15780240

# AT&T Bell Laboratories

## Document Cover Sheet for Technical Memorandum

**Title:** Crabs: the bitmap terror

| Author | Location | Ext. | Dept. |
|---|---|---|---|
| Luca Cardelli | MH 2C-458 | 5707 | 11271 |

| Document No. | Filing Case No. | Work Project No. |
|---|---|---|
| 11271-850701-10TMS | 39199-11 | 311403-0101 |

**Abstract**

Crabs is a graphic demo which violates most of the assumptions underlying well-structured window systems. It illustrates both the raw power of bitmap graphics and the restrictions which are usually imposed on its usage.

Pages of Text 23 Other Pages 2 Total 25
No. Figs. 0 No. Tables 0 No. Refs. 2

L. Cardelli
MH 2C-458

**Complete Copy**

Executive Directors 112
Directors 112
Department Heads 1127
MTS 11271

**Cover Sheet Only**

A. A. Penzias
1127 MTS
R. B. Ardis

**CI-II Review**

This document *does not contain* any of the types of information listed below and, in accordance with CI-II, *may* be furnished to AT&T-IS.

*Network Information* - Unpublished information related to the existing operation of or committed changes to intercarrier connection or the connection and/or operation of customer premises equipment with AT&T Communications network(s) by means of which regulated carrier services are furnished.

*Proprietary Information of Telecommunications Customers* - Unpublished information acquired from billing or message detail records that is related to telecommunications service provided by AT&T Communications to specifically identified customers and that finds a principal use in marketing (e.g., information describing the kinds and quantities of telecommunications service provided to an identified customer or information describing traffic and usage patterns of an identified customer).

*Nongeneric Software for Customer Premises Equipment (CPE) or Enhanced Services* - Any software that is of use by Information Systems in CPE products or enhanced services and that is not generic software.

MPM

**Approval:** BWKeough 7/3/85
A. G. Fraser Director

**Author Signature**

Luca Cardelli

Luca Cardelli

**For Use by Recipient of Cover Sheet:**

Computing network users may order copies via the *library* command; for information, type "man library" after logon. Otherwise:

1 Enter PAN if AT&T-BL (or SS# if non-AT&T-BL). _____
2 Fold this sheet in half with this side out.
3 Check the address of your local Internal Technical Document Service if listed; otherwise, use HO 4F-112. Use no envelope.
4 Indicate whether microfiche or paper copy is desired.

Internal Technical Document Service

| | |
|---|---|
| ( ) AK 2N-02 | ( ) IH 7K-101 |
| ( ) ALC 1B-102A | ( ) MV 1D-40 |
| ( ) CB 1C-336 | ( ) RD 20D-215 |
| ( ) HO 4F-112 | ( ) WH 3E-204 |

Please send a complete ☐ microfiche ☐ paper copy of this document to the address shown on the other side.

Subject: Crabs: the bitmap terror
Work Program- 311403-0101 -- File- 39199-11

date: July 1, 1985

from: Luca Cardelli

TM: 11271-850701-10TMS

*TECHNICAL MEMORANDUM*

## Laws and violations

A bitmap screen is a graphic universe where windows, cursors and icons live in harmony, cooperating with each other to achieve functionality and esthetics. A lot of effort goes into making this universe consistent, the basic law being that every window is a self contained, protected world. In particular: (1) A window shall not be affected by the internal activities of another window. (2) A window shall not be affected by activities of the window system not concerning it directly, i.e (2.1) it shall not notice being obscured (partially or totally) by other windows or obscuring (partially or totally) other windows, (2.2) it shall not see the *image* of the cursor sliding on its surface (it can only ask for its position).

Of course it is difficult to resist the temptation to break these rules. Violations can be destructive or non-destructive, useful or pointless. Useful non-destructive violations include programs printing out an image of the screen, or magnifying part of the screen in a *lens* window. Useful destructive violations are represented by the *pen* program, which allows one to scribble on the screen. Pointless non-destructive violations include a *magnet* program, where a moving picture of a magnet attracts the cursor, so that one has to continuously pull away from it to keep working. The first pointless, destructive program we wrote was *crabs*.

## History

The history of crabs is presented here with dates, times and people. Not that we kept notes, of course. The dates and times were reconstructed months later by looking at the creation date of files, and by what we could remember.

### *Prologue: Peek*

Crabs was written by Mark Manasse and me in November 1982, and evolved in about two days to its present form. The basic principles of law-violation were investigated a few months earlier (August 5, 1982) when Bart Locanthi brought in a Smalltalk videotape. It featured, among other things, a *peek* demo. This is a program which looks at a rectangular portion of the screen (controlled by moving the cursor around) and replicates it in its own screen space in real time. Beautiful self-referential effects are obtained when this window peeks at itself, or part of itself. This is a digital version of a video-camera looking at its own tv screen.

Copying data from another window, as peek does, can already be considered a violation of the rules. But what peek does is even worse because, for a given window, peek will only copy that part of the window which is visible on the screen (i.e. not obscured by other windows). This cannot be done by asking a window to access its data: a window is not aware of what parts are visible. This is stealing data directly from the screen. A well-structured graphics interface will not allow this, and one has to use low-level routines which are ***not meant to be used by normal people***. Needless to say, Bart and Mark rushed to implement it.

### *Step 1: QIX*

*November 16, 1982, dinner time.* Mark wanted to implement the QIX video game for our Blit terminals [AT&T 85] (knowledge of QIX is assumed here). A QIX screen can get very complicated, and there are complex rules about how things are allowed to move. Mark started figuring out clever data structures and algorithms to compute fast line operations. After a while I said, "Wait a second. Atari is selling arcade QIX machines and there is no way they can have enough memory to run those algorithms. How are they doing it?" After some thinking: "I bet they don't keep line segments in data structures, but they draw lines on a bitmap and (gosh!) they just look at what is in the bitmap to determine line intersections. Gee, this is awful." Although this was repulsive to our trained algorithmic minds, that was the germ of the crabs collision-detection trick. We never implemented QIX.

### *Step 2: Measles*

*November 16, later.* After a while Mark was convinced and we started implementing. We decided to start with a single QIX (i.e. a single line with two bouncing dots at the ends) for simplicity, and to use window boundaries to test the line intersection trick. Mark started dictating code and I typed it down. This was still a bit too hard, so we simplified it further: forget the QIX, let's just have little balls floating in the grey area between windows and bouncing against window borders. We would look at the raw screen bits to determine where a window border was (is there grey there?). Mark kept dictating, and after a while it was working. It was just about one page of code. Mark called this *measles*; we had a lot of measles bouncing around the screen. They were also bouncing off each other for free because they would see non-grey and change direction. This was very cheap and convenient: normally one would have to test the position of every measle against the position of every other measle to determine whether there is a collision.

### *Step 3: Angry Measles*

*November 17, very early.* Now a problem came up. We have all these measles bouncing around, and you create a new window and slap it on top of them. Suddenly those poor trapped measles have nowhere to go, no grey area to run to. They are frozen, paralyzed with terror, and buried underneath a window. Mark didn't like that at all, and came up with the concept of *angry measles*. When a measle gets buried underneath a window, it starts flashing so that it is visible through the window, as if saying "Hey, get that window off me". It turns out that little flashing things are very annoying to the human eye, and you would take the window away just to shut them up. At this point, tired and satisfied, we went to sleep.

### *Step 4: Hungry Measles*

*November 17, late morning.* I slept a lot less then Mark did. When I came in, I started showing measles to people. They thought it was cute stuff. Some objected to the flashing measles solution. We had considered many alternatives the night before, and I wasn't totally satisfied with that solution either. Dave MacQueen said something like "they should eat their way out." I thought that was a possibility, only sillier than most. After he left, however, that idea kept coming back. I went to look at the code (as I said, Mark did the dictating because he was more familiar with Blits then I was), and discovered that I could implement Dave's suggestion by changing a single line of code. That seemed to be easy enough, so I did it. When a measle was confronted with a non-gray area, it would change a little bit of that area to grey. Trapped measles could then build up grey regions and eventually escape.

The new version, *hungry measles*, had quite a different character. It wasn't cute, it was awesome. Those little balls would eat away your windows. If trapped, they would escape, leaving you wounded. There was no protection against them. You could set up barricades of windows to protect a part of the screen you wanted to work in, and they would erode them. They would infiltrate along the borders of the screen, where you are not allowed to put windows. You couldn't keep them all under control: they were too many, too quick. You couldn't get distracted.

### *Step 5: Crabs*

*November 17, afternoon.* I went up to the unix room and started the program on a terminal. People gathered, and several expressions of disgust were heard. Jim Weythman said "they look like crabs!".

Everybody knew instantly that that was the right name for it. I went back to my room and designed the basic crab icon. Mark came back. With his help, we prepared the crab icon so that it would look nice on a grey background. We made it so that crabs would move sidewise, and would turn around according to their prevalent direction. Crab legs would appear to move, because of an unexpected optical interaction with the grey background. We made the crabs window self-destruct so that there was no way of stopping crabs, short of rebooting the terminal. Finally, we allowed the crabs to see the image of the cursor on the screen, so that you could use the cursor to poke them (they would bite it, but the cursor regenerates). We showed it to Rob Pike again. He said "That's it, don't touch it any more".

## Impact

In the next few days, unaware people were exposed to crabs in the comfort of their own terminal ("Let me show you something..."). The question would always come up: "How do you stop them?" "You can't" "Yes, but how do you stop them?" Crabs could be downloaded remotely, on somebody else's terminal, while he was working. They could be left dormant (Rob's idea) during the lunch hour, to suddenly come up in the middle of the afternoon. They could be timed to start in the middle of an important demonstration. Once, Rob got them to eat (irrecoverably) part of a picture an artist was drawing on a Blit. The artist was offended, not by the damage picture, but by such inexplicable violation of what she considered to be laws of nature. Very soon, nobody could pass by Bell Labs without being exposed to crabs.

Programs were written to fight crabs on their own grounds. The idea was to run a program which would neutralize the crabs and allow you to keep working, without rebooting the terminal. Those program were either unsuccessful, or partially neutralized the crabs but made the Blit practically unusable. One day we got a program in the mail, called *squishcrabs*. It would poke the process table looking for a process which looked like crabs, and kill it. On top of that it would *squish* every crab on the screen to a black blob. That was cheating, but it worked. However, squishcrabs was too dependent on the process and program structure, and stopped working in later versions of the system.

In the following months Mark and I wrote many crab-like programs. Although interesting in their own way, none came close to the appeal crabs have. The best use we have for them is to make them fight overnight against crabs for screen territory, and watch the result in the morning. Crabs are still undefeated; they either wipe out the opposition, or come to a stable situation with crabs in one region of the screen and opponents in the other.

## Crab Rules

**1.** Crabs live on grey screen areas.

**2.** On grey areas they move around randomly, but smoothly. The orientation of the crab icon is determined by its direction of movement, so that they always appears to move sidewise.

**3.** When they bump into non-grey areas (including other crabs) they *bite* them by changing a little non-grey region into a grey region. After that they bounce off in a new random direction.

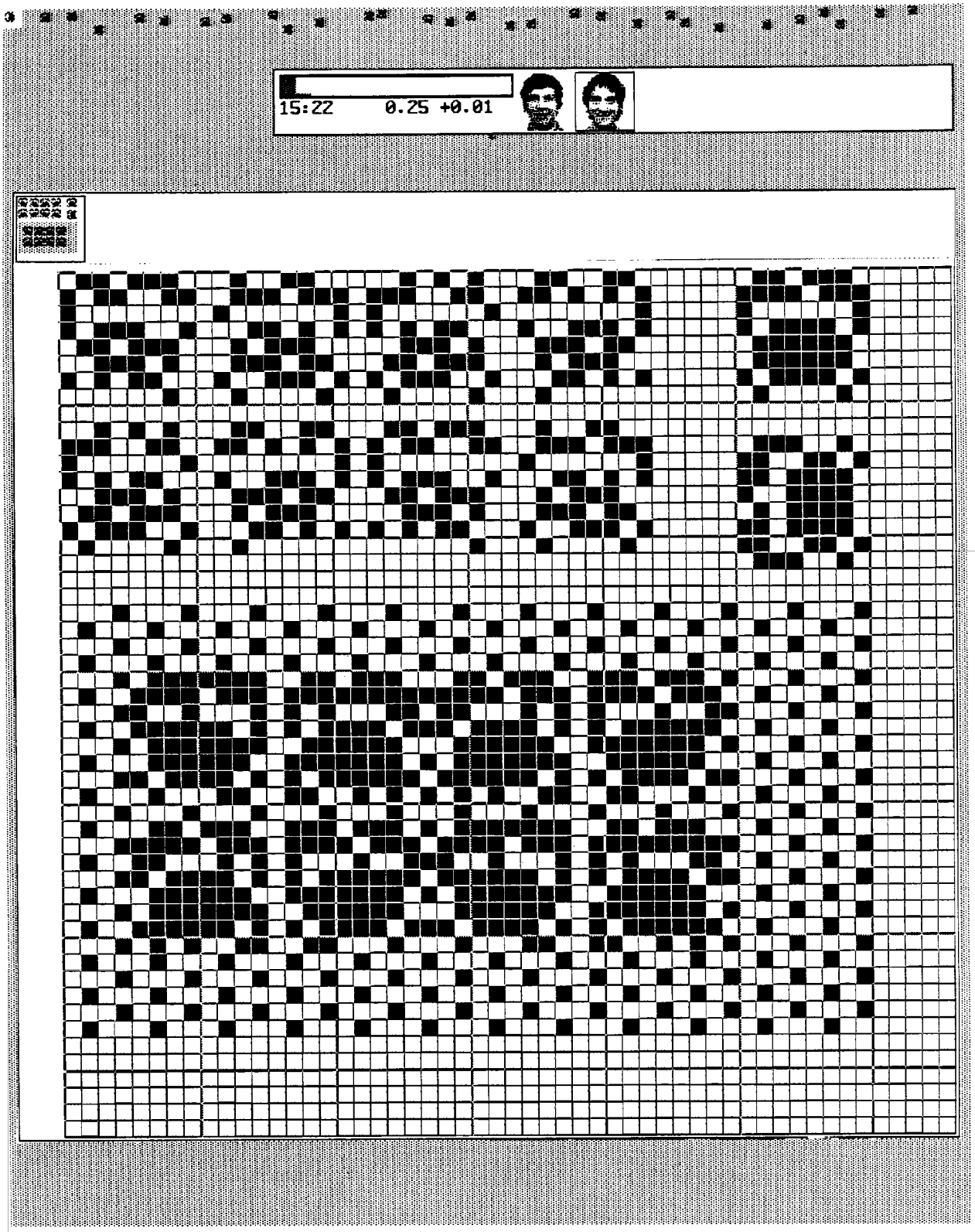
The crab-like (or insect-like) random motion on grey areas is obtained as follows. Every crab step is, in first approximation, determined by the current velocity. Every step has a probability (e.g. one in seven) of being subject to a deviation. If the deviation takes place, it is a small random perturbation (e.g. -1, 0 or +1) of the current velocity, independently chosen for the x and y components. There is a maximum speed (e.g. 7 pixels per step).

Every crab does the following:

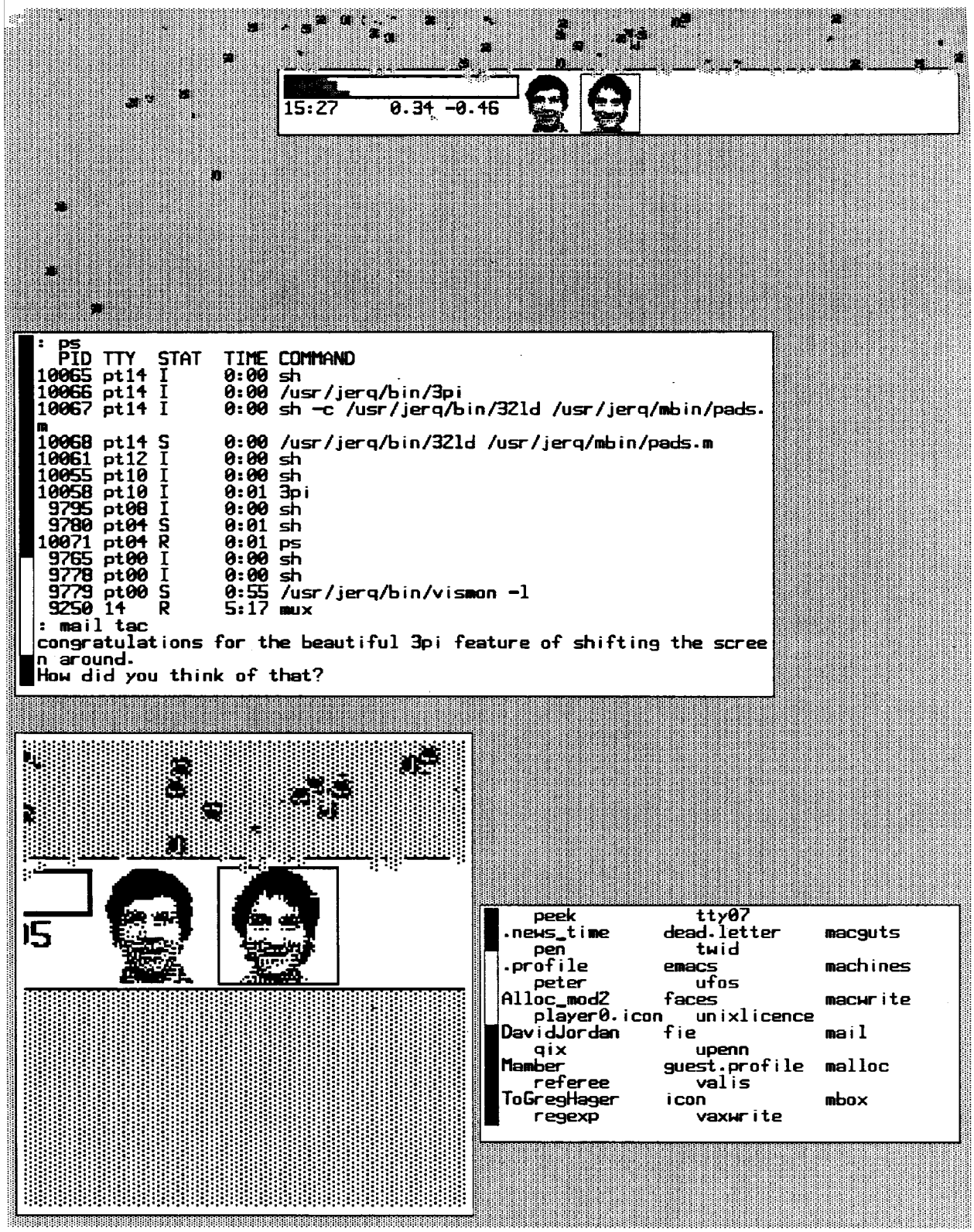
**0.** Draws itself in the initial position. Starts with a random direction and velocity.
**1.** Removes itself from the old position (by drawing itself in XOR mode).
**2.** Determines its new position, based on its current direction and velocity.
**3.** Looks to determine whether it is about to move on a grey area:
  Yes:
    **3.1.** Moves there. Goes to 4.
  No:
    **3.2.** Makes the new position grey by drawing a 4x4 grey pattern.
    **3.3.** Does not move. Picks a new random velocity, independent of the current velocity. Continues at 4.
**4.** Draws itself (in XOR mode) in the new position, as determined in 3.1 or 3.3.
**5.** Adds a random deviation to its velocity, as described above.
**6.** Back to 1.

Crab icons must be drawn in XOR mode, to be able to restore the background when the crab moves away. Unfortunately, if one draws a crab icon in XOR mode on a gray background, the crab itself gets *greyed*.To avoid that, crab icons are prepared so that they will look right when greyed. This is done by greying them beforehand (two XOR greying operations cancel) in all possible relative positions of the crab and the grey background. For the grey pattern we use, which repeats every two pixels vertically and every four pixels horizontally, there are 8 possible relative positions.

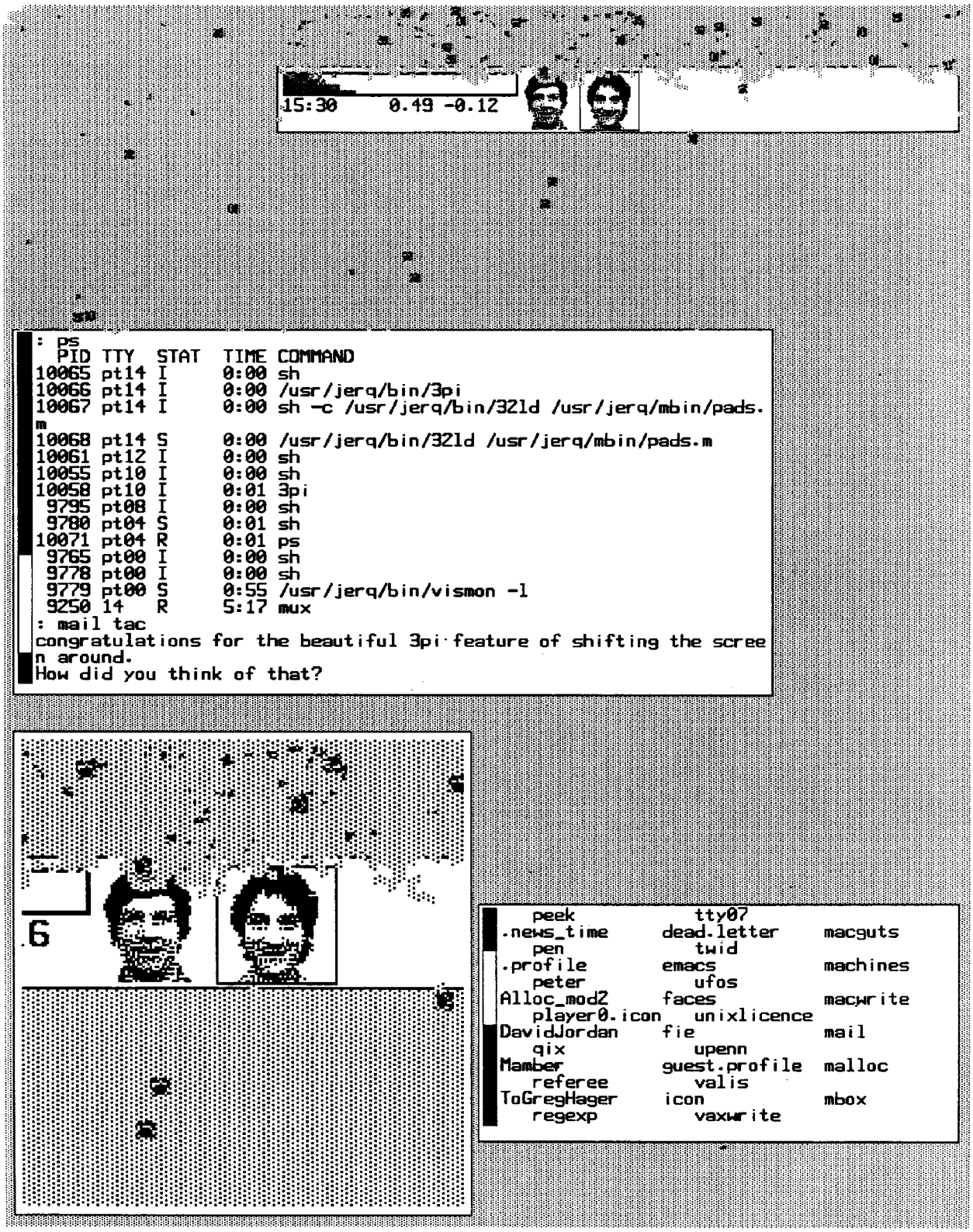
Some of the black pixels of the background immediately adjacent to a crab icon *stick* to it, visually. Depending on the speed of movement, this produces an optical illusion so that the crab legs appear to move.



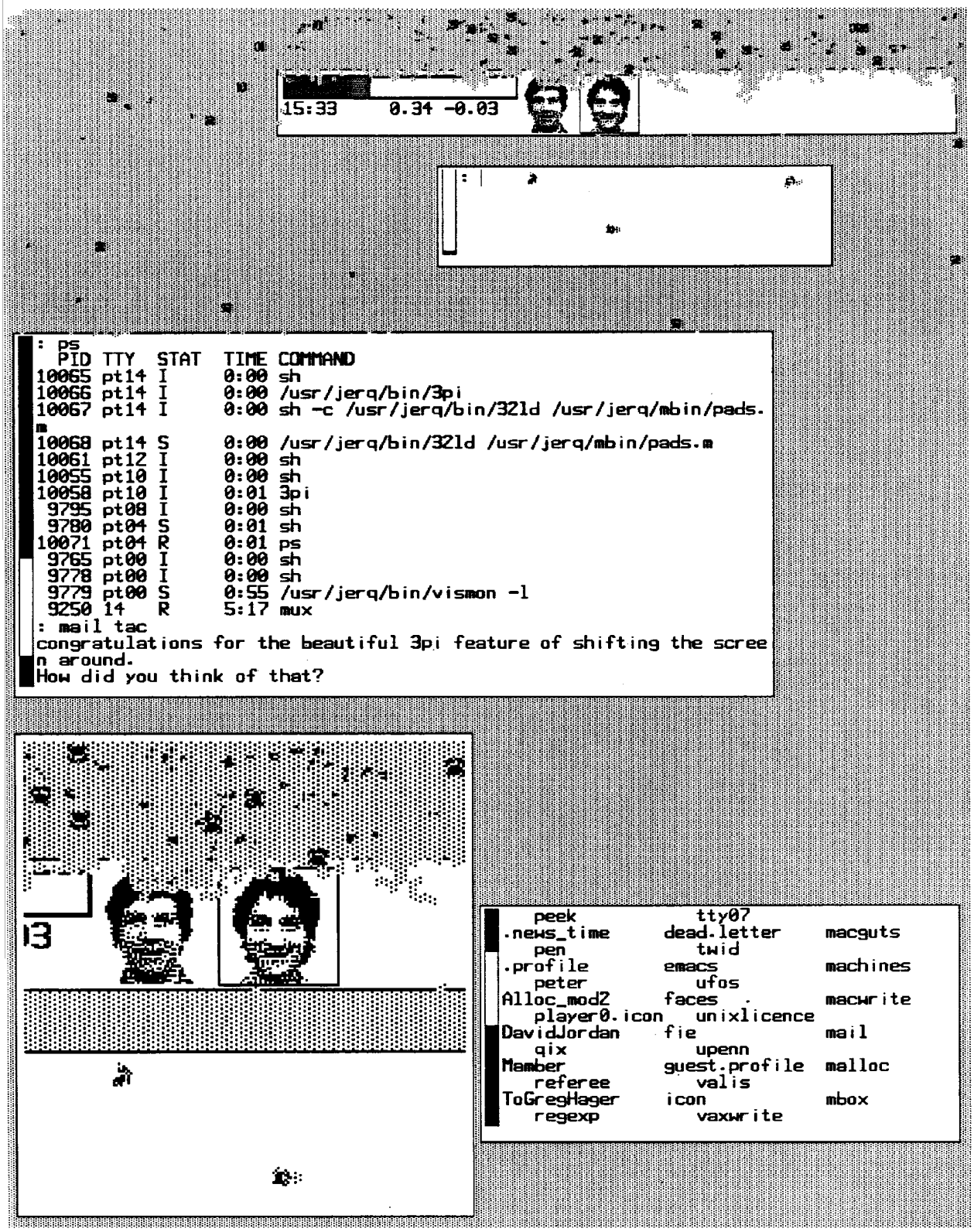

Thirty crabs start at the top, threatening the window with pictures of me and mark.
Center: magnified crabs on grey, in all possible displacements w.r.t. the background.



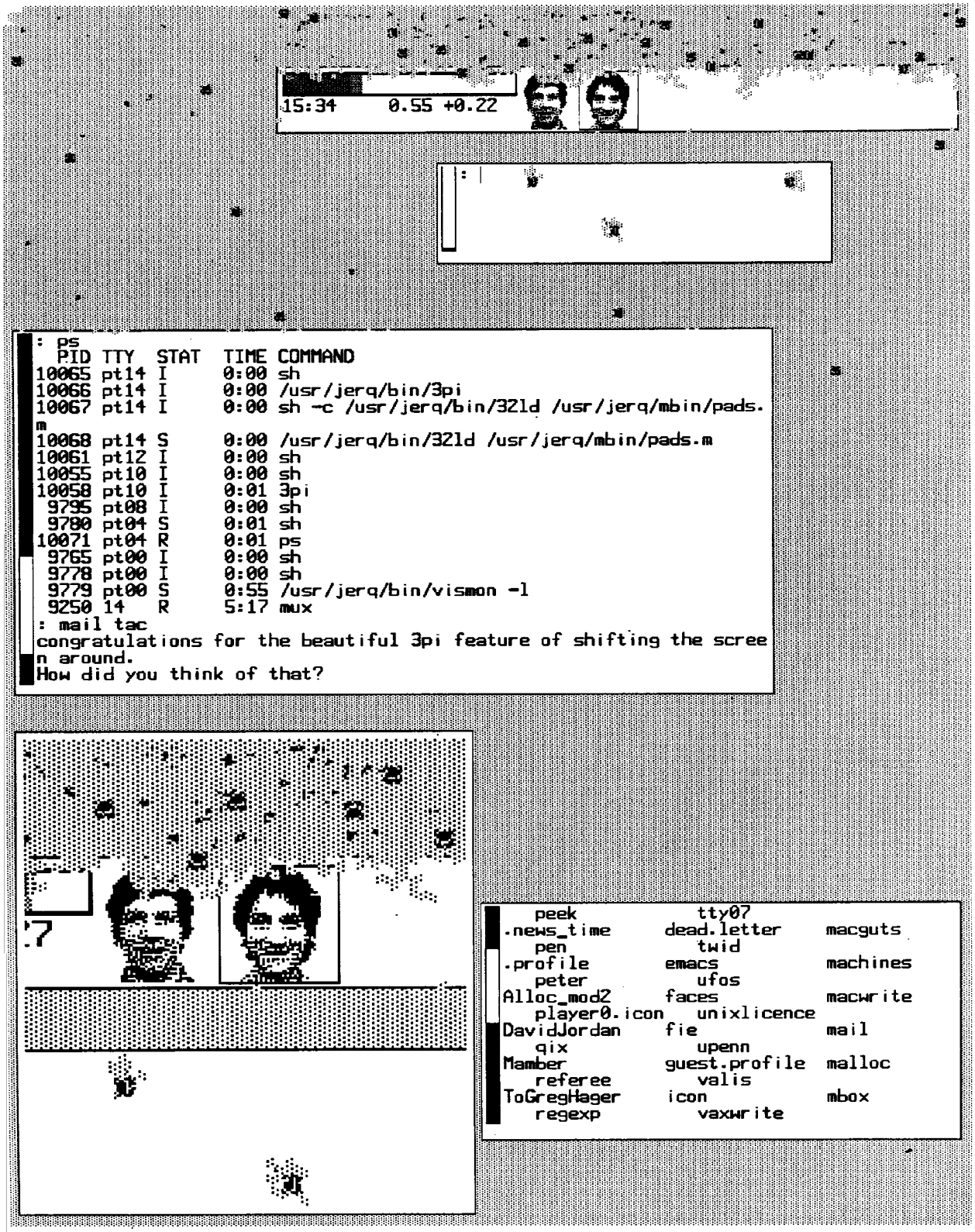

Crabs start eating the top window.
On the bottom left there is a "lens" window magnifying an area at the top of the screen.



