



# Getting Started

for Agentsheets version 1.3.0a2 & Visual AGenTalk version 0.8a2

Alexander Repenning  
Department of Computer Science and Center of LifeLong Learning  
Campus Box 430, University of Colorado, Boulder CO 80309  
(303) 492-1349, ralex@cs.colorado.edu  
Fax: (303) 492-2844

The purpose of this document is to get you started using Agentsheets and Visual AGenTalk. It is assumed that you know the basics of using a Macintosh but no programming experience is required. You will be stepped through the creation of a complete but small project from scratch.

---

## CONTENTS

<b>Tactile Programming .....</b>	<b>2</b>
<b>Hands-On: The Particles Project .....</b>	<b>3</b>
Defining Looks.....	3
Building a World.....	4
Defining Behaviors.....	4
Action Commands: Make Agents do Things.....	4
Condition Commands: Testing Circumstances.....	5
Behavior Editors and Rules .....	5
Dealing with Non-Determinism .....	7
State versus Transition.....	7
<b>Helpful things .....</b>	<b>8</b>
Procedural Abstraction and Messages.....	8
Short Cuts.....	8
More Documentation.....	9
Alpha: Limitations and known bugs .....	9

The research is supported by NSF (No. RED 925-3425 & Supplement), ARPA (No. CDA-940860), and Apple Computer Inc.

---

# TACTILE PROGRAMMING

Agentsheets has a history reaching back to 1989. We have created over 100 projects in Agentsheets in the past. Some of them as one afternoon hacks, others as multiple person-year projects. In this process we have experimented with extensions to object-oriented programming frame works, programming by example and with graphical rewrite rules. At the same time we realized the power of end-user programming with graphical rewrite rules but also their limitations. Visual AgenTalk is a recent (started in 1995) effort with which we explore new scaleable programming paradigms that would feature the same ease of use for end-user programming but allowing users to gradually move towards the power of some of the more traditional programming paradigms. Visual AgenTalk is an instance of what we call Tactile Programming extending visual with tactile perception. This is trivializing complex research issues, but in essence Visual Programming is employing visual perception to simplify programming by increasing the readability of programs. Tactile Programming does not question this goal but hopes to make programming more accessible to end-users by adding the *perception of manipulation* to visual perception. In Tactile Programming programs are no longer static representations nor is the notion of manipulation reserved to only edit programs. Instead, tactile program and their representations are dynamic including manipulation. In result, Tactile Programming turns into a unified program manipulation paradigm, that is a way to conceptualize programs, that supports the composition, comprehension and even sharing of programs through the World Wide Web. Tactile Programming is its very early infancy. Visual AgenTalk in its current form is a fresh seed of an evolving programming paradigm.

All this is fairly abstract especially because the benefits of tactile perception are hard to explain well in a static medium such as this document printed on paper. For a more complete philosophical background, software updates and agents to share visit our Web site:

<http://www.cs.colorado.edu/~l3d/systems/agentsheets/>

For a hands-on experience go on, plunge in and build your first project...

---

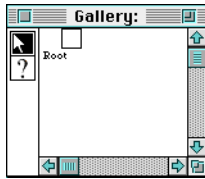
# HANDS-ON: THE PARTICLES PROJECT

Lets create a little world featuring particles adhering, in a very simplified way, to the laws of physics. This will allow you to build a simple sand watch, experiment with distribution and even explore the laws of conservation. The creation of this entire project from scratch will take less than an hour. The only potential time sink could be the attempt to draw really nice agent depictions (you have been warned).

---

## DEFINING LOOKS

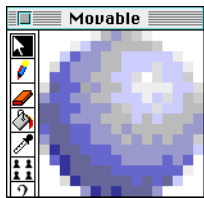
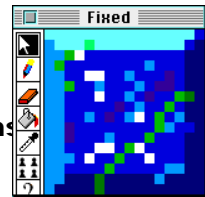
The first step is to create a couple of agents looking like particles.



