

Cellular Automaton Tool User Manual

Gesellschaft für Mathematik und Datenverarbeitung
Postfach 1316
D - 53731 Sankt Augustin 1

Cellular Automaton Tool
Version 8.10.93

User Manual

Author:
Georg Jünger

(C) Copyright 1993 by GMD, Sankt Augustin

Table of Contents

| | |
|---|----|
| 1. Introductory Notes | 4 |
| 2. Quick Tour through CAT | 5 |
| 2.1 Starting CAT | 5 |
| 2.2 Loading and Compiling a Program | 6 |
| 2.3 Executing and Controlling a Program | 7 |
| 2.4 Editing a Program | 9 |
| 2.5 Printing | 10 |
| 2.6 Window Handling | 11 |
| 2.7 Changing Properties and Appearance of the Cell Matrix | 11 |
| 2.8 Using the Online Help System | 14 |
| 2.9 Leaving CAT | 15 |
| 3. Complete Overview of Windows, Menus, Buttons and Boxes | 16 |
| 3.1 Windows | 16 |
| 3.1.1. CAT Main Window | 16 |
| 3.1.2. STATE Window | 17 |
| 3.1.3. RECIPE Window | 18 |
| 3.1.4. LIST Window | 18 |
| 3.1.5. Plane Window (Starring .CAS File) | 19 |
| 3.1.6. Palette Window (Starring .CAP File) | 19 |
| 3.2 Menues | 20 |
| 3.2.1. 'File' | 20 |
| 3.2.2. 'Edit' | 20 |
| 3.2.3. 'Search' | 21 |
| 3.2.4. 'Window' | 21 |
| 3.2.5. '?' (Help) | 21 |
| 3.2.6. Menu Accessible via State Control Button | 22 |
| 3.2.7. Menu Accessible via Palette Control Button | 22 |
| 3.2.8. Menu Accessible via Menu Button | 23 |
| 3.2.9. Local Menu Accessible via Right Mouse Key | 23 |
| 3.2.10. Local Menu Accessible via Shift / Right Mouse Key | 23 |
| 3.3 Buttons and Labels | 24 |
| 3.3.1. Buttons and Labels Contained in the Main Window | 24 |
| 3.3.2. Buttons and Labels Contained in the STATE Window | 24 |
| 3.4 Dialog Boxes | 25 |
| 3.4.1. 'Colors' Dialog Box | 25 |
| 3.4.2. 'Assign Value' Dialog Box | 26 |
| 3.4.3. 'Initialize States' Dialog Box | 26 |
| 3.4.4. 'Initialize Palette' Dialog Box | 27 |
| 4. CARP Instructions Reference Guide | 28 |
| % | 28 |
| * | 29 |
| + | 29 |
| - | 29 |
| := | 30 |
| < | 30 |
| <= | 30 |
| ◇ | 31 |
| = | 31 |
| > | 31 |
| >= | 32 |
| AND | 32 |
| Any8Sum | 32 |
| BarrelForm | 33 |
| Beep | 33 |
| BEGIN ... END | 34 |
| Brake | 34 |
| Comments in a CARP program | 34 |
| Colors | 35 |
| CONST | 35 |
| DelBrushes | 36 |
| DIV | 36 |

| | |
|--|----|
| EVENT | 36 |
| Expressions | 37 |
| FOR ... TO ...BY ... DO ... OD | 37 |
| GetX | 38 |
| GetY | 38 |
| Identifiers | 39 |
| IF .. THEN .. ELSE .. FI | 39 |
| INV | 39 |
| MOD | 40 |
| MooreSum | 40 |
| NeumannSum | 41 |
| NOT | 41 |
| OddCell | 42 |
| OR | 42 |
| PARALLEL DO | 42 |
| ParallelMethod | 43 |
| PillowForm | 43 |
| PipeForm | 44 |
| PIClipActive | 44 |
| PIClipAll | 44 |
| PIClipXY | 45 |
| PIFillRandom | 45 |
| PIFillUni | 45 |
| PIFillUpStairs | 46 |
| PROC | 46 |
| Random | 47 |
| Randomize | 47 |
| RECIPE | 48 |
| REF | 48 |
| RePaint | 49 |
| REPEAT .. UNTIL | 49 |
| RGBBrush | 50 |
| RGBPalette | 51 |
| RingForm | 51 |
| Self | 52 |
| SetLattice | 53 |
| SheetForm | 53 |
| SHL | 53 |
| ShowCell | 54 |
| ShowKind | 54 |
| ShowPlane | 55 |
| SHR | 55 |
| Statement | 56 |
| VAR | 57 |
| WHILE ... DO ... OD | 57 |
| WinClipActive | 57 |
| WinClipXY | 58 |
| WinClipAll | 58 |
| WrDCaps | 58 |
| WRITE | 59 |
| WrMCaps | 59 |
| WrPPars | 60 |
| XOR | 60 |
| XYBound | 61 |
| XYSize | 61 |
| YSize | 62 |
| Zet | 62 |
| Appendix A: Hardware and Software Requirements | 63 |
| Appendix B: Installation of CAT | 63 |
| Appendix C: Troubleshooting | 63 |
| Appendix D: Known Bugs | 63 |
| Appendix E: Runtime Message List | 64 |
| Appendix F: Compiler Error Message List | 65 |

1. Introductory Notes

The Cellular Automaton Tool (CAT) is a tool to build and visualize cellular automata in an easy-to-use way. Its pascal-like language CARP (**C**ellular **A**utomaton **P**rogramming Language) is an adequate means to build cellular automata of your own or to transfer samples from the literature.

First and foremost, CAT uses a matrix of colored cells (in the "STATE" window) to show the state of each cell of the cellular automaton for each generation. Apart from a representation in colors, the cell state can also be expressed in decimal or hexadecimal figures.

Both features, high-level language and visualization, should make it attractive to learn about the fascinating world of such models of the "real" world. On the same level of importance, CAT may be taken as a tool to learn about the parallel computing paradigm. Each cell is then taken as a processor that works concurrently with all other cell processors.

For both, beginners or advanced users, it is advisable to start with the "Quick Tour" and execute some sample programs. Later on, experienced users might continue with the CARP language description. He or she will soon grasp the power and possibilities of this language. Beginners should start by changing sample programs. Then, they may enlarge the complexity of their programs and the set of used CARP instructions step by step.

2. Quick Tour through CAT

2.1 Starting CAT

* To start CAT, doubleclick the CAT icon or choose the program manager and doubleclick CAT.EXE in the directory CAT has been installed in.



Figure 2.1 CAT icon

⚡ CAT comes up with its welcome panel. Depending on the CAT's last state and from the size of your data display, the CAT welcome panel may vary. Starting CAT you should be familiar with the picture below anyway.

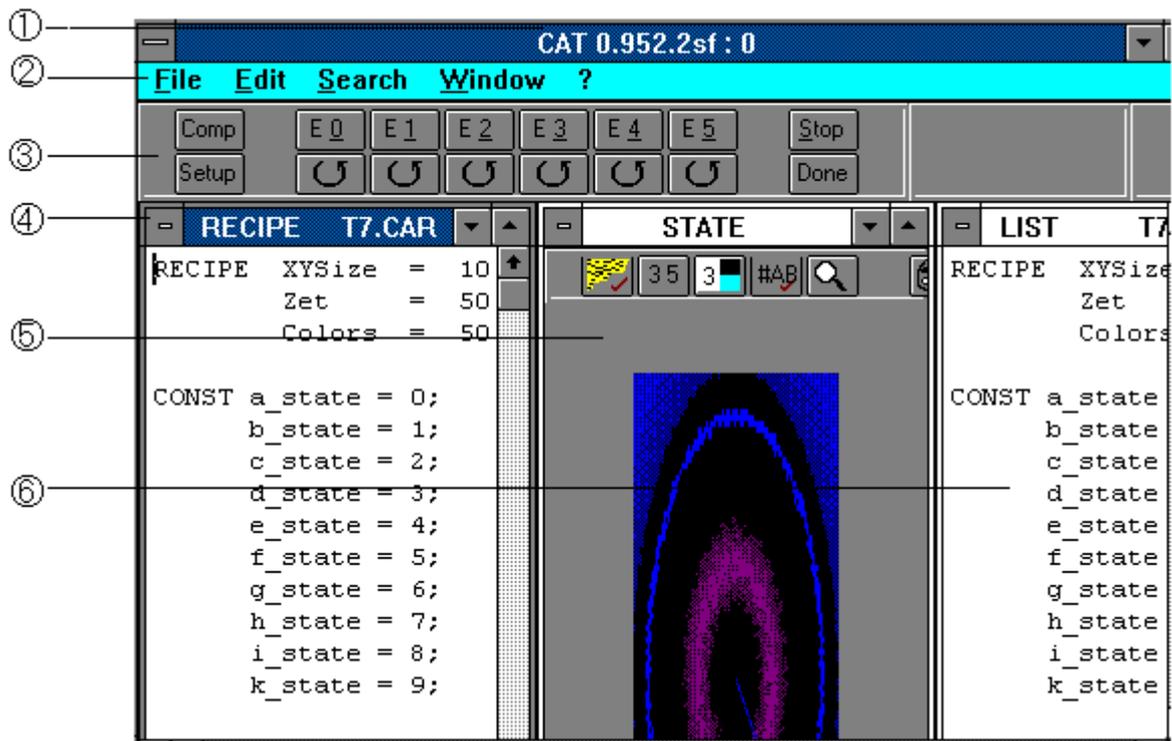


Figure 2.2 CAT welcome panel

The CAT welcome panel consists of

- ① a title bar with program name, version number and, separated by a colon, rounds of program execution,
- ② a menu and
- ③ a button bar,
- ④ a window RECIPE containing program text (may be empty),
- ⑤ a window STATE containing the matrix of cells,
- ⑥ a window LIST containing program output or error messages (may be empty).

2.2 Loading and Compiling a Program

Loading a Program

To load an existing program

* Select *Open* in menu *File*.

↳ CAT shows a file selection box to choose a program file.

The extension ".cat" is preset for models in CAT (a kind of project or workspace file). The list of directories shown in this example is probably different from your configuration.