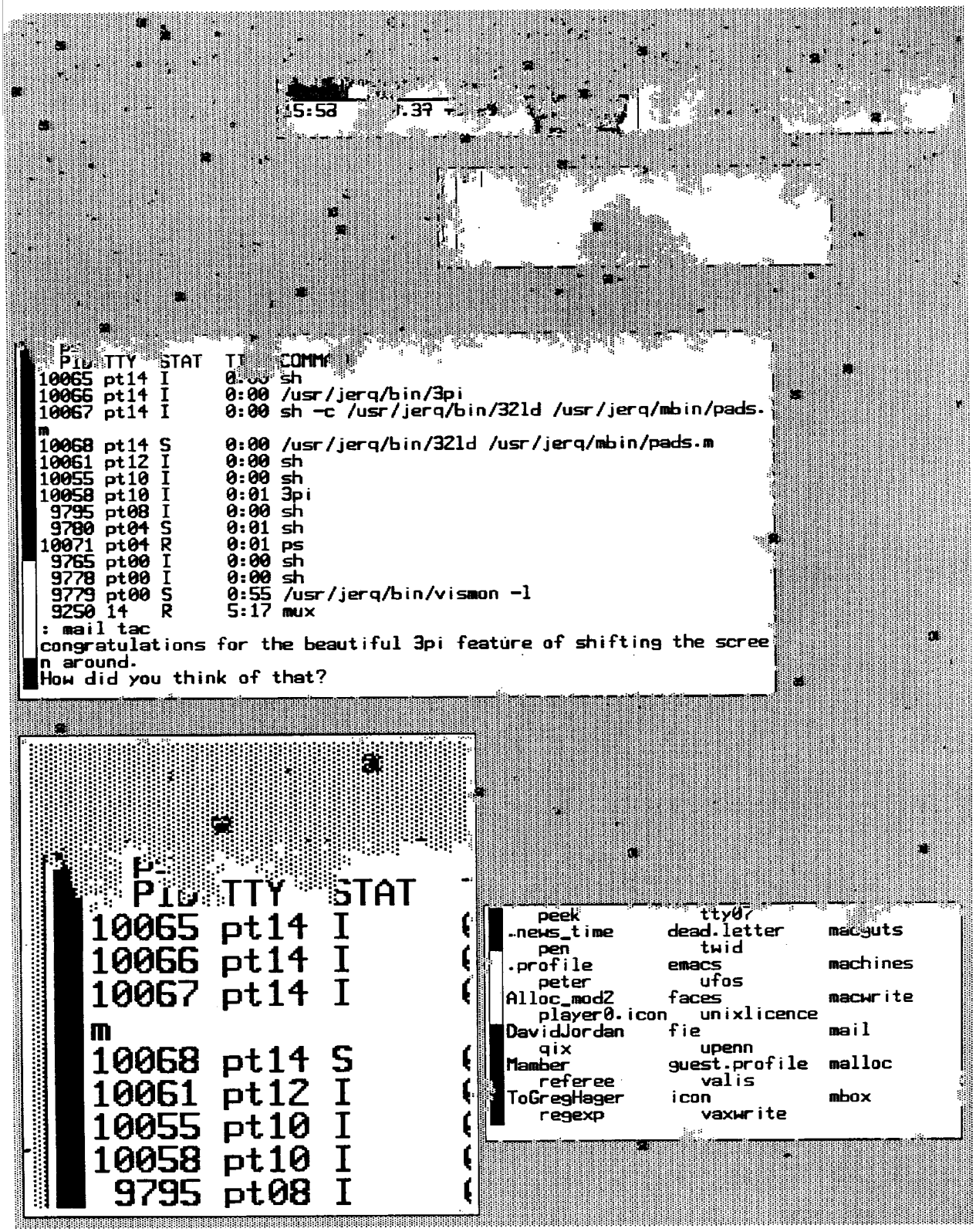

Killer crabs start eating their authors.
The top part of the screen is full of *crab-dirt* (a by-product of crab collisions).



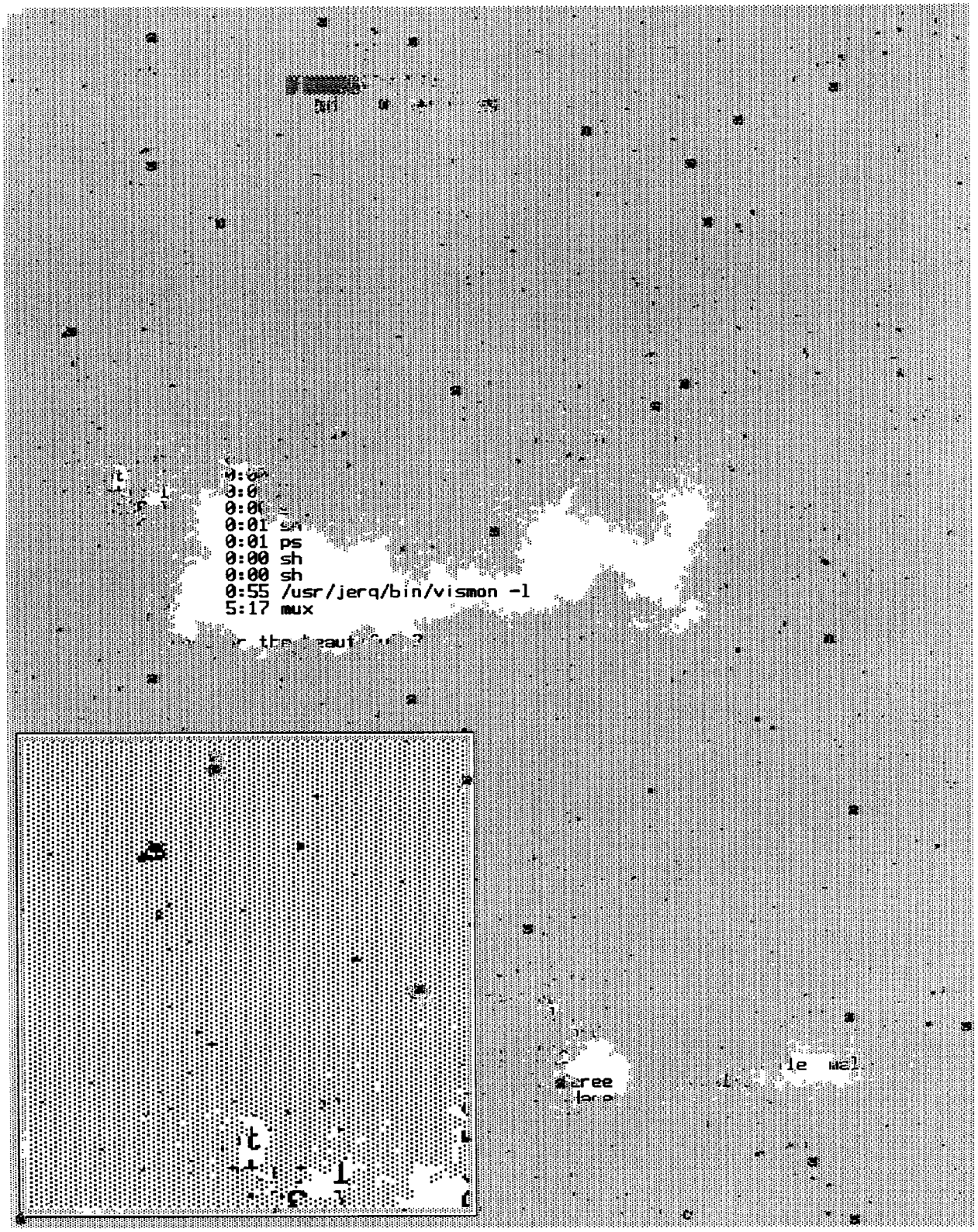

A new window is placed on top of three crabs.
The crabs start eating the window from underneath.




The three trapped crabs are now almost fully visible. This effect of *eating from underneath* is a totally unexpected non-obvious consequence of the crabs drawing algorithm.




The lens has been moved to show text being eaten away.



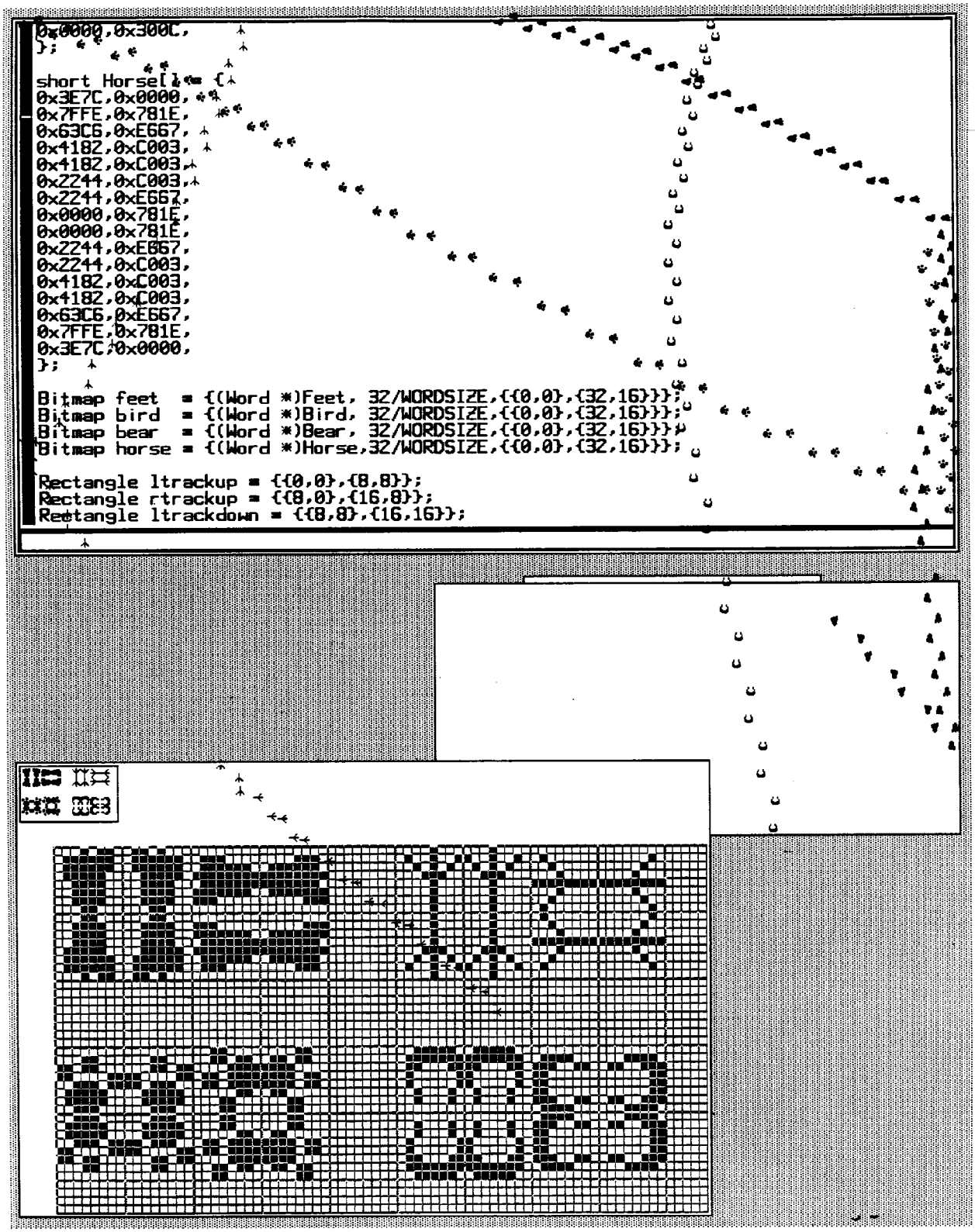

Eventually, crabs dominate the screen.
The lens window is almost unharmed because it regenerates.

## Tracks

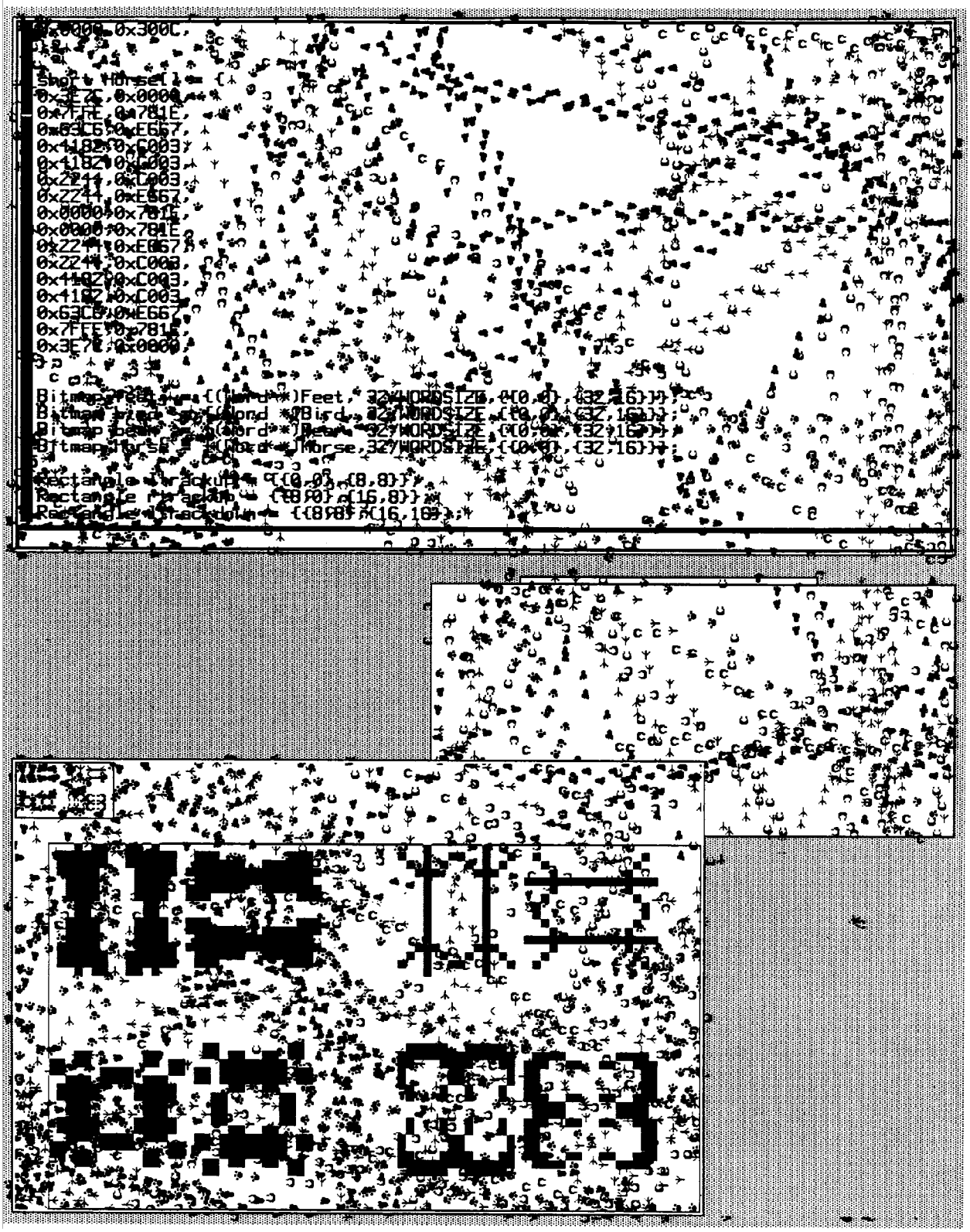
A few other crabs-related programs deserve mention. *Tracks* gives the illusion of invisible creatures walking on your windows and leaving footprints. There are cats, birds, unicorns and little people.

Tracks are not left on the background, only on windows. However tracks can cross grey regions and continue on another window. The random motion is obtained as in crabs, with slightly different parameters.

Tracks was written by myself as a crabs spin-off.




A few tracks appear.
Bottom left: magnified track icons.



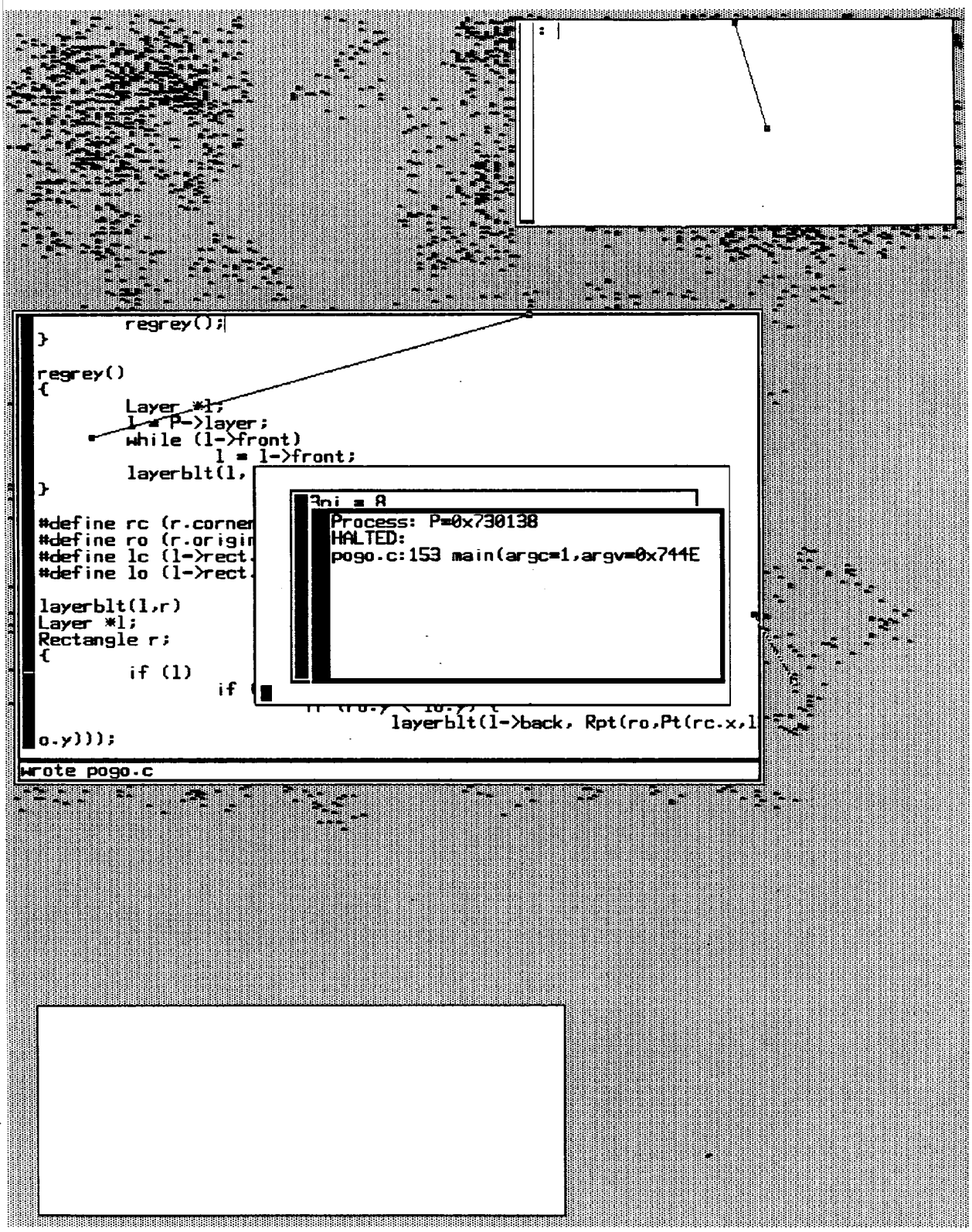
Many, many more tracks.

## Pogo Sticks

*Pogo-sticks* is another crabs-related program, written by Mark Manasse. A pogo stick is a pair of bouncers (dots) connected by a stick (line). Pogo sticks (or "pogos", for short) hate grey, and love any other color. In non-grey areas, the bouncers float freely, until they bump into grey areas. When that happens, the bouncers change direction and bounce off. The opposite bouncers of a pogo loosely attract each other.

A bouncer may overshoot a boundary between non-grey and grey, because of inertia, and get temporarily trapped in a grey area. In this situation the bouncer is continuously bouncing against grey, and assumes a kind of brownian motion. Fortunately, the opposite bouncer will very likely pull it out of trouble. If both bouncers are trapped in grey, the pogo may randomly wander in grey areas for a long time, looking like it is in an epileptic fit. A pogo in a large white space (like a window) tends to stay there; it is unlikely that it will have both bouncers outside at the same time and on the same side, so that they can wander off. It is likely that pogos will eventually migrate to the largest window available, and stay there most of the time.

When a bouncer is trapped in grey, it tries actively to make itself a home by turning all the grey it touches into black. Eventually this can create large black areas where the pogo can again float free.




Three pogos started at the top. Two of them have migrated to the lower window.

## Screen Wars

Different screen organisms, like crabs, tracks and pogo sticks, can interact in interesting ways when run concurrently. Each screen organism lives on some kind of screen territory and attacks some other kind of screen territory. When many organisms are present at the same time, they may fight for territory. An organism *wins* if it ends up controlling the whole screen and the other organisms loose all their natural territory. More often, some kind of equilibrium is reached and the screen is divided into domains of influence.

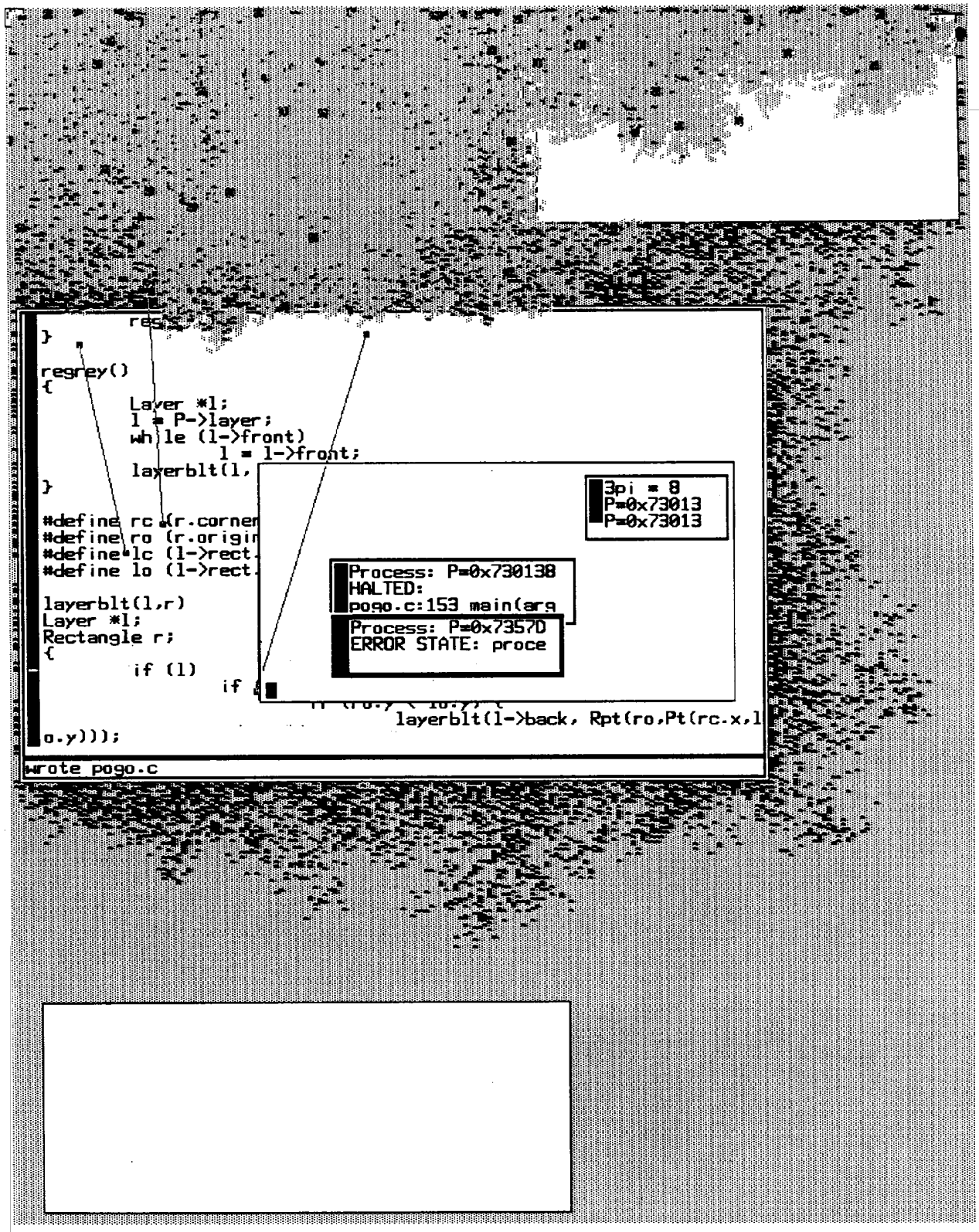
### *Crabs vs Pogos*

The following pictures show a 12-hour fight of crabs against pogo sticks for control of the screen. Eventually equilibrium is reached.

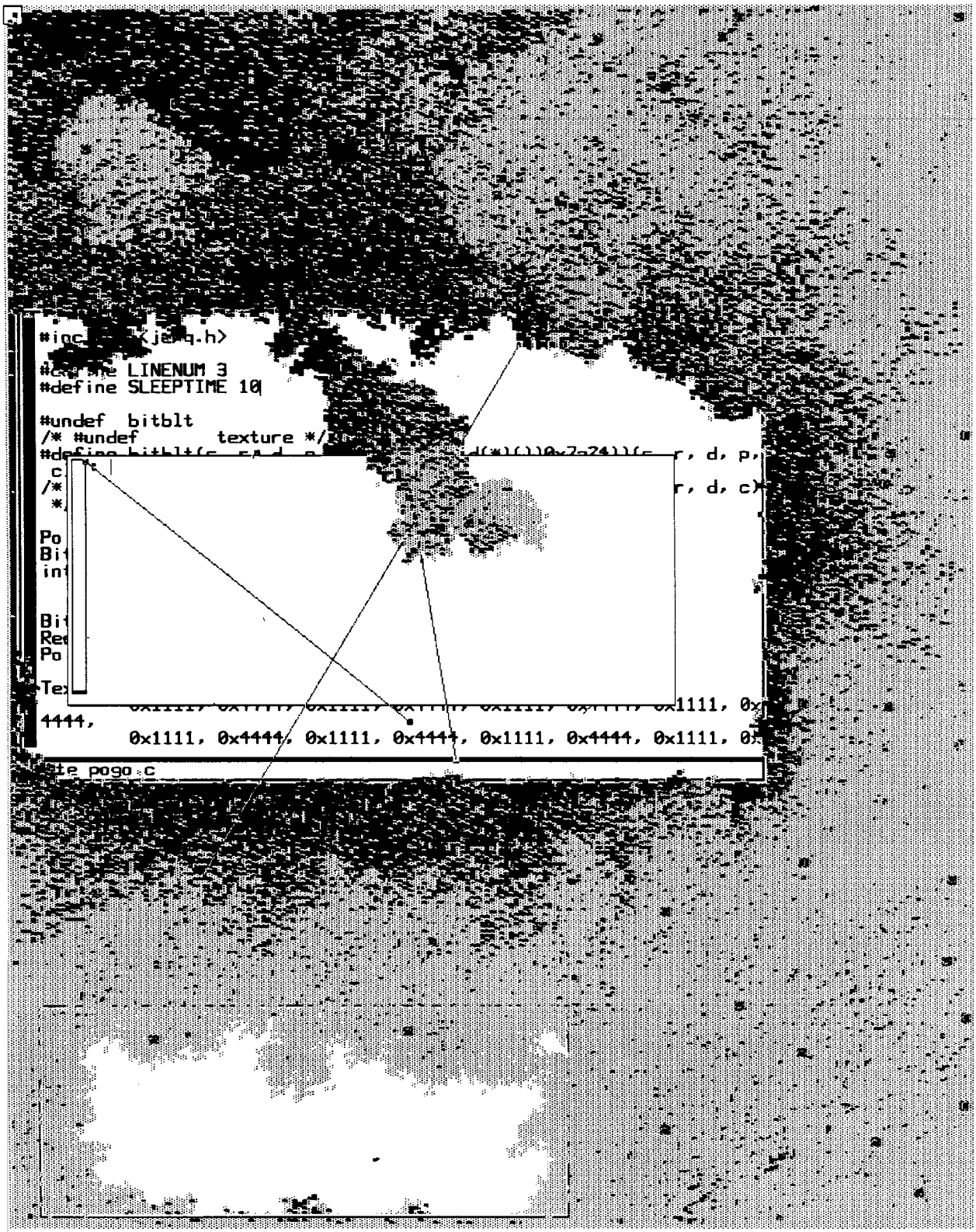
### *Other fights*

Tracks do not stand a chance against crabs, because tracks do not attack crab territory (unless they happen to step on a crab, in which case they leave a footprint there, but this is infrequent), while crabs attack tracks territory. Eventually, tracks loose their "footing".

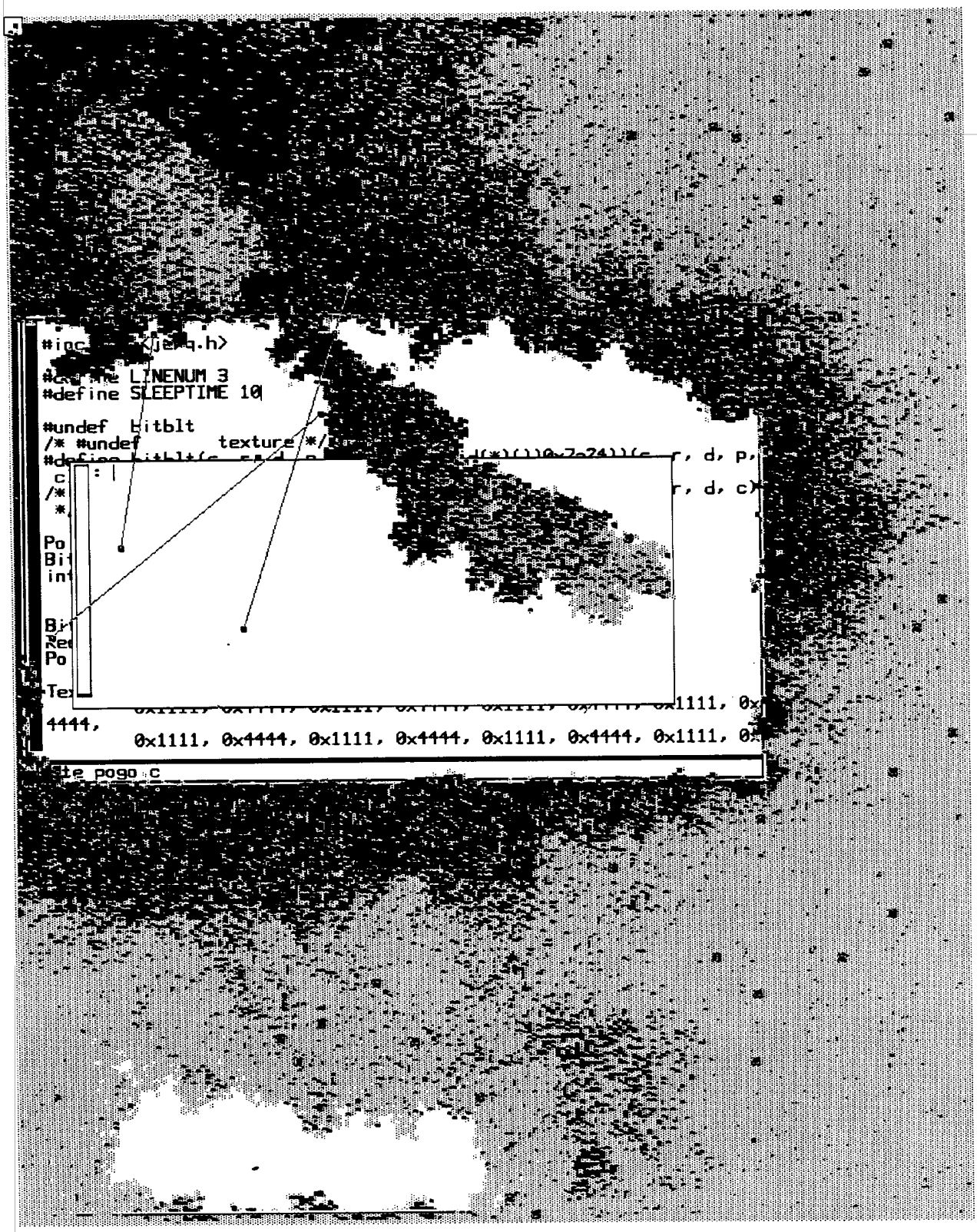
Tracks and pogos cooperate, and the result is a totally black screen.



