Massachusetts Institute of Technology
Artificial Intelligence Laboratory
February 1976

AI WORKING PAPER #120 LOGO WORKING PAPER #48

AN ACTOR-BASED COMPUTER ANIMATION LANGUAGE

by

Kenneth M. Kahn

Abstract

This paper reproduces an appendix of a doctoral thesis proposal that describes a language based on actor semantics designed especially for animation. The system described herein is built upon MacLisp and is also compatible with Lisp-Logo. The system was implemented to serve two functions: to provide a base system for the knowledge-based animation system which is described in Working Paper 119 (or Logo WP 47) and to experiment with various extensions of Logo to improve its value as an educational tool.

This work was supported in part by the National Science Foundation under grant number GJ-1049 and conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program.

TABLE OF CONTENTS An Actor-based Animation Language

Section I: Introduction

Some of the recent AI languages are based on a new view of computation sometimes called "actor" semantics. Carl Hewitt's PLASMA ([Hewitt 1975] and [Smith 1975]) and Alan Kay's SmallTalk [Goldberg 1974] are the best examples. The basic idea is to consider each entity within the system as something that is usually anthropomorphized as a "little person." Each "little person" or "actor" can receive messages asking it to do something, remember something, recall something, or send some messages to other actors. For animation this seems an ideal way to represent objects on the display. Each object is a process that can be arbitrarily smart. Charlie Brown can be an "actor" that can be told to walk, causing him to send the appropriate messages to his arms and legs and moving the rest of his body.

I have implemented a system in Lisp-Logo [Goldstein 1975] which enables one to define new objects, new object types, and the kinds of messages they can handle. For example, one can easily create a square named "George." George can be told many kinds of things like his size, speed of movement, or speed of rotation. George can be asked to do many things, all the things that turtles can do (FORWARD, BACK, RIGHT, LEFT, HIDE, SHOW, PENUP etc.) plus new acts like growing, or changing appearance. George can also be taught new things, or can be told to behave in ways other than his defaults. George also has a memory, you can tell George anything at all, his color, his friends, whatever.

One very important thing that George knows (though like everything in the system he can be told otherwise) is that he is a square. Presently, "Squares" know a few things, like how to draw themselves, or that after rotating 90 degrees that they look the same. Squares in turn know that

they are instances of "Object." Objects know how to do the turtle-like things mentioned above. Objects, in turn, know that they are instances of "Something", things that can receive messages, can pattern match those messages, and can perform memory functions. "Somethings" know how to learn new responses to new patterns. This entire hierarchy is very flexible and modifiable by the user. The basic process is the message is sent to some individual, "George", and if George has no patterns that match the message, he sends it off to the actor that he is a kind of. They in turn pass the buck, until either someone can handle the message, or an error message is generated. This is also a very useful default mechanism, if George is never told his size he can inherit it from "Square" or "Object".

Another feature of this system is the ability to have many different actors move on the screen with apparent parallelism. George can race against Sally. Danny's garden of flowers can be grown [Hillis 1976]. A stick figure can simultaneously move different limbs and change it's facial expression. "Movies" (or a list of display commands) can be produced that can be run forward or backwards at any speed the computer system is capable of or single stepped.

This system is intended to support the intelligent computer animator which is discussed in the companion AI Working Paper 119 (Logo 47) and, equally important, to be used by children. The system is hopefully a more powerful and natural means for doing simple programs for animation. The powerful ideas of "instances, classes and finding of the correct level of generality" and the "little person model of computation" are imbedded into the system. The hope is that through well-guided use of the system, some of these ideas will become more concrete to the children. Of course, all the usual reasons for teaching Logo to children remain in force (e.g. learning by doing, experience with debugging, becoming articulate in describing processes, and

exposure to and assimilation of powerful ideas). The use of the same actor-animation system by children and by the computer animator is very important for making the computer animator more accessible and understandable by the children using it. The idea is that if the children who programmed using the actor-animation system found it natural and intuitive then its use in the intelligent system would also be clear.

This view of programming as collections of actors, or a community of "little people", that send and receive messages from each other is very powerful. It is conducive to a modular, simple, natural representation of the knowledge needed for the application. Using an actor system one can model intelligence as a integrated community of rather limited individuals or in the more conventional manner as an integrated individual.

Another AI aspect of this system is the explicit "kind-of" hierarchy of actors. Each object is told what class it is a member of when it is created. When any object receives a message it cannot handle it passes the problem on to the class of which it is a member. The important concepts of instantiation, class membership, exceptions, placement of knowledge at the best level of generality, and inheritance of properties hopefully will flow from the proper use of this aspect of the system.