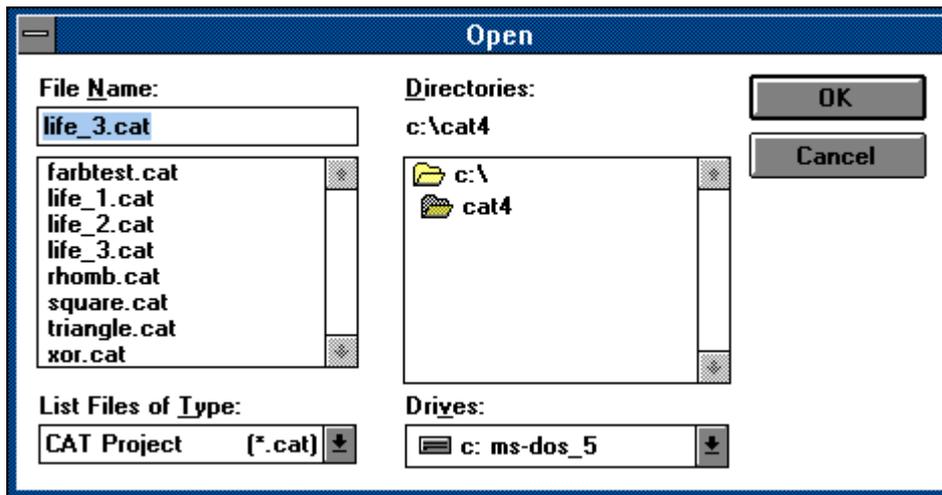


Figure 2.3 File selection box

* Select a file <model name>.cat (e.g. life_1.cat). (The dialog box is used as in Windows.)

↳ The chosen program file will pop up in the RECIPE window and the last state of the cell matrix will be displayed in the STATE window. Moreover, other attributes specific to this model will be restored (color palette, size and arrangement of CAT windows etc.).

Compiling a Program

* Click the button *Comp* in the left corner of the grey icon bar (see button 1 in figure 2.5).

↳ If compilation was successful nothing noticeable will happen.

Otherwise you will hear a short beep and this box will be shown:



Figure 2.4 CAT ERROR box

* Click the button *OK*.

↳ The box will disappear and you can now scroll inside the LIST window until you find the offending instructions. They will be flagged by the marks "(*** ERROR" and "***)" pointing out an error key number and the offending instruction. A sample: "(*** ERROR 140: Parser => unknown identifier *** Colordds ***)". The last word is the offending program instruction. Hints for error corrections can be found in Appendix F 'Compiler Message List'.

Compiling can also be achieved by clicking one of the buttons from "E0" to "E5". Before executing the code of the corresponding event (the button "E0" e.g. stands for event 0, see button 3 in figure 2.5), CAT checks whether the current CAT program has been already compiled. If not, CAT will compile it.

2.3 Executing and Controlling a Program

Some Constituents of a Cellular Automaton Tool Program (CARP Program)

To characterize the programming interface cellular automaton models are programmed by, we use the word **CARP**. CARP is an acronym for **Cellular Automaton Programming Language** and is a high-level language resembling PASCAL or MODULA.

Apart from definitions of variables or references, each CARP program has at least one element called "EVENT" starting with the keyword "EVENT" plus name plus semicolon and ending with another event or the keyword "END.". Events contain the program instructions, by which the state of each cell of the cell matrix is defined. Generally speaking, events are program parts existing independent of each other and therefore executable on demand. There is only one restriction: only one event can be executed at a time and can thus influence the state of the cell matrix; all other events are stopped in this case. Take a look at one of the sample programs to get an impression of this structure.

The event "SetUp" has a special function. Clicking the corresponding button (see button 2 in figure 2.5) executes instructions of the particular event "SetUp" that provides initialization of the cell matrix. It is implicitly executed if any other event is carried out. The "SetUp" event is not a necessary element of a CARP program. If it is missing default values for the color palette, the topology and other settings are used.

A description of the CARP programming interface is found on the pages 28 - 64.

Enlarging the Size of the STATE Window

Before working with the STATE window, bring it up and maximize it.

* Select *STATE* in the menu *Window*.

↙ The focus is set on the STATE window. (This might also be achieved by clicking in the STATE window if it is already visible.)

* Maximize the STATE window by clicking the maximize button in the top right corner (upward pointing triangle, see 6 in figure 2.5).

↙ The STATE window becomes maximized.

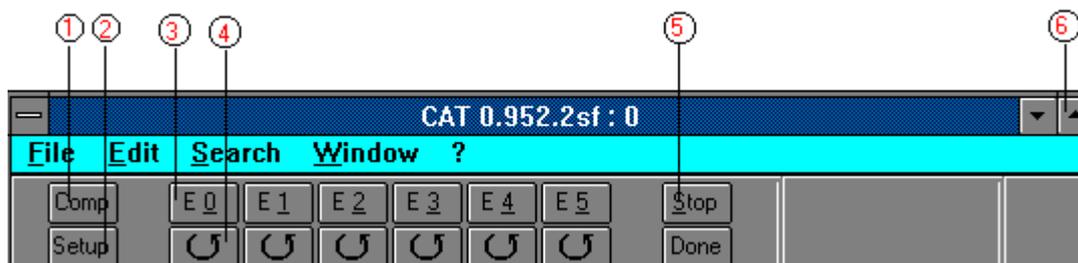


Figure 2.5 Event control buttons (Comp(ile) (1), SetUp (2), Event 0 Single Step (3), Event 0 Run (4), Stop (5), Maximize (6)

Starting Event 0

* Click the button E0 (button 3 in figure 2.5) in the grey button bar (depending on the size of your data display or on CAT's current window size, the bar may be stretched so that all buttons are aligned). Clicking E0 as opposed to clicking the run button (see button 4 in figure 2.5) means **stepwise** execution of event 1.

⚡ A pattern of pink and blue cells fills the STATE window representing a random dissemination of 'dead' and 'alive' cells. Event 0 should be used only once to initialize the cell matrix with a randomized dissemination of 'dead' and 'alive' cells. Depending on the properties of your hardware it may take a few seconds until the STATE window gets filled.

Moreover, you will notice four pieces of written information:

- ① After the colon, the title bar on the top of the CAT window contains the sum of rounds of all events whose code has been executed. Now it shows one round being completed.
- ② A label "Round" to the right of the ten STATE-window-related icons repeats the number of completed rounds.
- ③ A label on the right hand side of the icon bar contains the event number and the mode of execution. The keyword "Step" stands for stepwise execution, "Run", however, for execution in run mode.
- ④ A label "Time" contains the time passed by in hh:mm:ss,mm format, representing hours: minutes: seconds, milliseconds.

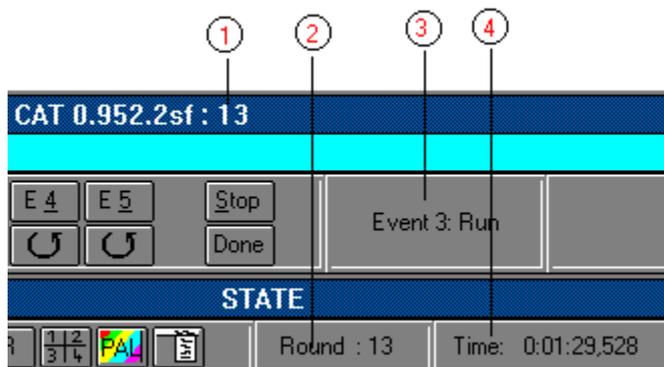


Figure 2.6 Event-related information

Starting Event 1 in Stepwise Mode

* Click the button E1 in the grey button bar.

⚡ After a short time the dispersion of 'dead' (blue) and 'alive' (red) cells changes according to the algorithm contained in the program text of event 1. For each cell the number of 'alive' neighbors is counted: is it 2 or 3, an 'alive' cell remains 'alive', otherwise it will 'die'. However, if a 'dead' cell is surrounded by just 3 'living' cells, it will become 'alive'. The equivalent labels are changed to 'Round: 2' and 'Event 2: Step'.

Note:

Each clicking of the buttons E0 to E5 will cause only **one** execution of the corresponding program text.

If you click a button (for example E4) that is not assigned to a corresponding event description in a CARP program, the cell matrix in the STATE window will remain unchanged.

Continuing Event 1 in Run Mode

* Now, click the run button below E1 (see button 4 in figure 2.5).

⚡ The program code of event 1 will now be executed continuously. The state of all cells will be computed and shown in the STATE window round by round. You will be informed about the actual round by the corresponding label 'Round'.

Stopping Program Execution

* Click the button "Stop" (see button 5 in figure 2.5).

⚡ The execution of any program code will be stopped. The STATE window will freeze the last state of the cell matrix. From here, you can restart any event in stepwise or run mode. You may also skip from one event to another by clicking different event buttons. If you click, for example, first the run button below E0 and then the run button below E1, the program code of event 0 will be executed until the next mouse click. When the current execution of event 0 is completed CAT will continue to execute the program code of event 1 without any interim step.

Hiding the Display of the Cell Matrix

Computing the new state of the STATE window for each generation of the cell matrix is quite CPU-time-consuming. If you want to elapse output of some generations, you can suppress the display of the cell matrix and speed up computation.

* Click the show / hide matrix button in the icon bar (see button 1 in figure 2.7).

⚡ Instead of the cell matrix, a grey background is displayed with a hint how to rebuild the cell matrix. Execution of the current event in run mode is continued as can be seen from the round counter in the title bar.



Figure 2.7 Graphic control buttons: Show/Hide matrix (1), Numeric state (2), Color mapping (3), Hexadecimal format (4), Magnifier (5), Color customizing (6), RePaint (7), State control (8), Palette customizing (9), Menu (10)

* Click the show / hide matrix button in the icon bar again.

⚡ The cell matrix is displayed in the STATE window again.

Note: Be patient if the code of any event is executed in run mode. It takes some time until the last execution of the event's program code is executed before CAT can switch to displayless mode. Another way to accelerate program execution is to show only every 10th or <n>th generation of the model by the corresponding use of the 'ShowPlane' instruction.

2.4 Editing a Program

Bring up the RECIPE window by means of the menu *Window* or click on this window if any part of it is visible.

* Click on any part of the text.

⚡ You will get a blinking text cursor and be able to write new text in insert mode.

To learn the different functions, change for example the value of the XYSIZE and return to the RECIPE window. If you chose a valid value (positive value < 130), program execution is continued

with new XSize and YSize values after pressing the corresponding event button. Otherwise, you will get an error message in the LIST window.

In general, most editing functions like *Cut*, *Copy* and *Paste* accessible by items of the *Edit* menu or by a local menu (see page 23) can be executed as in Windows 3.x, shortcuts may differ.

Note: Only the RECIPE window is ready for text editing, the LIST window is read-only.