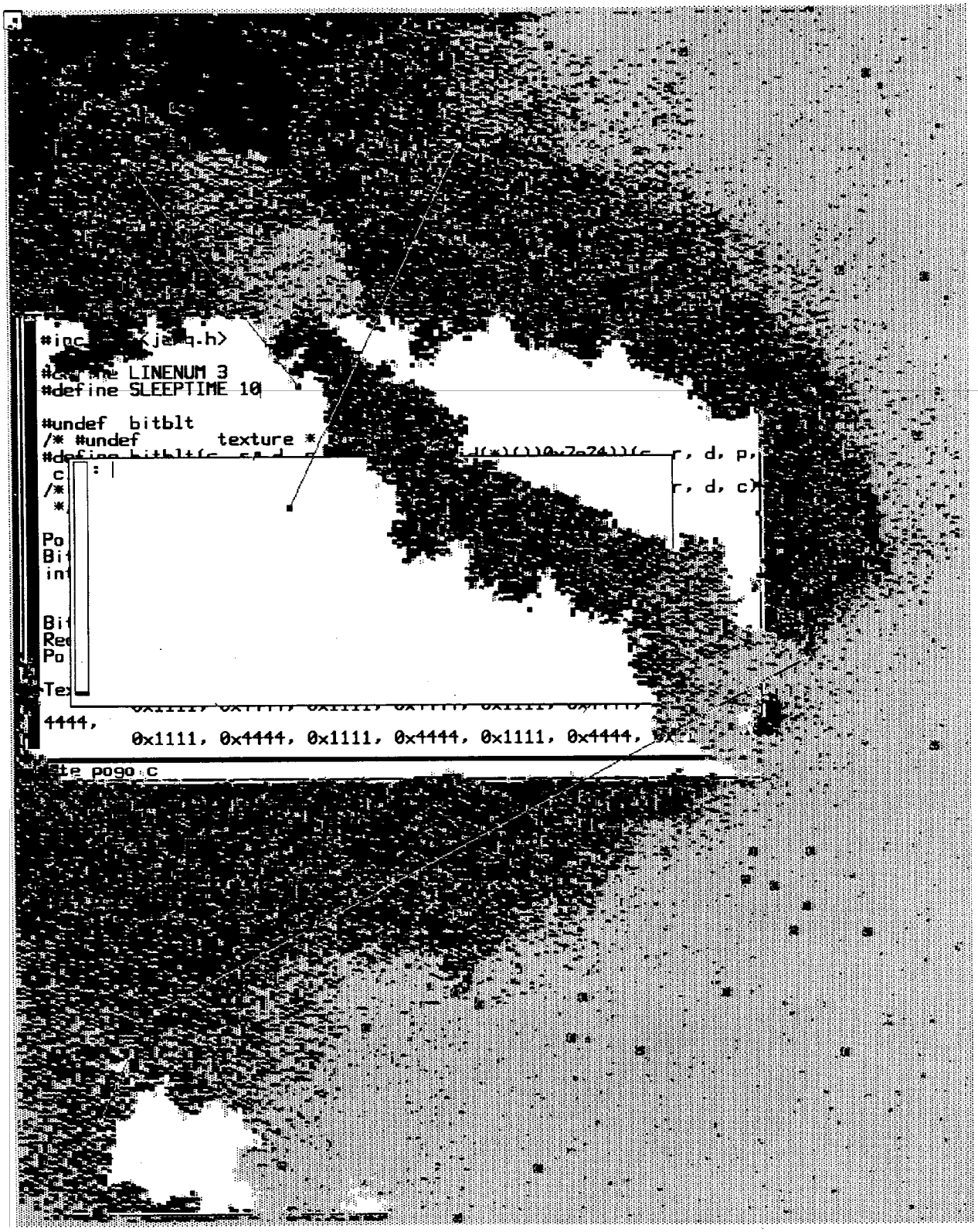

The crabs have eaten half of the upper window and have attacked the large window. Meanwhile the pogos have sprinkled black at the perimeter of the large window.



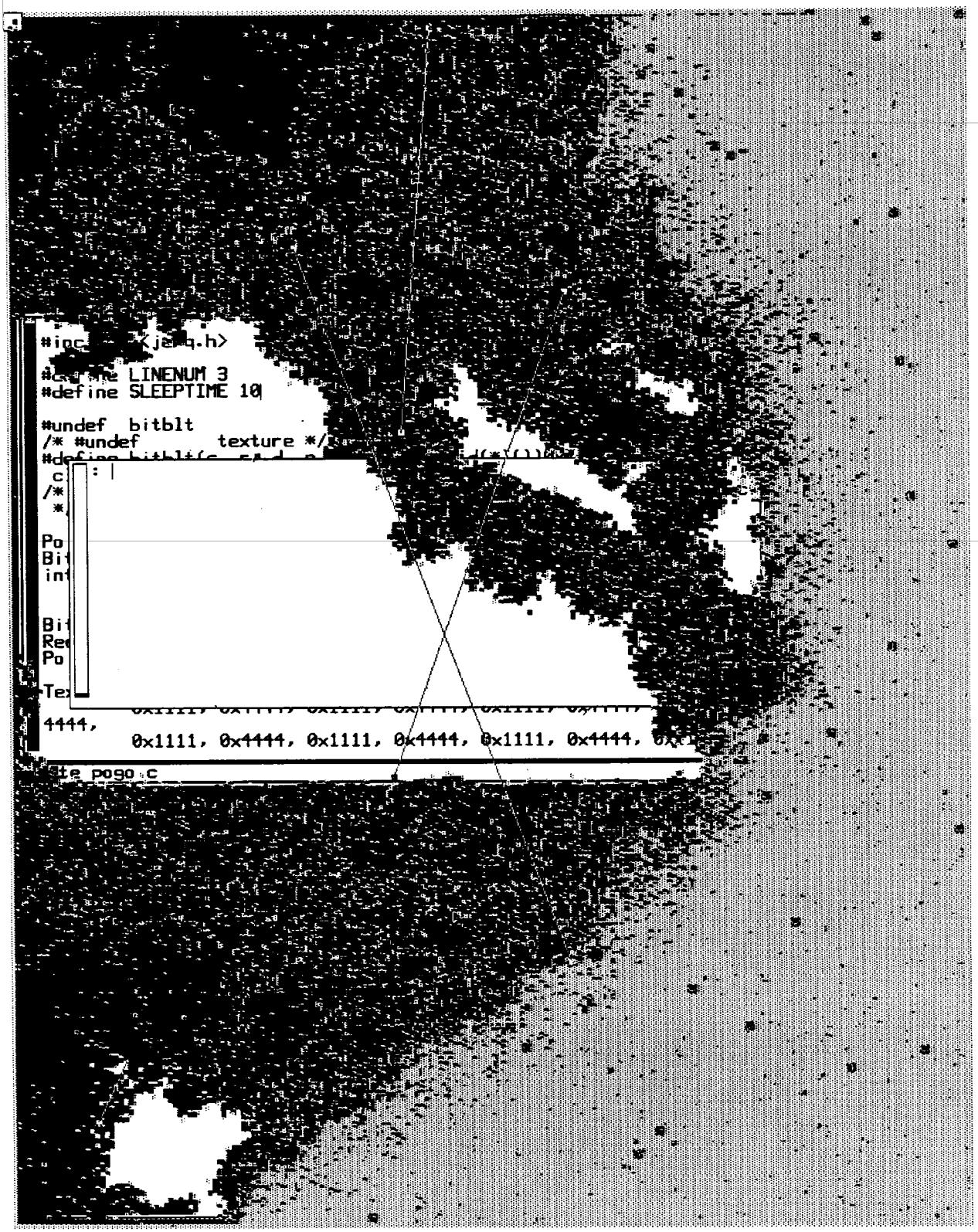

Pogos have control of the center and top left of the screen, except for a crab trapped in the top left which maintains its own grey territory, and another crab which goes deeper in the large window.




The crabs have control of the bottom and right side of the screen, which is now full of crab-dirt. The prisoner crabs keep working on their escape corridors. The pogos slowly gain territory.




One of the prisoners has escaped. It is not clear why the prisoners move almost coherently in one direction; maybe there is a slight bias in the random walk algorithm. (At this point I went to sleep)




Seven hours later the situation hasn't changed much. The other prisoner crab has escaped. The boundary between crabs and pogos is sharp and stable. The whole process took 12 hours.

MH-11271-LC-unix L. Cardelli

Atts.
References (1-2)
Appendix (I)

MH-11271-LC-unix L. Cardelli

Atts.
References (1-2)
Appendix (I)

**NAME**

crabs — graphical marine adventure game

**SYNOPSIS**

**crabs** [ **−i** ] [ **−s** duration ] [ **−v** velocity ] [ *number* ]

**DESCRIPTION**

In *crabs*, difficult situations are encountered in trying to kill or capture crustaceans swarming in a murky sea. You will have to work very rapidly to keep your territory free of seabed intruders. At first, you may even find it hard to keep a clear view of your surroundings, but later discoveries about the spirit of the game will suggest a solution.

There are several options.

**−i** causes the intruders to play intelligently, allowing them to avoid detection.

**−s** simplifies the game for the first *duration* time intervals. Default is 0. 5-10 is recommended for beginners, although you may want to forgo this option the first time, just to see how interesting it can get.

**−v** adjusts the velocity of the crabs, 1 being fastest. Default is 5.

*Number* specifies the number of intruders. Default is 30.

**FILES**

/usr/jerq/mbin/crabs.m — terminal program

**CRUSTACEANS**

Can be frustrating.

# Crab Genesis

## Introduction: laws and violations.

A bitmap screen is a graphic universe where windows, cursors and icons live in harmony, cooperating with each other to achieve functionality and esthetics. A lot of effort goes into making this universe consistent, the basic law being that every window is a self contained, protected world. In particular: (1) A window shall not be affected by the internal activities of another window. (2) A window shall not be affected by activities of the window system not concerning it directly, i.e (2.1) it shall not notice being obscured (partially or totally) by other windows or obscuring (partially or totally) other windows, (2.2) it shall not see the *image* of the cursor sliding on its surface (it can only ask for its position).

Of course it is difficult to resist the temptation of breaking these rules. Violations can be destructive or non-destructive, useful or pointless. Useful non-destructive violations include programs printing out an image of the screen, or magnifying part of the screen in a "lens" window. Useful destructive violations are represented by the "pen" program, which allows one to scribble on the screen. Pointless non-destructive violations include a "magnet" program, where a moving picture of a magnet attracts the cursor, so that one has to continuously pull away from it to keep working. The first pointless, destructive program we wrote was crabs.

The history of crabs is presented here with dates, times and people. Not that we kept notes, of course. The dates and times were reconstructed months later by looking at the creation date of files, and by what we could remember.

## Prologue: Peek

Crabs was written by Mark Manasse and me in November 1982, and evolved in about two days to its present form. The basic principles of law-violation where investigated a few months earlier (August 5, 1982) when Bart Locanthi brought in a Smalltalk videotape. It featured, among other things, a "peek" demo. This is a program which looks at a rectangular portion of the screen (controlled by moving the cursor around) and replicates it in its own screen space in real time. Beautiful self-referential effects are obtained when this window peeks itself, or part of itself. This is a digital version of a video-camera looking at its own tv screen.

Copying data from another window, as peek does, can already be considered a violation of the rules. But what peek does is even worse because, for a given window, peek will only copy that part of the window which is visible on the screen (i.e. not obscured by other windows). This cannot be done by asking a window to access its data: a window is not aware of what parts are visible. This is stealing data directly from the screen. A well-structured graphics interface will not allow this, and one has to use low-level routines which are not-meant-to-be-used-by-normal-people. Needless to say, Bart and Mark rushed to implement it.

Step 1: QIX

(November 16, 1982, dinner time) Mark wanted to implement the QIX video game for our blit terminals (knowledge of QIX is assumed here). A QIX screen can get very complicated, and there are complex rules about how things are allowed to move. Mark started figuring out clever data structures and algorithms to compute fast line operations. After a while I said, "Wait a second. Atari is selling arcade QIX machines and there is no way they can have enough memory to run those algorithms. How are they doing it?" After some thinking: "I bet they don't keep line segments in data structures, but they draw lines on a bitmap and (gosh!) they just look at what is in the bitmap to determine line intersections. Gee, this is awful." Although this was repulsive to our trained algorithmic minds, that was the germ of the crabs collision-detection trick. We never implemented QIX.

Step 2: Measles

(November 16, later) After a while Mark was convinced and we started implementing. We decided to start with a single QIX (i.e. a single line with two bouncing dots at the ends) for simplicity, and to use window boundaries to test the line intersection trick. Mark started dictating code and I typed it down. This was still a bit too hard, so we simplified it further: forget the QIX, let's just have little balls floating in the grey area between windows and bouncing against window borders. We would look at the raw screen bits to determine where a window border was (is there grey there?). Mark kept dictating, and after a while it was working. It was just about one page of code. Mark called this "measles"; we had a lot of measles bouncing around the screen. They were also bouncing off each other for free because they would see non-grey and change direction. This was very cheap and convenient: normally you would have to test the position of every measle against the position of every other measle to determine whether there is a collision.

Step 3: Angry Measles

(November 17, very early) Now a problem came up. We have all these measles bouncing around, and you create a new window and slap it on top of them. Suddenly those poor trapped measles have nowhere to go, no grey area to run to. They are frozen, paralyzed with terror, and buried underneath a window. Mark didn't like that at all, and came up with the concept of "angry measles". When a measle gets buried underneath a window, it starts flashing so that it is visible through the window, like saying "Hey, get that window off me". It turns out that little flashing things are very annoying to the human eye, and you would take the window away just to shut them up. At this point, tired and satisfied, we went to sleep.

Step 4: Hungry Measles

(November 17, late morning) I slept a lot less then Mark did. When I came in, I started showing measles to people. They though it was cute stuff. Some objected to the flashing measles solution. We had considered many alternatives the night before, and I wasn't totally satisfied with that

solution either. Dave MacQueen said something like "they should eat their way out". I thought that was a possibility, only sillier than most. After he left, however, that idea kept coming back. I went to look at the code (as I said, Mark did the dictating because he was more familiar with blits then I was), and discovered that I could implement Dave's suggestion by changing a single line of code. That seemed to be easy enough, so I did it. When a measle was confronted with a non-gray area, it would change a little bit of that area to grey. Trapped measles could then build up grey regions and eventually escape.

The new version "hungry measles" had quite a different character. It wasn't cute, it was awesome. Those little balls would eat away your windows. If trapped, they would escape, leaving you wounded. There was no protection against them. You could set up barricades of windows to protect a part of the screen you wanted to work in, and they would erode them. They would infiltrate along the borders of the screen, where you are not allowed to put windows. You couldn't keep them all under control: they were too many, too quick. You couldn't get distracted.

Step 5: Crabs

(November 17, afternoon) I went up to the machine room and started the program on a terminal. People gathered, and several expressions of disgust were heard. Jim Weythman said "they look like crabs!".

Everybody knew instantly that that was the right name for it. I went back to my room and designed the basic crab icon. Mark came back. With his help, we prepared the crab icon so that it would look nice on a grey background. We made it so that crabs would move sidewise, and would turn around according to their prevalent direction. We made the crabs window self-destruct so that there was no way of stopping crabs, short of rebooting the terminal. Finally, we allowed the crabs to see the image of the cursor on the screen, so that you could use the cursor to poke them (they would bite it, but the cursor regenerates). We showed it to Rob Pike again. He said "That's it, don't touch it any more".

Conclusions

In the next few days, unaware people were exposed to crabs in the comfort of their own terminal ("Let me show you something..."). The question would always come up: "How do you stop them?" "you can't" "yes, but how do you stop them?". Crabs could be downloaded remotely, on somebody else's terminal, while he was working. They could be left dormant (Rob's idea) during the lunch hour, to suddenly come up in the middle of the afternoon. They could be timed to start in the middle of an important demonstration. Once, Rob got them to eat (irrecoverably) part of a picture an artist was drawing on a blit. The artist was offended, not by the damage picture, but by such inexplicable violation of what she considered to be laws of nature. Very soon, nobody could pass by Bell Labs without being exposed to crabs.

Programs were written to fight crabs on their own grounds. The idea was to run a program which would neutralize the crabs and allow you to keep working, without rebooting the terminal. Those program were either unsuccessful, or partially neutralized the crabs but made the blit

practically unusable. One day we got a program in the mail, called "squishcrabs". It would poke the process table looking for a process which looked like it may be crabs, and killed it. On top of that it would "squish" every crab on the screen to a black blob. That was cheating, but it worked. However, squishcrabs was too dependent on the process and program structure, and stopped working in later versions of the system.

In the following months Mark and I wrote many crab-like programs. Although interesting in their own way, none came close to the appeal crabs have. The best use we have for them is to make them fight overnight against crabs for screen territory, and watch the result in the morning. Crabs are still undefeated; they either wipe out the opposition, or come to a stable situation with crabs in one region of the screen and opponents in the other.

Luca Cardelli

# Crabs

Rules:

(1) Crabs live on grey screen areas.
(2) On grey areas they move around randomly, but smoothly.
The orientation of the crab icon is determined by its direction of movement, so that they always appears to move sidewise.
(3) When they bump into non-grey areas (including other crabs) they "bite" them by changing a little non-grey region into a grey region. After that they bounce off in a new random direction.

The crab-like (or insect-like) random motion on grey areas is obtained as follows. Every crab step is, in first approximation, determined by the current velocity. Every step has a probability (e.g. one in seven) of being subject to a deviation. If the deviation takes place, it is a small random deviation (e.g. -1, 0 or +1) of the current velocity, independently chosen for the x and y components. There is a maximum crabs speed (e.g. 7 pixels per step).

Every crab does the following:

0. Draws itself in the initial position.
Starts with a random direction and velocity.

1. Removes itself from the old position (by drawing itself in XOR mode).

2. Determines its new position, based on its current direction and velocity.

3. Looks to determine whether it is about to move on a grey area:

Yes: 3.1. Moves there. Goes to 4.

No: 3.2. Makes the new position grey by drawing a 4x4 grey pattern.

3.3. Does not move. Picks a new random velocity, independent of the current velocity. Continues at 4.

4. Draws itself (in XOR mode) in the new position, as determined in 3.1 or 3.3.

5. Adds a random deviation to its velocity, as described above.

6. Back to 1.

Note: Crab icons must be drawn in XOR mode, to be able to restore the background when the crab moves away. Unfortunately, if one draws a crab icon in XOR mode on a gray background, the crab itself gets "greyed".
To avoid that, crab icons are prepared so that they will look right when greyed. This is done by greying them beforehand (two XOR greying operations cancel) in all possible relative positions of the crab and the grey background. For the grey pattern we use, which repeats every two pixels vertically and every four pixels horizontally, there are 8 possible relative positions.

Note: Some of the black pixels of the background immediately adjacent to a crab icon "stick" to it, visually. Depending on the speed of movement, this produces an optical illusion so that the crab legs appear to move.

Luca Cardelli

Crabs was written by myself and Mark Manasse on November 16 and 17, 1982.

Figures:

(Figure Crabs.0)
Top Right: crab icons in two orientations.
Top Left: greyed-out crabs.
Bottom: upward-looking crabs on grey, in all possible relative displacements w.r.t the background.

(Figure Crabs.1) Thirty crabs start at the top of the screen, threatening the top window with pictures of me and Mark.

(Figure Crabs.2) Crabs start eating the top window. On the bottom left there is a "lens" window magnifying an area at the top of the screen.

(Figure Crabs.3) Killer crabs start eating their authors. The top part of the screen is full of crab-shit, a by-product of crab collisions.

(Figure Crabs.4) A new widow is placed on top of three crabs. The crabs start eating the window from underneath.

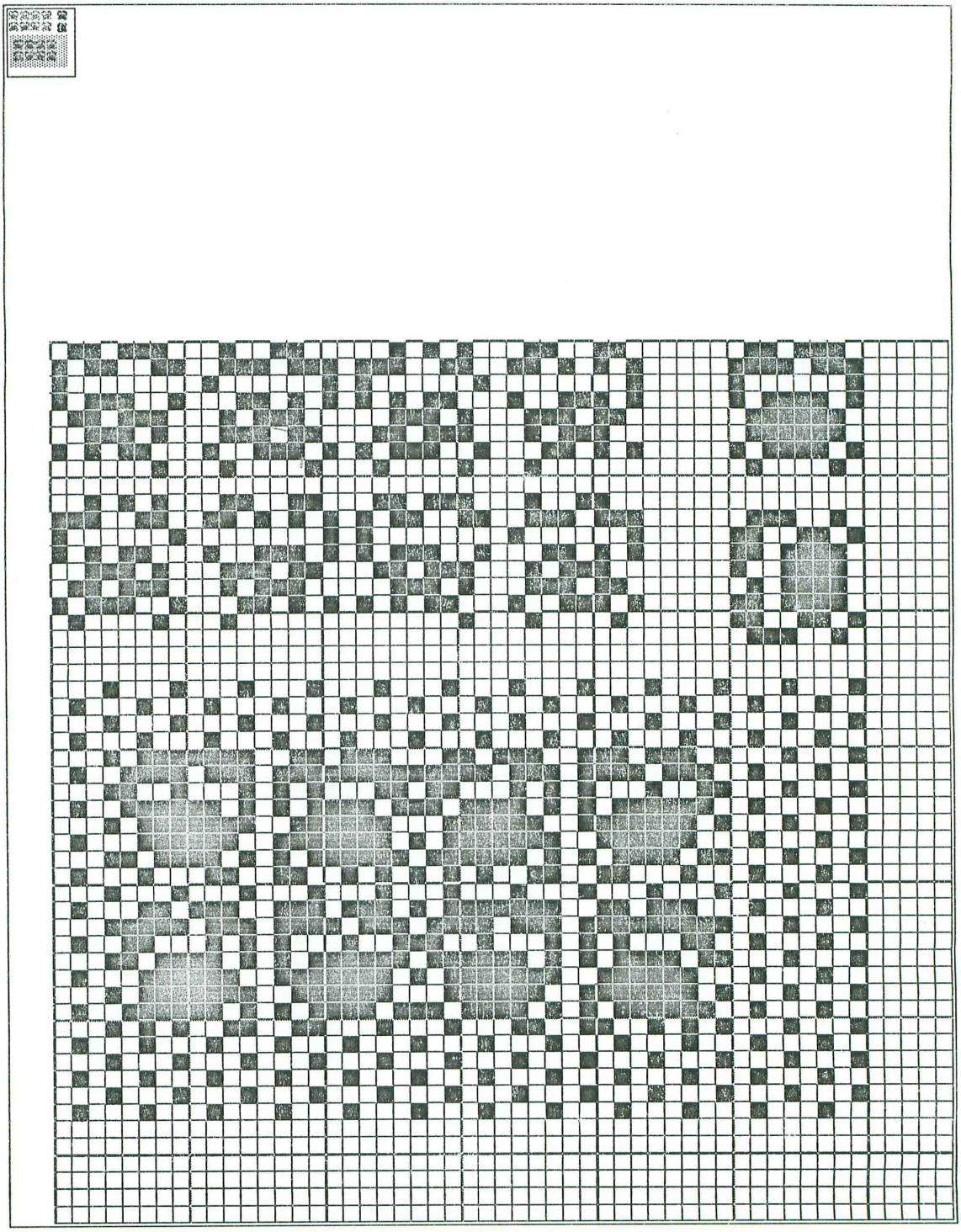
(Figure Crabs.5) The three trapped crabs are now almost fully visible. This effect of "eating from underneath" is a totally unexpected non-obvious consequence of the crabs drawing algorithm.

(Figure Crabs.6) One of the trapped crabs breaks loose.

(Figures Crabs.7 .. Crabs.9) More scenes of cannibalism and destruction.

(Figure Crabs.10) The lens has been moved to show text being eaten away.

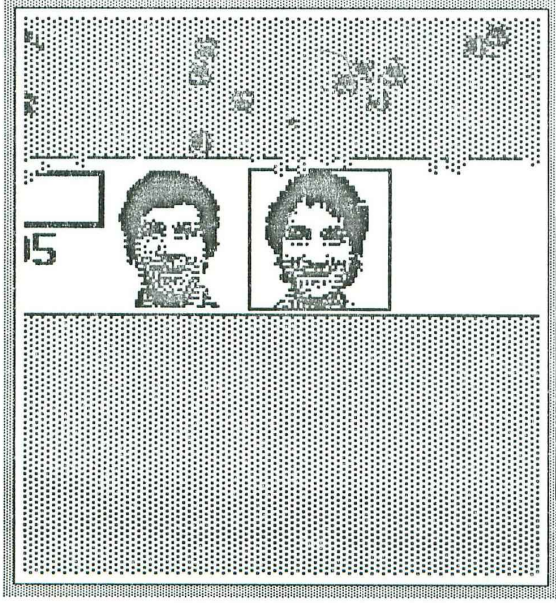
(Figures Crabs.11 .. Crabs.14) More of the same. The lens window is almost unharmed because it regenerates.



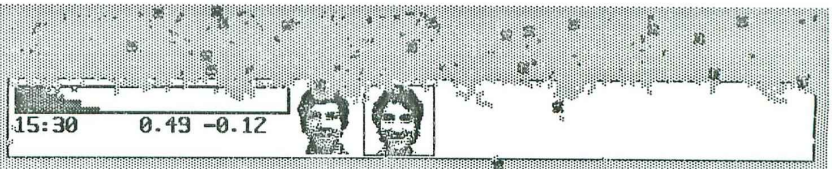

```
: ps
  PID TTY  STAT  TIME COMMAND
10065 pt14 I     0:00 sh
10066 pt14 I     0:00 /usr/jerq/bin/3pi
10067 pt14 I     0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.
m
10068 pt14 S     0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I     0:00 sh
10055 pt10 I     0:00 sh
10058 pt10 I     0:01 3pi
 9795 pt08 I     0:00 sh
 9780 pt04 S     0:01 sh
10071 pt04 R     0:01 ps
 9765 pt00 I     0:00 sh
 9778 pt00 I     0:00 sh
 9779 pt00 S     0:55 /usr/jerq/bin/vismon -1
 9250 14   R     5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the scree
n around.
How did you think of that?
```

```
   peek              tty07
.news_time       dead.letter     macguts
   pen               twid
.profile         emacs           machines
   peter             ufos
Alloc_mod2       faces           macwrite
   player0.icon      unixlicence
DavidJordan      fie             mail
   qix               upenn
Mamber           guest.profile   malloc
   referee           valis
ToGregHager      icon            mbox
   regexp            vaxwrite
```

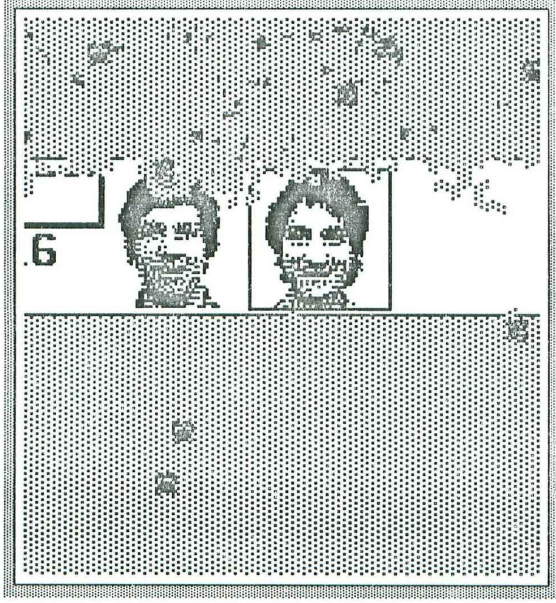

```
: ps
  PID TTY  STAT  TIME COMMAND
10065 pt14 I     0:00 sh
10066 pt14 I     0:00 /usr/jerq/bin/3pi
10067 pt14 I     0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.
m
10068 pt14 S     0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I     0:00 sh
10055 pt10 I     0:00 sh
10058 pt10 I     0:01 3pi
 9795 pt08 I     0:00 sh
 9780 pt04 S     0:01 sh
10071 pt04 R     0:01 ps
 9765 pt00 I     0:00 sh
 9778 pt00 I     0:00 sh
 9779 pt00 S     0:55 /usr/jerq/bin/vismon -1
 9250 14   R     5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the scree
n around.
How did you think of that?
```



