Height {fixnum}.

Access the size of a cell in pixels.

CLOSE-EVENT ()

Give user option to save sheet before closing if changes have been made.

COMPUTE-SCROLLBARS ()

Depending on the window size, cell size, and the dimensions of the sheet the scroll bars (if any) have to be adjusted.

COMPUTE-VISIBILITY ()

Determine the number of rows and columns visible.

DEFINE-DIMENSIONS (Number-Of-Rows Number-Of-Columns Number-Of-Layers &Key (Agents-Producer #'Default-Ager Producer))

in: Number-of-Rows Number-of-Columns Number-of-Layers {fixnum},
&key Agents-Producer {lambda-expression (<Row> <Column> <Layer>)
-> {list of: {Agent-Instance}}}
default #'default-agents-producer,

Determine the number of rows, columns and layers of the agent cube.

Adjust the size of the window to the number and size of the agents.

DEFINE-DISPATCH (Cell-Dispatch Layer-Dispatch)

in: Cell-Dispatch {lambda-expression ({list of: {fixnum}}) -> {list of: {fixnum}}},
Layer-Dispatch {lambda-expression ({list of: {Agent-Instance}})
-> {Agent-Instance} or {list of: {Agent-Instance}}}.

Define the dispatch function used to forward events like clicking at a cell to layers and within each layer to an agent instance.

- E a r l y D r a f t -

DOCUMENT-NAME (&Optional New-Name)
in: &optional New-Name {string}.
Document name = window title.

DOCUMENT-TYPE-NAME ()
out: Name {string}.
Return name of object.

FORWARD (Row Column Layer Height Width Selector &Rest Arguments)
in: Row Column {fixnum}, Layer {fixnum} or nil,
Height Width {fixnum}, Selector {symbol},
&rest Arguments {t}.
out: Result {t}.
Forward a message to a set of agent instances. This method is typically used when an agentsheet is referenced from another agentsheet. Determines the recipients using the cell/layer dispatch function.
Return the result of the message sent to the agent with max row column.
If the layer is not specified (e.g., layer = nil) then return the result of the agent on the 0-layer.

GALLERY (&Optional Layer New-Gallery)
in: &optional Layer {fixnum}, New-Gallery {Gallery-Instance}.
out: Galleries {Gallery-Instance or list of: {Gallery-Instance}}.
Access to galleries. Each layer is associated to a gallery.

KEY-EVENT (Char)
in: Char {Char}.
If there is a current keyboard focus agent then the character is forwarded to it using the SENSE-KEY message

KEYBOARD-FOCUS-CELL (&Optional Row Column)
in: &optional Row Column {fixnum}.
out: Position {list: <Row> <Column>}.
Define the cell receiving all future SENSE-KEY messages.

MARK-DIRTY ()
Mark the worksheet as dirty in order to prevent it from accidental close without save.

PROJECTION-LIST (&Optional List)
in: &optional List {list of: {fixnum}}.
out: List {list of: {fixnum}}.
Access to projection list.

REDRAW ()

REMOVE-ACTIVE-AGENT (Agent)
in: Agent {Agent}.
Remove the agent from the set of active agents.

RESIZE-EVENT ()
Invoked whenever the window size get changed interactively (e.g., using the mouse).

RUN-ACTIVE-AGENTS ()
Cycle through the active agents and send TASKS messages to them.

UPDATE-EVENT ()
Agentsheet window content needs to be redrawn.

Specific Methods:

APPEAR (&Key Explode-From)
in: &key Explode-From {list: <x1> <y1> <x2> <y2>}.
Initialize visibility and scoll bars

CLEAR-CELLS ()
Clear the window content.

CLICK-CELL (Row Column)
in: Row Column {fixnum} cell clicked.
Is called in case of clicking a specific cell. Dispatches the click event to a subset of agents at position <Row> <Column>.

CLICK-EVENT (X Y)
in: X Y {fixnum} local window cordinates.
Is called whenever the window content was clicked by the mouse.

DRAG-CELL-OUTLINES (Positions)
in: Positions {list of: {cons: <row> . <column>}}.
out: Offset {cons: <delta-row> . <delta-column>},
Upper-Left-Corner {cons: <row> . <column>}.
While mouse is pressed an outline of the agents at <Positions> can be dragged.