Any modification of the text will be indicated by a "Modified" mark in the status bar on the bottom of the RECIPE window. To the left of this mark, the cursor position is shown with its current x- and y-coordinates.

Pattern matching functions (*Find*, *Replace*, *Next*) can be found in the menu *Search*.

2.5 Printing

CAT provides two forms of printed output:

- printing the contents of the RECIPE (CARP program) or LIST window by aid of a local menu
- printing the contents of the STATE window by means of the state control button (see button 1 in figure 2.8). The state of each cell may be alternatively printed with its decimal or its hexa-decimal value. There is no facility for printing the colored cell matrix.

Printing the Contents of the LIST or the RECIPE Window

* Bring up the text window you want to print (LIST or RECIPE window).

⚡ The corresponding window will have the focus indicated by a blue title bar.

* Click the right mouse button upon the corresponding window.

⚡ A local menu pops up.

* Select *Print* and click OK in the print dialog box.

⚡ The corresponding file is printed on your standard printer.

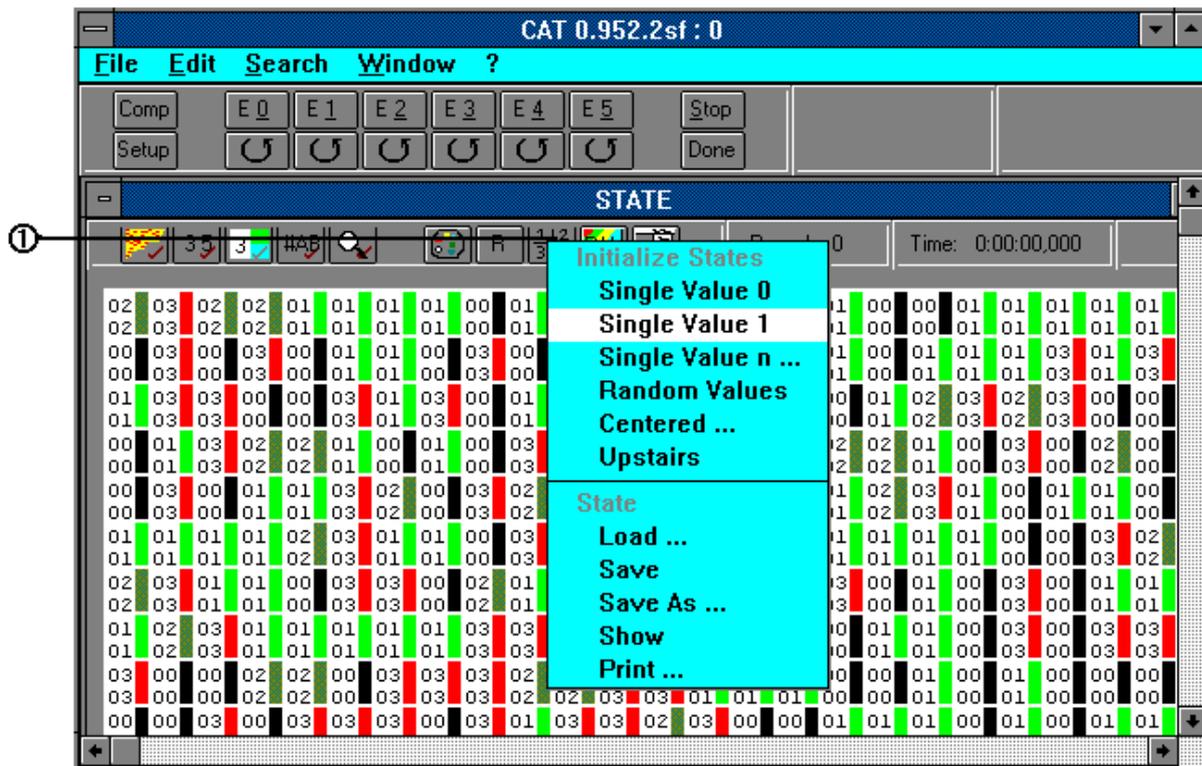


Figure 2.8 STATE window with numeric cell and color mapping representation

Printing the Contents of the STATE Window

- * Bring up the STATE window.
- ↳ The STATE window gets the focus indicated by a blue title bar.
- * Click the state control icon (see button 1 in figure 2.8).
- ↳ A menu pops up.
- * Select *Print* and click OK in the print dialog box.
- ↳ A text file representing the current state of the .CAS file (**CAT State** file) is printed on your standard printer. The cells are represented by their decimal value (default). Click the hexadecimal format button (button 4 in figure 2.7) before printing if you want to shift to hexadecimal output.

Note: There are restrictions on printing large STATE files although CAT automatically tries to choose an appropriate smaller font. Setting XYSIZE to values below 30 should work anyway.

2.6 Window Handling

Each CAT model of a cellular automaton contains at least three windows. This number might be increased by windows created by means of the *Show* or *Print* function. In this case, you may find it difficult to work your way through the CAT environment. Therefore, there are some functions in the menu *Window* to aid your orientation.

Bringing up all CAT Windows at a Time

- * Select *Tile* in menu *Window*.
- ↳ All windows currently contained in your CAT environment are shown at the expense of the space of each window. You can now select and then maximize the window you want to work with or resize one window at the expense of other windows.

Bringing up all CAT Windows in Overlapping Mode

- * Select *Cascade* in menu *Window*.
- ↳ All CAT windows are now shown overlapping each other, the first being presented in an acceptable size. If you want to put the focus on a different window, click on any of its parts to bring it up.

Bringing up a Specific CAT Window

- * Click the menu *Window*.
- ↳ All windows currently contained in your CAT environment are identified in the bottom part of this window by their names.
- * Drag the mouse pointer to the name of the window you want to bring up.
- ↳ The corresponding window comes up and may be maximized if necessary.

Note: The last method is advisable if you are already sure about the window you want to work with. To save window space, you also may iconify a window, e.g. the LIST window. CAT windows may not be closed by using the close button.

2.7 Changing Properties and Appearance of the Cell Matrix

Apart from defining size, state and colors of the cell matrix by certain instructions of the CARP program you can interactively change the appearance and some properties of the cell matrix. This is done by the group of buttons from the numeric cell state button to the palette customizing button (see buttons 2 to 9 in figure 2.7).

Showing the Cell State by Numbers

* Click the numeric cell state button (button 2 in figure 2.7)

↙ A small figure in the top left corner of each cell comes up, representing its current state. The figure is partially surrounded by the color that is associated with this value (see the cell matrix in figure 2.8 showing the numeric cell state and the color mapping).

Regaining the previous appearance can be achieved by clicking the same icon a second time.

Note: If your CAT model has a great number of cells, CAT may find no font being small enough and may thus refuse to show the cell state by figures (error message is "Window is too small to display the state"). A workaround is in some cases to reduce XYSize in the CARP program or to maximize the STATE window.

Showing the Color Mapping Entry Associated to each Cell

* Click the color mapping button (button 3 in figure 2.7).

↙ A small figure in the top left corner of each cell comes up, representing the according color palette entry. This entry is a pointer to a RGB triple defining how a certain state of a cell is represented by a specific color (see the cell matrix in figure 2.8 showing the numeric cell state and the color mapping).

Regaining the previous appearance can be achieved by clicking the same icon a second time.

Note: If your CAT model has a great number of cells CAT might refuse to show the color palette entry (error message is "Window is too small to display the color mapping").

A workaround is to reduce XYSize in the CARP program or to maximize the STATE window. If the cells are big enough it is possible to show the figure of the cell state and the color palette entry at the same time. The cell state is shown at the top of a cell, the color palette entry at the bottom. For more information on how to handle the color mapping and color palettes see page 50 and 51.

Switching from Hexadecimal to Decimal Figure Display

Depending on your preferences, it might be adequate to change the number format from hexadecimal to decimal.

* Enable figure display of the cell state or the color mapping. Then click the hexadecimal format button (button 4 in figure 2.7).

↙ The representation of the figures will change from decimal format to hexadecimal.

* To return to the previous state click the same button again.

↙ The representation of the figures will change again.

Removing the Frame around the Cell Matrix

Depending on the (optional) value of XYBound in the CARP program, your STATE window has a narrower or broader frame of darker cells. These cells are a kind of "virtual" continuation of the cell matrix beyond its edges according to the chosen topology. This border area may be displayed or hidden.

* Click the magnifier icon (button 5 in figure 2.7).

↙ The frame of the plane window caused by the XYBound value disappears and the cells become slightly bigger.

This, of course, works only if XYBound is defined.

Editing the Cell Plane

In general, it is much too time-consuming and complicated to give a whole cell matrix a certain shape cell by cell. This is easier to achieve by using instructions of your CARP program or by means of menu items of the state control button (see 8 in figure 2.7). Nevertheless, it is sometimes sufficient to initialize the whole cell matrix with an identical value for each cell by means of the state control button and to alter certain cell states 'manually'.

* Initialize the cell matrix by clicking the state control button and choosing *Single Value 0*.

↳ The cell matrix will adopt a uniform color.

* Click the left mouse button on a cell you want to change.

↳ The cell adopts the next value that is inside the range of possible different states (depending on the Zet value of your CARP program). The change of the cell state is mostly (depending on the chosen color palette) revealed by a different color.

* Click the right mouse button on the same cell.

↳ The cell will adopt its genuine state.

Note: Each click on the cell will make it adopt the next state of its defined <n> states. If e.g. only two states are defined the cell state will toggle between these two states.

Additionally, you may also edit the cells of the STATE window if the state of each cell is expressed by figures (number display mode). This is a good means to control the mouse clicks.

Clicks on the right mouse key decrement the state value, clicks on the left mouse key increment the state value.

Changing the Color Mapping for one Cell

There are about 230 colors defined by a triple of red, green and blue values that can be used to characterize a certain cell state. Changing the color mapping for a cell is done in two steps.

* Click on one cell of the kind whose color mapping you want to change. Then click the color customizing button (button 6 in figure 2.7).