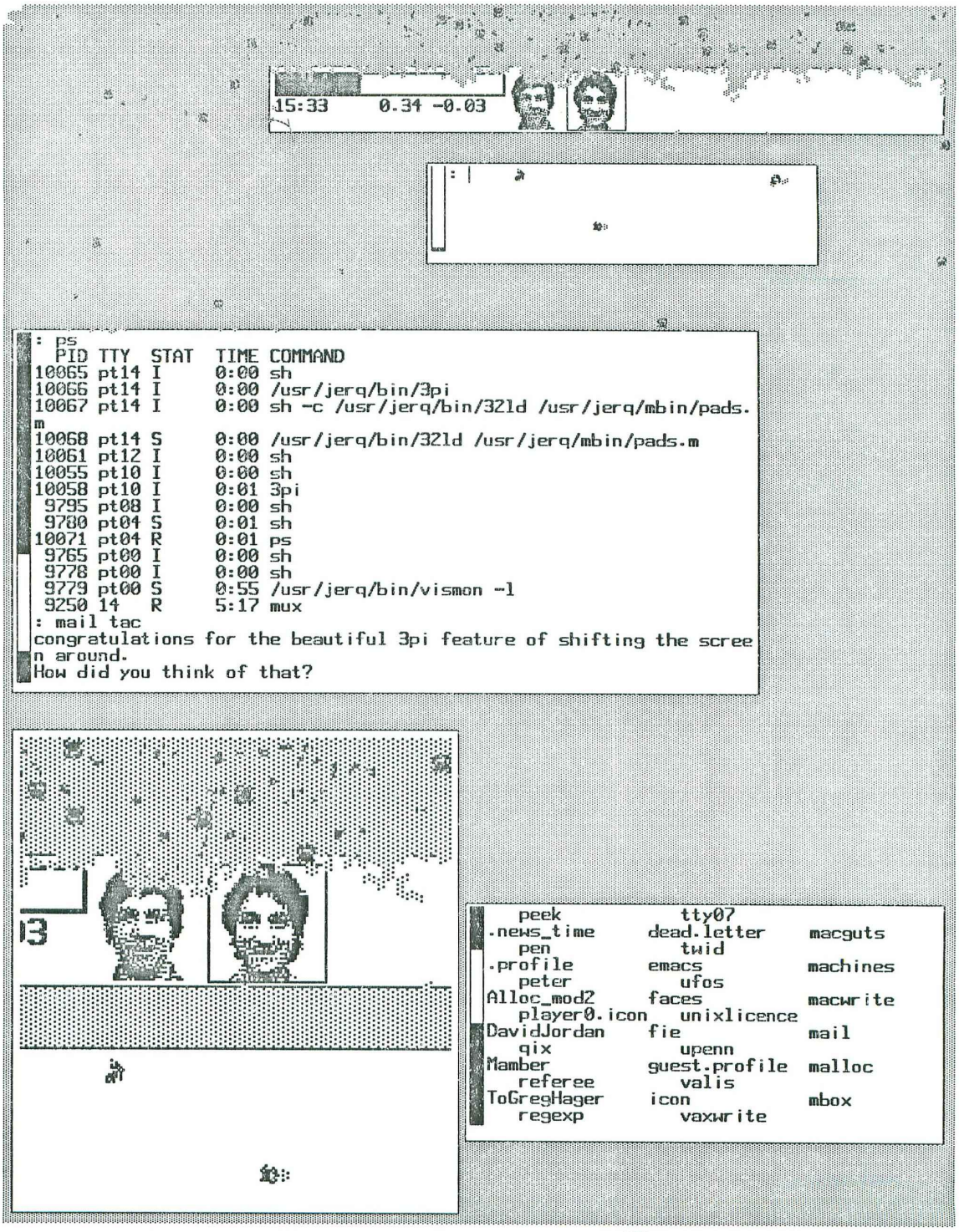
```
   peek              tty07
.news_time        dead.letter     macguts
   pen               twid
.profile          emacs           machines
   peter             ufos
Alloc_mod2        faces           macwrite
   player0.icon      unixlicence
DavidJordan       fie             mail
   qix               upenn
Mamber            guest.profile   malloc
   referee           valis
ToGregHager       icon            mbox
   regexp            vaxwrite
```




```
: ps
  PID TTY  STAT  TIME COMMAND
10065 pt14 I     0:00 sh
10066 pt14 I     0:00 /usr/jerq/bin/3pi
10067 pt14 I     0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.
m
10068 pt14 S     0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I     0:00 sh
10055 pt10 I     0:00 sh
10058 pt10 I     0:01 3pi
 9795 pt08 I     0:00 sh
 9780 pt04 S     0:01 sh
10071 pt04 R     0:01 ps
 9765 pt00 I     0:00 sh
 9778 pt00 I     0:00 sh
 9779 pt00 S     0:55 /usr/jerq/bin/vismon -1
 9250 14   R     5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the scree
n around.
How did you think of that?
```

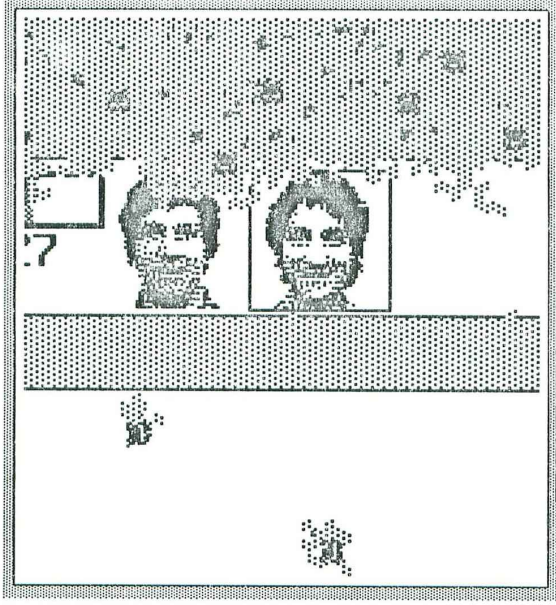



```
   peek              tty07
.news_time        dead.letter     macguts
   pen               twid
.profile          emacs           machines
   peter             ufos
Alloc_mod2        faces           macwrite
   player0.icon      unixlicence
DavidJordan       fie             mail
   qix               upenn
Mamber            guest.profile   malloc
   referee           valis
ToGregHager       icon            mbox
   regexp            vaxwrite
```

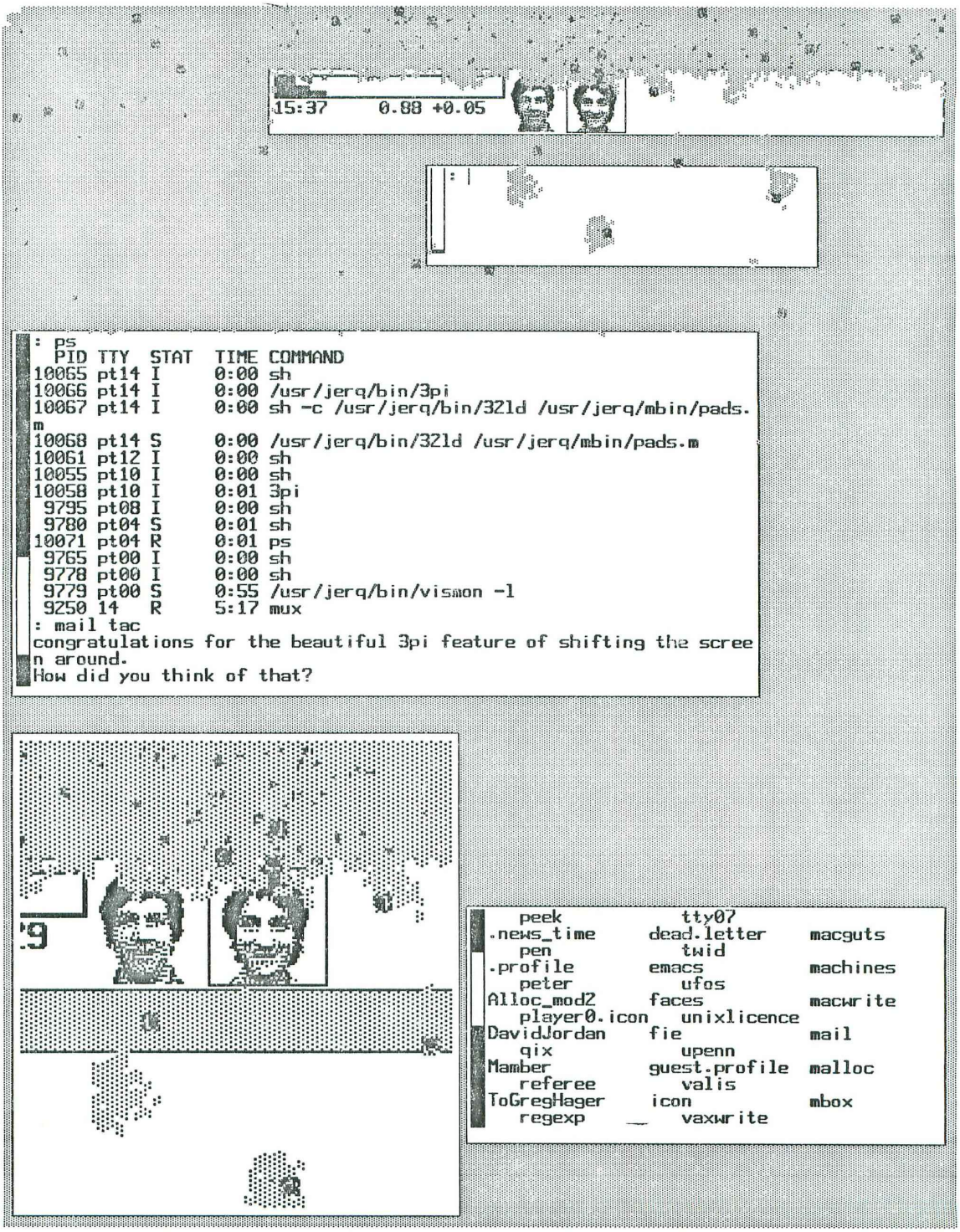

15:33 0.34 -0.03
: ps
PID TTY STAT TIME COMMAND
10065 pt14 I 0:00 sh
10066 pt14 I 0:00 /usr/jerq/bin/3pi
10067 pt14 I 0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.
m
10068 pt14 S 0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I 0:00 sh
10055 pt10 I 0:00 sh
10058 pt10 I 0:01 3pi
9795 pt08 I 0:00 sh
9780 pt04 S 0:01 sh
10071 pt04 R 0:01 ps
9765 pt00 I 0:00 sh
9778 pt00 I 0:00 sh
9779 pt00 S 0:55 /usr/jerq/bin/vismon -1
9250 14 R 5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the scree
n around.
How did you think of that?
peek tty07
.news_time dead.letter macguts
pen twid
.profile emacs machines
peter ufos
Alloc_mod2 faces macwrite
player0.icon unixlicence
DavidJordan fie mail
qix upenn
Mamber guest.profile malloc
referee valis
ToGregHager icon mbox
regexp vaxwrite

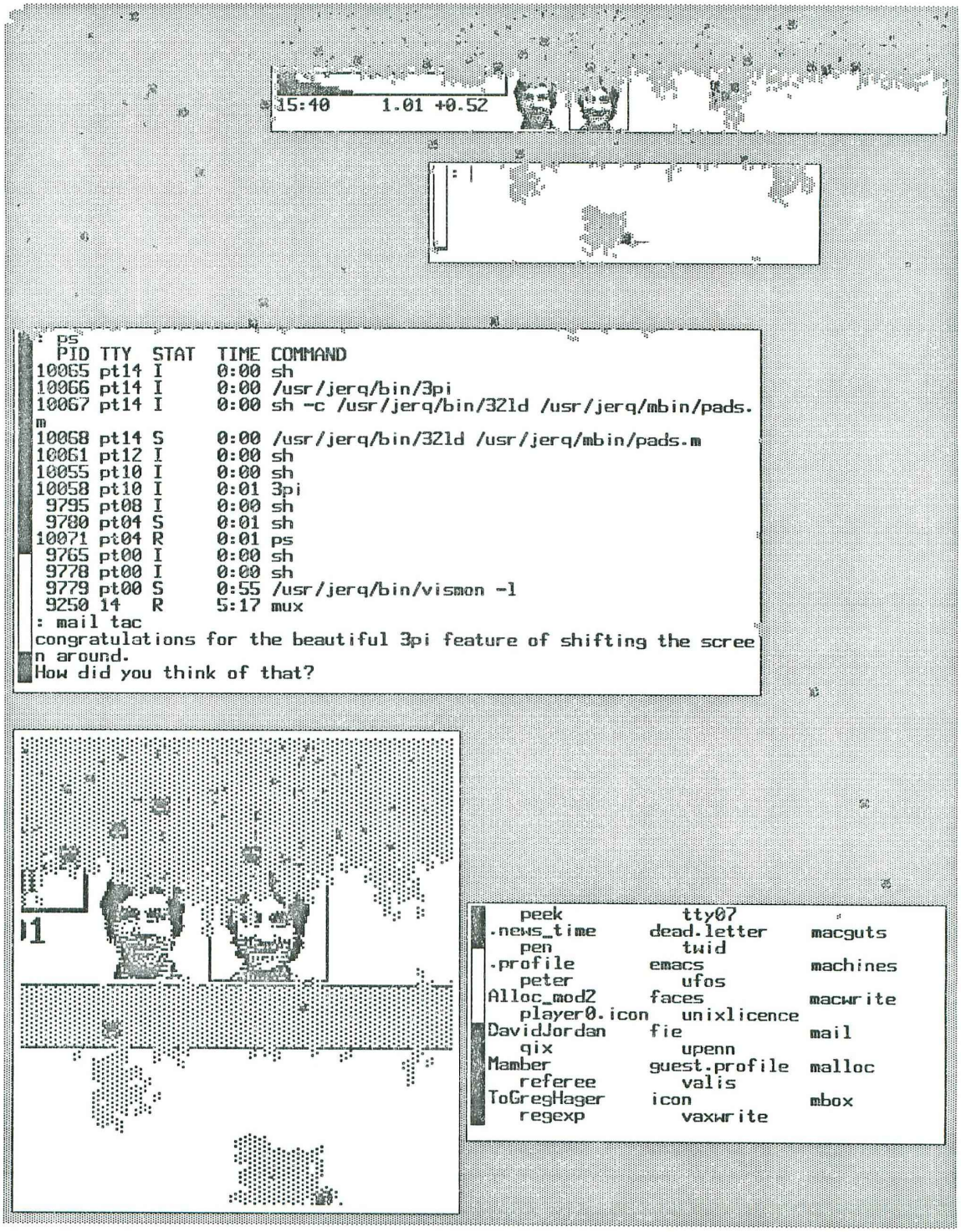


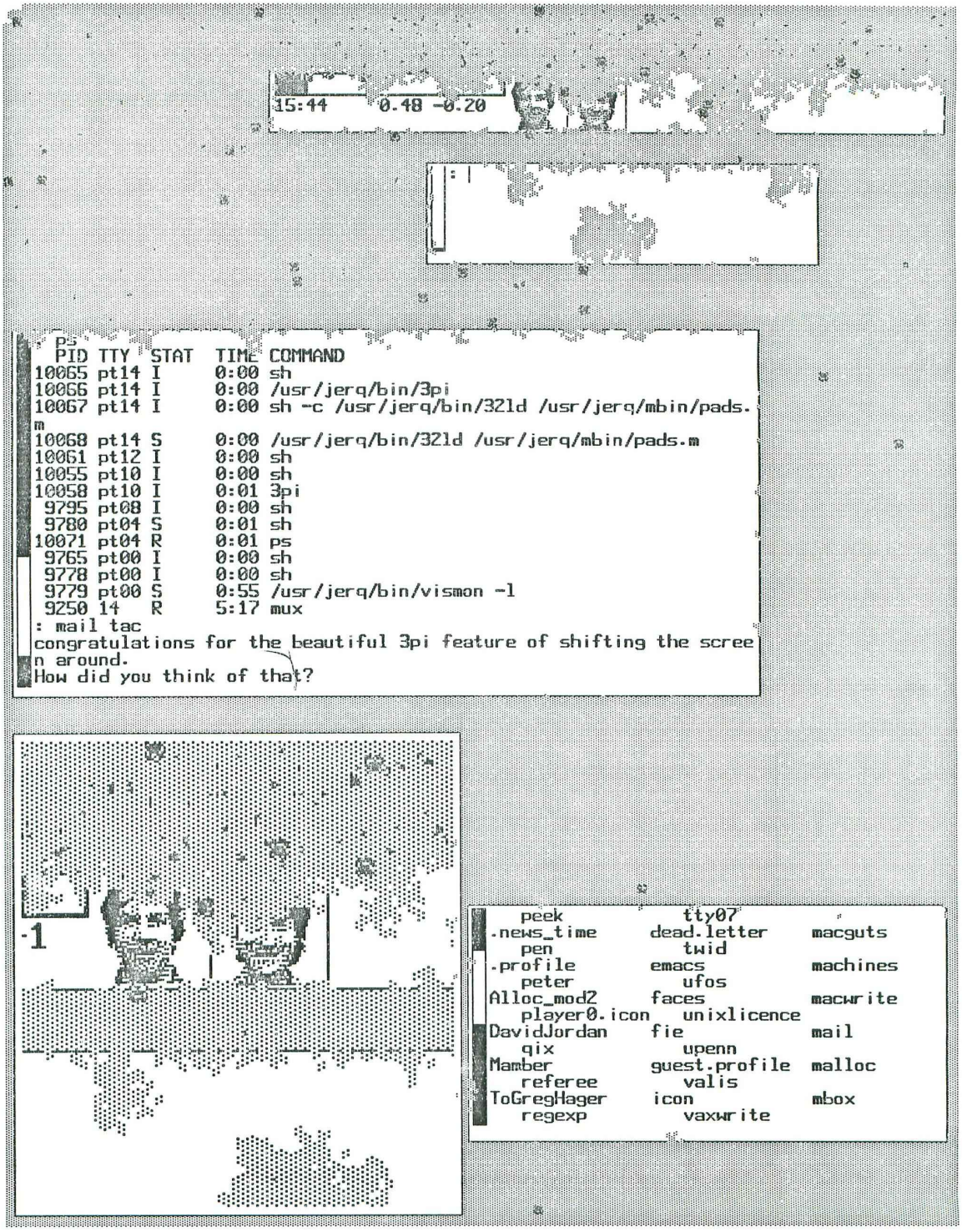
```
: ps
  PID TTY  STAT  TIME COMMAND
10065 pt14 I     0:00 sh
10066 pt14 I     0:00 /usr/jerq/bin/3pi
10067 pt14 I     0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.
m
10068 pt14 S     0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I     0:00 sh
10055 pt10 I     0:00 sh
10058 pt10 I     0:01 3pi
 9795 pt08 I     0:00 sh
 9780 pt04 S     0:01 sh
10071 pt04 R     0:01 ps
 9765 pt00 I     0:00 sh
 9778 pt00 I     0:00 sh
 9779 pt00 S     0:55 /usr/jerq/bin/vismon -1
 9250 14   R     5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the scree
n around.
How did you think of that?
```

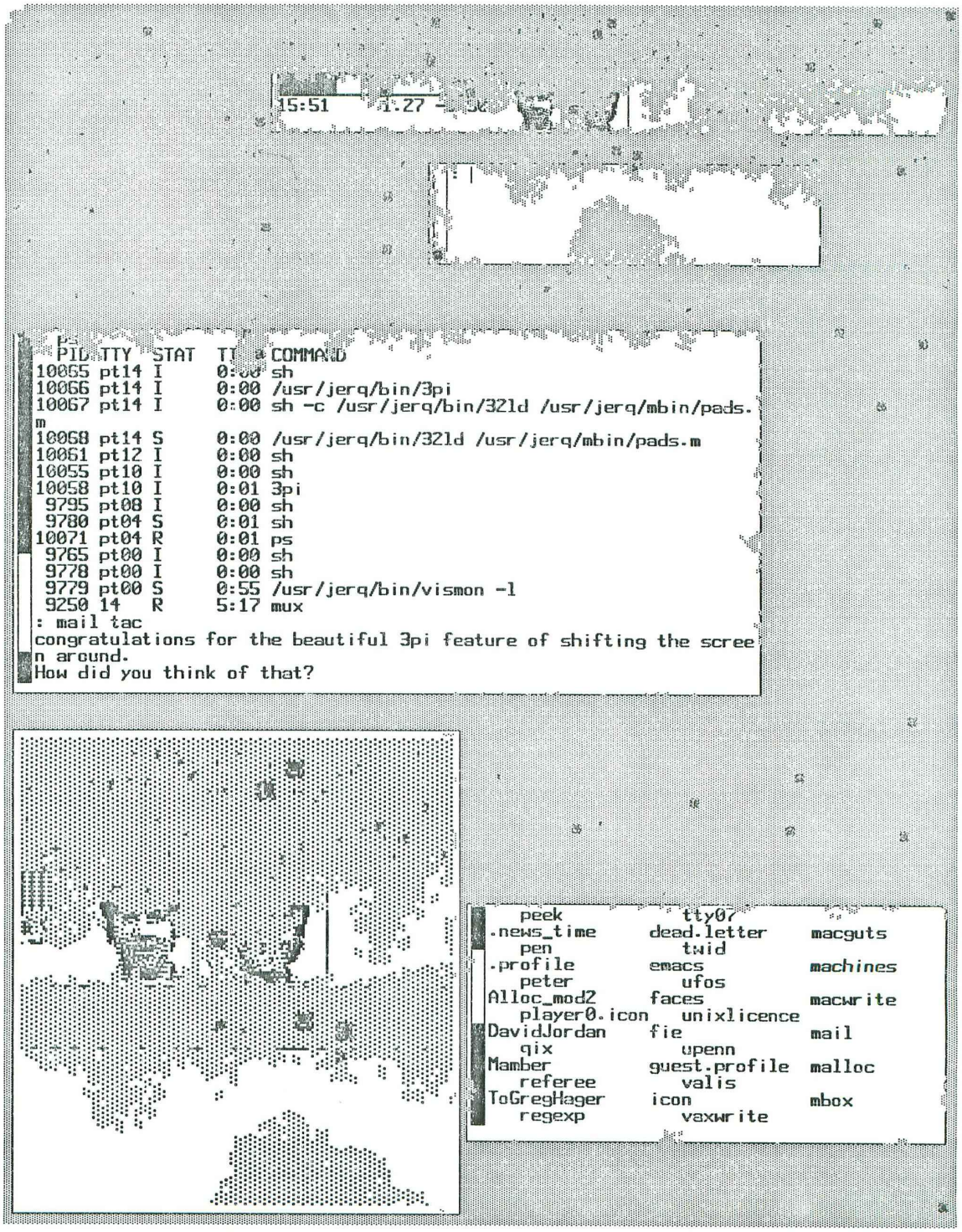


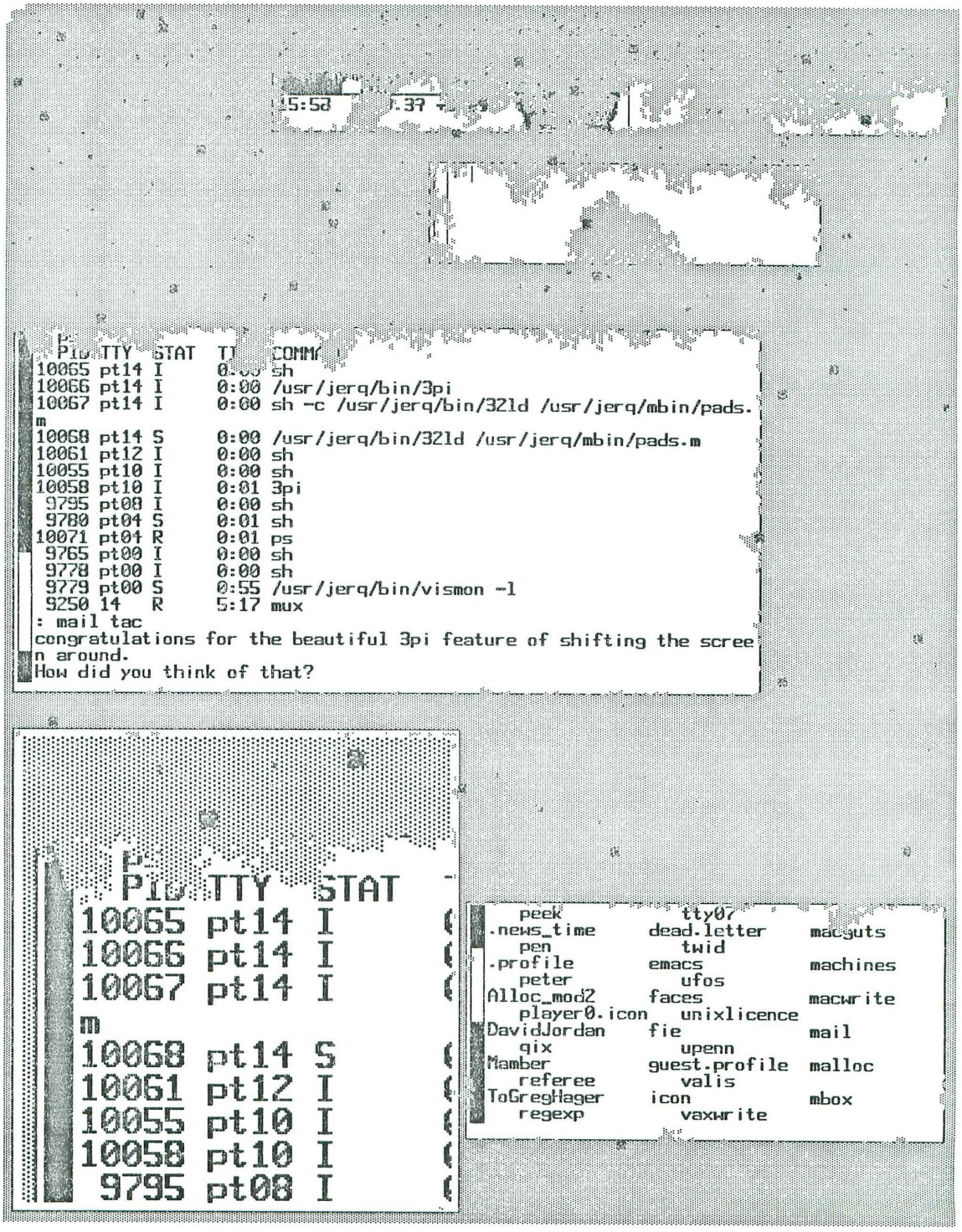
```
   peek              tty07
.news_time        dead.letter     macguts
   pen               twid
.profile          emacs           machines
   peter             ufos
Alloc_mod2        faces           macwrite
   player0.icon      unixlicence
DavidJordan       fie             mail
   qix               upenn
Mamber            guest.profile   malloc
   referee           valis
ToGregHager       icon            mbox
   regexp            vaxwrite
```

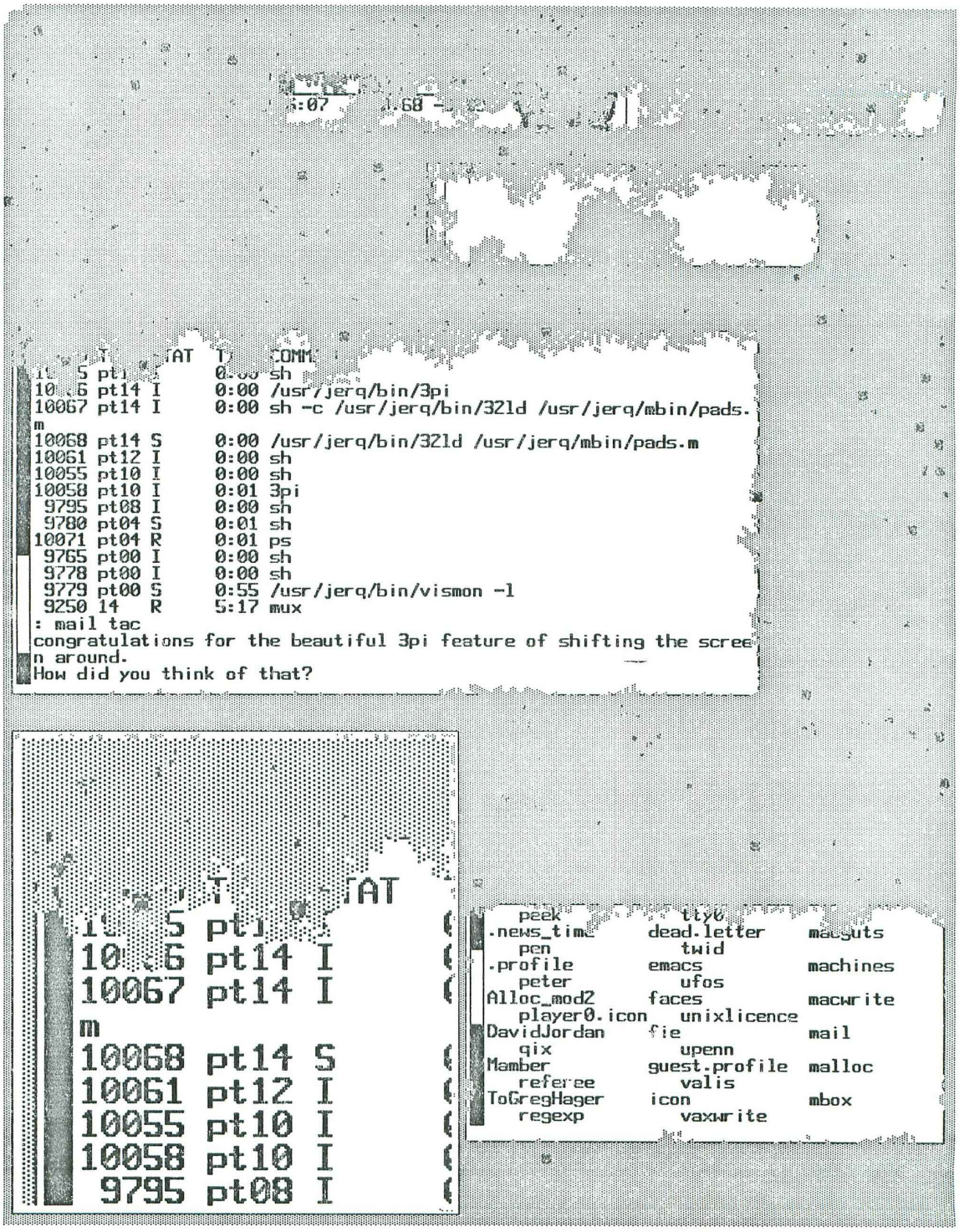

15:37 0.88 +0.05
: ps
PID TTY STAT TIME COMMAND
10065 pt14 I 0:00 sh
10066 pt14 I 0:00 /usr/jerq/bin/3pi
10067 pt14 I 0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10068 pt14 S 0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I 0:00 sh
10055 pt10 I 0:00 sh
10058 pt10 I 0:01 3pi
9795 pt08 I 0:00 sh
9780 pt04 S 0:01 sh
10071 pt04 R 0:01 ps
9765 pt00 I 0:00 sh
9778 pt00 I 0:00 sh
9779 pt00 S 0:55 /usr/jerq/bin/vismon -l
9250 14 R 5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the screen around.
How did you think of that?
peek
.news_time
pen
.profile
peter
Alloc_mod2
player0.icon
DavidJordan
qix
Mamber
referee
ToGregHager
regexp
tty07
dead.letter
twid
emacs
ufos
faces
unixlicence
fie
upenn
guest.profile
valis
icon
vaxwrite
macguts
machines
macwrite
mail
malloc
mbox