Start a new project with **File New Project...** (a short way of referring to the **Project...** menu command in the hierarchical **File** menu under the **New** heading). In the first dialog box type in "Particles" as the name of the project. In the second dialog box specify the depiction size in pixels to be 16 x 16. This is kind of small which is good for this specific projects since you will need to be able to see many agents at the same time. The default size of 32 x 32 is same size as icons on the Macintosh desktop. As a result of this operation you get a *depiction gallery* which is one of the essential parts of a project. The new gallery (Figure below) comes with a single depiction called Root.

The plan is to create two kinds of agents, *movable* particles that can fall down and *fixed* particles are fixed in space. Select the Root depiction and hit return. This allows changing the name of a depiction. Call it "movable". Create an additional depiction using the **Gallery New...** menu command. Change its name to "fixed".

The look of the depictions created should imply at least somewhat their behavior. Edit the depiction either manually, by double clicking it, or by grabbing it anywhere from the screen. The Fixed depiction, on the right, got created manually using the Agentsheets depiction editor and the Color Palette featured in the **Applications** menu.





The movable depiction, on the left, got created by drawing a ball in Canvas and then using the **Gallery GrabScreenAnySize** command in Agentsheets to create an anti-aliased miniature. This is a technique to create very nice depictions. Other great source of pictures that can be turned into anti-aliased miniatures is the Word Wide Web. Any picture displayed with your favorite Web browsers can directly be turned into a miniature in Agentsheets.

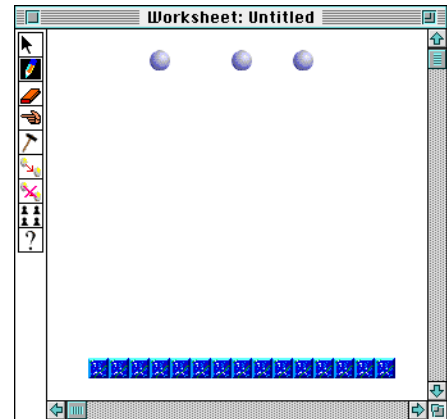
The **Gallery GrabAgain** menu command allows you to grab once defined portions of your screen again at any point in time. This is a very helpful feature if you use a dynamic picture sources such as a live camera in which case you can grab different images at different moments to create a number of depictions representing your agents state. For instance, you could use facial expression of a person. Also, you can grab a 3D model using a QuickDraw 3D viewer as the source of multiple representations. OK, here we are starting to do it; spending too much time on creating look. Note, however, while the difference between creating a blob look and slick anti-aliased look can be many hours of hard work it also can mean the difference between a blah project and one that really draws people in. But now enough of this. This is a good time to save the gallery, call it "Components", and to build a world.

---

## BUILDING A WORLD

Having our basic component we are ready to create a world. Via the **File New**  **Worksheet** menu command you create a new worksheet, i.e., a place in which agents live and interact with each other and with you the user.

Select the "Fixed" depiction in the gallery, activate the worksheet just created by clicking at it and select the draw tool in the worksheet . Your first click at the worksheet with the draw tool selected will bring up a dialog box asking you if you like to create a new agent class called "Fixed" or if you are happy with using just an generic agent. Select Fixed. Repeat these steps for the "Movable" depiction and draw a scene like to the one on the left.



The scene is completely passive at this point lets change this by defining some behaviors.

---

## DEFINING BEHAVIORS

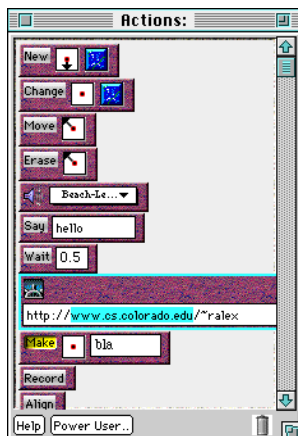
You may think that now it's time for the hard part but this is not really so. The notion of tactile programming featured by Visual AgenTalk will help to **gradually** develop more complex behaviors. In tactile programming not only the things in your application world are objects but also the element of your programming world. In essence tactile programming enhances object-orientation programming by elevating the programming world objects from textual or even visual objects to the level of tactile objects that can be manipulated by end-users.