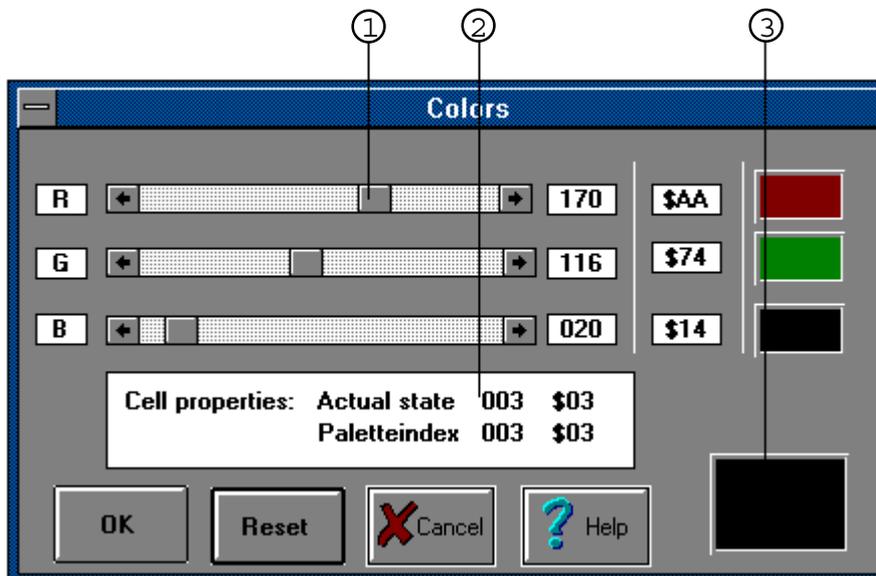


Figure 2.9 'Colors' selection box

↳ A box pops up with three slider scroll bars representing the colors red, green and blue (see 1 in figure 2.9) and, within a white area, statements about the current cell and its corresponding color palette index to be changed (see 2 in figure 2.9).

* Click on one of the three slider scroll bars and drag it to the left or to the right.

↳ The corresponding color control panel on the right side of the box changes its color very quickly, whereas the other two smaller control panels remain unchanged. The decimal and hexadecimal

value for the color being changed is also displayed. The summarizing color panel in the bottom right corner changes proportionally to the changes of its component colors (see 3 in figure 2.9).

* Change all three colors until you find a color value you wish to save. Click the *OK* button.

⚡ All cells of this kind are now assigned to the changed color mapping value and show their new color in the *STATE* window.

Note: Changes of a single color mapping or a whole color palette (the sum of all color mappings) are only valid within a certain model. The *CARP* language offers additional ways to define color palette settings (see pages 50 and 51). In this case, the color selection box may be of use to get the defining decimal or hexadecimal values of an adequate color in a handy way.

2.8 Using the Online Help System

CAT provides an online help system with hypertext features.

* Click the question mark in the menu bar (see 1 in figure 2.10).

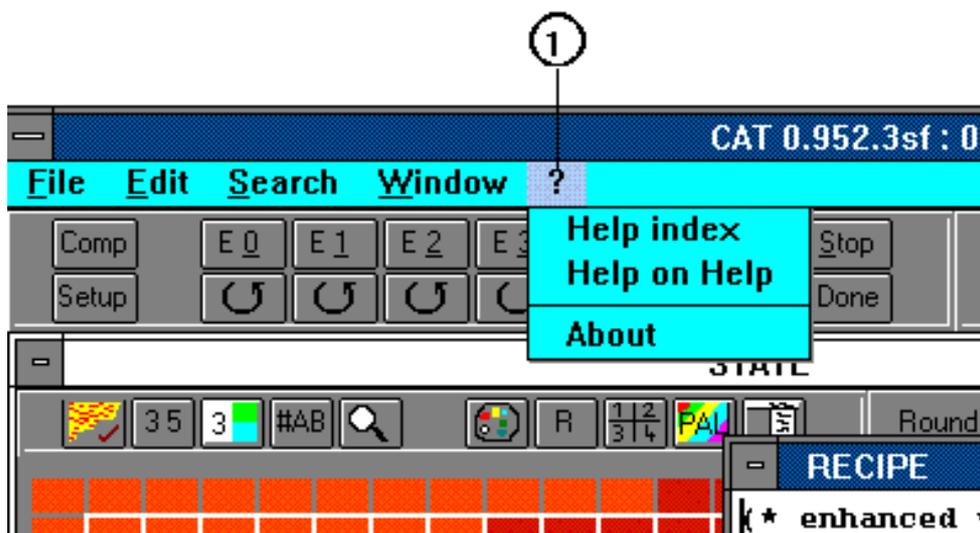


Figure 2.10 Access to the help system

⚡ A menu pops up containing the items *Help index*, *Help on Help*, *About* (see figure 2.10).

* Select *Help index*.

⚡ A window *CAT Helpsystem* comes up, containing an overview of all provided keywords. Any colored keyword can be activated by a mouse click and leads you to a short information text. To return to the point of the help system where you started from

* click the *Contents* or *Back* button of the Help window.

⚡ You find yourself back on the overview page of the help system. To leave the helpsystem,

* click either the close button in the top left corner of the "CAT Helpsystem" window or use the *Quit* command of the *File* menu.

⚡ The "CAT Helpsystem" window will close.

Note: To activate the online help system you can alternatively press the <F1> key. You will then come directly to the overview page of the help system or to the page which was displayed during the previous interrogation of the help system.

To get help information on the help system itself and its usage, click the question mark (see button 1 in figure 2.9) and select *Help on Help*.

If you want to print text from the online manual select *Print* from the *File* menu.

2.9 Leaving CAT

CAT can be left if no code of any event is executed. To leave

- * doubleclick the close button in the top left corner of the "CAT " window (see 1 in figure 3.1).
- ↳ The "CAT" window will close.

Note: Alternatively, you can select *Close* in the system menu that pops up if you click the close button (see 1 in figure 2.10) or you can select *Exit* in menu *File* (see item 3 in figure 2.10). If you want to quit CAT only temporarily, you can iconify the CAT program by selecting *Symbol* in the system menu. If the next application you want to use is rather memory-consuming, it is advisable to quit CAT completely.

3. Complete Overview of Windows, Menus, Buttons and Boxes

3.1 Windows

3.1.1. CAT Main Window

The CAT main window pops up after clicking the CAT icon. According to the size of your screen or the size of the main window CAT was last left in the buttons of the button panel (3) may appear aligned.

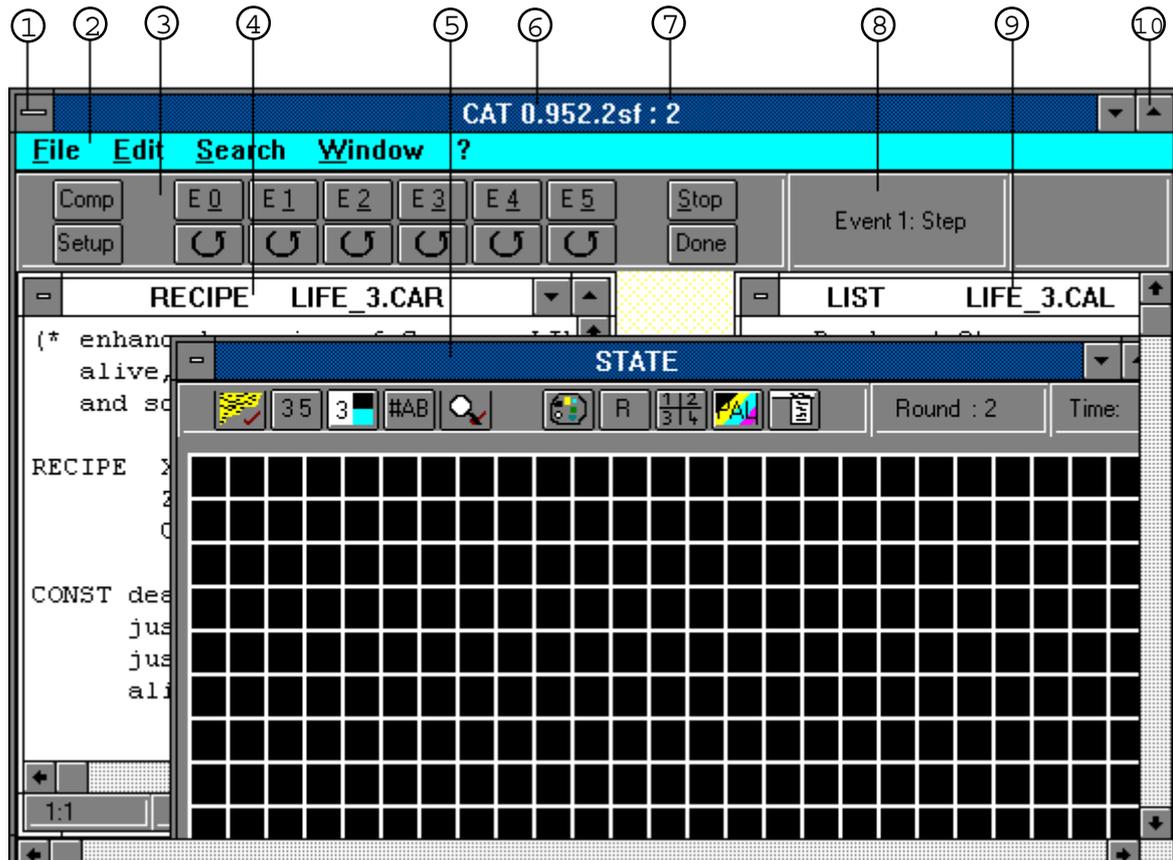


Figure 3.1 CAT Main window

| Name | Function |
|-------------------|--|
| ① Close button | Closes the CAT window and finishes the CAT application. |
| ② Menu panel | Provides access to different menus. |
| ③ Button panel | Provides different buttons for event control. |
| ④ RECIPE window | Contains your CARP program. |
| ⑤ STATE window | Contains the cell matrix. |
| ⑥ Program label | Program name and version number. |
| ⑦ Round counter | One of the two generation or round counters. |
| ⑧ Event label | Contains the current event name and runtime mode (Step / Run). |
| ⑨ LIST window | Contains compiler and runtime messages and output caused by your CARP program. |
| ⑩ Maximize button | Enlarges your CAT application to the whole screen. |

3.1.2. STATE Window

The STATE window displays the cell matrix of your current cellular automaton. The cell matrix may be customized in many ways (see pages 11 - 12 and 24 - 27).