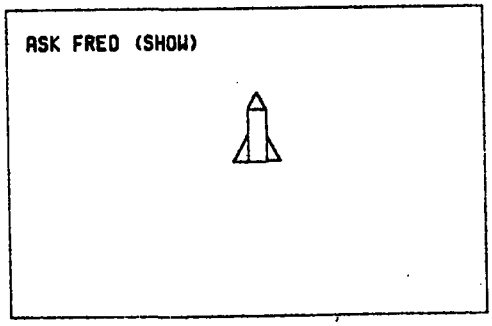
To both facilitate the use of the system and to give the user a good start in what is hopefully the right direction, the top nodes of the hierarchy can be predefined. Currently the default hierarchy consists of an actor called "Something" at the top node. "Something" can receive messages to remember, forget, replace or find items. It receives messages for editing actors including ones to insert, remove, replace, interchange or correct receivers. It also can create new instances of actors, print out the script and memory of an actor, and accept messages to be done at a later time. "Object" is an actor directly below "Something" and as such inherits all of its abilities.

In addition "Object" can behave like a Logo turtle on the tv display. "Objects" also can move across the screen at a particular speed, can rotate at any speed, can revolve around a point, move away from a point, grow and shrink. "Object" also remembers various items such as its rotational speed, size, and speed thereby providing default values to all its inferiors. There are other sub nodes of "Something" such as "Movie" and "Universe" which are discussed later. Under development are "Composite-something" and "Composite-object" which know how to send appropriate messages to their parts. In the following examples "Rocket" and "Flower" which are instances of "Object" are used for illustrative purposes.

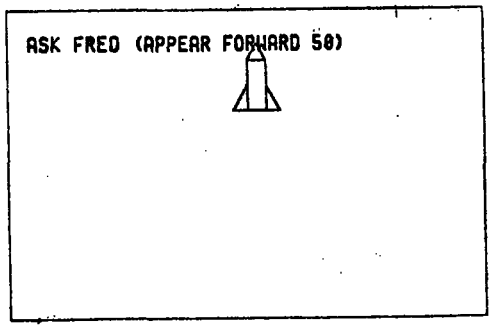
Section II: An Example

```
ASK ROCKET (MAKE FRED)
```

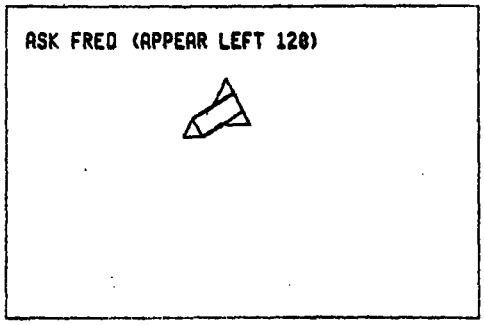
ASK is the basic message passing command, in this case the message "Make Fred" is being sent to the actor "Rocket." "Rocket" does not know how to handle messages of this form so it passes the message to "Object" who also ignorant of such messages. The message is finally sent to "Something" which can match the message with one of its patterns and it creates a new actor named "Fred" which is a kind of "Rocket."



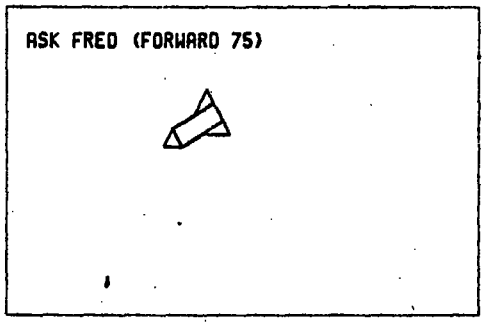
Fred is asked to show himself. He knows nothing about "showing" and asks "Rocket" who asks "Object" which can handle the message. It asks Fred what turtle procedure draws him and Fred doesn't know so asks Rocket who answers with the name of a standard Lisp (or Logo) procedure. It then asks Fred for his position, heading and size and then invokes the Lisp (or Logo) procedure.



"Object" is passed this message via "Fred" and "Rocket" and Fred is asked to Hide and then to appear at the place 50 steps forward. The word "APPEAR" is there to distinguish this type of message from those in which the movement is gradual as described later.



Again "Object" handles this type of message and Fred is asked to Hide, then to rotate to the left 120 degrees and then to show.