The part of Visual AgenTalk presented here is based on production rules expressing behaviors as IF THEN rules. Visual AgenTalk is not limited to production rules. Other applications have used Visual AgenTalk to create tactile versions of more traditional programming paradigms.


---

## Action Commands: Make Agents do Things


Get the Actions palette via the **Applications ActionPalette** command.



Each action command from the command palette on the left, e.g., the move command, , is a tactile object that can be direct manipulated in a couple of ways. Command parameters, e.g., the direction parameter, , can be direct manipulated to define their value. The entire command can be dragged and dropped to a number of different locations. It can be moved to some other place in the palette, on top of agents and, as we will see later, into behavior editors.

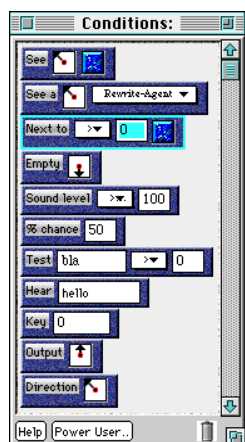
To explore how agents deal with commands just drag them onto the agent that should execute the command. Hint: use the label area, , as the area to grab commands. Dragging, for instance, the move command onto any of our agents will make them move in the direction indicated by the direction



parameter. Other commands allow your agents to play sounds, talk via the speech synthesizer (if you have the Speech manager of Apple installed), change their look, open up page on the Web via Netscape and many more things. You cannot only create new commands yourself but you can even share them with others via the world wide web. This, however, is not covered in this introductory document, you need the Visual AgentTalk designer manual.

To drag and drop a command onto an agent will make the agent *execute* the command *once* in its current context. This allows you at any point in time to have any agent execute any command you want to see what it does. A command that is currently executing provides special visual feedback by changing its frame to a yellow and black striped pattern . This shows you **what** is executing, **when** and **how long**. This makes especially sense when commands are embedded in programs.

## Condition Commands: Testing Circumstances

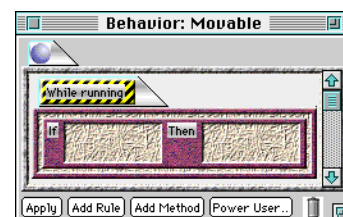
Condition commands, found in the **Applications** menu, are used to test the circumstances agents are in.





Like action commands, condition commands can be dragged and dropped onto agents. Dropping the Next-To  command will test if more than one agent out of the 8 immediate neighbors look like . If the condition is true then it provides the same kind of feedback like the actions but if it is false then it will blink and play an alert sound. In the scene we created this condition is false for all movable agents and true for some fixed agents. Additional commands allow you to test attributes, check the sound level of the microphone (if you have one), poll if keys on the keyboard are pressed and many other things.

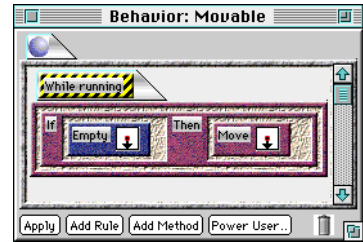
## Behavior Editors and Rules

In contrast to executing actions and conditions once by dragging them directly onto agents *behavior editors* allow you to assign permanent behaviors to agents. Double clicking any of the Movable agents **in the worksheet** will bring up a behavior editor (right).



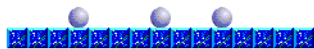
A behavior editor can contain any number of *methods*. A method is the combination of a *trigger* with a set of rules. A new behavior editors comes up with a single method containing the while running  trigger and one empty rule. The trigger defines **when** a method get executed. The while running trigger will make the method execute periodically if the worksheet is in running mode (see below).

A naive model of gravity suggest that things fall down if there is no supporting object below them. This becomes our first rule. Drag an Empty  condition into the *condition box*, the box between the If and the Then label, of the rule and a Move action into the *action box* of the rule. The direction parameter of Empty and Move should point down to refer to the agent below.