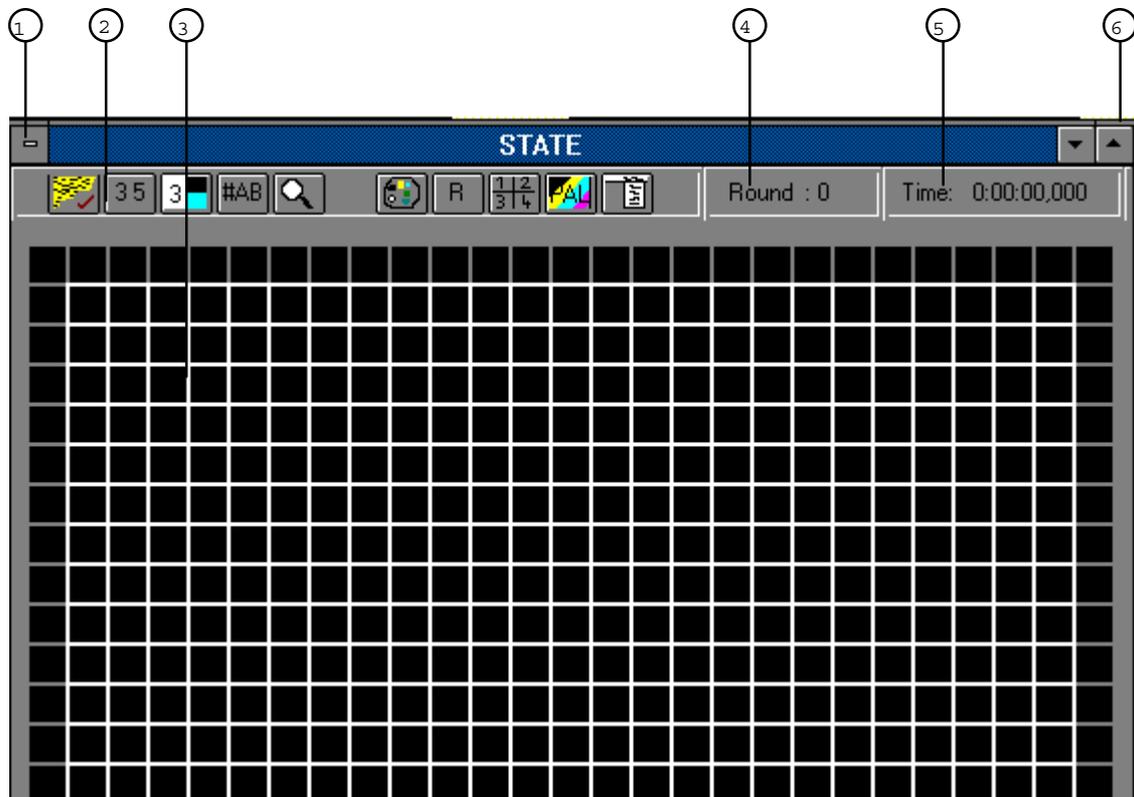


Figure 3.2 STATE window

| Name | Function |
|-------------------|---|
| ① Close button | Disabled for this window. |
| ② Button panel | Allows customization of several properties (color mapping, color palette, figure format etc. Further description on p. 24 and p.11 - 14). |
| ③ Cell matrix | Area in which the cells are displayed with their current color and/or figure values. |
| ④ Round counter | Second round or generation counter. |
| ⑤ Time label | Shows the time passed by in hh:mm:ss,msms (hours : minutes : seconds, milliseconds). |
| ⑥ Maximize button | Enlarges the STATE window to all available CAT space. |

3.1.3. RECIPE Window

The RECIPE window is the place where you can enter your CARP program (see p. 28 - 62) by which your cellular automaton is programmed.

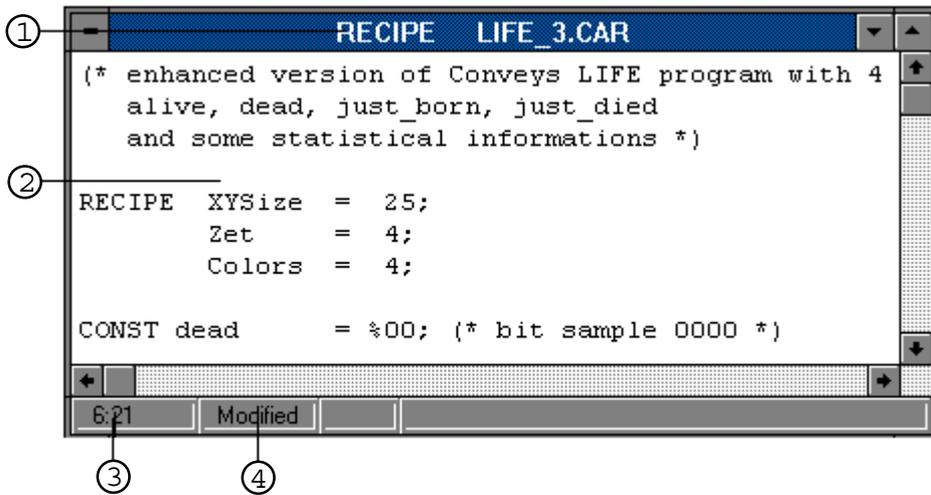


Figure 3.3 RECIPE window

| Name | Function |
|-------------------------|--|
| ① Window and file label | Indicates the file name of your CARP program (extension .CAR). |
| ② CARP program | Editable CARP program text. |
| ③ Cursor position | Shows the current cursor position with its equivalent x- and y-values. |
| ④ 'Modified' mark | Appears as soon as anything in this window has been changed. |

Editing work is supported by functions of the *Edit* (see p.20), *Search* (see p.21) and the two local menus (see p.23). The latter are invoked either by clicking the right mouse key or by clicking the right mouse key while holding the shift key pressed. Local menu functions also include printing and font customization.

3.1.4. LIST Window

The non-editable LIST window contains runtime, compiler error messages and messages created by your CARP program.

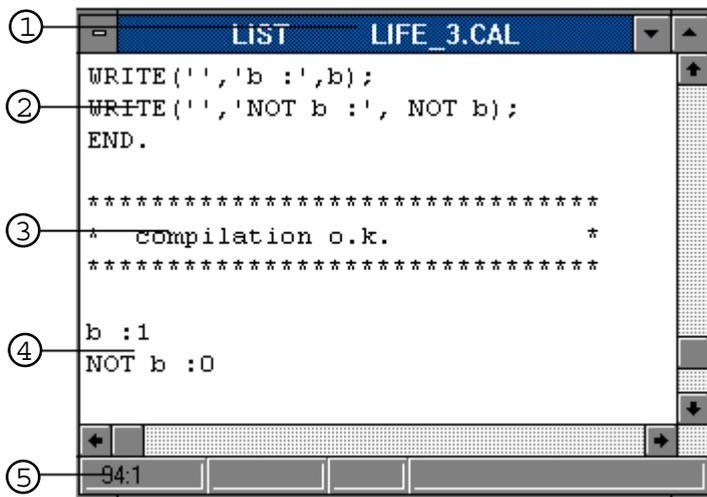


Figure 3.4 LIST window

| Name | Function |
|----------------------|---|
| ① File name | Indicates the name of the corresponding file (extension .CAL). |
| ② CARP program | Repeated CARP program (not editable). |
| ③ Compilation result | Indicates whether compilation has been successful or not. You will find the offending code and a short error classification framed in '****' marks where it occurs in the program text. |
| ④ Text output | Text effected by corresponding CARP instructions. |
| ⑤ Cursor position | Indicates the current cursor position with its x- and y-values. |

3.1.5. Plane Window (Starring .CAS File)

The Plane window is a text window where all cells of the cell matrix appear with their current state in figure format (decimal or hexadecimal). The Plane window pops up after pressing the state control button and selecting 'State' | 'Show'.

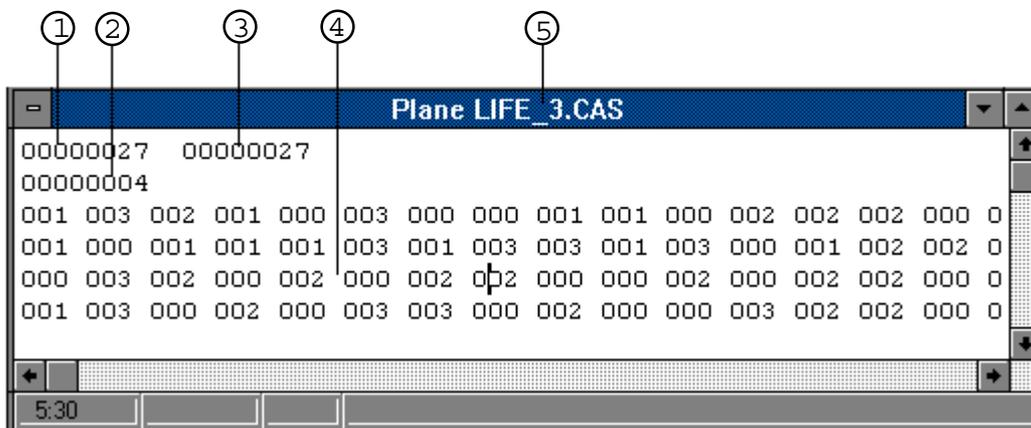


Figure 3.5 Plane window

| Name | Function |
|-----------------------------|---|
| ① X-value | Gives the current X-value, eventually increased by the values for the border area (see XYBound settings). |
| ② Zet-value | Number of states of a cell defined by the Zet setting. |
| ③ Y-value | Gives the current Y-value, eventually increased by the values for the border area (see XYBound settings). |
| ④ cell matrix figure format | Current state of each cell in figure format. |
| ⑤ Window and file label | Indicates the name of the corresponding file (extension .CAS) |

3.1.6. Palette Window (Starring .CAP File)

The Palette window displays the current values of the color palette (see the glossary at the end of this manual). The Plane window pops up after pressing the palette customizing button and selecting 'Palette' | 'Show'.