This time Fred is told to go forward, so he asked for his speed, he has none in this example and asks "Rocket" for his speed which is 25. He then is asked to (APPEAR FORWARD 25) *and to plan on continuing the rest (50) on the next tick of the "clock".*

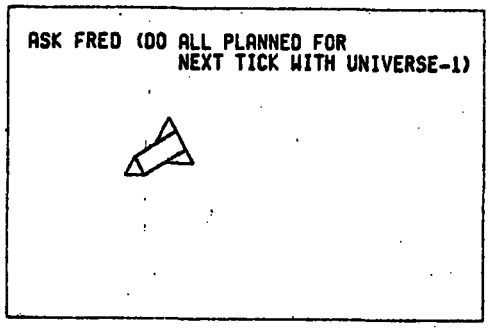
```
ASK FRED (WHAT SPEED ?)

25
```

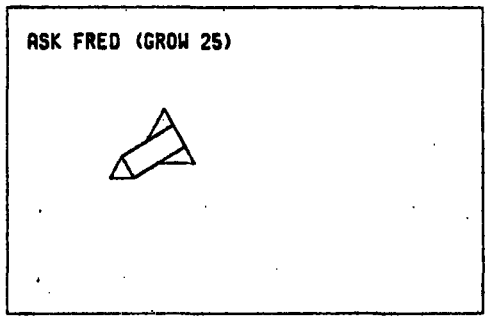
In responding to the previous message Fred was asked what his speed was. Fred, as is true of all "Somethings", has a memory. This memory is a general relational data base and it is also used to maintain the state of actors. The message "What ..." indicates that the value found to is to be returned.

```
ASK FRED (REPLACE SPEED 50)
```

Fred is asked to remove the item about his speed (there is none in this case) from his memory and to remember the item "(SPEED 50)".



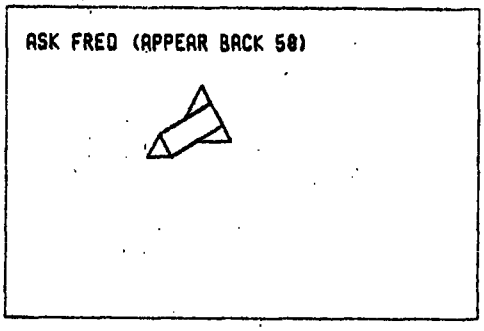
Fred is asked to do all the things that he had planned to do on the next "clock" tick.[1] *He asks himself what things he had planned then and does them. In this case the only thing that was planned was* **"(FORWARD 50)"** *which was left over from earlier. Since his speed is now 50 he can do it all and appears forward 50 steps. "Universe-1" is an actor that can be asked for all the other actors that are currently on the screen, so that interactions are possible. For example, collision or avoidance can be implemented by asking the other actors where they are and maybe even where they are planning on going.*



Fred is told to grow, which causes a message to be sent to him to hide, then to replace his size with his old size plus 25 and finally to reappear again.

---

1 To some people this wordy style of programming is distasteful. I could just as well have defined the message to be "(TICK UNIVERSE-1)". It is very important, however, that the code be as clear and easy to read as possible. The difficulty in typing can be overcome by simple human engineering aids, for example, a special "help" button which, when pushed, could finish the line to the extent possible, saving much typing and preventing misspellings.



Now when Fred is told to do anything an image of him with his new size moves.

```
ASK ROCKET '(IF RECEIVE SHOOT MISSLE WITH SPEED ?SPEED TO GO ?DISTANCE THEN
        DO-THE-FOLLOWING:
                ASK ROCKET (MAKE MISSLE)
                ASK MISSLE (REPLACE SPEED :?SPEED)
                ASK MISSLE (REPLACE SIZE (QUOTIENT
                                          (ASK :SELF (WHAT SIZE ?))
                                          4))
                ASK MISSLE (REPLACE STATE (ASK :SELF (WHAT HERE ?)))
                ASK MISSLE (SHOW)
                ASK MISSLE (FORWARD :?DISTANCE THEN HIDE))
```

The behavior of any actor in the system can be extended. The "if receive ..." message is matched by "Something" which adds a new receiver to the actor that received the message. In this case, "Rocket" is sent the message asking it that if it receives any messages of the form: the word "shoot" followed by the words "missle with speed", then any word, then the words "to go" followed by only one more word,[2] then call the first word "?speed" and the second word "?distance."[3] Then do the following series of things:

> *(1) create a rocket named "Missle" (it is possible to make the name "Missle" local to this receiver or to have a unique name generated)*
>
> *(2) ask the newly created "Missle" to replace its speed with the number that in the message that corresponded to the word "?SPEED" in the pattern (a fancier version could easily add the rocket's present speed with ":speed")*

---

2 If the pattern was SHOOT MISSLE WITH {NUMBERP ?SPEED} TO GO {NUMBERP ?DISTANCE} then it will match only if the words following "SHOOT" and "GO" are numbers.