- E a r l y D r a f t -

The new relative position is returned. This method has no side effect, i.e., the agents are NOT moved.

DRAG-INTO (Row Column From-Delta-Row From-Delta-Column)

in: Row Column From-Delta-Row From-Delta-Column {fixnum}.

If after a click the mouse has not been released and the mouse is dragged over a cell border then the drag-into message is issued.

DRAW-CELLS (&Optional (Row 0) (Column 0) Height Width)

in: &optional Row Column {fixnum} default 0,
Height Width {fixnum}.

Make the agents draw their depiction. Draws ALL agents within the active layers denoted by the projection list.

FINISH-DRAGGING (Row Column From-Delta-Row From-Delta-Column)

in: Row Column From-Delta-Row From-Delta-Column {fixnum}.

Issued when mouse key(s) are released after dragging.

HORIZONTAL-SCROLL-EVENT (Position)

A scroll due to clicking of the scroll bar is due.

VERTICAL-SCROLL-EVENT (Position)

A scroll due to clicking of the scroll bar is due.

Unique Methods:

LOAD-FROM-PATHNAME (A-Pathname)

RESTORE-AGENT (Agenttype Row Column Layer Arglist)

in : the agent class, and row,column, layer to place into.

arglist - list of instance variable name / value pairs

Places the specified agent with specified depiction in the current sheet

(self) at the specified row, column, layer. Restores all instance variables.

RESTORE-WINDOW-GEOMETRY (X Y Width Height)

SAVE-TO-PATHNAME ()

SAVE-WINDOW-GEOMETRY (File)

Instance Variables:

Own:

From Class: AGENTSHEET

ACTIVE-AGENTS initform: NIL {list of: {Active-Agent}}

AGENTS initform: NIL {Agents}

CELL-DISPATCH-FUNCTION initform: #'IDENTITY {lambda-expression ({list of: {fixnum}}) -> {list of: {fixnum}}}

CELL-HEIGHT initform: 15 {fixnum} height of agent cell in pixels

CELL-WIDTH initform: 15 {fixnum} width of agent cell in pixels

COLUMN-S initform: 0 {fixnum} left column in window

COLUMNS initform: 8 {fixnum} number of columns

DIRTY initform: NIL non-nil after any worksheet modifying operation.

Set to nil again after save.

GALLERIES initform: NIL {array of {Gallery-Instances}}

KEYBOARD-FOCUS-CELL initform: NIL {cons <Row> <Column>}

LAYER-DISPATCH-FUNCTION initform: #'FIRST {lambda-expression ({list of: {Agent-Instance}})
-> {Agent-Instance} or {list of: {Agent-Instance}}}

LAYERS initform: 1 {fixnum} number of sheet layers

PROJECTION-LIST initform: '(0) {list of: {fixnum}} Only layer#s within this list will be drawn.

ROW-S initform: 0 {fixnum} top row in window

ROWS initform: 8 {fixnum} number of rows

UPDATE-REGION initform: NIL {list <Row> <Column> <Height> <Width>}

VISIBLE-COLUMNS initform: 0 {fixnum} number of columns fully or partially visible in window.

VISIBLE-ROWS initform: 0 {fixnum} number of rows fully or partially visible in window.

7.2. CLASS: LINK-AGENTSHEET

Class Object: LINK-AGENTSHEET

Super Class of: ICON-EDITOR-SHEET, GRAPHER-AGENTSHEET

Sub Class of: AGENTSHEET

Instance Methods:

Own:

From Class: LINK-AGENTSHEET

Generic Methods:

ADD-SELECTION (Row Column)

in: Row Column {fixnum}.

Add a cell to the set of selections.

BROADCAST-SELECTION (Except-Row Except-Column Selector &Rest Arguments)

in: Except-Row Except-Column {fixnum}, Selector {symbol}, &rest Arguments {t}.

Send a message to all members of the selection.

CELL-ORIGIN (Row Column)

in: Row Column {fixnum}.

Links need to be redrawn.

CLOSE-EVENT ()

Give user option to save sheet before closing if changes have been made.

COPY ()

Copy all selected agents and put them into the clipboard.