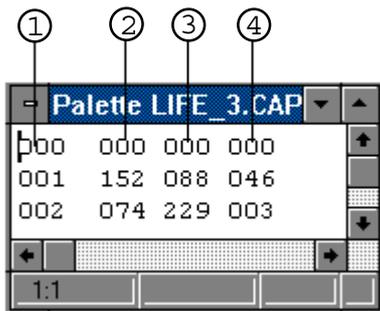
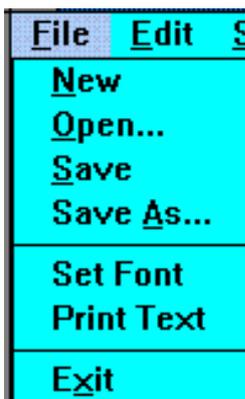


Figure 3.6 Palette window

| Name | Function |
|------------------|--|
| ① Mapping number | Identifying number by which a state is assigned to a certain color. |
| ② Red-value | First part of the RGB triple defining the 'Red' share of the color. |
| ③ Green-value | Second part of the RGB triple defining the 'Green' share of the color. |
| ④ Blue-value | Third part of the RGB triple defining the 'Blue' share of the color. |

3.2 Menues

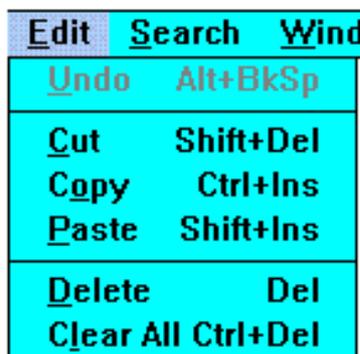
3.2.1. 'File'



| Name | Function |
|-------------------|--|
| <i>New</i> | Creates a new empty cellular automaton environment. |
| <i>Open...</i> | Opens an existing cellular automaton environment (.CAT file) i.e. a kind of workspace or project file. |
| <i>Save</i> | Saves current cellular automaton under a preset name. |
| <i>Save As..</i> | Saves current cellular automaton under a new name. |
| <i>Set Font</i> | Customizes font for all text windows (global). |
| <i>Print Text</i> | Prints the current text window. |
| <i>Exit</i> | Leaves CAT. |

Figure 3.7 Menu *File*

3.2.2. 'Edit'



| Name | Function |
|------------------|---|
| <i>Undo</i> | Makes the last editing action undone. |
| <i>Cut</i> | Deletes a marked text portion and keeps it in the temporary text buffer. |
| <i>Copy</i> | Copies a marked text portion and keeps it in the temporary text buffer. Text remains unchanged. |
| <i>Paste</i> | Inserts text copied to the temporary text buffer at the current cursor position. |
| <i>Delete</i> | Deletes a marked text portion. If no text is marked, the character to the left of the current cursor position is deleted. |
| <i>Clear All</i> | Deletes all the text. Be careful! This action cannot be made undone by the <i>undo</i> function. |

Figure 3.8 Menu *Edit*

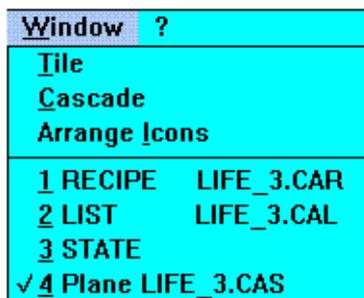
Most of the above items may only be selected if a text portion has been marked or if the temporary text buffer has been filled.

3.2.3. 'Search'

Figure 3.9 Menu *Search*

| Name | Function |
|-------------------|--|
| <i>Find...</i> | Finds a search pattern in the current text window and highlights it if found. |
| <i>Replace...</i> | Finds a search pattern in the current text window and replaces it by another text. |
| <i>Next</i> | Finds next position of a search pattern in the current text window. Precondition: a search pattern has been already defined. |

3.2.4. 'Window'

Figure 3.10 Menu *Window*

| Name | Function |
|---------------------------|---|
| <i>Tile</i> | Shares the available space equally for all CAT windows. |
| <i>Cascade</i> | Makes a stack of all CAT windows such the window labels are all visible. The last window is completely visible. |
| <i>Arr. Icons</i> | If CAT windows have been iconified, all icons are put in order on the bottom of the screen. |
| <i>RECIPE</i> | Brings up the RECIPE window. |
| <i>LIST</i> | Brings up the LIST window. |
| <i>STATE</i> | Brings up the STATE window. |
| <i><wind. name></i> | Brings up another CAT window if present (Plane, Palette). |

3.2.5. '?' (Help)

Figure 3.11 Menu *Help*

| Name | Function |
|---------------------|---|
| <i>Help index</i> | Gives an overview on all available help items. |
| <i>Help on Help</i> | Gives information on how to use the online help system. |
| <i>About</i> | Gives information about CAT, especially about the people who created CAT. |

Alternatively, the online help system may be invoked by pressing the <F1> key.

3.2.6. Menu Accessible via State Control Button

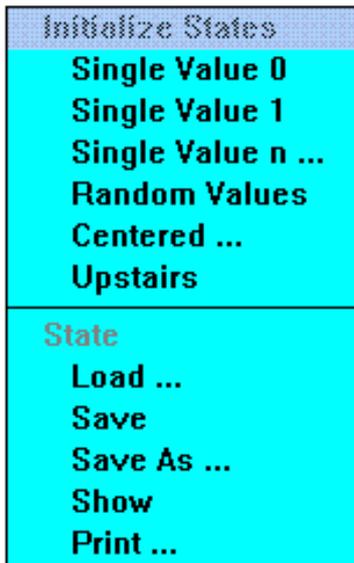


Figure 3.12 Menu *State Control Button*

| Name | Function |
|---------------------|--|
| <i>S. Value 0</i> | Initializes each cell of the cell matrix with value 0. |
| <i>S. Value 1</i> | Initializes each cell of the cell matrix with value 1. |
| <i>S. Value n..</i> | Initializes each cell of the cell matrix with value n. A dialog box prompts you for the value needed (see p.26). |
| <i>Rand. Value</i> | Initializes the cell matrix with random values inside the range of defined states. |
| <i>Centered</i> | Initializes the cell matrix with one extreme value in the center and the other extreme value in the corners. You are prompted for the settings required (see p.26). |
| <i>Upstairs</i> | Initializes the cell matrix with stair-like patterns. The width of the steps depends on the XYSIZE and ZET values. |
| <i>Load...</i> | Loads an existing cell matrix saved as .CAS file (state). |
| <i>Save</i> | Saves the current state of the cell matrix under the preset name (.CAS file). |
| <i>Save As..</i> | Saves the current state of the cell matrix under a new name (.CAS file). |
| <i>Show</i> | Shows the current cell matrix represented by the corresponding .CAS file in the Plane window (see p. 19). |
| <i>Print...</i> | Prints the current cell matrix represented by the equivalent .CAS file on your standard printer. |

3.2.7. Menu Accessible via Palette Control Button



Figure 3.13 Menu *Palette Control Button*

| Name | Function |
|--------------------|---|
| <i>B/W blend</i> | Defines a color palette with a blend of colors from black to white. Granularity depends on the ZET value. |
| <i>Color blend</i> | Defines a color palette with a blend of all defined colors. The granularity of the blend depends on the ZET value. |
| <i>By param.</i> | Defines a color palette with different initial and terminal colors and differences on the distribution of colors. You are prompted for all settings required (see p. 27). |
| <i>Load</i> | Loads an existing color palette (.CAP file) and overwrites the existing one. |
| <i>Save</i> | Saves the current color palette under the preset name (.CAP file). |
| <i>Save As..</i> | Saves the current color palette under a new name (CAP file). |
| <i>Show</i> | Shows the current color palette as a text window (.CAP file) in the Palette window (see p. 19 - 20). |
| <i>Print</i> | Prints the current color palette (.CAP file). |

3.2.8. Menu Accessible via Menu Button



Figure 3.14 Menu Menu Button

| Name | Function |
|---------------------|--|
| <i>Show Color</i> | Shows the states of all cells of the matrix in color representation. |
| <i>Show State</i> | Shows the states of all cells of the matrix in figure format. |
| <i>Col. Mapping</i> | Shows the color mapping of each cell of the matrix in figure format. |
| <i>Hex Display</i> | Toggles between decimal or hexadecimal figure format inside the STATE window. |
| <i>Zoom</i> | Displays or suppresses the border area around the cell matrix (if defined by a corresponding XYBound setting). |
| <i>Refresh</i> | Renews the display of the cell matrix. |

The items accessible via the menu button offer the same functions as some of the buttons of the STATE window. Usage of one of these two kinds of triggering functions is a question of preference. As the corresponding buttons the items of this menu have toggle logic: selecting them twice will lead to the state where you started from.

3.2.9. Local Menu Accessible via Right Mouse Key



Figure 3.15 Menu Local Menu (I)

| Name | Function |
|--------------------|---|
| <i>New</i> | Deletes the contents of the current text window and creates a new empty text window. |
| <i>Overload</i> | Loads an existing file substituting the current text window. Be careful! File extensions are not checked. |
| <i>Save</i> | Saves the current text window under the preset name. |
| <i>Save As..</i> | Saves the current text window under a new name. |
| <i>Insert File</i> | Loads and inserts a text file at the current cursor position. |
| <i>Set Font</i> | Customizes font for the current text windows (only local). |
| <i>Print</i> | Prints the current text window. |
| <i>Hide Window</i> | Hides the current window and shows the window on top of the window stack. |

In contrast to the functions of the 'File' menu (see p. 20), the effect of the above items is only **local**.

3.2.10. Local Menu Accessible via Shift / Right Mouse Key



Figure 3.16 Menu Local Menu (II)

| Name | Function |
|---------------|---|
| <i>Undo</i> | Makes the last editing action undone. |
| <i>Cut</i> | Deletes a marked text portion and keeps it in the temporary text buffer. |
| <i>Copy</i> | Copies a marked text portion and keeps it in the temporary text buffer. Text remains unchanged. |
| <i>Paste.</i> | Inserts text copied to the temporary text buffer at the current cursor position. |
| <i>Delete</i> | Deletes a marked text portion. |

Most of the above items may only be selected if either a text portion has been marked or the temporary text buffer has been filled.

3.3 Buttons and Labels

3.3.1. Buttons and Labels Contained in the Main Window



Figure 3.17 Buttons and labels of the main window

| Name | Function |
|------------------|--|
| ① Comp button | Compiles the CARP program you have written in the RECIPE window. No reaction means a successful compilation. Program errors, however, are reported in the LIST window. |
| ② Setup button | Carries out the code of the special event SetUp. This event should contain definitions for a color palette, for a certain topology etc. |
| ③ Event 0 / Step | Carries out the code of the event 0 in single step mode. |
| ④ Event 0 / Run | Carries out the code of the event 0 in run (endless) mode. |
| ⑤ Stop button | Stops the execution of any event. |
| ⑥ Done button | Sorry! Not yet implemented. |
| ⑦ Event label | Appears as soon as the first event has been triggered and indicates the current event and the mode it is running in. |

Event buttons from E 0 to E 5 and the event button SetUp are assigned to the corresponding event definitions of your CARP program.

You may also trigger an event in single step mode by pressing <ALT> and the number of the corresponding event. <ALT> 3 is, for example, equivalent to clicking the E 3 / Step button.

Clicking any event button (SetUp, E 0 to E 5 in run or step mode) effects an implicit compilation if the CARP program has been modified since the last compilation.