3 The ":" in front of the names is a convention necessary to be compatible with Lisp-Logo. It is also used in messages to indicate that the value of the atom, not the atom is intended.

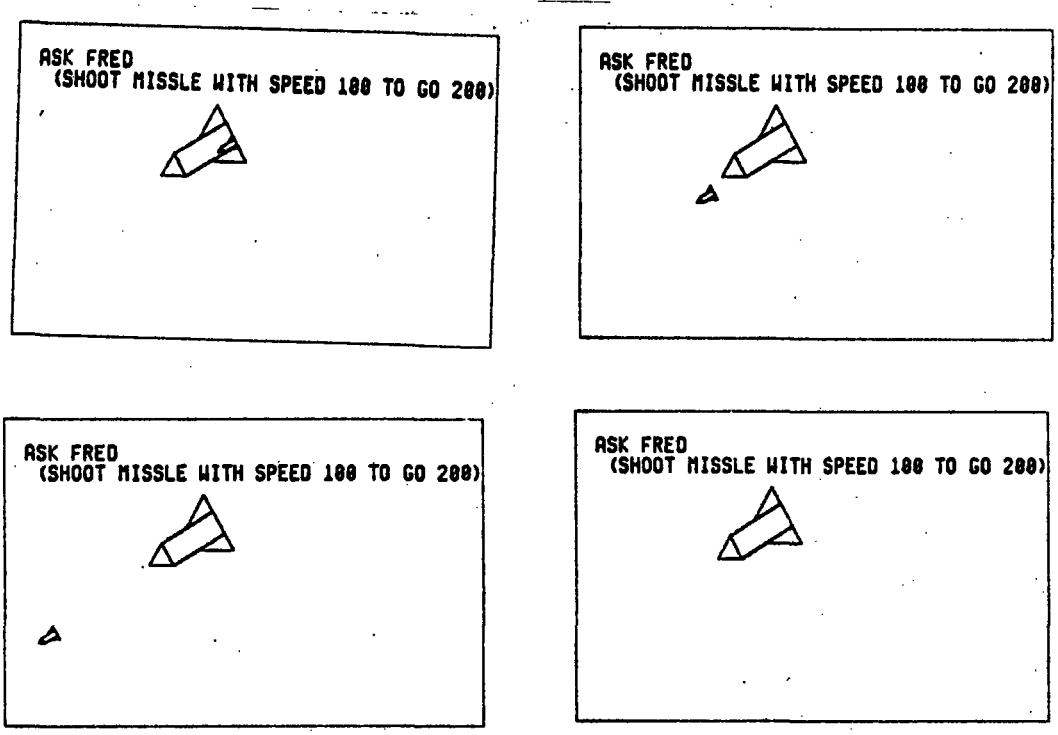
(3) replace the size of the missle with 1/4 of the size of the actor that received the message which is always called "self"

(4) replace the state of the missle with the state of the actor receiving the message; this way the missle appears where the shooter is, rather than the default which is the center of the screen

(5) the missle is asked to show itself

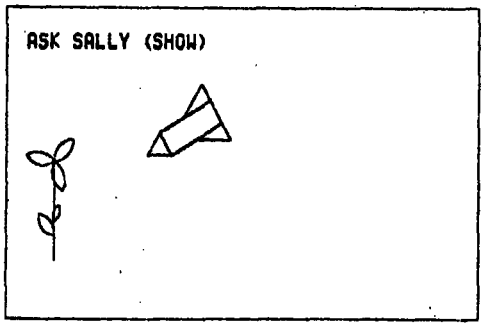
(6) it is told to go forward the last word in the message

(7) finally, it is told to hide when the finished moving forward (a fancier version might explode).



Fred is given this newly defined type of message and then asks "Rocket" to try to handle it. It can, and the above procedure is executed with the speed of the missle being 100 and the distance it is to travel being 200.

```
ASK FLOWER (MAKE SALLY)
ASK SALLY (REMEMBER SIZE 50)
```



A flower named Sally is created, given a size and asked to show.