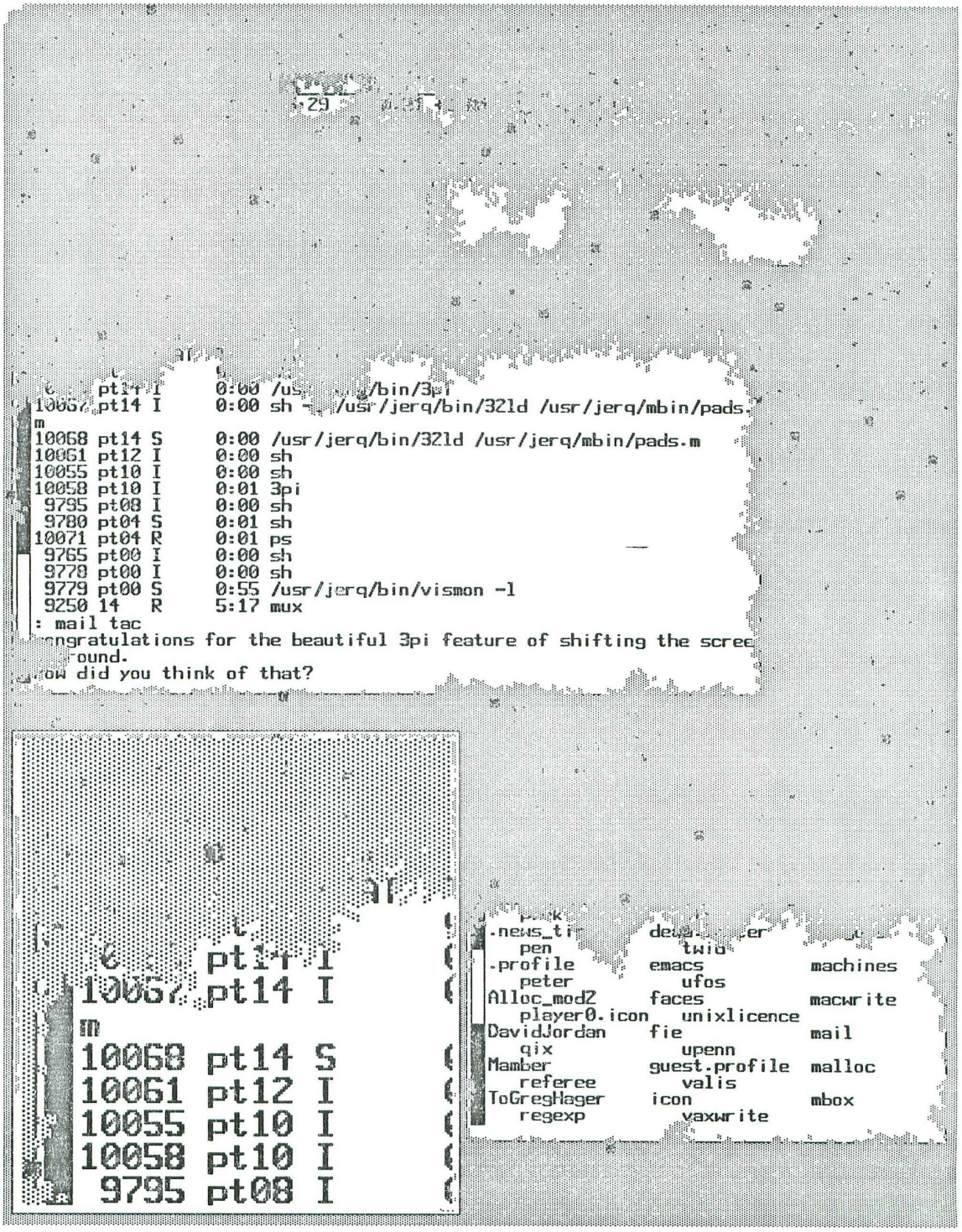

15:40 1.01 +0.52
: ps
PID TTY STAT TIME COMMAND
10065 pt14 I 0:00 sh
10066 pt14 I 0:00 /usr/jerq/bin/3pi
10067 pt14 I 0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10068 pt14 S 0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I 0:00 sh
10055 pt10 I 0:00 sh
10058 pt10 I 0:01 3pi
9795 pt08 I 0:00 sh
9780 pt04 S 0:01 sh
10071 pt04 R 0:01 ps
9765 pt00 I 0:00 sh
9778 pt00 I 0:00 sh
9779 pt00 S 0:55 /usr/jerq/bin/vismon -l
9250 14 R 5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the screen around.
How did you think of that?
peek
.news_time
pen
.profile
peter
Alloc_mod2
player0.icon
DavidJordan
qix
Mamber
referee
ToGregHager
regexp
tty07
dead.letter
twid
emacs
ufos
faces
unixlicence
fie
upenn
guest.profile
valis
icon
vaxwrite
macguts
machines
macwrite
mail
malloc
mbox

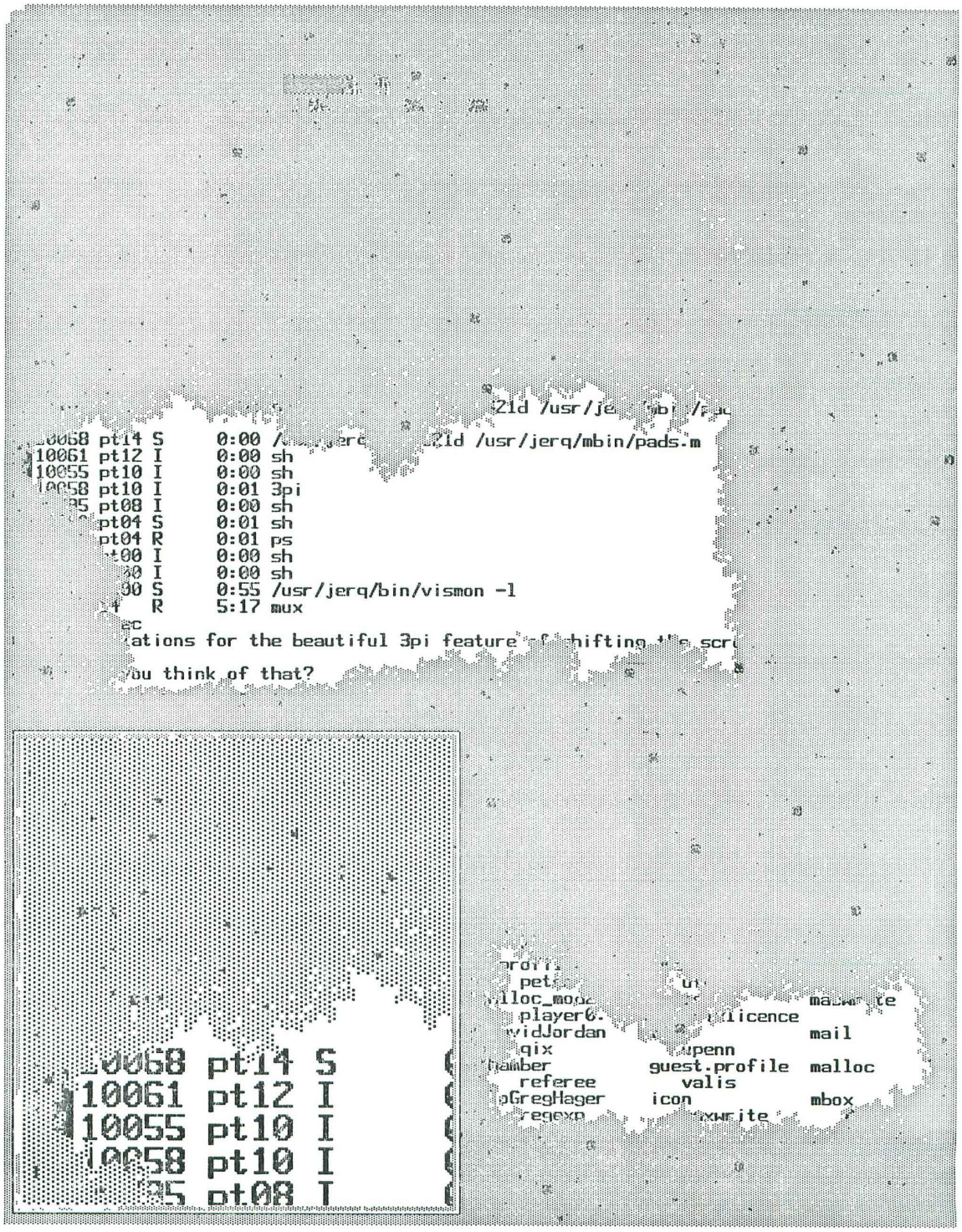

15:44 0.48 -0.20
: |
ps
PID TTY STAT TIME COMMAND
10065 pt14 I 0:00 sh
10066 pt14 I 0:00 /usr/jerq/bin/3pi
10067 pt14 I 0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.
m
10068 pt14 S 0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I 0:00 sh
10055 pt10 I 0:00 sh
10058 pt10 I 0:01 3pi
9795 pt08 I 0:00 sh
9780 pt04 S 0:01 sh
10071 pt04 R 0:01 ps
9765 pt00 I 0:00 sh
9778 pt00 I 0:00 sh
9779 pt00 S 0:55 /usr/jerq/bin/vismon -l
9250 14 R 5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the scree
n around.
How did you think of that?
-1
peek tty07
.news_time dead.letter macguts
pen twid
.profile emacs machines
peter ufos
Alloc_mod2 faces macwrite
player0.icon unixlicence
DavidJordan fie mail
qix upenn
Mamber guest.profile malloc
referee valis
ToGregHager icon mbox
regexp vaxwrite

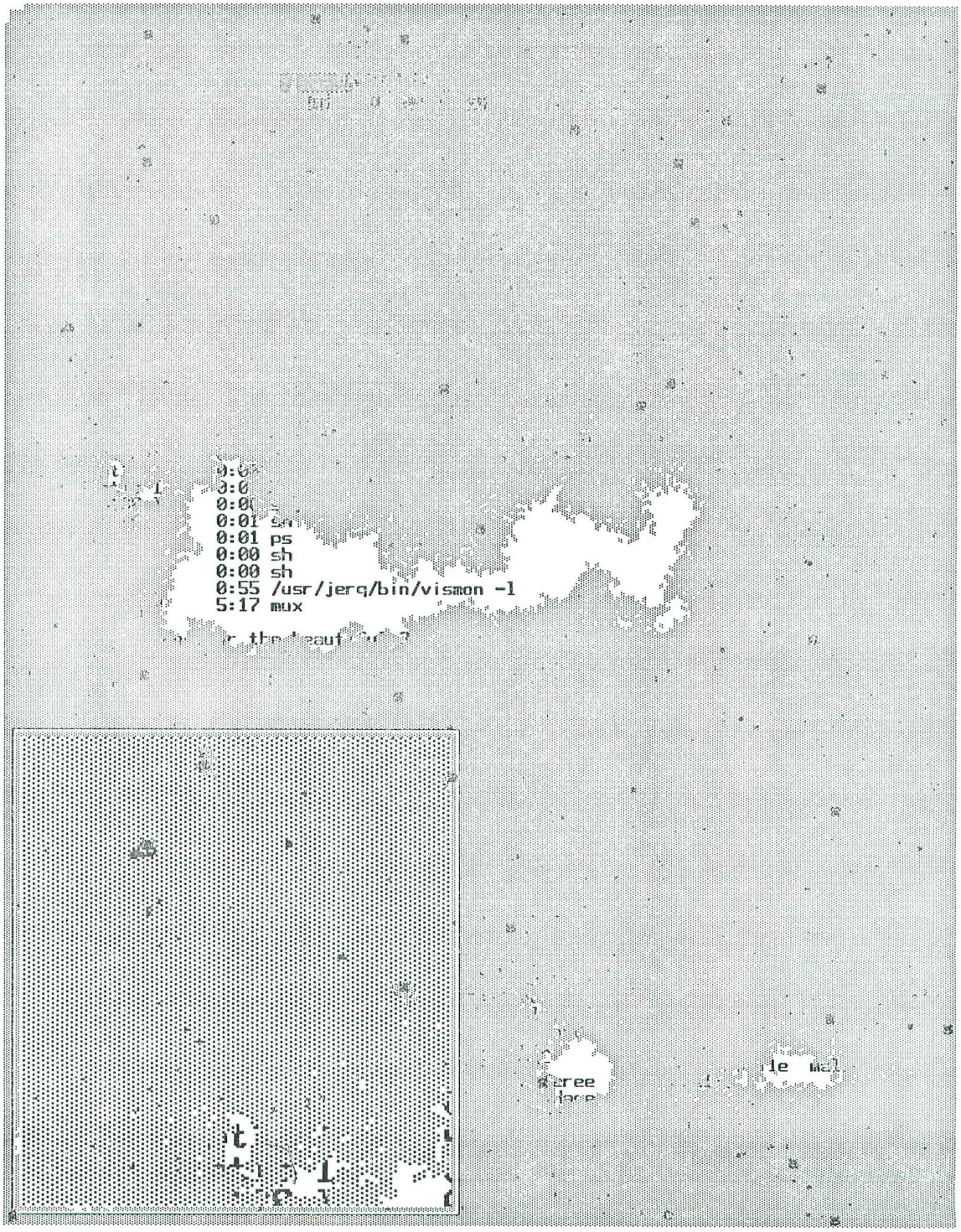

15:51
ps
PID TTY STAT TI COMMAND
10065 pt14 I 0:00 sh
10066 pt14 I 0:00 /usr/jerq/bin/3pi
10067 pt14 I 0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10068 pt14 S 0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I 0:00 sh
10055 pt10 I 0:00 sh
10058 pt10 I 0:01 3pi
9795 pt08 I 0:00 sh
9780 pt04 S 0:01 sh
10071 pt04 R 0:01 ps
9765 pt00 I 0:00 sh
9778 pt00 I 0:00 sh
9779 pt00 S 0:55 /usr/jerq/bin/vismon -l
9250 14 R 5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the screen around.
How did you think of that?
peek
.news_time
pen
.profile
peter
Alloc_mod2
player0.icon
DavidJordan
qix
Mamber
referee
ToGregHager
regexp
tty07
dead.letter
twid
emacs
ufos
faces
unixlicence
fie
upenn
guest.profile
valis
icon
vaxwrite
macguts
machines
macwrite
mail
malloc
mbox

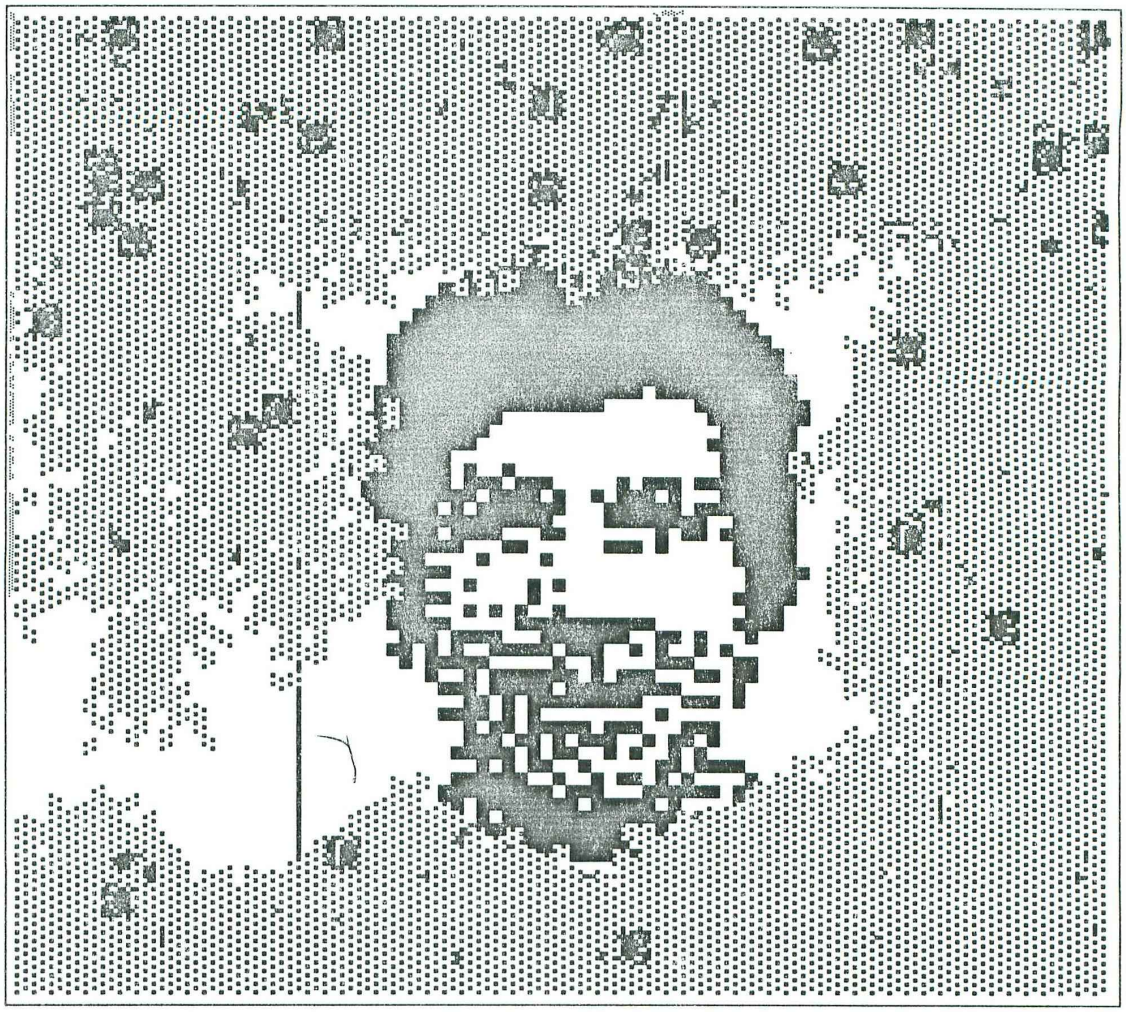

PID TTY STAT TIME COMM
10065 pt14 I 0:00 sh
10066 pt14 I 0:00 /usr/jerq/bin/3pi
10067 pt14 I 0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10068 pt14 S 0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I 0:00 sh
10055 pt10 I 0:00 sh
10058 pt10 I 0:01 3pi
9795 pt08 I 0:00 sh
9780 pt04 S 0:01 sh
10071 pt04 R 0:01 ps
9765 pt00 I 0:00 sh
9778 pt00 I 0:00 sh
9779 pt00 S 0:55 /usr/jerq/bin/vismon -l
9250 14 R 5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the screen around.
How did you think of that?
ps
PID TTY STAT
10065 pt14 I
10066 pt14 I
10067 pt14 I
m
10068 pt14 S
10061 pt12 I
10055 pt10 I
10058 pt10 I
9795 pt08 I
peek tty07
.news_time dead.letter macguts
pen twid
.profile emacs machines
peter ufos
Alloc_mod2 faces macwrite
player0.icon unixlicence
DavidJordan fie mail
qix upenn
Mamber guest.profile malloc
referee valis
ToGregHager icon mbox
regexp vaxwrite

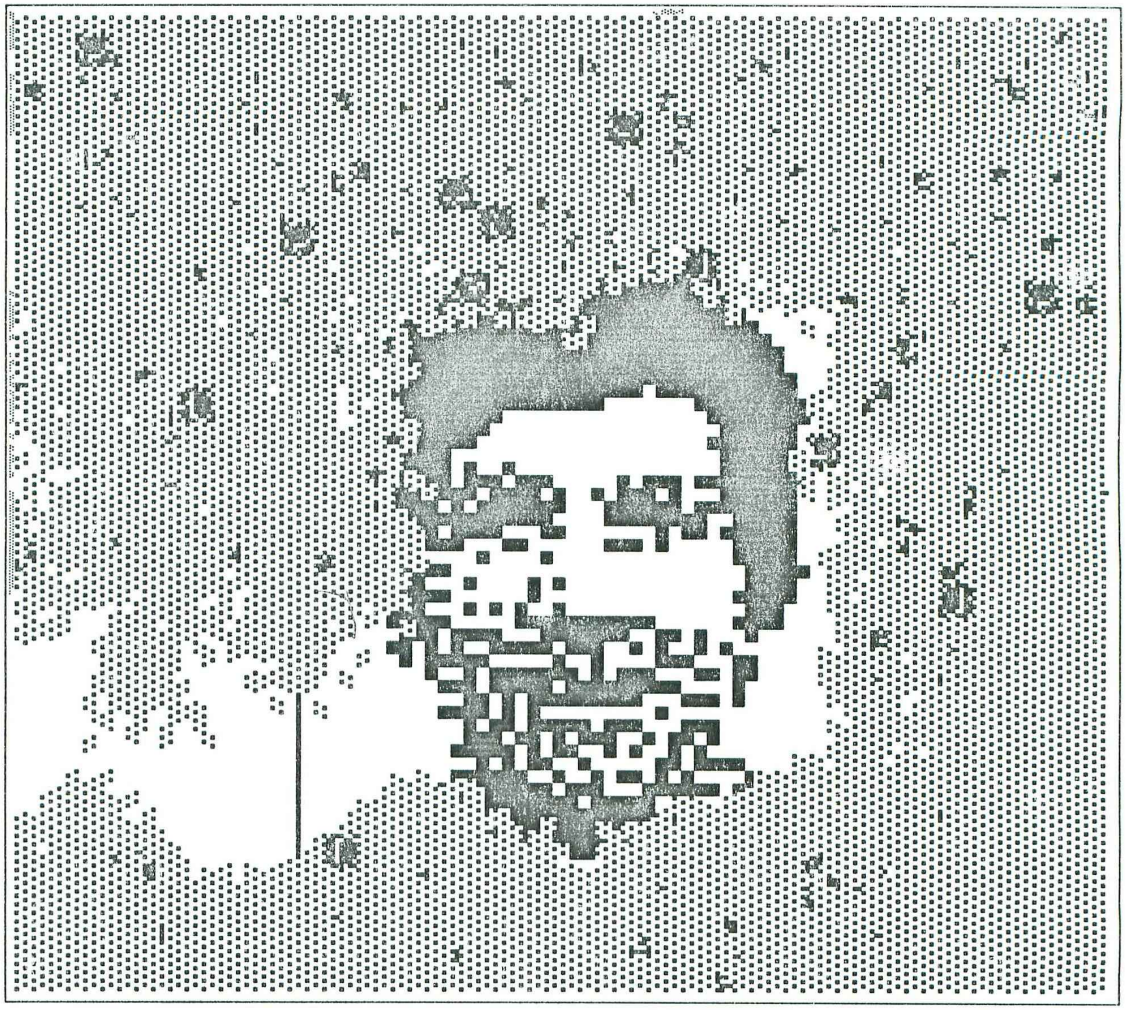

T AT T COMM
5 pt1 0:00 sh
6 pt14 I 0:00 /usr/jerq/bin/3pi
10067 pt14 I 0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.
m
10068 pt14 S 0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I 0:00 sh
10055 pt10 I 0:00 sh
10058 pt10 I 0:01 3pi
9795 pt08 I 0:00 sh
9780 pt04 S 0:01 sh
10071 pt04 R 0:01 ps
9765 pt00 I 0:00 sh
9778 pt00 I 0:00 sh
9779 pt00 S 0:55 /usr/jerq/bin/vismon -l
9250 14 R 5:17 mux
: mail tac
congratulations for the beautiful 3pi feature of shifting the scree
n around.
How did you think of that?
T AT
5 pt
6 pt14 I
10067 pt14 I
m
10068 pt14 S
10061 pt12 I
10055 pt10 I
10058 pt10 I
9795 pt08 I
peek tty6
.news_time dead.letter macguts
pen twid
.profile emacs machines
peter ufos
Alloc_mod2 faces macwrite
player0.icon unixlicence
DavidJordan fie mail
qix upenn
Mamber guest.profile malloc
referee valis
ToGregHager icon mbox
regexp vaxwrite

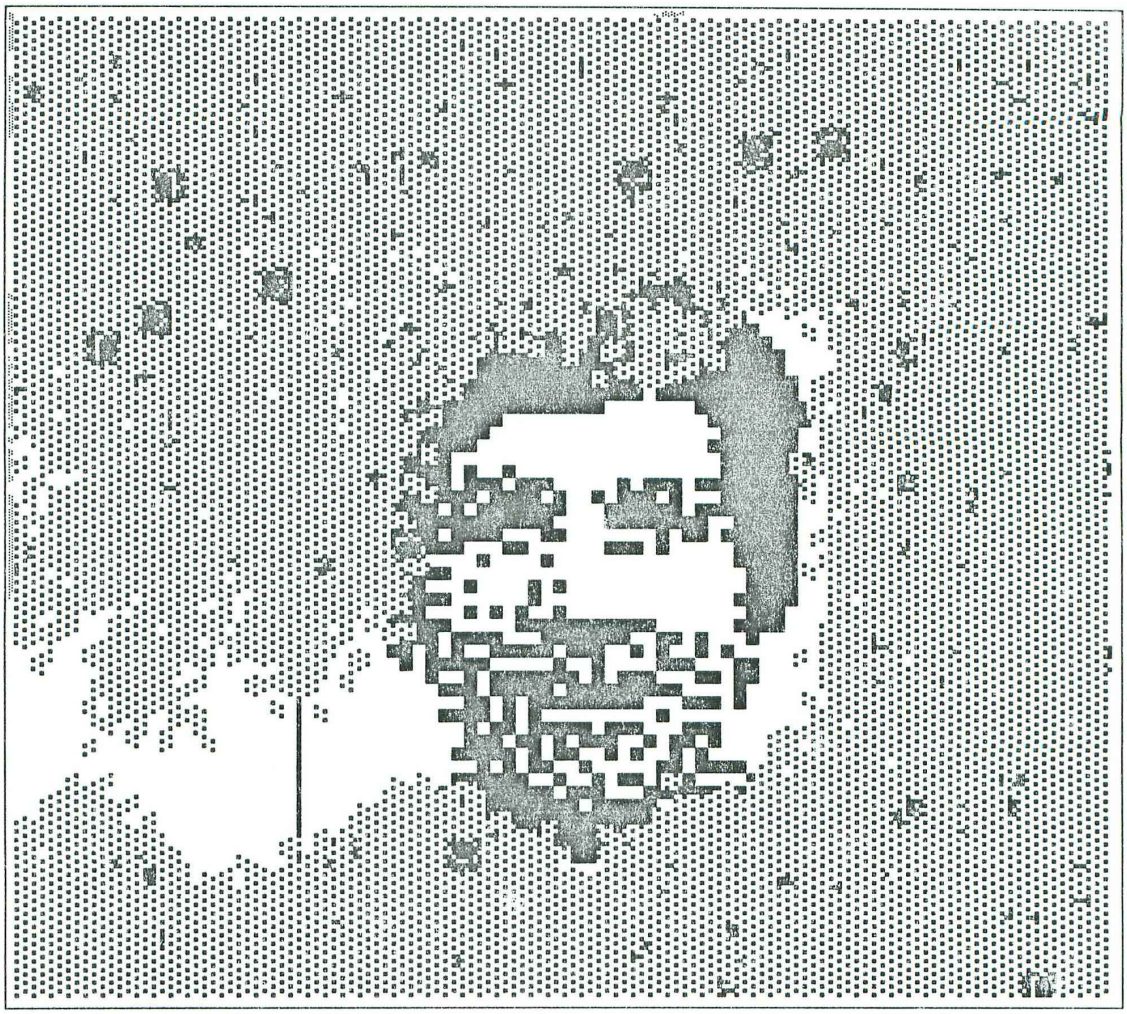

pt14 I 0:00 /usr/jerq/bin/3pi
10057 pt14 I 0:00 sh -c /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10068 pt14 S 0:00 /usr/jerq/bin/32ld /usr/jerq/mbin/pads.m
10061 pt12 I 0:00 sh
10055 pt10 I 0:00 sh
10058 pt10 I 0:01 3pi
9795 pt08 I 0:00 sh
9780 pt04 S 0:01 sh
10071 pt04 R 0:01 ps
9765 pt00 I 0:00 sh
9778 pt00 I 0:00 sh
9779 pt00 S 0:55 /usr/jerq/bin/vismon -l
9250 14 R 5:17 mux
: mail tac
ngratulations for the beautiful 3pi feature of shifting the scree
ound.
ow did you think of that?
pt14 I
10057 pt14 I
10068 pt14 S
10061 pt12 I
10055 pt10 I
10058 pt10 I
9795 pt08 I
.news_time
pen
.profile
peter
Alloc_mod2
player0.icon
DavidJordan
qix
Mamber
referee
ToGregHager
regexp
twid
emacs
ufos
faces
unixlicence
fie
upenn
guest.profile
valis
icon
vaxwrite
machines
macwrite
mail
malloc
mbox

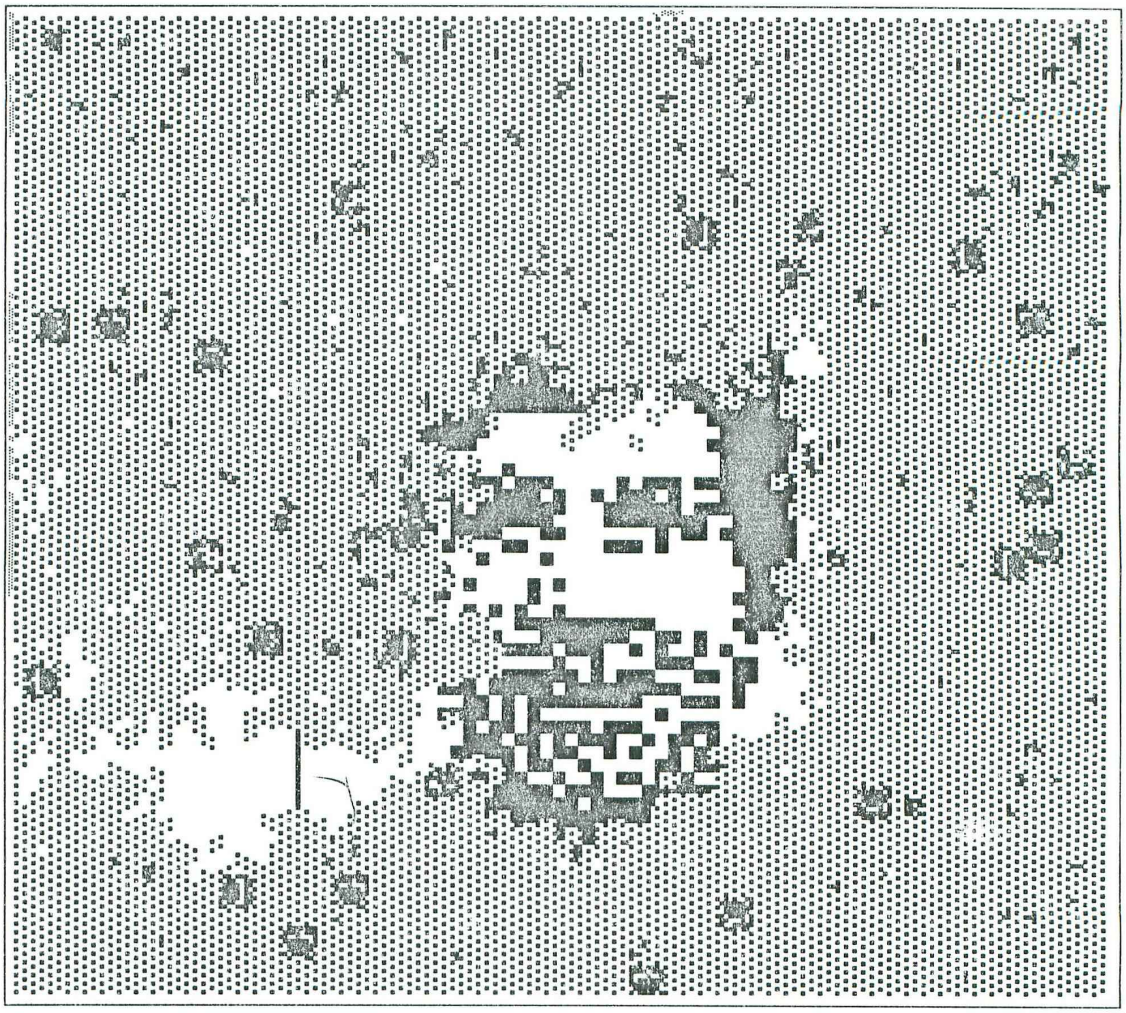

21d /usr/jerq/mbin/pads.m
0:00 sh
0:00 sh
0:01 3pi
0:00 sh
0:01 sh
0:01 ps
0:00 sh
0:00 sh
0:55 /usr/jerq/bin/vismon -l
5:17 mux
ations for the beautiful 3pi feature of shifting the scr
ou think of that?
10061 pt12 I
10055 pt10 I
10058 pt10 I
pet
player0.
widJordan
qix
guest.profile
malloc
referee
valis
icon
mbox
mail