3.3.2. Buttons and Labels Contained in the STATE Window

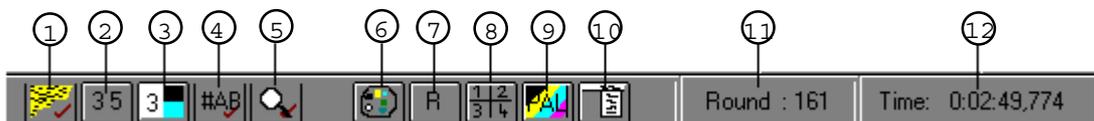


Figure 3.18 Buttons and labels of the STATE window

| Name | Function |
|--------------------------|---|
| ① Show / hide matrix b. | Switch to show or suppress display of the cell states by colors. |
| ② Numeric state b. | Switch to display the state of each cell by figures. |
| ③ Color mapping b. | Displays the color mapping entry that is assigned to a single cell. |
| ④ Hexadecimal format b. | Switch to show the numbers in hexadecimal or decimal format. |
| ⑤ Magnifier b. | Switch to show or suppress the display of the border area. |
| ⑥ Color customizing b. | Changes the color that is assigned to a color mapping entry. |
| ⑦ Repaint button | Paints the STATE window with its cell matrix again. |
| ⑧ State control b. | Opens the menu <i>State</i> (initialization and other functions, see p.22). |
| ⑨ Palette customizing b. | Opens the menu <i>Palette</i> (changing and customizing a whole palette, see p.22). |

- Ⓣ Menu button Offers the same functions as menu items that are provided by some of the above buttons, see p.23).
- (11) Round counter Shows the number of executed rounds.
- (12) Time label Shows the time passed by in hours : minutes : seconds , milliseconds format (hh:mm:ss,msms).

Many of these buttons have toggle switch logic. The actual state of a toggle switch is indicated by a little red hook.
 Many of the functions executed by the above buttons can also be effected by corresponding CARP procedures.

3.4 Dialog Boxes

3.4.1. 'Colors' Dialog Box

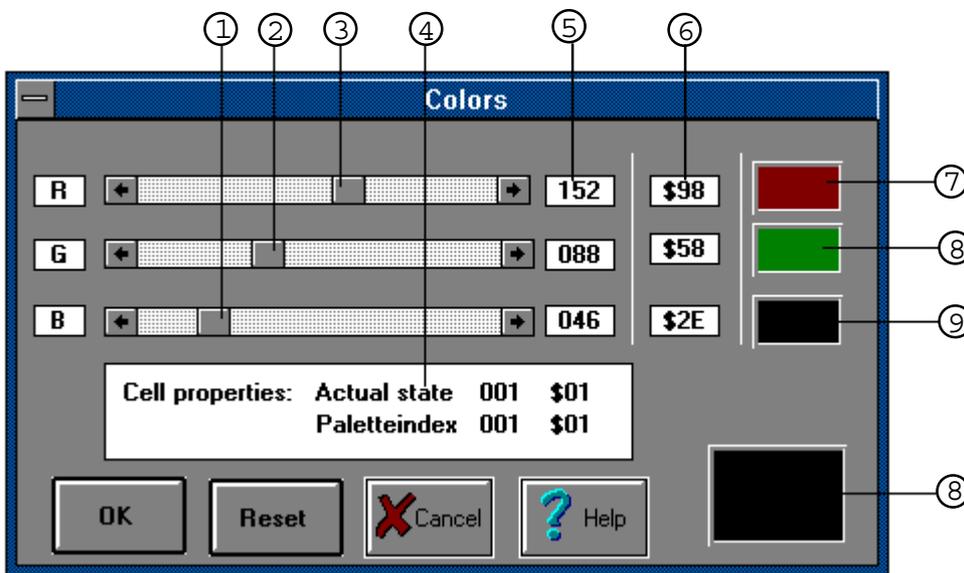


Figure 3.19 'Colors' dialog box

The 'Colors' dialog box pops up after clicking the color mapping button (see 6 in figure 3.18) or after clicking shift and right mouse button having one cell selected in the STATE window.

| Name | Function |
|--------------------------|--|
| ① Slider for color blue. | Defines the blue portion. Result is visible at ⑩. |
| ② Slider for color green | Defines the green portion. Result is visible at ⑧. |
| ③ Slider for color red | Defines the red portion. Result is visible at ⑦. |
| ④ Cell properties | Gives state and assigned color mapping values of current cell. |
| ⑤ Decimal color value | Shows actual value for selected color in decimal format. |
| ⑥ Hexadec. color value | Shows actual value for selected color in hexadecimal format. |
| ⑦ Control panel 'red' | Shows the current color setting for color element 'red'. |
| ⑧ Control panel 'green' | Shows the current color setting for color element 'green'. |
| ⑨ Control panel 'blue' | Shows the current color setting for color element 'blue'. |
| ⑩ Mixed color panel | Shows the color created by mixing the three above elements. |

3.4.2. 'Assign Value' Dialog Box

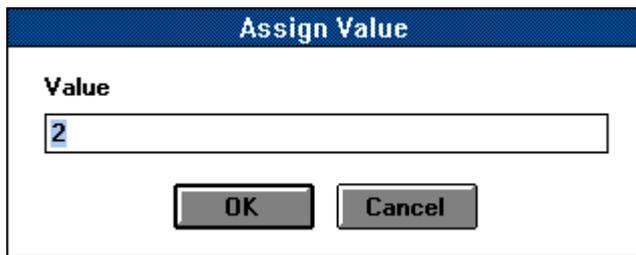


Figure 3.20 'Assign Value' dialog box

The 'Assign Value' dialog box pops up after clicking the state control button (see 8 in figure 3.18) and selecting *Initialize States | Single Value n....* You are prompted for the value you want to assign to each cell of the cell matrix. To take effect this value must be inside the range of defined states (cp. Zet setting).

3.4.3. 'Initialize States' Dialog Box

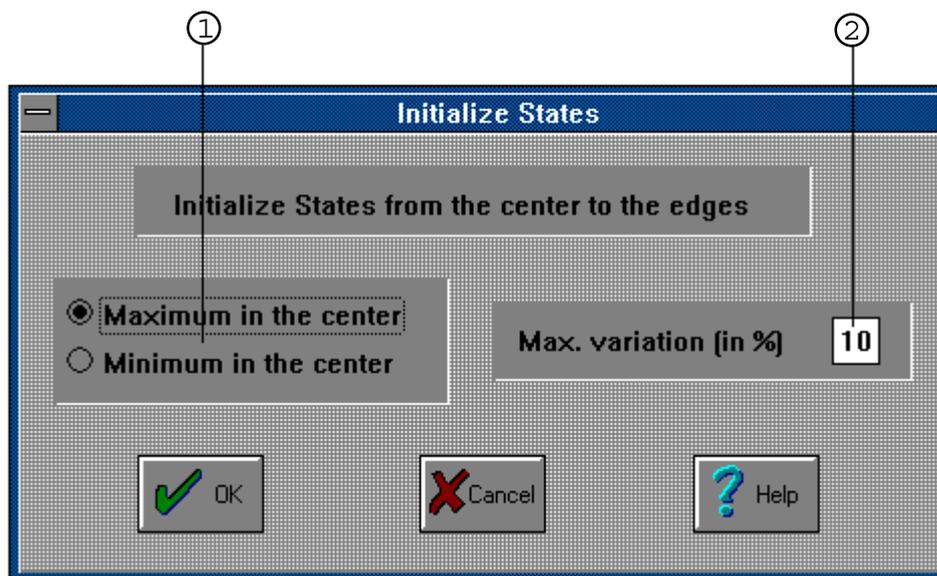


Figure 3.21 'Initialize States' dialog box

The 'Initialize States' dialog box pops up after clicking the state control button (see 8 in figure 3.18) and selecting *Initialize States | Centered...*

| Name | Function |
|--------------------------|---|
| ① Center options buttons | Defines distribution of values with maximum in the center or at the edges. |
| ② Variation text box | Allowed variation of color distribution (in %). (May be neglected in most cases). |

3.4.4. 'Initialize Palette' Dialog Box

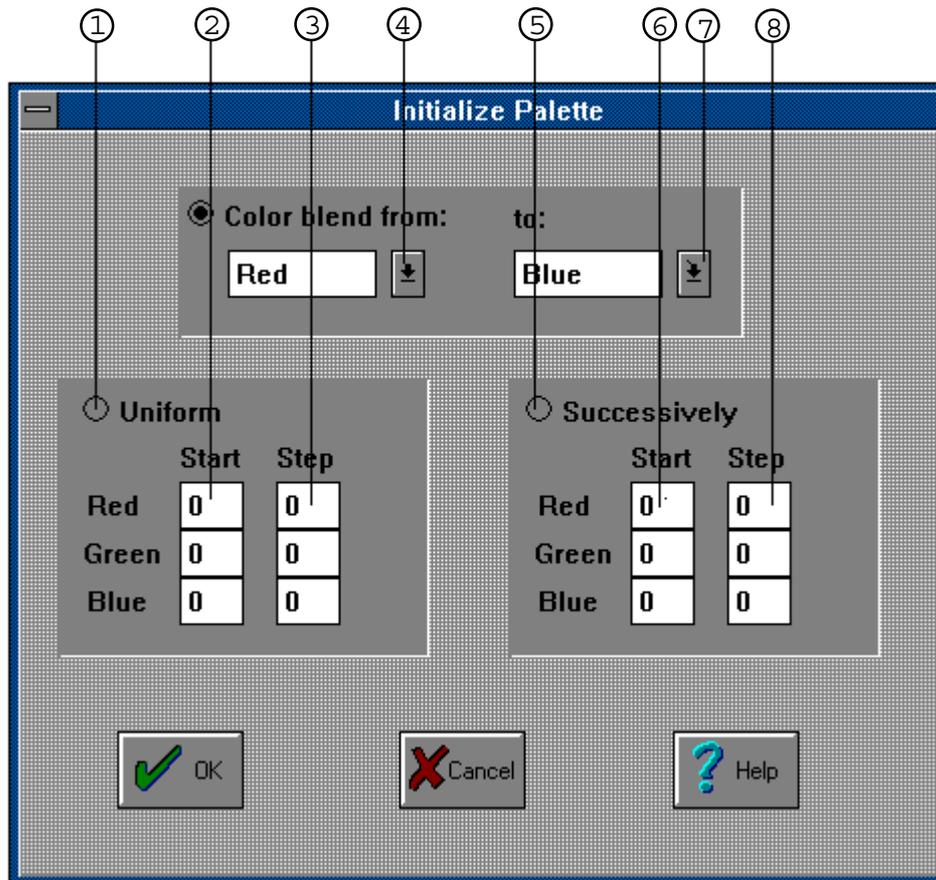


Figure 3.22 'Initialize Palette' dialog box

The 'Initialize Palette' dialog box pops up after clicking the palette customizing button (see 9 in figure 3.18) and selecting *Initializing Palette | By parameters...*

| Name | Function |
|-----------------------------|--|
| ① 'Uniform' option b. | All three elements of the RGB triple are increased uniformly by their corresponding 'Step' value from the first to the last defined color. |
| ② 'Start' text box (u.) | Defines the start value for each color (0 - 230). |
| ③ 'Step' text box (u.) | Defines the interval by which the color value increases. |
| ④ 'blend from' drop-down b. | Defines the color where the color blend begins at. |
| ⑤ 'Successively' option b. | At first, only the 'red' value is increased by its 'Step' value until the upper limit (255) has been reached. Then the 'green' value is increased while the 'red' value remains constant and so on. Choose high 'Step' values to use this option properly. |
| ⑥ 'Start' text box (s.) | Defines the start value for each color (0 - 230) |
| ⑦ 'blend to' drop-down b. | Defines the color where the color blend ends. |
| ⑧ 'Step' text box (s.) | Defines the interval by which the color value increases. |

A good means to observe the changes of the color palette caused by the 'Uniform' or 'Successively' option is to display the color palette by means of the *Palette | Show* function (see p. 22).