Now we can use "Flower" and "Rocket" to make a little animated movie. Suppose we want a movie in which Sally the flower is just peacefully swaying back and forth in the wind. Then Fred the rocket flies by and shoots a "Missle" at Sally. Fred flies away and the "Missle" heads right towards poor Sally. As a surprise ending, however, the missle could be filled with water and Sally could grow larger as she continues to sway in the wind.

```
ASK SALLY (PLAN: SWAY 10 DEGREES 12 TIMES NEXT)
```

"Flowers" can be asked to accept "sway" messages which cause them to go left and then right the specified number of degrees. The "Plan:" part is the same kind of message that "Forward" produced previously. Sally does nothing on receiving this message other than remember to do it with the next tick of the "clock."

```
ASK FRED (PLAN: FORWARD 300 NEXT)
ASK FRED (PLAN: SHOOT MISSLE WITH SPEED 50 TO GO 150 NEXT)
ASK FRED (PLAN: RIGHT 90 THEN SHRINK 100 AFTER 2 MORE TICKS)
ASK SALLY (PLAN: GROW 60 AFTER 6 MORE TICKS)
```

More events are scheduled, such as Fred being told to begin going forward 300 steps and then shoot a missle with a speed of 50 to go 150 steps. Two "clock" ticks later Fred will start to turn right 90 and when finished turning will shrink away. Notice that he will be turning while he still has some steps to go forward and thus will plot a polygonal course. Six frames into the movie Sally will be begin to grow. (The numbers of ticks in this example were chosen simply because they caused thing to happen at the right time. Much of this would become simpler if one could plan events relative to other events. This is a problem I hope to tackle soon.)

```
ASK MOVIE (MAKE SHOOTING 12 TICKS LONG IN UNIVERSE-1)
```

Here "Movie" is asked to make a movie called "Shooting" that is 12 frames long. It in turn asks "Universe-1" to send to a "tick" message to all the actors with things planned. It will stop either after 12 ticks or sooner if no more things are planned. The "Screen" asks "Shooting" to remember each display command in addition to doing them, so that they can be played back at a speed that is not limited by the time it takes to send and interpret all the messages.

ASK SHOOTING (SHOW)

"Shooting" is asked to show its record of the running of the display commands it remembered when the movie was made. Stills from the movie can be seen in Figure A.