The selection remains intact.

CUT ()

Remove all selected agents and put them into the clipboard

DRAG-SELECTION ()

If there are multiple selections drag them to a new location by

cutting them on the old location and pasting them to the new one.

The operation does NOT affect the clipboard.

FORWARD-TO-FIRST-SELECTION (Selector &Rest Arguments)

in: Selector {symbol}, &rest Arguments {t}.

out: Result {t}.

Send a message to the first selection, i.e., the one created

with a click, not shift click.

NUMBER-OF-SELECTIONS ()

out: Number {fixnum}.

Return the number of selected cells.

PASTE (&Optional Row Column)

in: &optional Row Column {fixnum}

Copy agents in the agent clipboard into the agentsheet.

If <row> and <column> are not provided then the content of clipboard

will be pasted into the upper left corner of the agentsheet.

REDRAW ()

REMOVE-SELECTION (Row Column)

in: Row Column {fixnum}.

Remove a cell from the set of selections.

RESET-SELECTION ()

Remove all selections.

SELECTION-MEMBER-P (Row Column)

in: Row Column {fixnum}.

Return t if <Selection> is a member of the selected cells.

UPDATE-EVENT ()

Agentsheet window content needs to be redrawn.

Specific Methods:

ADD-LINK (Link)

in: Link {Link}.

Add a new link visible link to the set of links.

DRAW-LINKS (&Optional Row Column Height Width &Key Some-Links)

in: &optional Row Column Height Width {fixnum}

&key Some-Links {list of: {Links}}.

- E a r l y D r a f t -

Draw all links or just <Some-Links> intersecting the specified area of cells.

LINK (Row1 Column1 Row2 Column2 Name &Key (Color *Light-Gray-Color*) Label (Style :Solid) (Arrow T))
in: Row1 Column1 Row2 Column2 {fixnum},
Name {symbol},
&key Color {Color} default *Light-Gray-Color*, Label {string},
Style {keyword} default :solid,
Arrow {boolean} default t.
out: Link {Link}.
Link a pair of cells.
<Name> is used as reference to communicate from agent to agent.
If <Arrow> is non nil then the link is a arc; communication is only one way.
Draws the link if the color is non-nil and the window is visible.

LINK-SELECTION (Name &Key (Color *Light-Gray-Color*) Label (Style :Solid) (Arrow T))
in: Name {symbol},
&key Color {keyword} default *Light-Gray-Color*, Label {string}, Style {keyword} default :solid
Arrow {boolean} default t.
Create a binary link between two cells. The cell selected first is the source,
the cell selected secondly is the destination of the link.
If less than two cells are selected nothing happens.

REMOVE-LINK (Link)
in: Link {Link}.
Remove a visible link.

UNDRAW-LINK (Link &Optional Refresh-Agents)
in: Link {Link}, &optional Refresh-Agents {boolean}.
Undraw a link. If <Refresh-Agents> is set to a non-nil value then refresh all the
agents which have been covered by the link.

UNLINK-SELECTION ()
Remove ALL links from first to second selection.

Unique Methods:
LOAD-FROM-PATHNAME (A-Pathname)
SAVE-TO-PATHNAME ()

Instance Variables:
Own:
From Class: LINK-AGENTSHEET
AGENT-CLASS-NAME initform: 'TEXT-NODE-AGENT
LINKS initform: NIL {list of {link}}
SELECTED-CELLS initform: NIL {list of {cons <Row> <Column>}}

7.3. CLASS: ICON-EDITOR-SHEET

Class Object: ICON-EDITOR-SHEET

Agentsheet to display and edit color icons.

Super Class of: COLOR-PALETTE-SHEET

Sub Class of: LINK-AGENTSHEET

Class Methods:

Own:

From Class: ICON-EDITOR-SHEET

Generic Methods:

```
NEW (Icon &Key (Name "Untitled") (Pixel-Size 8) (Agent-Class-Name 'Icon-Editor-Color-Pixel-Agent) Explode-
in: Icon {Icon},
    &key Name {string} default 'Untitled' Pixel-Size {fixnum} default 8,
    Agent-Class-Name {symbol} default 'icon-editor-color-pixel-agent,
    Explode-From {list: <x1> <y1> <x2> <y2>}.
out: Sheet {ICON-EDITOR-SHEET}.
Create instance of ICON-EDITOR-SHEET.
```

Instance Methods:

Own:

From Class: ICON-EDITOR-SHEET

Generic Methods:

ICON (&Optional New-Icon)

INITIALIZE-AGENTS (Agent-Class-Name Width Height)

in: Agent-Class-Name {symbol}, Width Height {fixnum}.