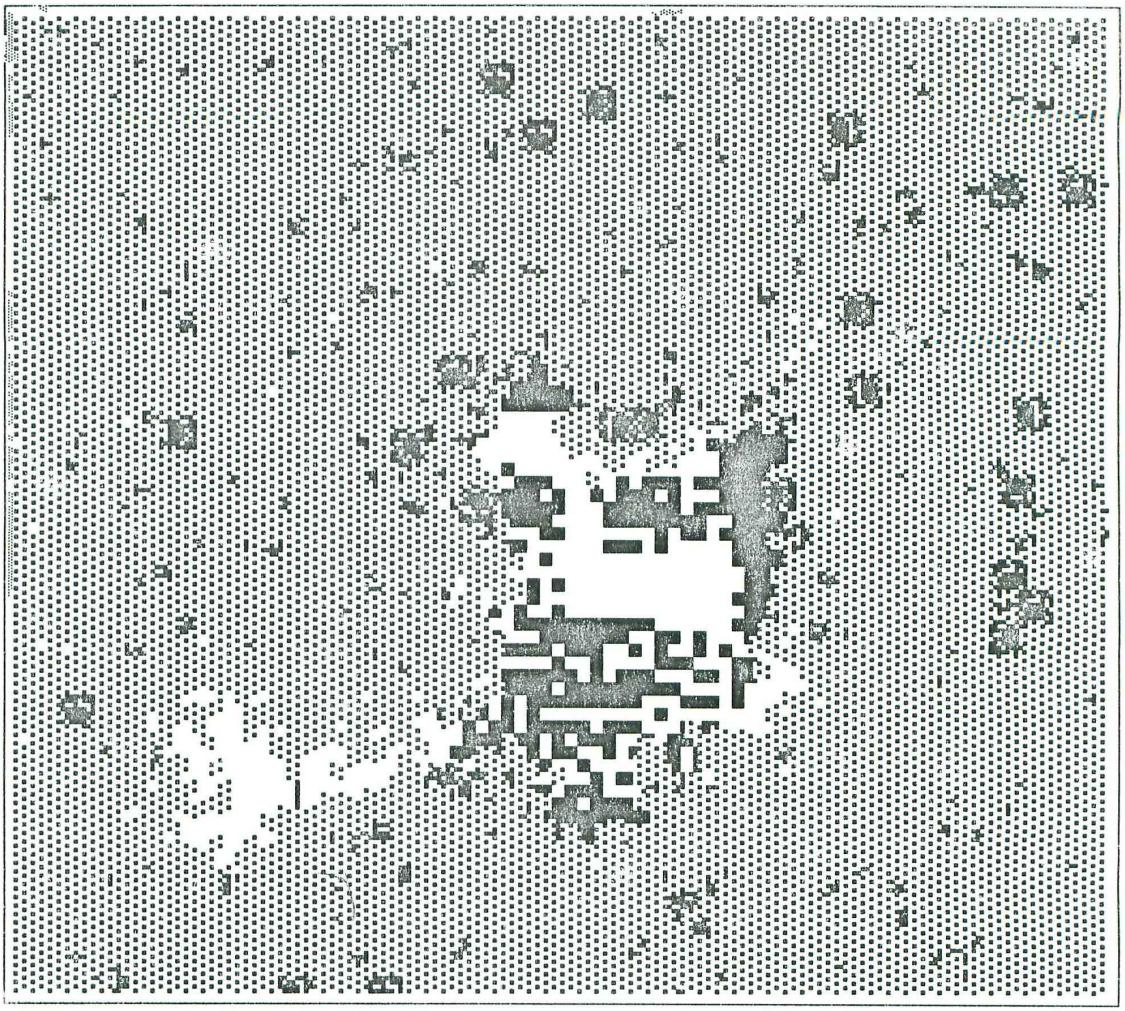

0:01 sh
0:01 ps
0:00 sh
0:00 sh
0:55 /usr/jerq/bin/vismon -1
5:17 mux

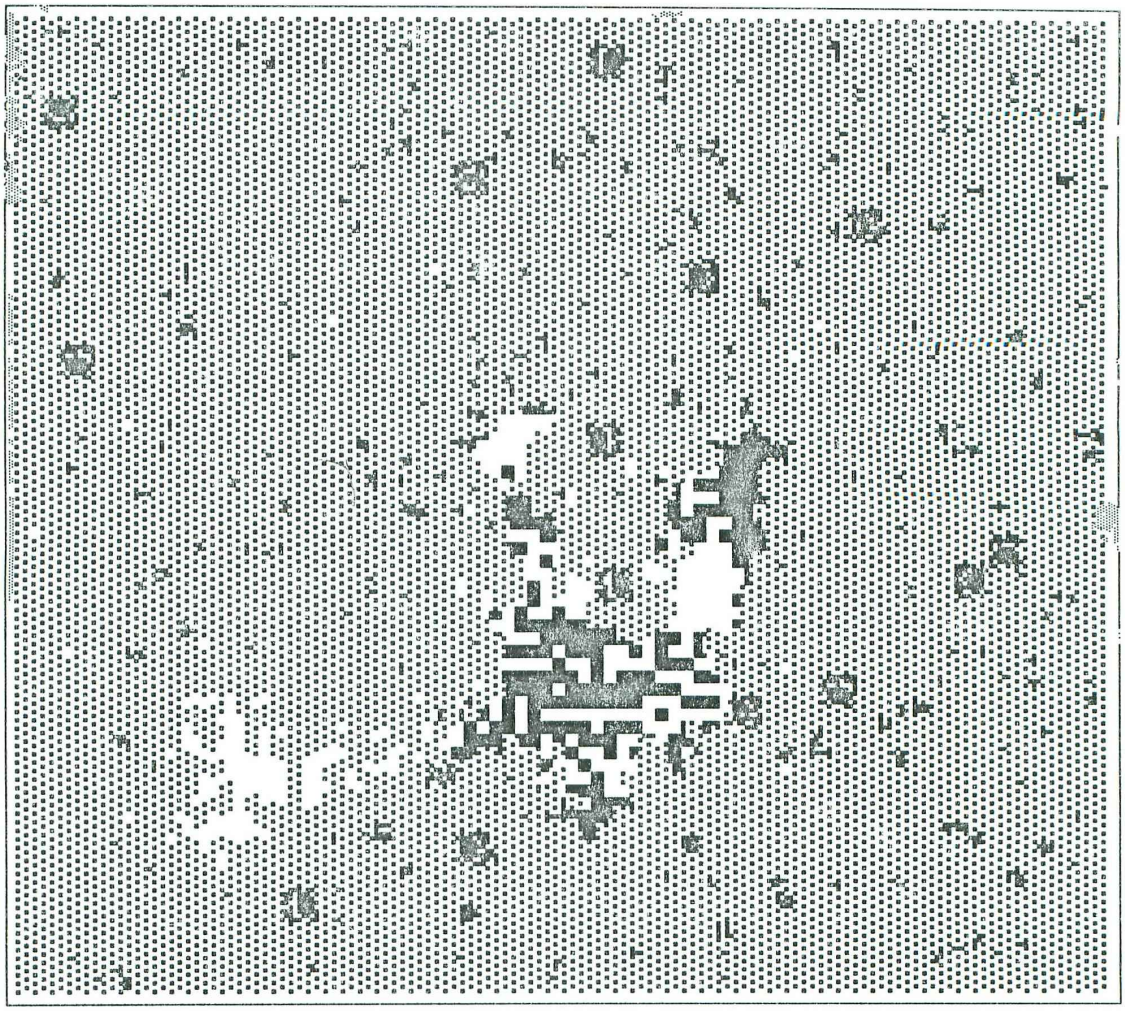


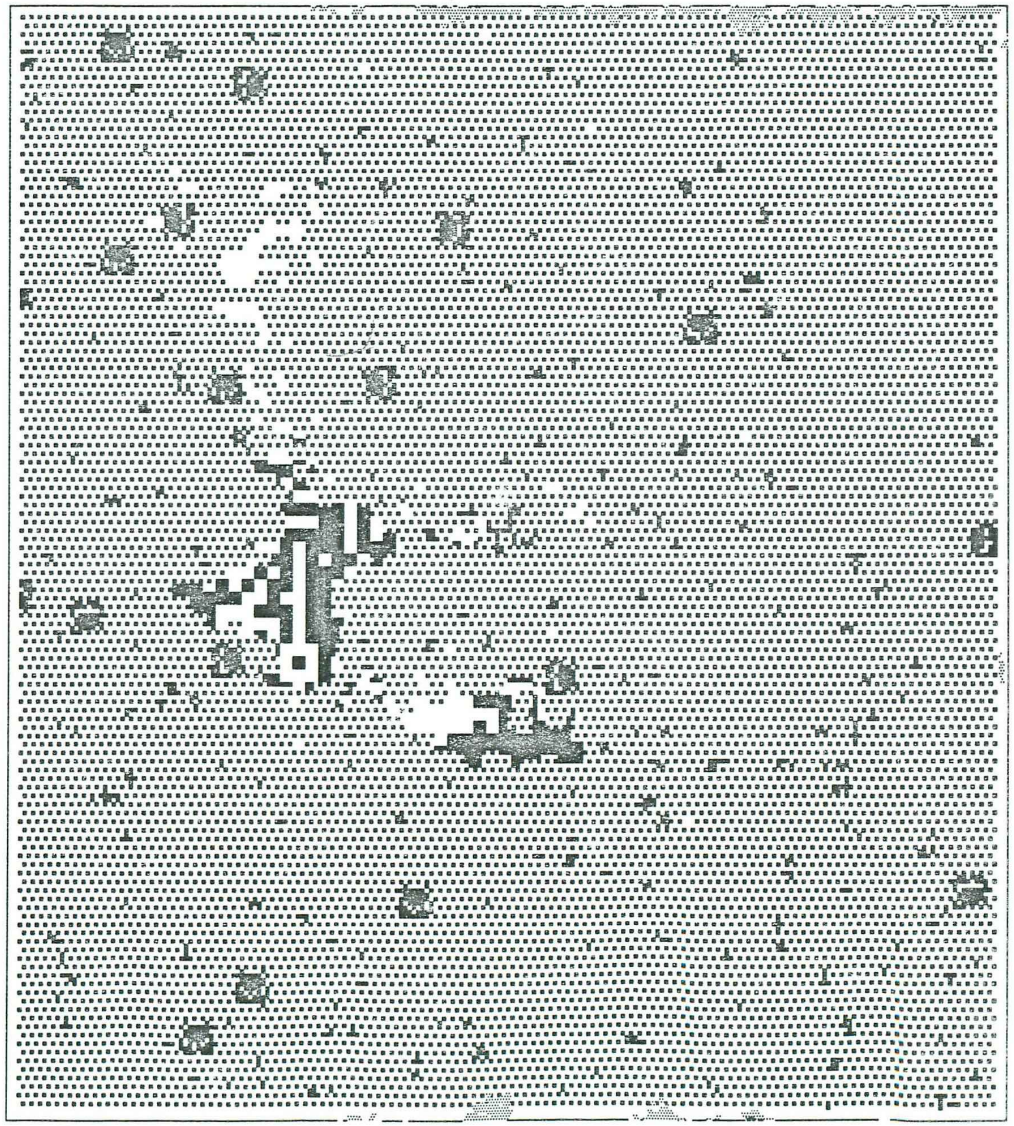


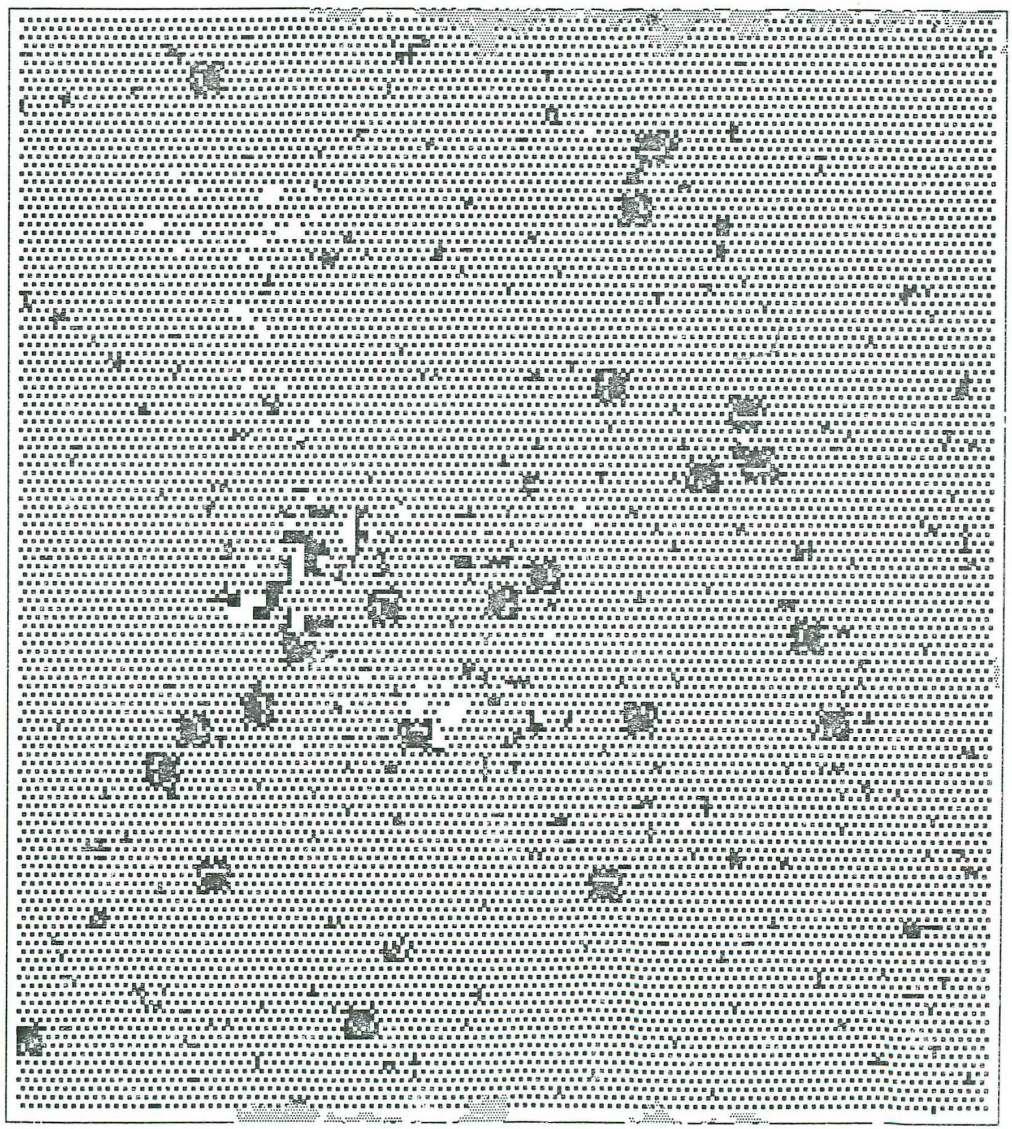












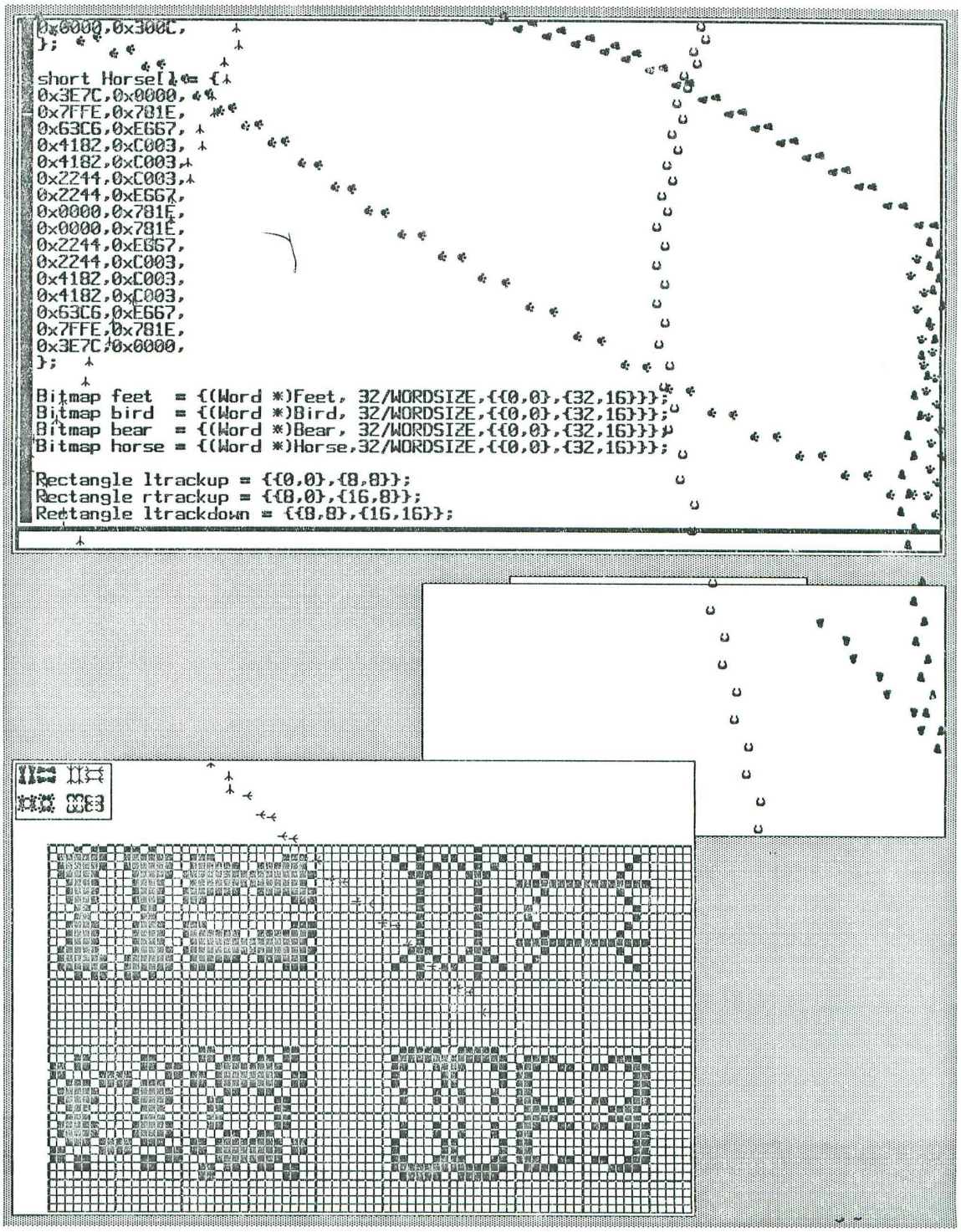
"Tracks" gives the illusion of animals walking on your windows and leaving footprints. There are cats, birds, unicorns and little people.

Tracks are not left on the background, only on windows. However tracks can cross grey regions and continue on another window.

The random motion is obtained as in crabs, with slightly different parameters.

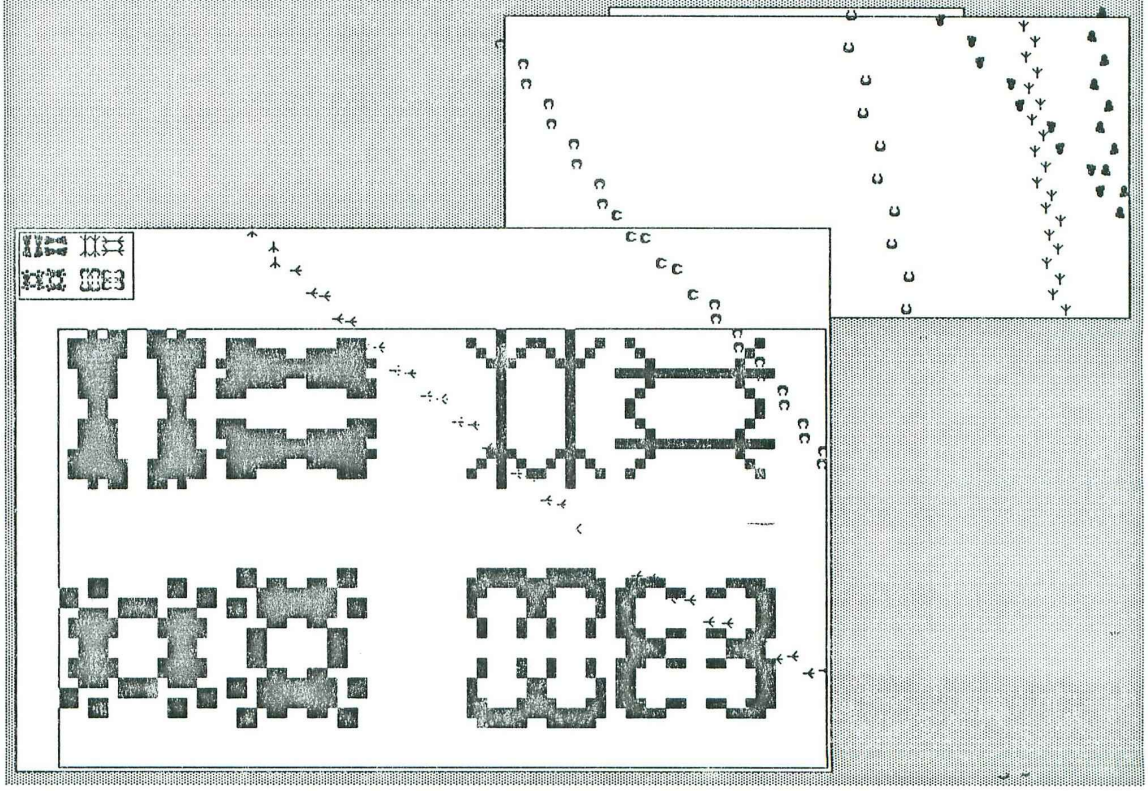
Luca Cardelli

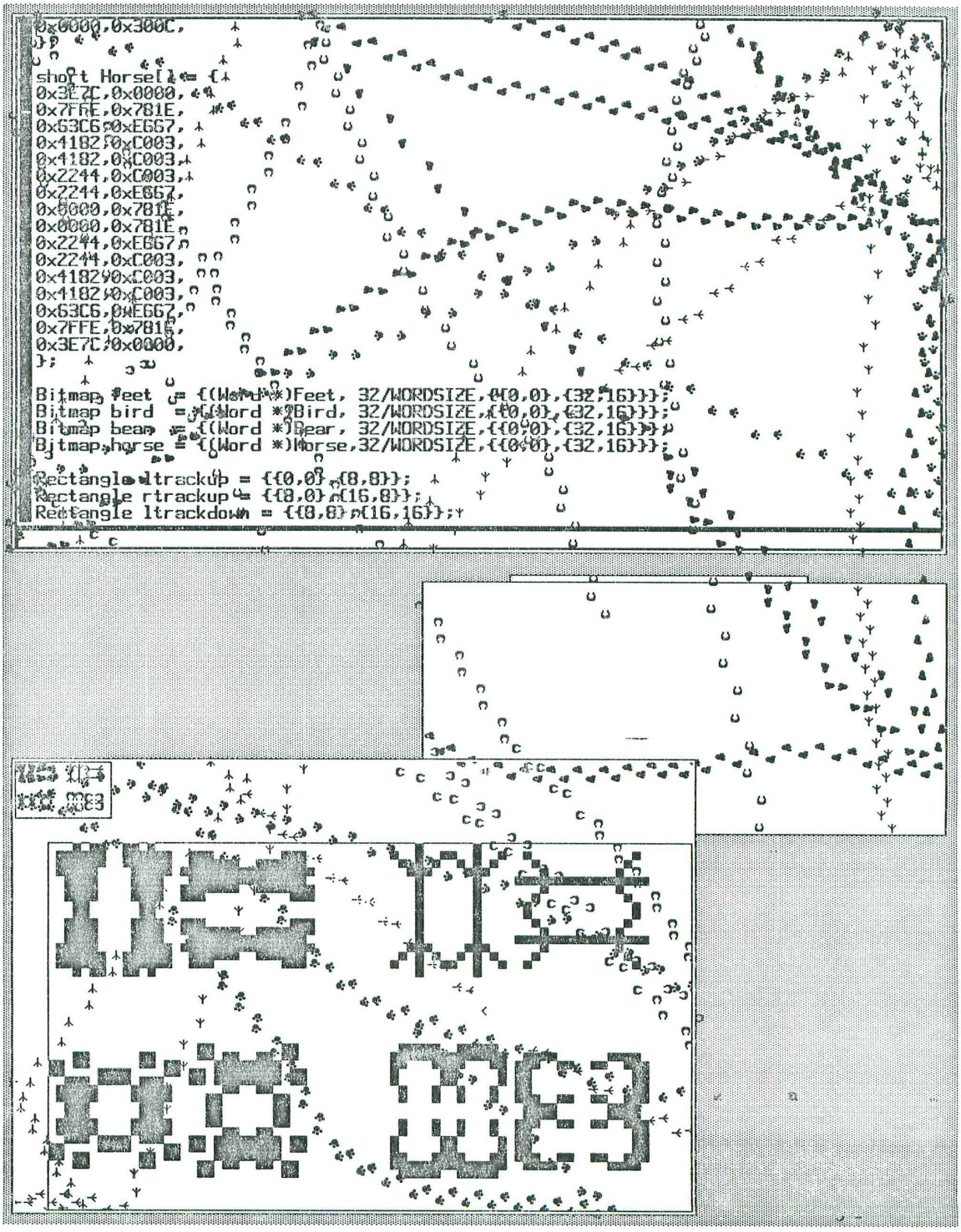
Tracks was written by myself as a crabs spin-off.

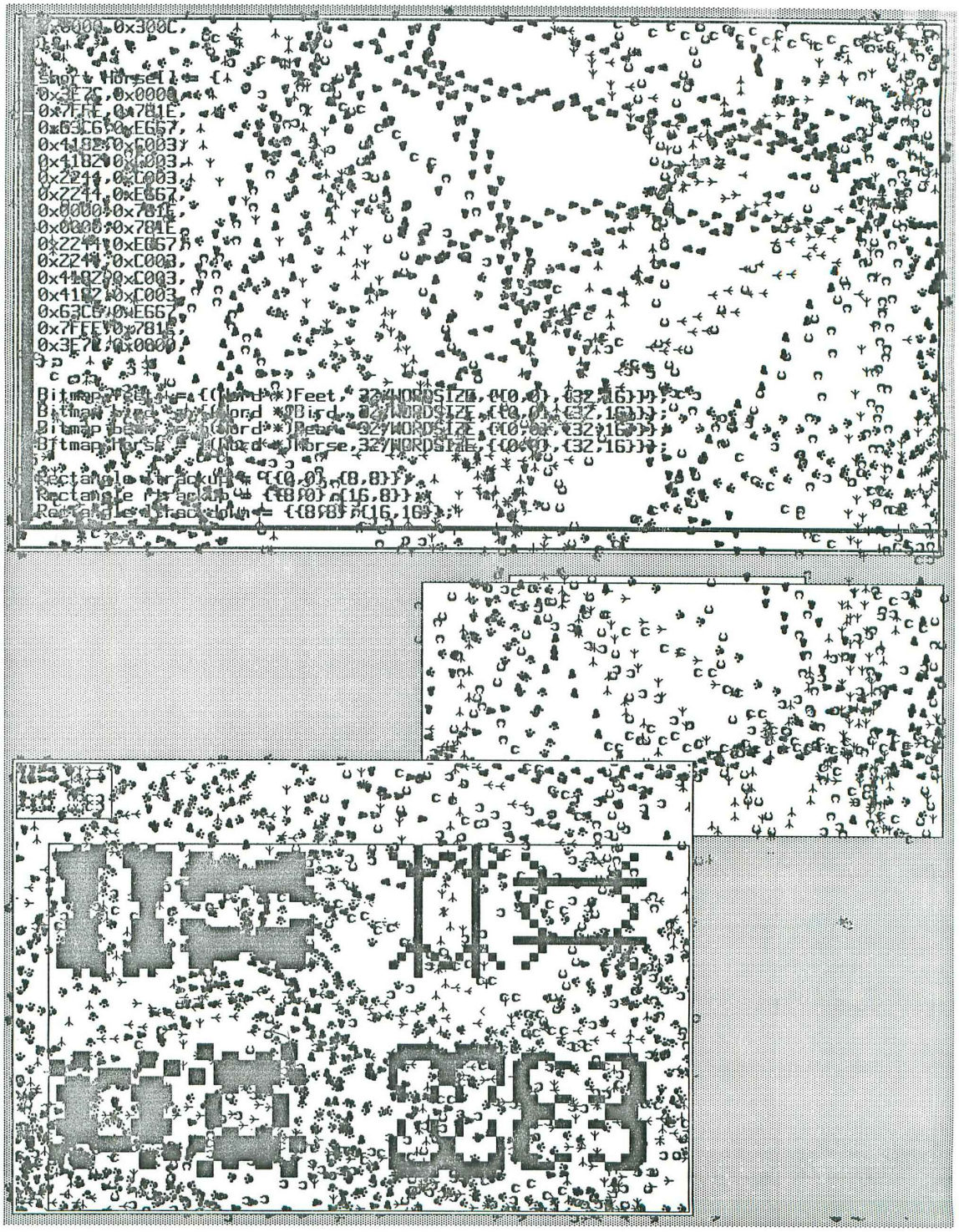


```
0x0000,0x300C,
};

short Horse[] = {
0x3E7C,0x0000,
0x7FFE,0x781E,
0x63C6,0xE667,
0x4182,0xC003,
0x4182,0xC003,
0x2244,0xC003,
0x2244,0xE667,
0x0000,0x781E,
0x0000,0x781E,
0x2244,0xE667,
0x2244,0xC003,
0x4182,0xC003,
0x4182,0xC003,
0x63C6,0xE667,
0x7FFE,0x781E,
0x3E7C,0x0000,
};

Bitmap feet  = {(Word *)Feet, 32/WORDSIZE,{{0,0},{32,16}}};
Bitmap bird  = {(Word *)Bird, 32/WORDSIZE,{{0,0},{32,16}}};
Bitmap bear  = {(Word *)Bear, 32/WORDSIZE,{{0,0},{32,16}}};
Bitmap horse = {(Word *)Horse,32/WORDSIZE,{{0,0},{32,16}}};

Rectangle ltrackup = {{0,0},{8,8}};
Rectangle rtrackup = {{8,0},{16,8}};
Rectangle ltrackdown = {{8,8},{16,16}};
```

```
0x0000,0x300C,
};

short Horse[] = {
0x3E7C,0x0000,
0x7FFE,0x781E,
0x63C6,0xE667,
0x4182,0xC003,
0x4182,0xC003,
0x2244,0xC003,
0x2244,0xE667,
0x0000,0x781E,
0x0000,0x781E,
0x2244,0xE667,
0x2244,0xC003,
0x4182,0xC003,
0x4182,0xC003,
0x63C6,0xE667,
0x7FFE,0x781E,
0x3E7C,0x0000,
};

Bitmap feet  = {(Word *)Feet, 32/WORDSIZE,{{0,0},{32,16}}};
Bitmap bird  = {(Word *)Bird, 32/WORDSIZE,{{0,0},{32,16}}};
Bitmap bear  = {(Word *)Bear, 32/WORDSIZE,{{0,0},{32,16}}};
Bitmap horse = {(Word *)Horse,32/WORDSIZE,{{0,0},{32,16}}};

Rectangle ltrackup = {{0,0},{8,8}};
Rectangle rtrackup = {{8,0},{16,8}};
Rectangle ltrackdown = {{8,8},{16,16}};
```




0x0000,0x300C,
short Horse[] = {
0x3E7C,0x0000,
0x7FFE,0x781E,
0x63C6,0xE667,
0x4182,0xC003,
0x4182,0xC003,
0x2244,0xC003,
0x2244,0xE667,
0x0000,0x781E,
0x0000,0x781E,
0x2244,0xE667,
0x2244,0xC003,
0x4182,0xC003,
0x4182,0xC003,
0x63C6,0xE667,
0x7FFE,0x781E,
0x3E7C,0x0000,
};
Bitmap feet = {(Word *)Feet, 32/WORDSIZE,{{0,0},{32,16}}};
Bitmap bird = {(Word *)Bird, 32/WORDSIZE,{{0,0},{32,16}}};
Bitmap bear = {(Word *)Bear, 32/WORDSIZE,{{0,0},{32,16}}};
Bitmap horse = {(Word *)Horse,32/WORDSIZE,{{0,0},{32,16}}};
Rectangle ltrackup = {{0,0},{8,8}};
Rectangle rtrackup = {{8,0},{16,8}};
Rectangle ltrackdown = {{8,8},{16,16}};



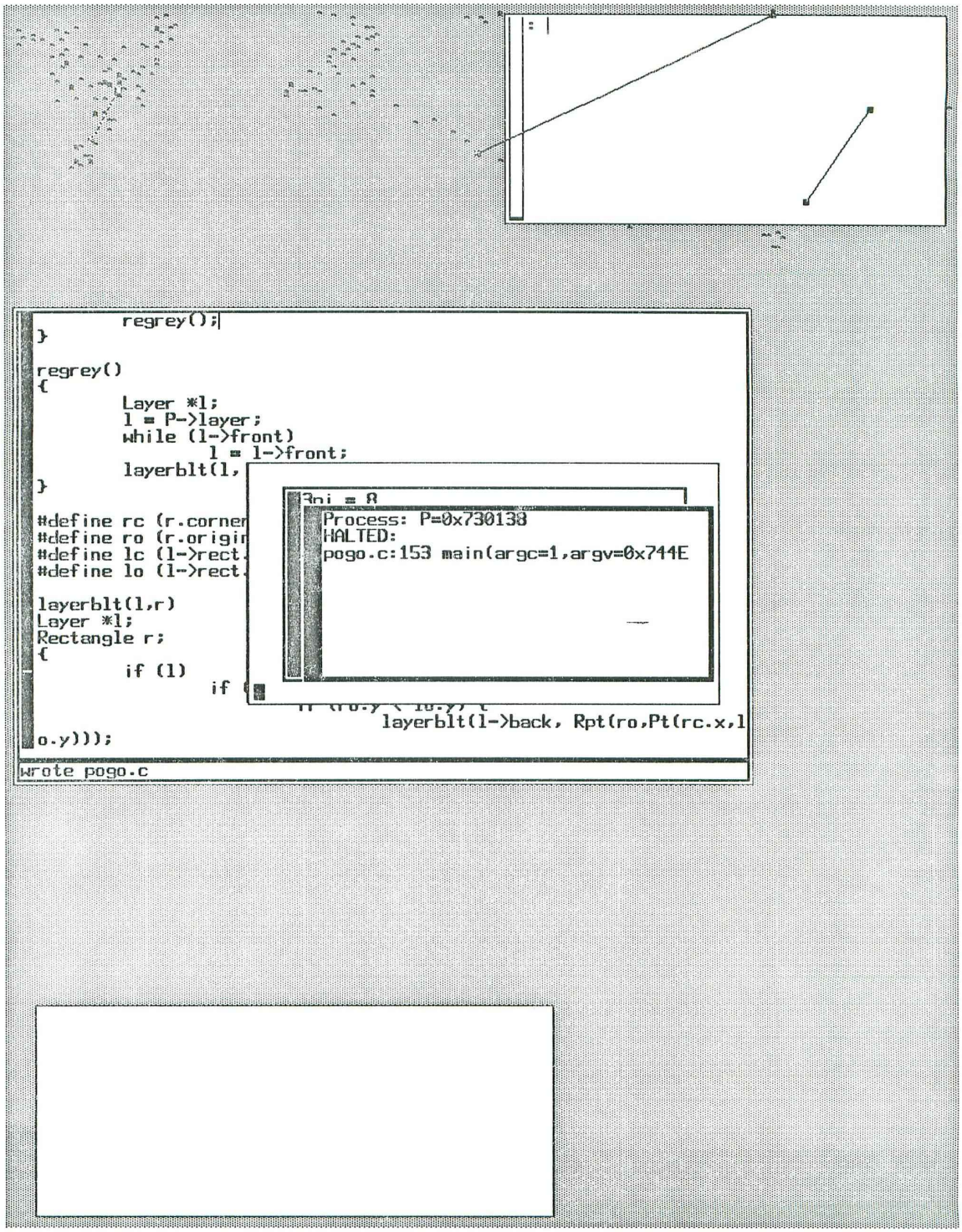
A pogo stick is a pair of bouncers (dots) connected by a stick (line).
Pogo sticks (or "pogos", for short) hate grey, and love any other
color. In non-grey areas, the bouncers float freely, until they bump into
grey areas. When that happens, the bouncers change direction and
bounce off. The opposite bouncers of a pogo loosely attract each other.