The behavior editor serves as a scratch pad. Behaviors constructed can at any time and any level by explored. Like using the command palettes you can drag and drop action and condition commands out of the behavior editor on agents to see what they do. You can also drag action and condition boxes, rules and even entire methods onto agents.

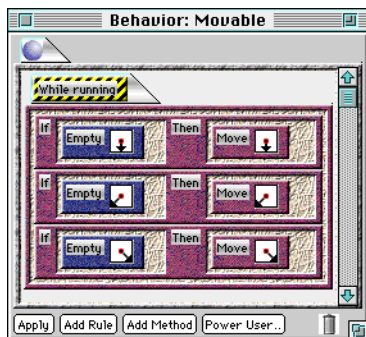
Test the rule by dragging it onto any agent (movable or fixed). The execution feedback will show you that first all conditions get checked top to bottom (there is only one so far), and if none of them fails then all the actions are executed again top to bottom. If you repetitively keep the Movable agent executing the rule until it sits on top of the Fixed agents, or if you simply use the mouse to drag it down there, you will note that when executing the rule again the empty condition will no longer hold (it will blink) and the move action will consequently no longer be executed.




Being fairly confident with the gravity rule *apply* the behavior to the agent by pressing the apply button in of the Behavior editor. Now run the worksheet via the **Worksheet Run** menu command. Behavior is applied

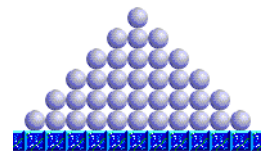
to all instances of the same agent class. Consequently, applying the rule will make **all** the Movable agents fall to the ground made up of Fixed agents.

Agents can still be manipulated when the worksheet is running. That is, agents can be added, erased or simply be moved around in the worksheet. With the simple behavior given so far Movable agents can be stacked on top of each other.

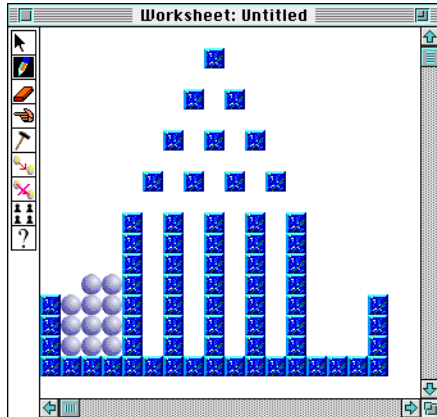


This behavior does not seem to be consistent with physics. The Movable agents look like balls that would not be stackable. Add two rules to deal to extend the behavior using the "Add Rule" button of the behavior editor. The two new rules look similar to the previous one but they check in the diagonal and make agents move diagonally. Make sure to keep the first rule on top of the list.

To test these rules drag the entire method onto a Movable agent by grabbing it on the method tab  (not the trigger part of it). The rules will be executed with visual feedback from top to bottom. After applying the new behavior to the Movable agent run the worksheet and drop a number of movable agents from the same spot. They pile up nicely.

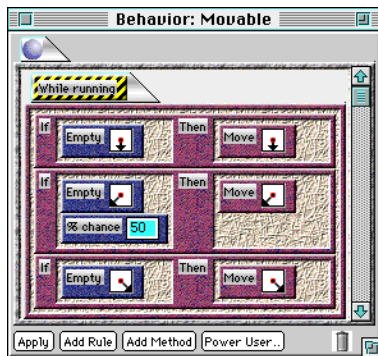


## Dealing with Non-Determinism

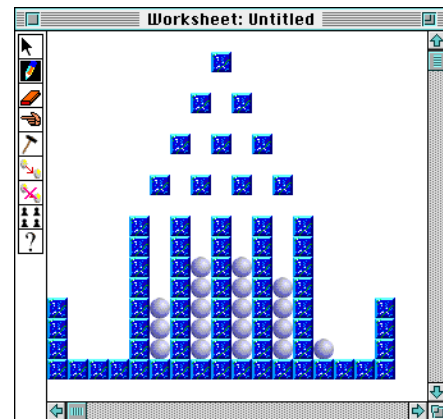


In many simulations it is necessary to deal with non-determinism, i.e., to be able to include elements of chance. The scene, on the left, resulting from creating a more complex world with Fixed agents and dropping a number of Movable agents on top reveals a problem.