Create a new agent cube if necessary or re-use existing agent cubes from Agent Repository if compatible with <Agent-Class-Name> <Width> <Height> request.

REQUIRES-SAVING ☹

Never save a icon editor sheet.

Unique Methods:

UPDATE-EVENT ☹

Class Variables:

Own:

From Class: ICON-EDITOR-SHEET

AGENTS-REPOSITORY value: #<HASH-TABLE :TEST EQL size 2/60 #x819021> {list of: {list: {list: <Agent-Class-Nc <Width> <Height>} {Agents}}}.

Instance Variables:

Own:

From Class: ICON-EDITOR-SHEET

ICON initform: NIL {CIconHandle}

TOOLS initform: '(POINT DRAW ERASE FILL SAMPLE GET-INFO)

7.4. CLASS: COLOR-PALETTE-SHEET

Class Object: COLOR-PALETTE-SHEET

Color Palette Sheets are used for interactive color selection by the user. Color Palette Sheets represent standard color tables of depth 1, 2, 4, 8 depending on the screen pixel depth.

Super Class of:

Sub Class of: ICON-EDITOR-SHEET

Class Methods:

Own:

From Class: COLOR-PALETTE-SHEET

Generic Methods:

NEW (&Key (Name "Colors") (Pixel-Size 8))

in: &key Name {string} default 'Colors' Pixel-Size {fixnum} default 8.

out: Sheet {ICON-EDITOR-SHEET}.

Create a color palette agentsheet.

Instance Methods:

Own:

From Class: COLOR-PALETTE-SHEET

Generic Methods:

UPDATE-EVENT () extension:

Also draw selection frames.

Instance Variables:

Own:

From Class: COLOR-PALETTE-SHEET

LEFT-MARGIN initform: 0

TOOLS initform: NIL no tools required

7.5. CLASS: GRAPHER-AGENTSHEET

Class Object: GRAPHER-AGENTSHEET

A grapher-agentsheet can represent a partially ordered set.

Super Class of: COLOR-GALLERY-AGENTSHEET

Sub Class of: LINK-AGENTSHEET

Instance Methods:

Own:

From Class: GRAPHER-AGENTSHEET

Generic Methods:

DEFINE-CHILDREN-FUNCTION (Function)

in: Function {lambda (Node) -> {list of <Node>}}.

The children function has to return the children of each node or nil if there are none.

ROOTS (&Rest New-Roots)

in: &rest Roots {symbol}.

out: Roots {list of: {symbol}}.

If <Roots> is omitted then the current root will be returned. Otherwise <Roots> is used as the new root and the window refreshed.

TEXT-MODE (Mode)

in: Mode {boolean}

Choose between text mode and depiction mode.

Specific Methods:

ADD-CHILD-TO-SELECTION (Link-Name Value &Optional Class-Name)

in: Link-Name {symbol}, Value {t}, &optional Class-Name {symbol}.

out: Row Column {fixnum}, Agent {Agent}..

Creates a child of all the selected agents, link the selected agents to it, and return its coordinate.

ADD-ROOT (Value)

in: Value {t}.

Add additional root node.

CANNONIZE ()

Make a modified graph canonical again.

LAYOUT ()

Map the logical node topology to agents and links in the Agentsheet.