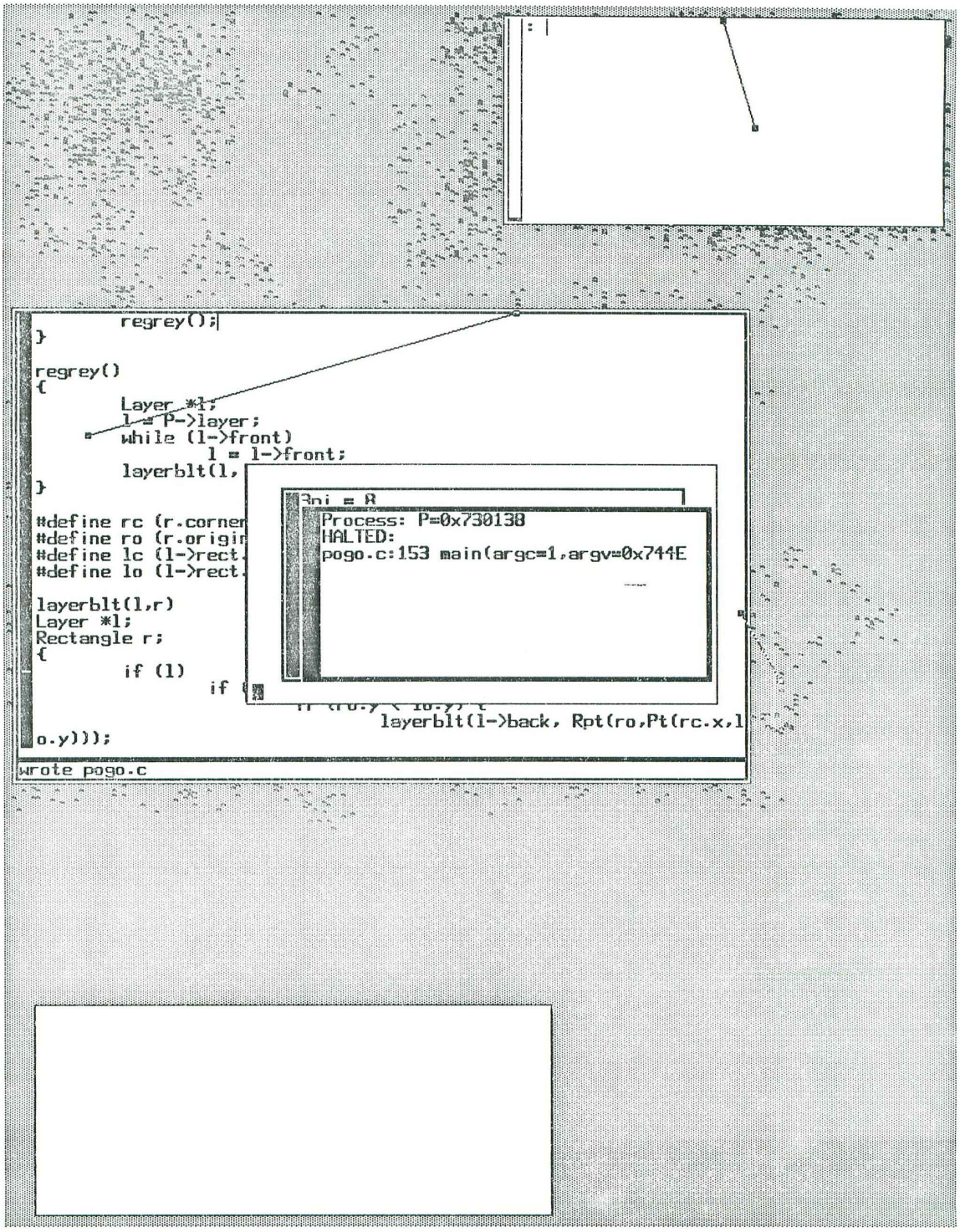
A bouncer may overshoot a boundary between non-grey and grey, because of inertia,
and get temporarily trapped in a grey area. In this situation the bouncer is
continuously bouncing agains grey, and assumes a kind of brownian motion.
Fortunately, the opposite bouncer will very likely pull it out of trouble.
If both bouncers are trapped into grey, the pogo may randomly wander in grey
areas for a long time, looking like it is in a epileptic fit.

However, when a bouncer is trapped in grey, it tries actively to make itself a
home but turning all the grey it touches into black. Eventually this can create
large black areas where the pogo can again float free.

Luca Cardelli

Pogo was written by Mark Manasse, as a crabs spin-off.


regrey();
}
regrey()
{
Layer *l;
l = P->layer;
while (l->front)
l = l->front;
layerblt(l,
}
#define rc (r.corner
#define ro (r.origin
#define lc (l->rect.
#define lo (l->rect.
layerblt(l,r)
Layer *l;
Rectangle r;
{
if (l)
if (
layerblt(l->back, Rpt(ro,Pt(rc.x,l
o.y)));
Process: P=0x730138
HALTED:
pogo.c:153 main(argc=1,argv=0x744E
wrote pogo.c


:
regrey();
}
regrey()
{
Layer *l;
l = P->layer;
while (l->front)
l = l->front;
layerblt(l,
}
#define rc (r.corner
#define ro (r.origi
#define lc (l->rect.
#define lo (l->rect.
layerblt(l,r)
Layer *l;
Rectangle r;
{
if (l)
if (
Process: P=0x730138
HALTED:
pogo.c:153 main(argc=1,argv=0x744E
layerblt(l->back, Rpt(ro,Pt(rc.x,l
o.y)));
wrote pogo.c

# Screen Wars

Here we see crabs fighting against pogo sticks for the control of the screen.

(Figure Pogo.1) Three pogo sticks start at the top of the screen.

(Figure Pogo.2) Two pogos have migrated to the lower window.
A pogo in a large white space tends to stay there for long periods, because it is very unlikely that it will have both bouncers outside at the same time and on the same side, so that they can wander off. It is likely that pogos will eventually migrate to the largest window available.

(Figure ScreenWars.1) The third pogo also migrated to the large window, and at the top of the screen there are new intruders: crabs.

(Figure ScreenWars.2) Here the crabs have eaten half of the upper window and have attacked the large window. Meanwhile the pogos have sprinkled black at the perimeter of the window.

(Figure ScreenWars.3) A pogo wandered in the top left corner, aquiring a lot of black territory. At the same time a single crab was trapped inside the large window by the black stuff pogos drop, and deeply eroded it. The crabs have totally eaten the top window and have invaded the lower regions of the screen.
The pogos are gaining territory around the window, where the black stuff keeps the crabs away.

(Figure ScreenWars.4) Pogos have control of the center and top left of the screen, except for a crab trapped in the top left which maintains its own grey territory, and another crab which goes deeper and deeper in the large window. The crabs have control of the bottom and right side of the screen, which is now full of crab-shit (crab-shit is the product of a crab biting another crab; it is a non-obvious side-effect of the crabs drawing algorithm).

(Figure ScreenWars.5) More of the same.

(Figure ScreenWars.6) The pogos keep slowly gaining territory. One of the prisoner crabs has escaped, the other one is moving to the right. It is not clear why these crabs are moving coherently, over large periods, in one direction; maybe there is a slight bias in the random walk algorithm.
(At this point I went to sleep)

(Figure ScreenWars.7) Seven hours later the situation hasn't changed much, except that the other prisoner crab has escaped. The boundary between crabs and pogos is sharp and stable. The whole process took 12 hours.

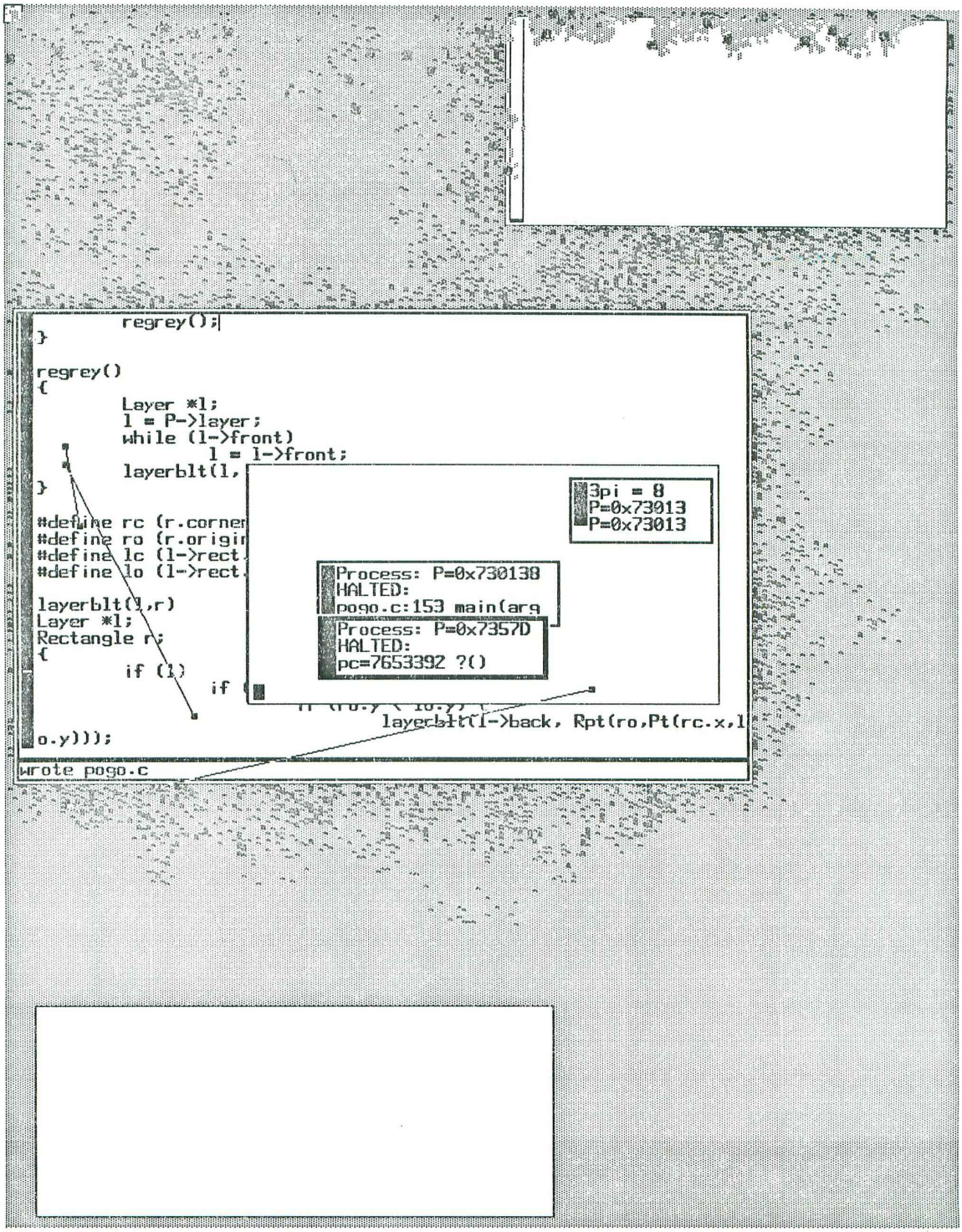
Other fights:

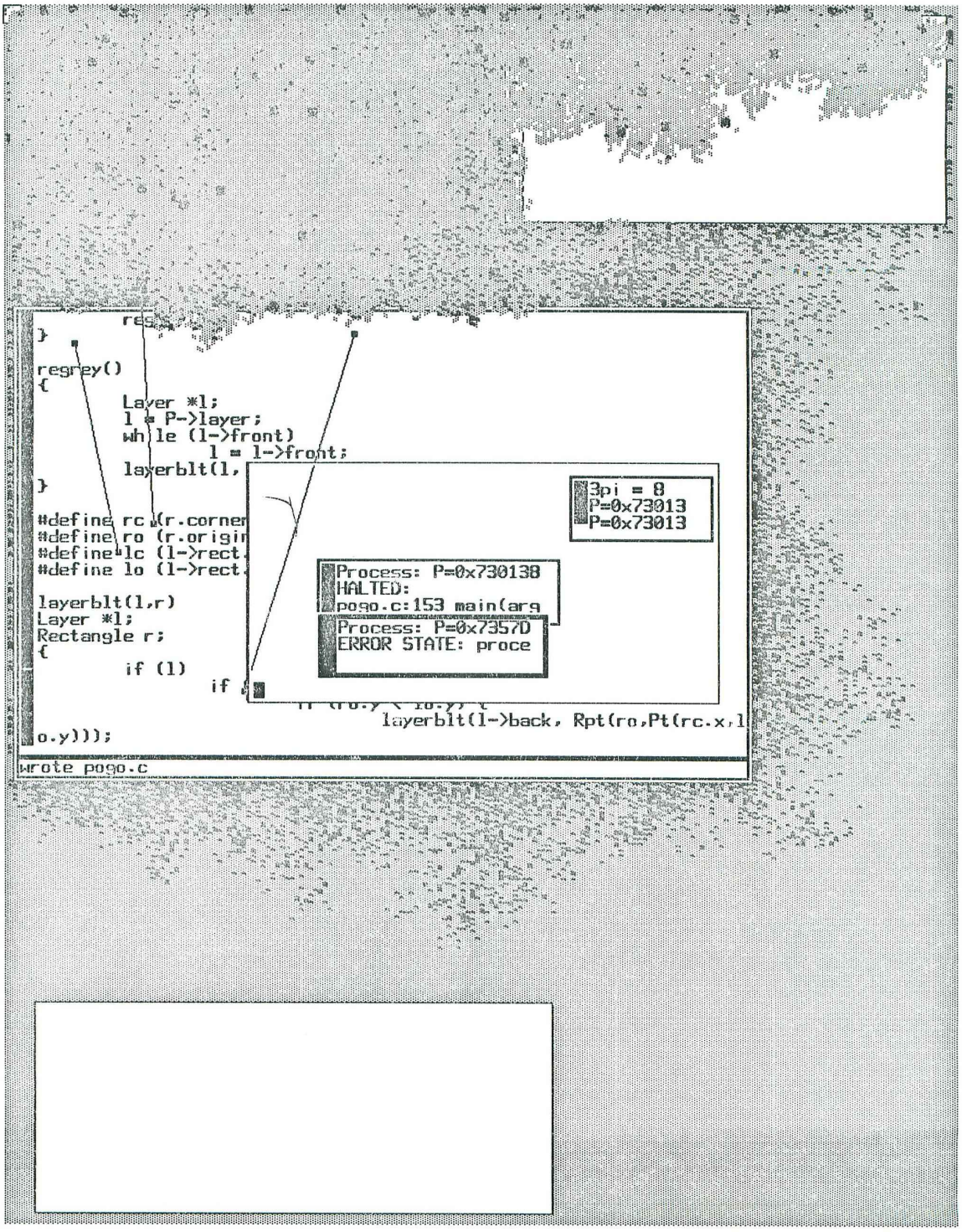
Tracks do not stand a chance against crabs, because tracks do not attack crab territory (unless they happen to step on a crab, in which case they leave a footprint there, but this is unfrequent), while crabs attack tracks territory. Eventally, tracks loose their "footing".

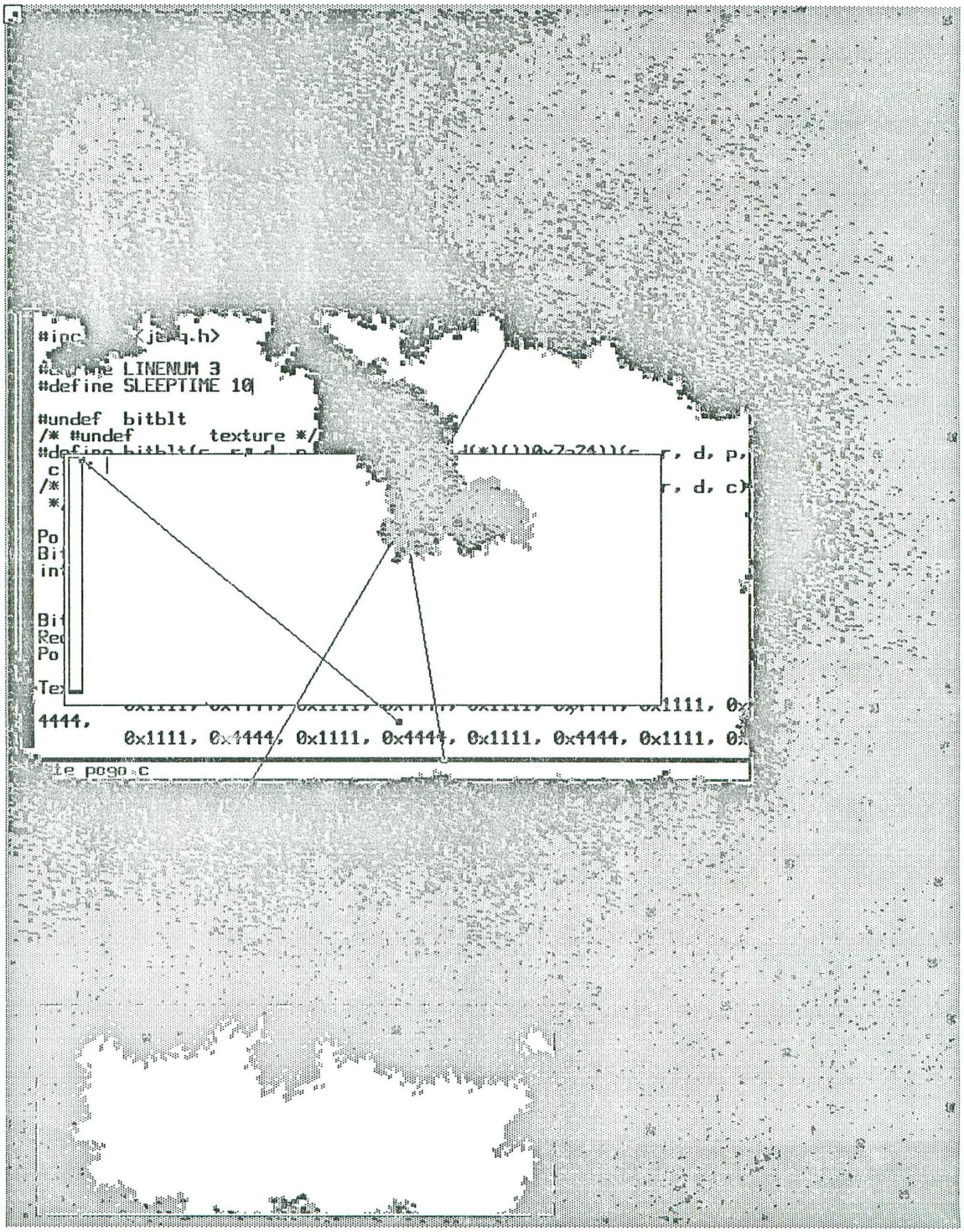
Tracks and pogos cooperate, and the result is a totally black screen.

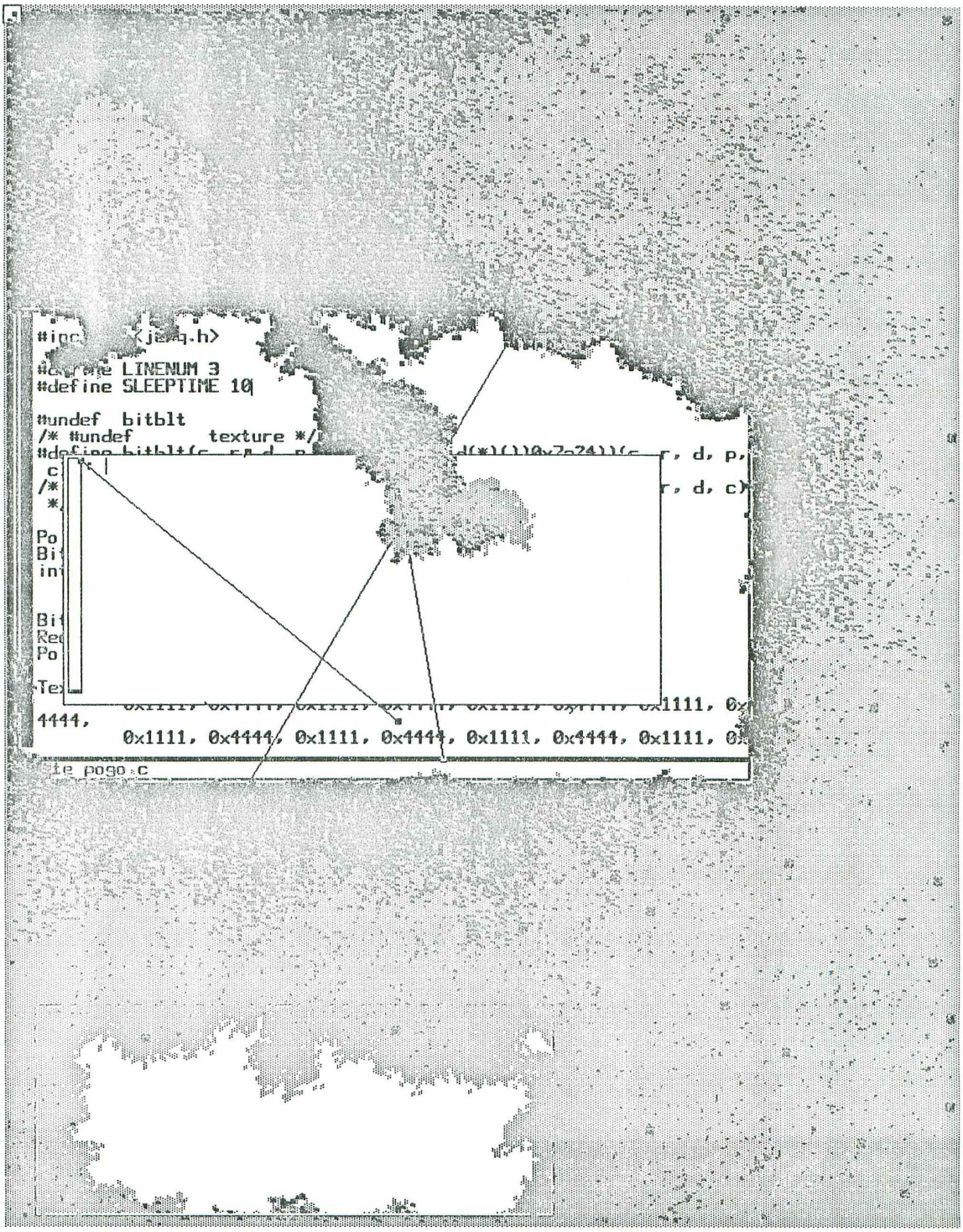
Luca Cardelli

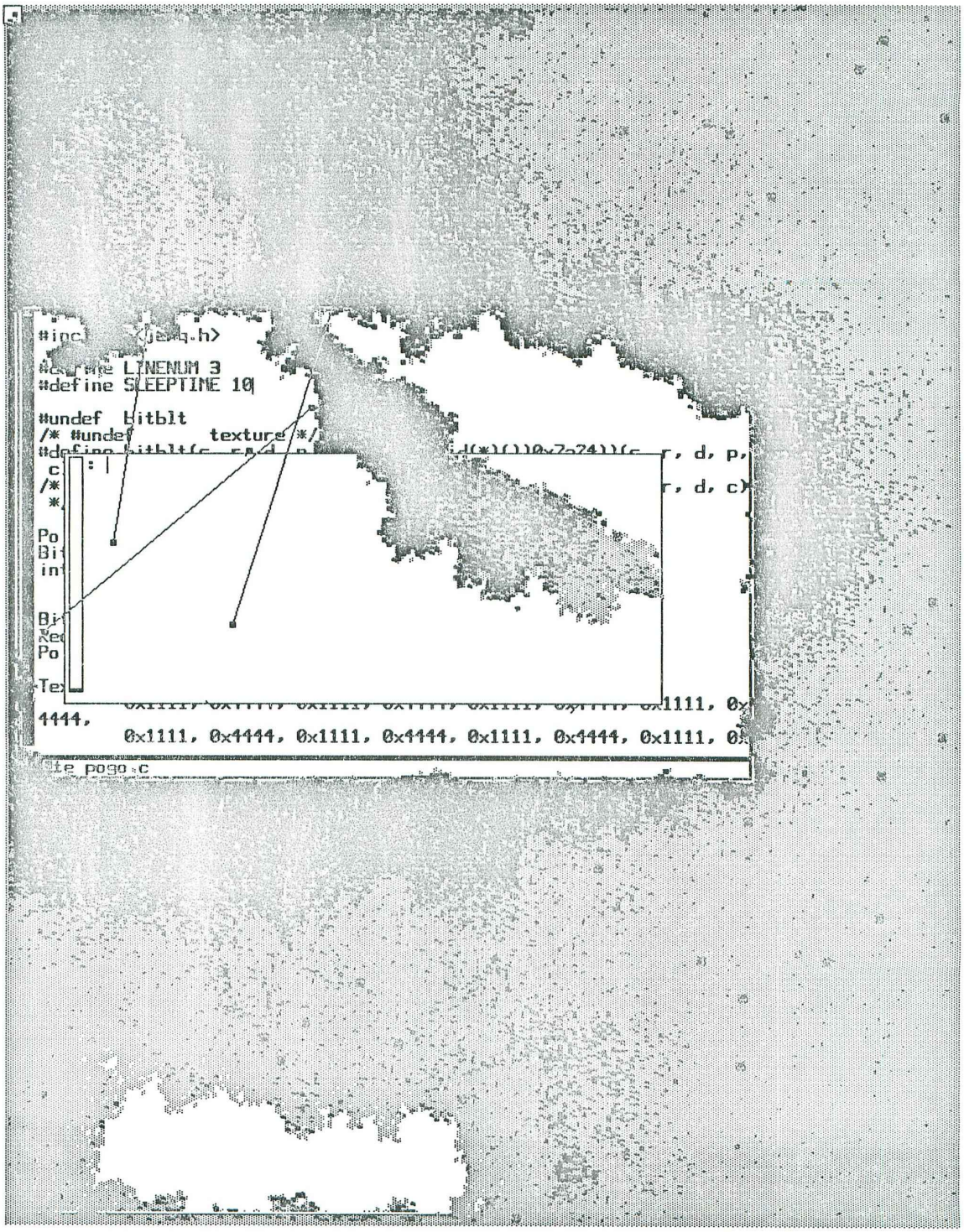
P.S. some inconsistencies in the figures, e.g. windows appearing and disappearing, are due to the fact that I have to use a couple of windows to print out the screen dumps, and I have to fight pogos and crabs while doing that.