All the particles end up on the left because of the order in which the rules are executing. If a Movable agent can go left **and** right it will always go left. Changing the order of the move left/right rules would make all the movable agents move to the other side. Adding a chance 50% condition, instead, will change the distribution.




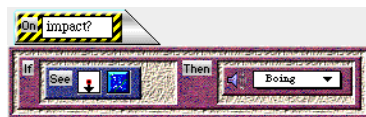
Now movable agents will fall with same chance to left as they fall to the right. This is the famous Patienko experiment demonstrating normal distribution.

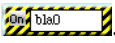


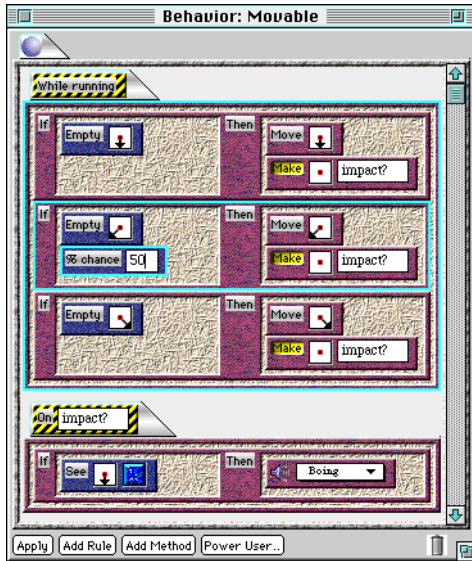
## State versus Transition

One of the trickier problems with programming can be to create behaviors that are based on *state transitions* rather than on states. For instance it is simple to detect the state of a Movable agent being

adjacent above a fixed agent with a rule such as  but it is much harder to detect something like an *impact* which is a state transition from a state of not being adjacent to later being adjacent. This rule would not only apply after the Movable agent moved on top of a Fixed one but would keep on applying. As a result the boing sound would keep on playing indefinitely even after the impact event. In state-based programming approaches such as graphical rewrite rules it is possible but quite tricky to get around these problems. Visual AgenTalk's ability to create named methods and to call them explicitly at specific moments can be used in this case to solve the transition problem.



Create a new method by pressing the "Add Method" Behavior Editor button. New methods come by default with a On trigger, . Change the name of this method to "impact?" and put the previous rule into this method.



All that is left to do is to call this method at the moment of the transition. With the previous rules of moving there were three cases when movement could occur. Put Make commands after each Move command with the "impact?" message resulting in our final behavior

Don't forget to press Apply. This concludes the example.

## HELPFUL THINGS

### PROCEDURAL ABSTRACTION AND MESSAGES

The ability to name methods and to call them cannot only be used in specific cases such as the previous transition problem but it is a very general way to achieve procedural abstraction by allowing the separation of behavior definition and reference.



The Make command is not just used to call your own methods but also to send message to you neighbors. For instance, this command will send the explode message to the agent to the right of you. This command should be read as "make the agent to my right explode."

### SHORT CUTS

Some helpful shortcuts:

- **Quick Command Palettes:** Double clicking at action or condition boxes will bring up the action and condition command palettes respectively. Double clicking triggers will bring up the trigger palette.
- **Drag Forwarding:** Dragging an action or condition command onto a rule, instead of into action or condition box, will automatically forward the command into the right box.
- **Drag 'n' Copy:** Dragging action or condition commands from one rule into another will copy them. Dragging rules from one method into another will copy them. Dragging methods from one Behavior editor into another will copy them.

---

## MORE DOCUMENTATION

This is only a quick start manual helping you to create one simple project. For more documentation consult our Web site: <http://www.cs.colorado.edu/~l3d/systems/agentsheets/>