Instance Variables:

Own:

From Class: GRAPHER-AGENTSHEET

CHILDREN-FUNCTION initform: NIL {lambda (Node) -> {list of <Node>}}

NODES initform: (MAKE-HASH-TABLE) {hashtable of: {Agent}}

ROOTS initform: NIL {list of: {symbol}}

TEXT-MODE initform: T {boolean} if nil then symbols are interpreted as depiction names

7.6. CLASS: COLOR-GALLERY-AGENTSHEET

Class Object: COLOR-GALLERY-AGENTSHEET

Super Class of:

Sub Class of: COLOR-DEPICTION-STORAGE, GRAPHER-AGENTSHEET

Class Methods:

Own:

From Class: COLOR-GALLERY-AGENTSHEET

Generic Methods:

ACTIVE-GALLERY-AGENTSHEET (&Optional Gallery-Agentsheet)

in: &optional Gallery-Agentsheet {Gallery-Agentsheet}.

out: Gallery-Agentsheet {{Gallery-Agentsheet}}.

Access to the current active gallery agentsheet.

FORWARD (Selector &Rest Arguments)

in: Selector {symbol} &rest Arguments {list of: {t}}.

out: Result {t}.

Forward the message to the currently active gallery agentsheet instance if there is one.

IMAGINE ()

out: Color-Gallery-Agentsheet {Color-Gallery-Agentsheet}.

Create a new Color-Gallery-Agentsheet and prepare-contents.

NEW (&Key (Window-Show T)) extension:

out: Color-Gallery-Agentsheet {Color-Gallery-Agentsheet}.

Create COLOR-GALLERY-AGENTSHEET with root.

Instance Methods:

Own:

From Class: COLOR-GALLERY-AGENTSHEET

Generic Methods:

ACTIVATE-EVENT ()

ADD-DEPENDENCY (Node-Name Children)

in: Node-Name {symbol}, Children {list of: {symbol}}.

Define a new dependency.

ADD-OPEN-WORKSHEET (&Optional Worksheet)

in: optional Worksheet {instance}.

out: list of open worksheets.

Adds Worksheet to list of currently open worksheets.

AGENTSHEET-CLASS-NAME (&Optional New-Name)

in: &optional New-Name {symbol}.

out: Agentsheet-Class-Name {symbol}.

Access to class name of worksheet associated with gallery.

BACKGROUND-AGENT-CLASS (&Optional Background-Agent-Class)

in: &optional Background-Agent-Class {symbol}.

out: Background-Agent-Class {symbol}.

BROADCAST-OPEN-WORKSHEETS (Selector &Rest Arguments)

in: Selector {symbol}, &rest Arguments {t}.

Broadcasts method (selector) to all currently open worksheets.

CLASS-DEFINITION-FILE (&Optional New-Class-Definition-File)

in: optional New-Class-Definition-File {string}.

out: Class-Definition-File {string}.

CLOSE-EVENT ()

Make sure this sheets gets no longer used

DEFINE-CLASS-FILE (Pathname)

in: Pathname {pathname}.

This file will be load the next time the gallery is load.

Typically this file contains the all class definition and the depiction -> class relations.

DEFINE-END-USER-DEPICTIONS ()

Define the currently selected depictions as the set relevant to end users

DOCUMENT-NAME (&Optional New-Name)

- E a r l y D r a f t -

in: &optional New-Name {string}.
Document name = window title.

DOCUMENT-TYPE-NAME ()
out: Name {string}.
Return name of object.

DRAW-LINKS (&Optional Row Column Height Width &Key Some-Links)
Only draw the links in visual programming designer view

INIT () extension:
Fill in agents.

KEY-EVENT (Char)
in: Char {character}.
Gallery keyboard short cuts.

NUMBER-OF-SELECTED-DEPICTIONS ()
out: Number {fixnum}.
Returns the number of depictions currently selected.

PREPARE-CONTENTS ()
This gets called before sheet APPEARS in case a sheet gets IMAGINED.

REMOVE-OPEN-WORKSHEET (Worksheet)
in: Worksheet {instance}.
out: list of open worksheets.
Removes Worksheet from list of currently open worksheets.

SELECT-END-USER-DEPICTIONS ()
Select all the depictions defined for the end-user view

SELECTED-BITMAPS ()
out: Icons {alist: <Name> <Icon>}.
Return the set of currently selected Icons.

SND-SOUND-FILE (&Optional New-Snd-Sound-File)
in: optional New-Snd-Sound-File {string}.
out: Snd-Sound-File {string}.

SOUND-FOLDER (&Optional New-Sound-Folder)
in: optional New-Sound-Folder {string}.
out: Sound-Folder {string}.