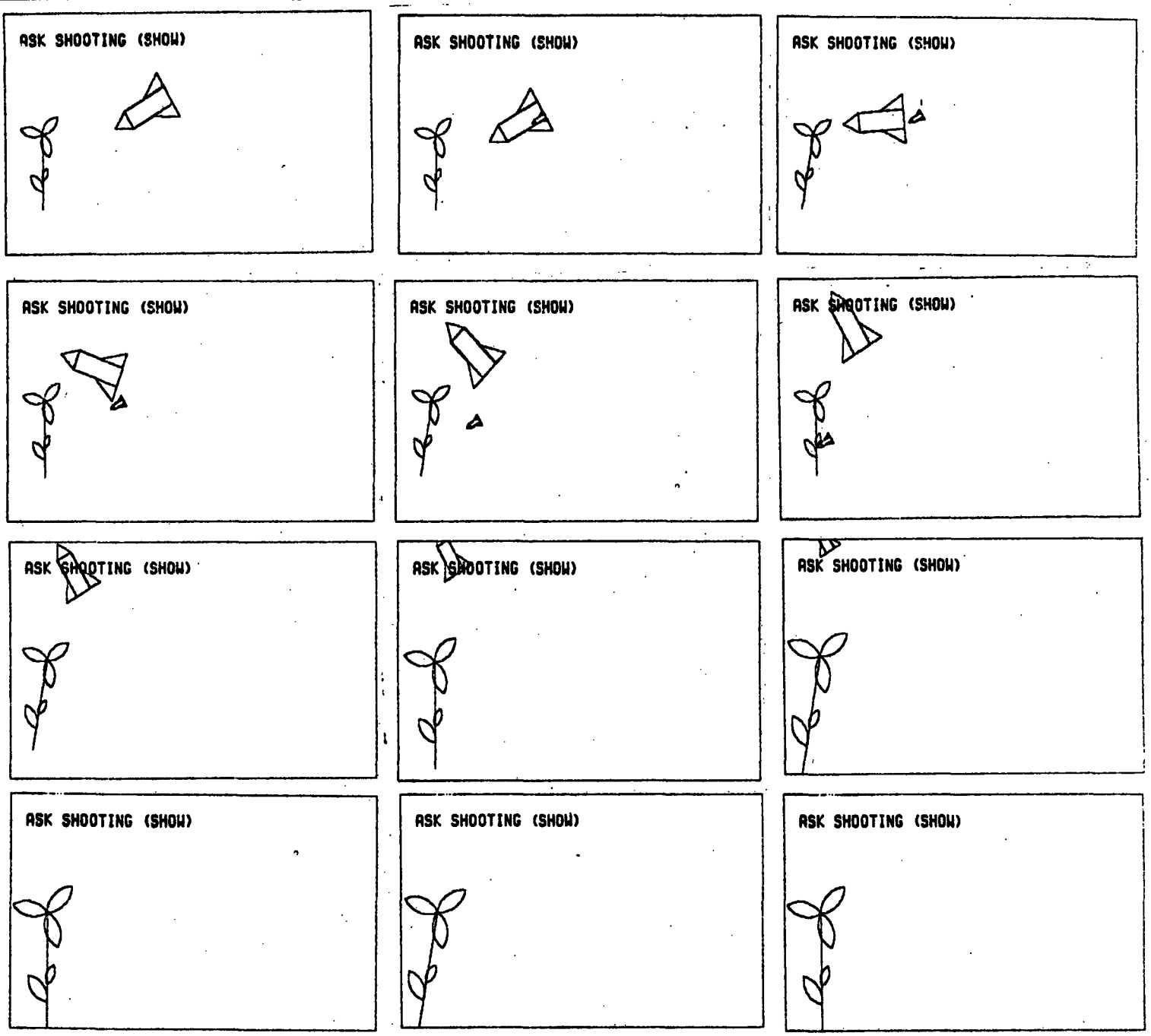


Figure A

Scenes from the "Shooting" Movie

Section III: How to Grow a Flower Garden

In a paper by Danny Hillis called, "Ten Things to do with a Better Computer", he has an example of how to grow a garden in an actor-like system [Hillis 1976]. He describes a garden in which seeds are born, wait, grow into flowers, create new seeds, continue growing and die.

In my animation system, his garden can be implemented in a fairly straight-forward manner as follows:

```
TO DEFINE.FLOWER
;Logo syntax can be used if my system is loaded into Lisp-Logo
10 ASK OBJECT (MAKE FLOWER)
20 ASK FLOWER (REMEMBER SIZE 10)

;This tells flower to remember that the default size of flowers is 10. "Remember" is the standard kind
of message for telling any actor to remember something
30 ASK FLOWER (REMEMBER DRAW USING DRAW-FLOWER)
;This tells flower that the Logo procedure called "draw-flower"
;is to be used to draw instances of "Flower."
END

TO DEFINE.SEED
10 ASK SOMETHING (MAKE SEED)
; Seed is not an Object, since it does not do turtle-like things.
20 ASK SEED (IF RECEIVE ?SEED (START) THEN DO.SEED.THING :?SEED)
;This simply lets seeds take a "start" message and then calls the appropriate procedure.
END

TO DO.SEED.THING :SEED
10 LOCAL A.FLOWER
;a local name for the flower that seed will spawn is needed
20 ASK FLOWER (MAKE A.FLOWER)
30 ASK A.FLOWER (APPEAR RIGHT 90)
;Flowers are "Objects" and so can take any turtle-like command
40 ASK A.FLOWER (APPEAR FORWARD (* 100 (RANDOM)))
50 ASK A.FLOWER (APPEAR LEFT 90)
60 ASK A.FLOWER (PLAN: SHOW IN 10 TICKS)
;The message transmission ASK A.FLOWER (SHOW) will occur after "A.flower"
;has received 10 "Tick" messages.
70 REPEAT 15 (ASK A.FLOWER (PLAN: GROW 10 AFTER 2 MORE TICKS)
;This schedules the call, ASK A.FLOWER (GROW 10),
```

```
; 15 times, each time 2 ticks after the last.
80 ASK SEED (PLAN: ASK (SEED (MAKE)) (START) AT THAT TIME)
; This means at that time create another seed and tell it to start at the same time as the last thing
scheduled, which was after line 70 was run.
90 ASK A.FLOWER (PLAN: HIDE AFTER 60 MORE TICKS)
END


ASK (ASK SEED (MAKE)) (START)
;This starts the first seed off which will start the others off later.
ASK MOVIE (MAKE GARDEN 1000)
The default universe sends ticks to all the flowers and seed involved, the things they had planned are
run, and all display and turtle commands are stored in the movie "Garden".
ASK GARDEN (REMEMBER SPEED 10)
ASK GARDEN (SHOW)

;These two message transmissions result in the movie being shown at 10 frames per second. First, a
flower appears and grows and and then another starts to grow, then another and after a while the
first flower disappears and so on.
```

Section IV: Message Passing

Message passing is the mechanism of communication between two actors. A pattern matching procedure called "Match" is used to decode the incoming message. This provides great flexibility in the syntax of the messages sent and received, and enables the use of much more readable commands.