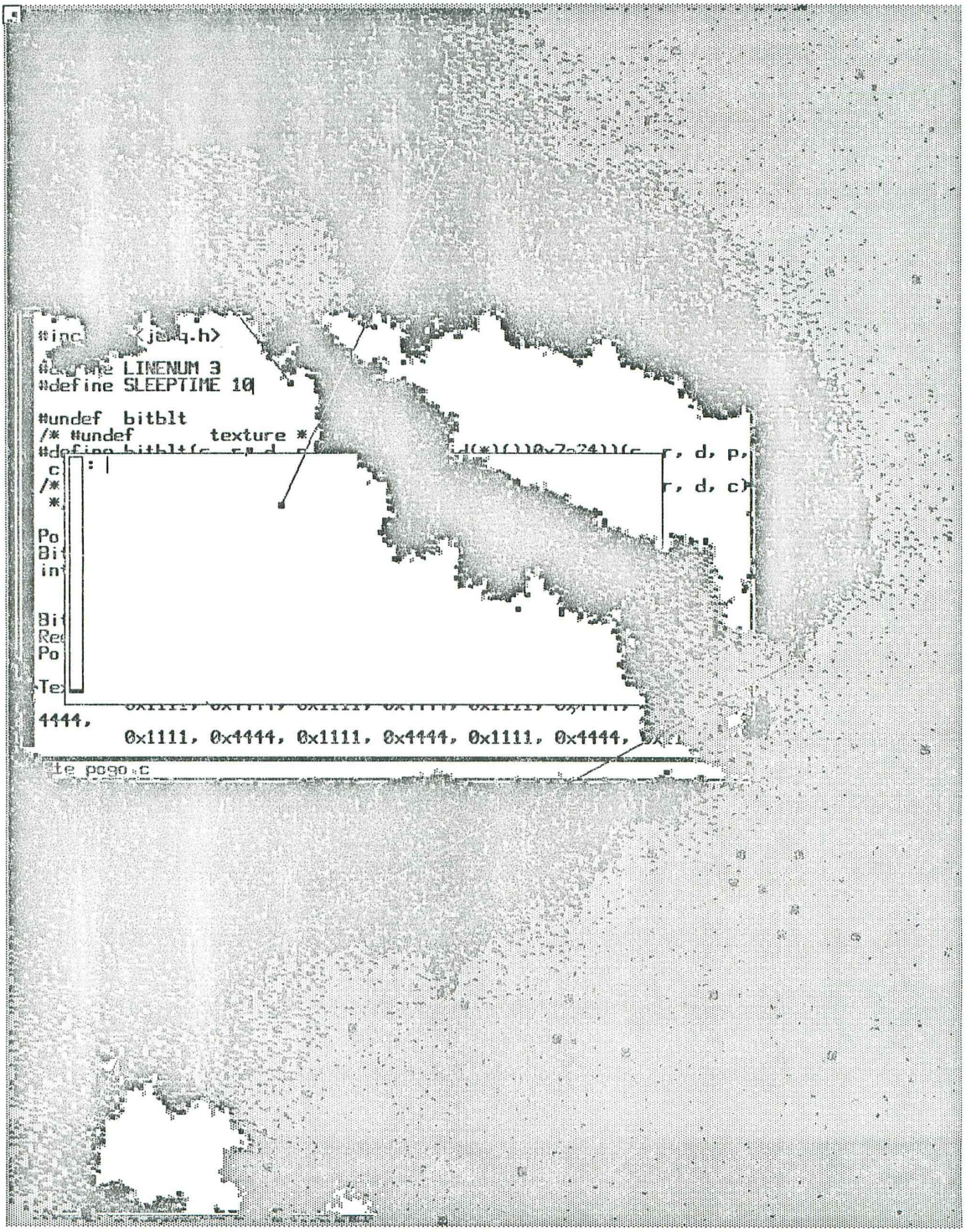

regrey();
}
regrey()
{
Layer *l;
l = P->layer;
while (l->front)
l = l->front;
layerblt(l,
}
#define rc (r.corner
#define ro (r.origin
#define lc (l->rect.
#define lo (l->rect.
layerblt(l,r)
Layer *l;
Rectangle r;
{
if (l)
if (
layerblt(l->back, Rpt(ro,Pt(rc.x,l
o.y)));
wrote pogo.c
3pi = 8
P=0x73013
P=0x73013
Process: P=0x730138
HALTED:
pogo.c:153 main(arg
Process: P=0x7357D
HALTED:
pc=7653392 ?()

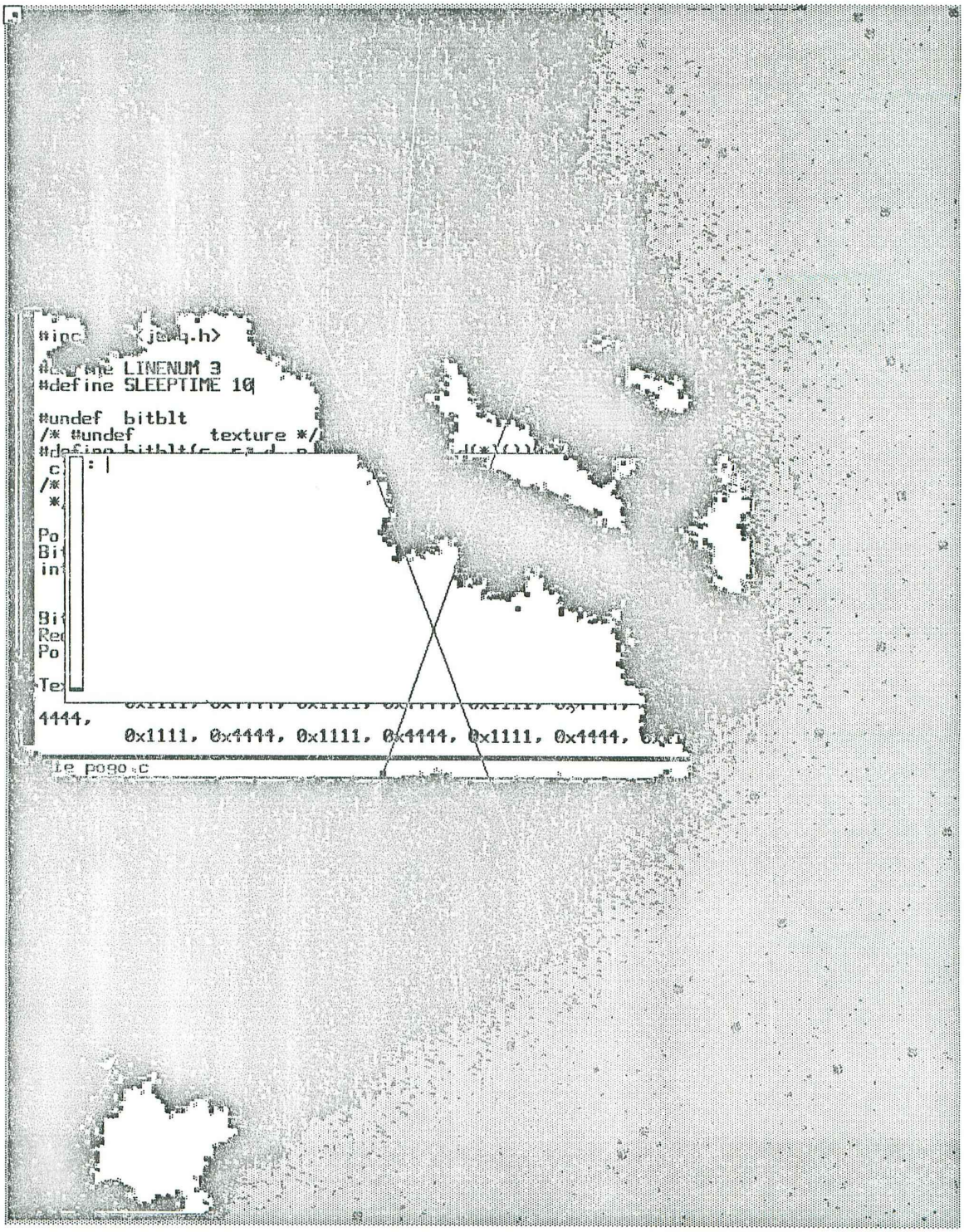

}
regrey()
{
Layer *l;
l = P->layer;
while (l->front)
l = l->front;
layerblt(l,
}
#define rc (r.corner
#define ro (r.origin
#define lc (l->rect.
#define lo (l->rect.
layerblt(l,r)
Layer *l;
Rectangle r;
{
if (l)
if (
layerblt(l->back, Rpt(ro,Pt(rc.x,l
o.y)));
3pi = 8
P=0x73013
P=0x73013
Process: P=0x730138
HALTED:
pogo.c:153 main(arg
Process: P=0x7357D
ERROR STATE: proce
wrote pogo.c

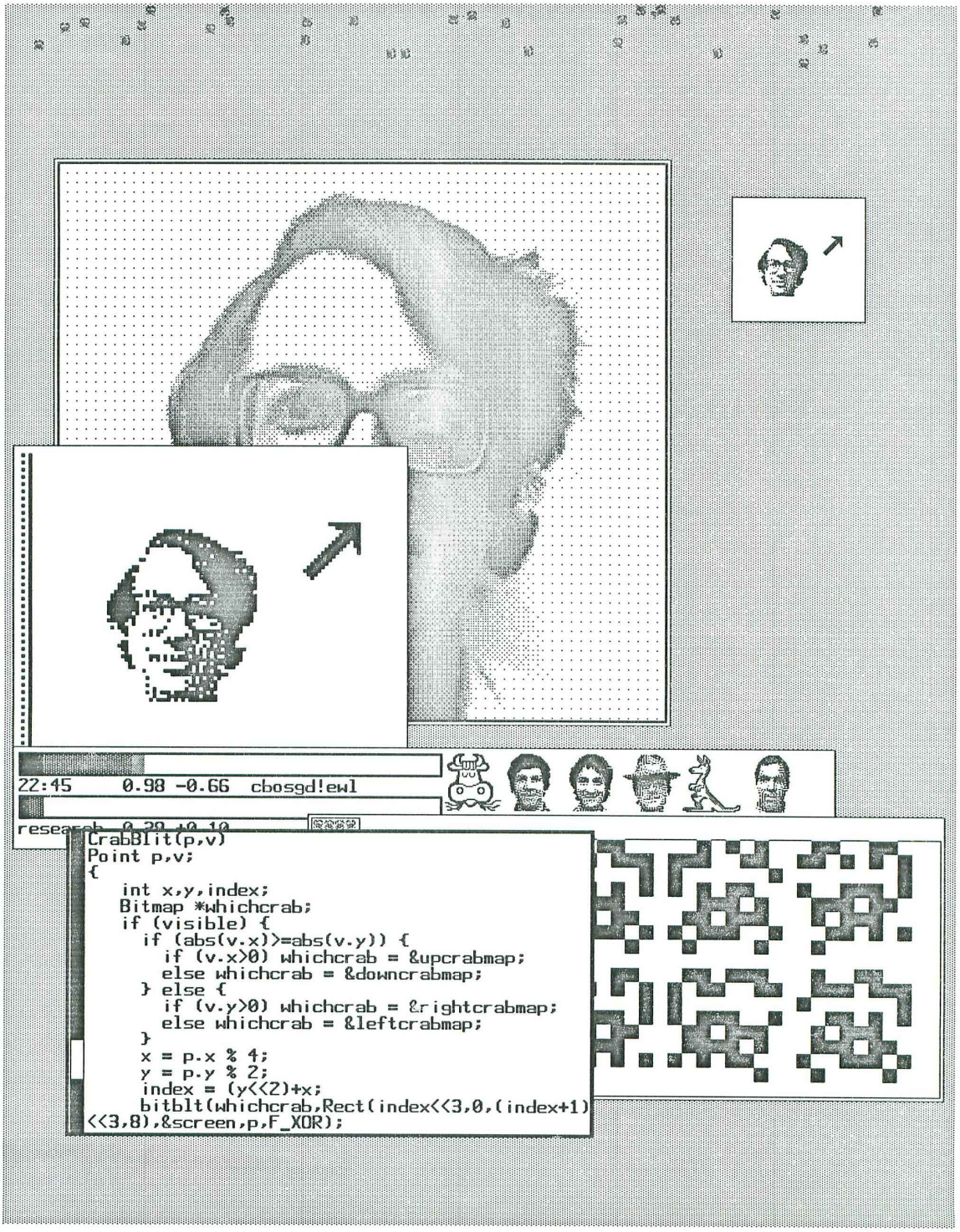

#undef bitblt
/* #undef texture */
#define LINENUM 3
#define SLEEPTIME 10
0x1111, 0x4444, 0x1111, 0x4444, 0x1111, 0x4444, 0x1111, 0x
te pogo.c

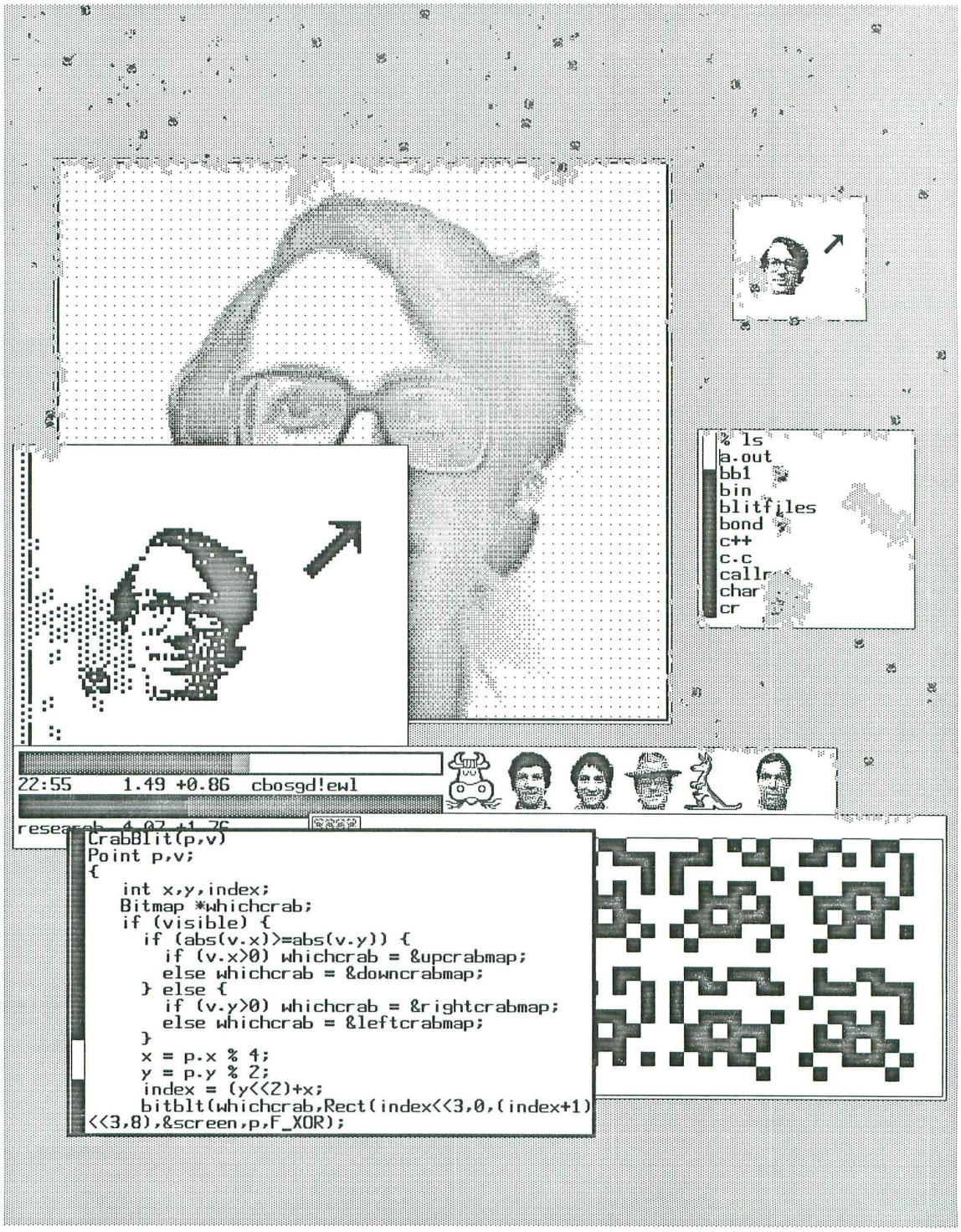

#inc <jerq.h>
#define LINENUM 3
#define SLEEPTIME 10
#undef bitblt
/* #undef texture */
0x1111, 0x4444, 0x1111, 0x4444, 0x1111, 0x4444, 0x1111, 0x
ie pogo.c

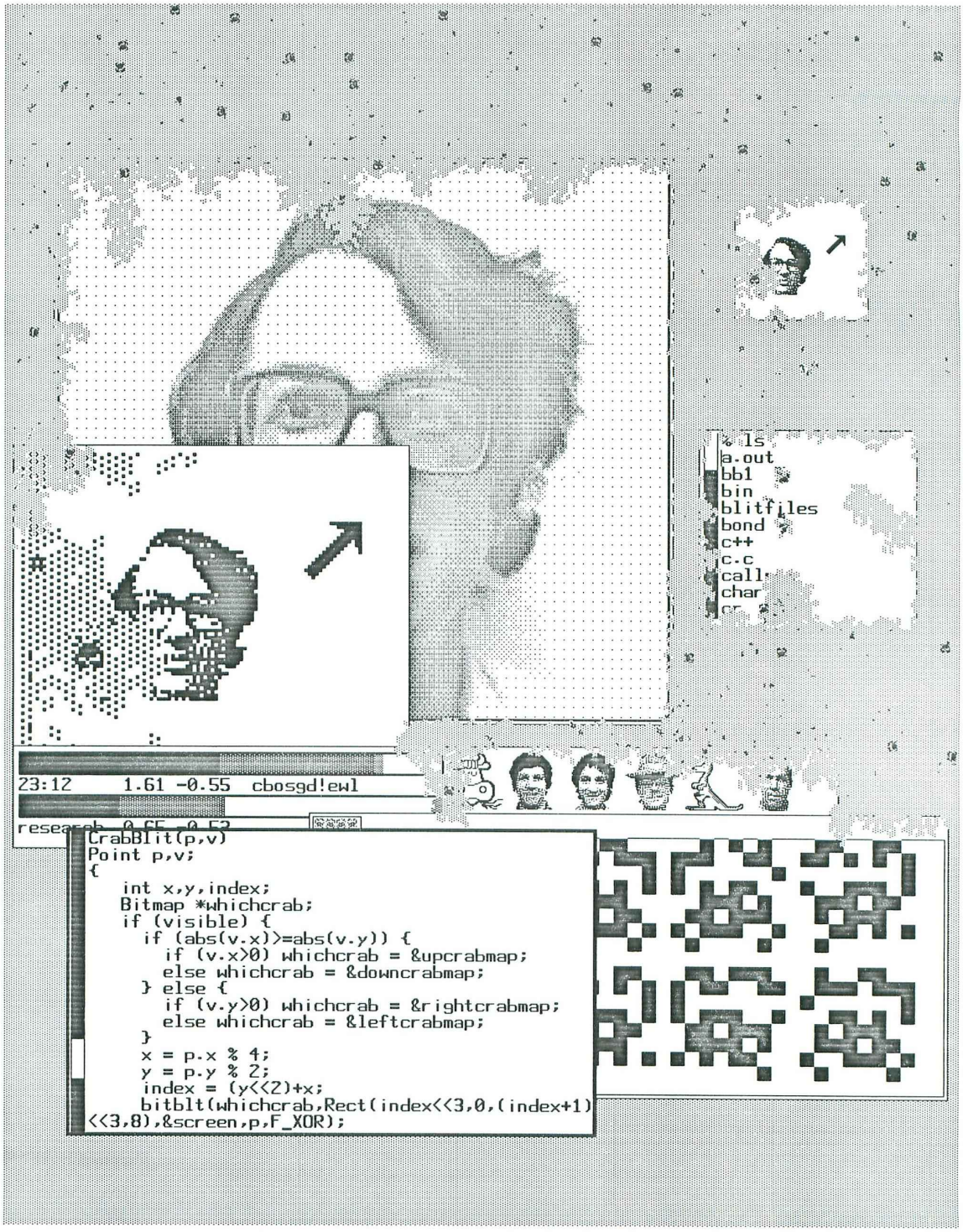

#define LINENUM 3
#define SLEEPTIME 10
#undef Bitblt
/* #undef texture */
0x1111, 0x4444, 0x1111, 0x4444, 0x1111, 0x4444, 0x1111, 0x
4444,
0x1111, 0x4444, 0x1111, 0x4444, 0x1111, 0x4444, 0x1111, 0x
ie pogo.c

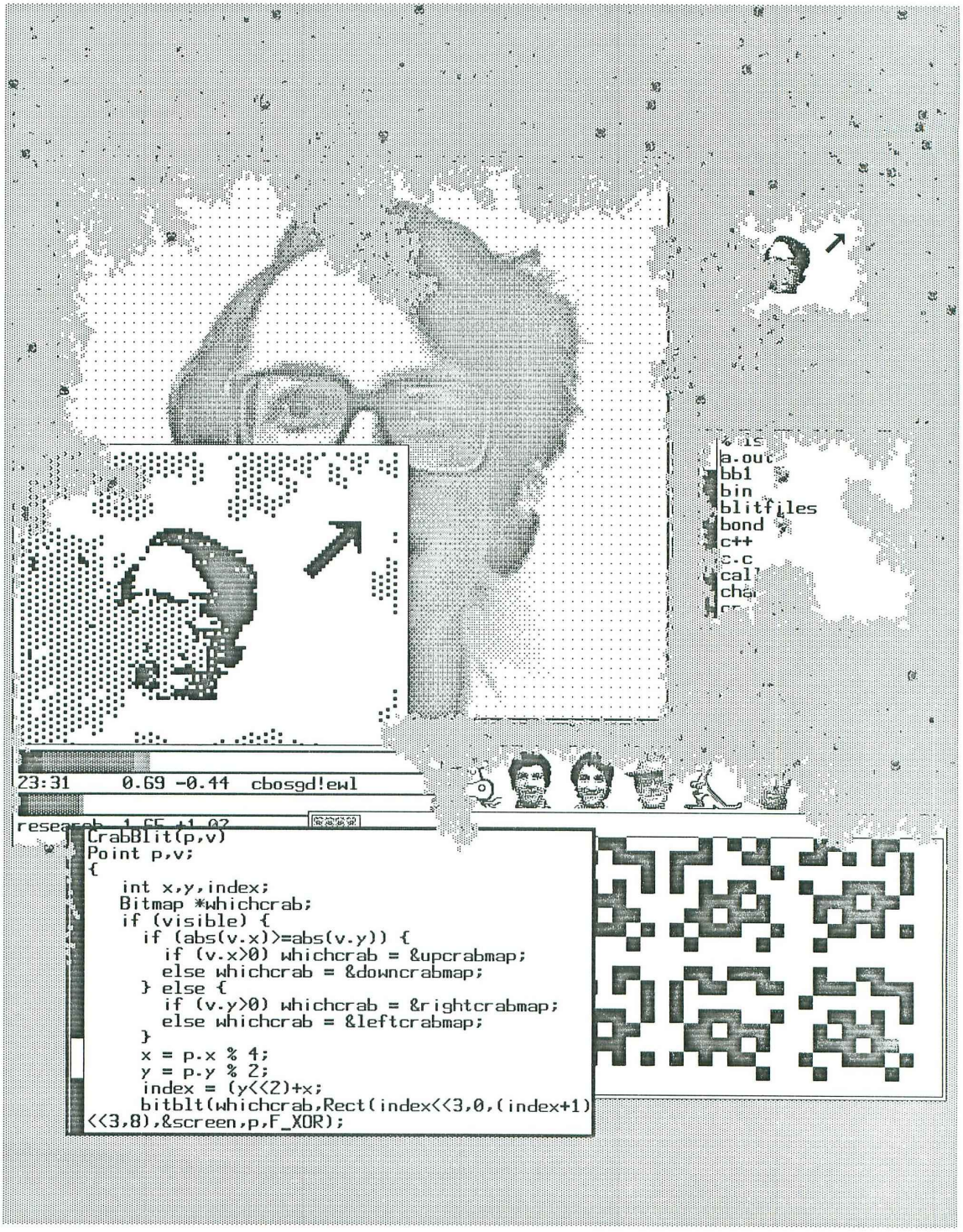

#inc <jagq.h>
#define LINENUM 3
#define SLEEPTIME 10
#undef bitblt
/* #undef texture *
4444,
0x1111, 0x4444, 0x1111, 0x4444, 0x1111, 0x4444,
te pogo.c

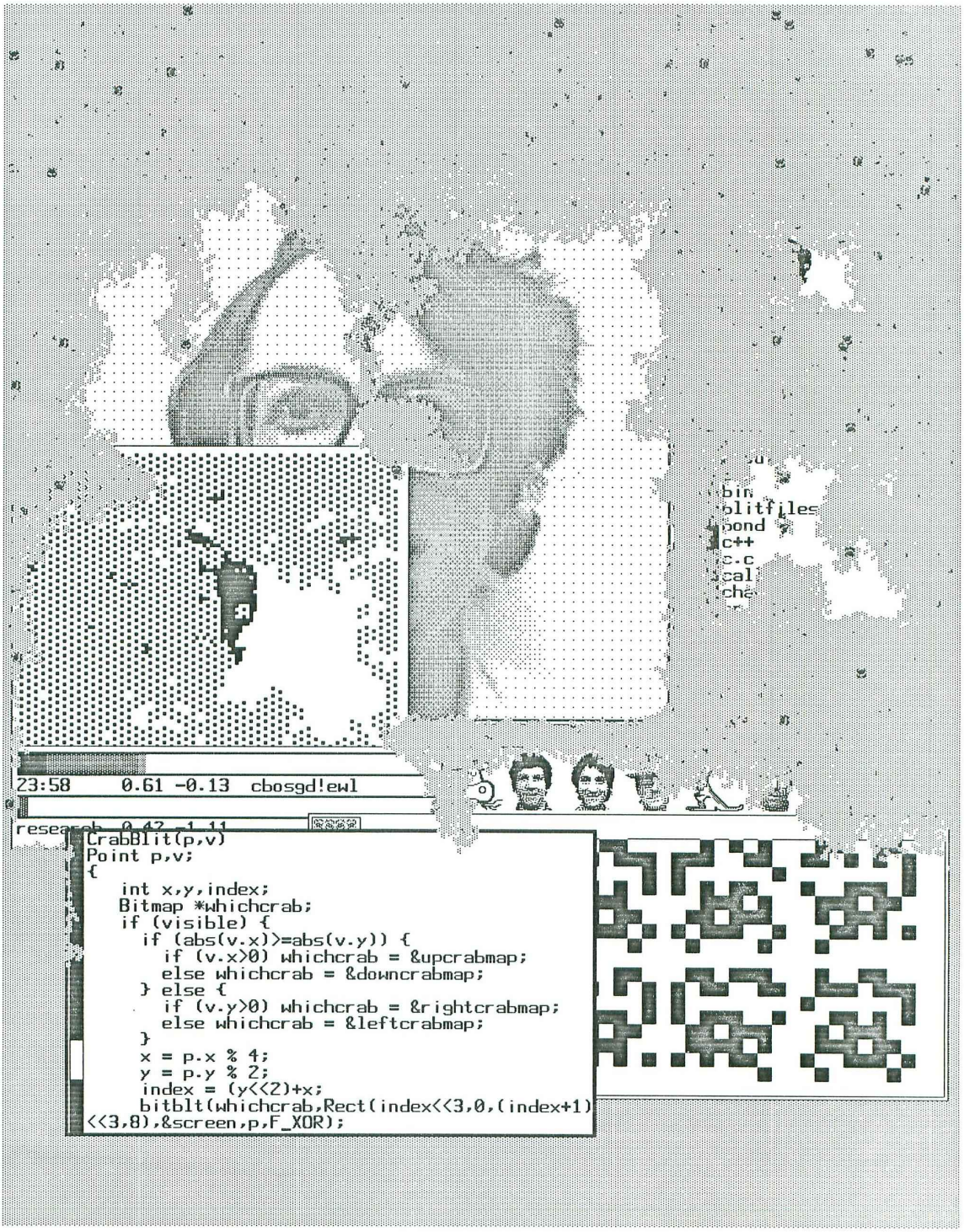

#define LINENUM 3
#define SLEEPTIME 10
#undef bitblt
/* #undef texture */
0x1111, 0x4444, 0x1111, 0x4444, 0x1111, 0x4444,


22:45 0.98 -0.66 cbosgd!ewl
CrabBlit(p,v)
Point p,v;
{
int x,y,index;
Bitmap *whichcrab;
if (visible) {
if (abs(v.x)>=abs(v.y)) {
if (v.x>0) whichcrab = &upcrabmap;
else whichcrab = &downcrabmap;
} else {
if (v.y>0) whichcrab = &rightcrabmap;
else whichcrab = &leftcrabmap;
}
x = p.x % 4;
y = p.y % 2;
index = (y<<2)+x;
bitblt(whichcrab,Rect(index<<3,0,(index+1)
<<3,8),&screen,p,F_XOR);


% ls
a.out
bb1
bin
blitfiles
bond
c++
c.c
callr
char
cr
22:55 1.49 +0.86 cbosgd!ewl
CrabBlit(p,v)
Point p,v;
{
int x,y,index;
Bitmap *whichcrab;
if (visible) {
if (abs(v.x)>=abs(v.y)) {
if (v.x>0) whichcrab = &upcrabmap;
else whichcrab = &downcrabmap;
} else {
if (v.y>0) whichcrab = &rightcrabmap;
else whichcrab = &leftcrabmap;
}
x = p.x % 4;
y = p.y % 2;
index = (y<<2)+x;
bitblt(whichcrab,Rect(index<<3,0,(index+1)
<<3,8),&screen,p,F_XOR);


% ls
a.out
bb1
bin
blitfiles
bond
c++
c.c
call
char
23:12 1.61 -0.55 cbosgd!ewl
CrabBlit(p,v)
Point p,v;
{
int x,y,index;
Bitmap *whichcrab;
if (visible) {
if (abs(v.x)>=abs(v.y)) {
if (v.x>0) whichcrab = &upcrabmap;
else whichcrab = &downcrabmap;
} else {
if (v.y>0) whichcrab = &rightcrabmap;
else whichcrab = &leftcrabmap;
}
x = p.x % 4;
y = p.y % 2;
index = (y<<2)+x;
bitblt(whichcrab,Rect(index<<3,0,(index+1)
<<3,8),&screen,p,F_XOR);


% ls
a.out
bb1
bin
blitfiles
bond
c++
c.c
cal
cha
23:31 0.69 -0.44 cbosgd!ewl
CrabBlit(p,v)
Point p,v;
{
int x,y,index;
Bitmap *whichcrab;
if (visible) {
if (abs(v.x)>=abs(v.y)) {
if (v.x>0) whichcrab = &upcrabmap;
else whichcrab = &downcrabmap;
} else {
if (v.y>0) whichcrab = &rightcrabmap;
else whichcrab = &leftcrabmap;
}
x = p.x % 4;
y = p.y % 2;
index = (y<<2)+x;
bitblt(whichcrab,Rect(index<<3,0,(index+1)
<<3,8),&screen,p,F_XOR);


bin
blitfiles
bond
c++
c.c
cal
che
23:58 0.61 -0.13 cbosgd!ewl
CrabBlit(p,v)
Point p,v;
{
int x,y,index;
Bitmap *whichcrab;
if (visible) {
if (abs(v.x)>=abs(v.y)) {
if (v.x>0) whichcrab = &upcrabmap;
else whichcrab = &downcrabmap;
} else {
if (v.y>0) whichcrab = &rightcrabmap;
else whichcrab = &leftcrabmap;
}
x = p.x % 4;
y = p.y % 2;
index = (y<<2)+x;
bitblt(whichcrab,Rect(index<<3,0,(index+1)
<<3,8),&screen,p,F_XOR);