TOOL-ICON-FILE (&Optional New-Tool-Icon-File)
in: optional New-Tool-Icon-File {string}.
out: Current Application Tool-Icon-File {string}.

WORKSHEET-FOLDER (&Optional New-Worksheet-Folder)
in: optional New-Worksheet-Folder {string}.
out: Worksheet-Folder {string}.

Specific Methods:

CLONE-DEPICTION ()
Create a new node by applying a clone operation to the selected set of Icons,
insert it into the graph and refresh the window.

CREATE-ROOT-NODE (Name Icon)
in: Name {symbol}, Icon {Icon}.
Initialize gallery with a first node.

DELETE-SELECTED-DEPICTIONS ()
Delete all selected depictions.
Non-terminal depictions cannot be deleted -> create a warning.

LOAD-FROM-PATHNAME (Pathname)
in: Pathname {string}.
Load icon and dependency files related to <Pathname>.

NEW-WORKSHEET ()
out: Worksheet {Agentsheet}.
Create a new worksheet compatible in cell size and background
agent to gallery.

RENAME-SELECTED-DEPICTION (New-Name)
in: New-Name {symbol}.
Rename a depiction, update sheet.If there are multiple depictions
selected use the first one

REPLACE-BITMAP (Name Icon)
in: Name {symbol}, Icon {Icon}.
Replace the Icon stored in gallery named <Name> with <Icon>.

SAVE-TO-PATHNAME ()
Save the gallery's icons and dependencies into files.

- E a r l y D r a f t -

```
VIEW (&Optional New-View)
  in: &optional New-View {keyword :vp-designer, :end-user}.
  out: View {keyword :vp-designer, :end-user}.
  Define a view specific to some type of user
Unique Methods:
  NEW-DEPICTION ()
Class Variables:
  Own:
  From Class: COLOR-GALLERY-AGENTSHEET
    ACTIVE-GALLERY-AGENTSHEET value: #<OPUS instance, __, a COLOR-GALLERY-AGENTSHEET>
Instance Variables:
  Own:
  From Class: COLOR-GALLERY-AGENTSHEET
    AGENT-CLASS-NAME initform: 'COLOR-GALLERY-AGENT
    AGENTSHEET-CLASS-NAME initform: 'LINK-AGENTSHEET {symbol}
    BACKGROUND-AGENT-CLASS-NAME initform: 'SELECTABLE-BACKGROUND-AGENT {symbol}
    CLASS-DEFINITION-FILE initform: NIL {string}
    DEPENDENCIES initform: (MAKE-HASH-TABLE) {symbol} -> {list of: {symbol}}
    END-USER-DEPICTIONS initform: NIL {list of: {symbol}}
    END-USER-VIEWPORT-SIZE initform: '(100 . 100)
    OPEN-WORKSHEETS initform: NIL {list of: {Agentsheets}}
    SND-SOUND-FILE initform: NIL if non-nil load these snd resources
    SOUND-FOLDER initform: NIL folder where play-from-file sounds are stored
    TOOL-ICON-FILE initform: NIL if non-nil load these cursors
    TOOLS initform: (LIST 'POINT 'GET-INFO) {list of: {symbol}}
    VIEW initform: VP-DESIGNER type of view, one of :vp-designer :end-user
    VP-DESIGNER-VIEWPORT-SIZE initform: '(100 . 100)
    WORKSHEET-FOLDER initform: NIL if non-nil use this as default worksheet folder
```

7.7. CLASS: HYPER-AGENTSHEET

Class Object: HYPER-AGENTSHEET

A hyper agentsheet is part of a so-called hyper agent.
Communication takes place in both directions.

Super Class of:

Sub Class of: AGENTSHEET

Instance Methods:

Own:

From Class: HYPER-AGENTSHEET

Generic Methods:

HYPER-AGENT-LINK (Agent Container)

in: Agent {Agent-Instance}, Container {Agentsheet-Instance}.

Set the links to the hyper agent and its container.

Instance Variables:

Own:

From Class: HYPER-AGENTSHEET

HYPER-AGENT initform: NIL {Agent-Instance} link to hyper agent

HYPER-AGENT-CONTAINER initform: NIL {Agentsheet-Instance}