The following is a list of some of the messages that any instance of "Something" can receive:

(1) The "RECEIVE" message which enables an actor to increase the set of messages it understands,

(2) The memory messages, "REMEMBER," "WHAT," "REPLACE," and "FORGET" which provide a powerful relational memory to each actor. These are also used to inspect the position, heading, speed, size and the like of an object and to update those quantities.

(3) "MAKE" and "UNMAKE" messages which create or destroy instances,

(4) Two kinds of "PRINT" messages which prints out the script or the memory of an actor in a form designed to be easy to read,

(5) A series of structural editing commands for inserting, deleting, and replacing receivers or their parts,

(6) "PLAN:" message for scheduling things to do at later times,

(7) The "TICK" message for telling an actor to do all that he or she should during the smallest quantum of time and providing him or her with access to the names of other actors, enabling interactions.

Instances of "Object" can, by passing the buck to "Something", receive all the above messages. They can also receive the "turtle" commands, i.e. "FORWARD," "BACK," "RIGHT," "LEFT," "HIDE," "SHOW," "PENUP," "PENDOWN," of which there are two varieties. One, the object moves gradually across the screen at some speed while the other the object disappears and appears at the new position.

Section V: Ticks Plans and Movies

Using the system, one can write procedures that move one object, then another and back again. It will not, however, look as if they are moving simultaneously since the interpretation and execution of the commands is slower than the maximum duration for persistence of vision of occur. A solution is to have a scheduler run things at the appropriate time, and then save away the display commands to be run later. These saved-away commands, called a "Movie", can be run later with the appearance of parallelism. This was done in earlier implementations but the current one distributes the responsibility to each actor to remember what it will do later. The scheduler, or "Universe" as it is called, just sends "tick" or "increment the time" messages to all actors it knows about. Movies can be remembered or just the code run, depending on the message to the "Universe" or "Movie."

If one wants an event to be dependent upon the occurrence of another event then the appropriate actors must check for the occurrence of the event when it receives a tick message. For example, if Lucy is told to scream if Snoopy comes too close then the "Lucy" actor must check to see where Snoopy is whenever she receives a tick message. If Lucy was told to scream if anybody came near, she would need to know the names of everybody around. That is why name of the current universe comes along with each tick message. Lucy can ask each actor where he or she is.

When an actor is told to plan something it always relative to that own actor's internal time. His or her time is incremented with each tick. Associated with each time are the things that actor plans to do at that time. After the actor remembers the things planned it tells the current universe that it has things to do and would like to be placed on the mailing list for receiving tick messages.

The universe when told to run will send ticks to each actor with things to do. When an actor has nothing left to do it tells the universe who stops sending it ticks. When the universe has run the number of ticks asked of it or there are no more actors with things to do it stops.

The screen is an actor that receives display messages like "Put George Forward 100". When a movie is being made the screen can send messages asking the movie to remember the display messages the screen received. The movie can then be run later without sending any messages except those to the display.

Section VI: How does one talk about this?

In teaching children to use the animation system one should have a consistent vocabulary and set of concepts. Are the words "Something," "Universe," and "Object" reasonable names for these general entities? I think not, but haven't thought up better ones. How should one talk about the difference between "Square", the general square, and any instantiation of squares? Will there often be a confusion between the actor "George", his "script", and his image on the screen? Identity becomes very strange to talk about, since anything about an actor can be changed, what it looks like, how it acts, even what it is a kind of. (Though this programming style may be discouraged to avoid this confusion.) Only its name is permanent.

Explaining the ticks, plans and universes might bring up problems. How should one talk about, or think about, many separate processes going on in parallel? I don't really know.

This system lends itself very well to the "little man" vocabulary. Each actor is a little person, who knows a few things, can be told to do a few things, and can be taught new tricks. This little person receives messages and sends them to others. This view of programming lends itself very well to a variation of playing "turtle." The children can play games where they are actors and pass the messages around. This link between reality and this programming style I hope will make the system seem more intuitive and natural.

Section VII: Efficiency Issues

One may worry that such a system will run too slowly to be useful for working with children or for building an intelligent system on top of it. The message passing and matching involved are much slower than more traditional mechanisms. The basic use of hierarchies is slow, since each actor seldom can respond directly to a message but needs to pass it on to the class which it is a member.

The answer to this objection is standard. One should let a compiler worry about such efficiencies. I have implemented a few macros that when possible replace actor transmissions with the code that they would invoke. I also plan to "compile" patterns in the receivers to run faster. The price for some of these hacks is less flexibility. If the transmission "ASK FRED (APPEAR FORWARD 100)" is replaced by the action part of the receiver in "Object" with "Fred" and the amount instantiated properly. However, telling Fred's immediate superior a new way to handle "forward" messages will not affect his behavior if he is "compiled."

There are other efficiency hacks that may be worthwhile, for example, in the memory system for the actors. I plan, however, to follow the principle that the code should be written clearly and simply and that efficiency hacks should be below the surface and transparent to the user.

Section VIII: Extensions and Improvements Planned

One useful extension would be the addition of primitives for joining and breaking apart "Object" actors. For example, one may want to join a triangle actor and a square actor to make a house actor. Or one may want to have a face accept messages as well as any of it parts. A person may get into a car, so that temporarily any movement of the car should also change the state of the person. Some progress has been made here, so that simple composite objects can be defined but more needs to be done.

The scheduling and interaction of events is very important and needs to be extended. Events should be able to be scheduled relative to other events or internal clocks. "Object" should be able to accept messages to handle simple interactions. For example, one should be able to tell an "Object" to go forward until it runs into another "Object".

Another improvement being considered is the ability to handle partial messages. For example, if an object receives a forward message without a number as the second word, then instead of the present response of printing out an error message, it should prompt the user with a question like, "How much should Fred go forward?"

Taking this idea one step further I plan to have a "help" button that can be pushed at any time. If one has typed only part of a message and then the help button, then the actor may be able to finish part or all of the message for the user. This feature will hopefully alleviate many of the problems of typing long names and messages that make the code more readable.

It should be clear by now that this system is far from complete and I welcome any suggestions or criticism.

Bibliography

[Adler 1976]
Adler, M.
"Understanding Peanuts Cartoons"
Progress in Perception
Department of Artificial Intelligence Research Report No. 13
University of Edinburgh, December 1975

[Baecker 1969]
Baecker, R.
"Interactive Computer-Mediated Animation"
MIT EE Ph.D. Thesis 1969 MAC-TR-61

[Goldberg 1974]
Goldberg, A.
"Smalltalk and Kids -- Commentaries"
Learning Research Group, Xerox Palo Alto Research Center, 1974 Draft

[Goldstein 1974]
Goldstein, I. P.,
Understanding Simple Picture Programs,
MIT AI Laboratory AI-TR-294, September 1974

[Goldstein 1975]
Goldstein I., Lieberman H., Bochner H., Miller M.
"LLOGO: An Implementation of LOGO in LISP"
MIT-AI Memo 307, March 4, 1975

[Goldstein 1976]
Goldstein I., Papert S.
"AI, Language and the Study of Knowledge"
MIT-AI Memo 337, 1976

[Hewitt 1975]
Hewitt C., Smith B.
"Towards a Programming Apprentice"
IEEE Transactions on Software Engineering SE-1, March 1975

[Kahn 1976]
Kahn, K.
"Three Interactions Between AI and Education"
Machine Representations Of Knowledge
ed. Elcock, E., Michie D.

D. Reidel Publishing Company, Dordrecht, Holland, in press

[Minsky 1975]
Minsky, M.
"A Framework for Representing Knowledge"
The Psychology of Computer Vision
ed. Winston, P.
McGraw-Hill Book Company, New York, 1975

[Papert 1971a]
Papert S.
"Teaching Children Thinking"
MIT-AI Memo 247, October 1971

[Papert 1971b]
Papert S.
"Teaching Children To Be Mathematicians vs. Teaching About Mathematics"
MIT-AI Memo 249

[Parke 1972]
Parke, F. I.
"Computer Generated Animation of Faces"
Univ. of Utah Report CSc-72-120, June 1972

[Schank 1975]
Schank, R.
Conceptual Information Processing
North Holland Publishing Company 1975

[Schulz 1962]
Schulz C.
"All This and Snoopy, Too"
Fawcett Crest Publications, Inc. Greenwich, Conn. 1962

[Schulz 1975]
Schulz C.
Peanuts Jubilee -- My Life and Art with Charlie Brown and Others
Holt, Rinehart and Winston, New York 1975

[Simmons 1975]
Simmons, R.
"The Clowns Microworld"
Theoretical Issues in Natural Language Processing
ed. Schank R. and Nash-Weber B. June 1975

[Smith 1975]
Smith B. and Hewitt C.
"A Plasma Primer"
MIT-AI Working Paper 92, October 1975

[Sutherland 1963]
Sutherland, I.
"Sketchpad: A Man-Machine Graphical Communication System"
MIT Lincoln Lab TR-296 1963