4. CARP Instructions Reference Guide

%

Syntax

```
% (operand)
```

Remarks

The % operator precedes a sequence of ones (1) and zeroes (0) that are interpreted as a bit sequence. Therefore, the operand may only consist of a sequence of 1s and 0s.

By this, you can do bit manipulation in a more explicit form compared to treating integers as bit values.

Example

```
RECIPE  XYSize  = 60;
        Zet     = 4;
        Colors  = 4;

CONST  dead      = %00; (* bit sequence 00 *)
       just_died = %01; (* bit sequence 01 *)
       just_born = %10; (* bit sequence 10 *)
       alive     = %11; (* bit sequence 11 *)
       (*
       (*           A *)
       (*           | *)
       (*         "alive bit" *)

REF    east      [1,0];
       west      [-1,0];
       north     [0,-1];
       south     [0,1];
       north_ea  [1,-1];
       north_we  [-1,-1];
       south_ea  [1,1];
       south_we  [-1,1];

PROC  add_second_bit; (* procedure evaluates second bit of*)
      (* neighbors that indicates alive *)
      (* state and returns sum of found bits*)

BEGIN
RETURN ((east XOR %01) SHR 1) + ((west XOR %01) SHR 1)
      + ((north XOR %01) SHR 1) + ((south XOR %01) SHR 1)
      + ((north_ea XOR %01) SHR 1) + ((north_we XOR %01) SHR 1)
      + ((south_ea XOR %01) SHR 1) + ((south_we XOR %01) SHR 1)
END add_second_bit;
```

*

Syntax $(\text{operand}) * (\text{operand})$ **Remarks**

The * operator multiplies two operands. As operands are allowed: integer constants, variables or procedures that return an integer.

Example

```
VAR a, b;  
  
EVENT 1;  
b:= 17;  
a := b * 4;
```

+

Syntax $(\text{operand}) + (\text{operand})$ **Remarks**

The + operator adds two operands. As operands are allowed: integer constants, variables or procedures that return an integer.

Example

```
VAR temp;  
CONST c = 2345;  
  
EVENT 1;  
temp:= c + 37;
```

-

Syntax $(\text{operand}) - (\text{operand})$ **Remarks**

The - operator subtracts two operands. As operands are allowed: integer constants, variables or procedures that return an integer.

Example

```
VAR a, b;  
  
EVENT 1;  
b:= 17;  
a := b - 4;
```

:=

Syntax

```
(VAR) identifier | Self := expression;
```

Remarks

The assignment procedure '=' attributes the value of an expression right to the assignment procedure to any variable or the cell Self standing left to this procedure.

Example

```
EVENT E1;
PARALLEL DO
  Self := OddCell;
OD;
ShowPlane;
```

<

Syntax

```
(operand) < (operand)
```

Remarks

The < operator compares two operands in respect to size. If the first operand is smaller than the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
WHILE b < i DO
  a := b + 48
OD;
```

<=

Syntax

```
(operand) <= (operand)
```

Remarks

The <= operator compares two operands in respect to size. If the first operand is smaller or equal than the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
IF b <= limit
THEN b := Any8Sum (a, b, c, d, e, f, g, h);
FI;
```

**Syntax**

(operand) <> (operand)

Remarks

The <> operator compares two operands in respect to unequality. If the first operand is unequal to the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
IF a <> limit
THEN a := a +1
ELSE a := limit
FI;
```

**Syntax**

(operand) = (operand)

Remarks

The = operator compares two operands in respect to equality. If the first operand is equal to the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
WHILE a = b DO
  Self := add_four_positions;
OD;
```

**Syntax**

(operand) > (operand)

Remarks

The > operator compares two operands in respect to size. If the first operand is greater than the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
IF b > a
THEN b := a;
FI;
```

>=

Syntax

```
(operand) >= (operand)
```

Remarks

The >= operator compares two operands in respect to size. If the first operand is greater or equal compared to the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
WHILE a >= 0 DO
  a := a -1;
OD;
```

A

AND

Syntax

```
(operand) AND (operand)
```

Remarks

The AND operator connects two operands and returns true, if both operands are true. All other cases return false.

If the cells of your CAT model may only have the state 0 or 1, you may treat cells with the AND operator, too.

Example

```
a := 8;
IF (a > limit) AND (b = 9)
  THEN a := Self FI;
```

Any8Sum

Syntax

```
Any8Sum ( n1, n2, n3, n4, n5, n6, n7, n8 );
```

Remarks

Any8Sum adds the state values of neighbors, variables or constants that follow as 8 parameters.

Example

```
REF knight_t_l [-1,-2]; (* possible jumps of *)
knight_t_r [1,-2]; (* knights *)
knight_b_l [-1,2];
knight_b_r [1,2];
knight_mt_l [-2,-1];
knight_mt_r [2,-1];
knight_mb_l [-2,1];
```

```

        knight_mb_r    [2,1];

EVENT E1;
PARALLEL DO
    Self := Any8Sum
        (knight_t_l, knight_t_r, knight_b_l, knight_b_r,
         knight_mt_l, knight_mt_r, knight_mb_l, knight_mb_r);
OD;
ShowPlane;

```

B

BarrelForm

Syntax

```
BarrelForm;
```

Remarks

The topology BarrelForm forms a virtually barrelshaped matrix, i.e. the right and left edges of the cell matrix are mutually copied to the opposite edge.

```

c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
Topology BarrelForm (c = copied cell)

```

Example

```

EVENT SetUp;
BarrelForm;
PlClipActive;
ShowPlane;

```

Beep

Syntax

```
Beep ( n );
```

Remarks

Returns n beeps.

This procedure is useful if you want to mark a crucial state of your cellular automaton model by an acoustic signal.

Example

```

CONST max_value = 5478;
VAR x;
EVENT E0;
...
IF x >= max_value THEN Beep(1) FI;

```

BEGIN ... END

Syntax

```
BEGIN
  statement;
  [statement;]
  ...
  [statement;]
END;
```

Remarks

Instructions bracketed by the keywords BEGIN and END may be used as an additional means for structuring a CARP program. Usage is optional.

Example

```
(* Compound statement used within an "IF" statement *)
IF First < Last THEN
  BEGIN
    Temp := First;
    First := Last;
    Last := Temp;
  END;
FI;
```

Brake

Syntax

```
Brake;
```

Remarks

Stops the current event.

Example

```
FOR x := 1 TO x < 10 BY 2 DO
  WRITE (x);
  IF (x + 3) = 5
    THEN Brake
    ELSE y := x + 1
  FI;
OD x;
```

C

Comments in a CARP program

Syntax

```
(* string *)
```

Remarks

To keep your program self-explanatory even for later times, use comments in your CARP program. Use pairs of "(" and ")" respectively to indicate start or end of a comment. Comments may comprise several lines.

Example

```
REF left[-1,0]; up[0,-1]; right[0,1];
(* x counts negative for referenced cells
   on the top of cell Self *)
```

Colors

Syntax

```
Colors = n;
```

Remarks

Colors defines the number of available colors. The color actually assigned to a certain state may be either interactively set by means of the color customizing button or by means of the RGBBrush procedure.

Example

```
RECIPE XYSize = 140;
       XYBound = 3 ;
       Zet = 20;
       Colors = 20;
```

CONST

Syntax

```
CONST
  identifier = expression;
```

Remarks

A constant declaration (CONST) defines an identifier, which denotes a constant value within the block containing the declaration. A constant identifier cannot be included in its own declaration. You can only assign a value to a constant during the declaration.

Expressions used in constant declarations must be written in such a way that the compiler can evaluate them at compile time.

A string cannot be assigned to a constant. If possible, use instead the WRITE procedure with a string parameter.

Examples

```
(* Constant Declarations *)

CONST
  limit = 65000;
  KeyCode = 943762;
```

D

DelBrushes

Syntax

```
DelBrushes;
```

Remarks

Deletes all color palette entries, which may be defined by means of the RGBBrush or the RGBPalette procedure. Be careful! The color palette has to be redefined after the entries have been deleted by the DelBrushes procedure.

Deleting of color palette entries may also affect the color display of MS Windows or other Windows applications.

Example

```
EVENT SetUp;
DelBrushes; (* all color palette entries
             are now lost *)
RGBPalette (10, 0,20, 0,20, 0,20);
            (* color palette is now redefined *)
```

DIV

Syntax

```
(operand) DIV (operand)
```

Remarks

The DIV operator divides two operands and returns an integer as result of the whole expression. As operands are allowed: integer constants, variables or procedures that return an integer.

Example

```
VAR a, b;
EVENT 1;
b:= 17;
a := b DIV 4; (* a is now 4 *)
```

E

EVENT

Syntax

```
EVENT [E<n>] | [SetUp];
      statements;
[END.] | [EVENT E <n+1>;]
```

Remarks

An event is a program part that starts with the keyword "EVENT" plus identifier number plus semicolon and ends with the next keyword "EVENT" or the keyword "END.". The identifier number must be in the range from 0 to 5. An event is a program unit, which may be triggered by its

corresponding single step or run button or by the SetUp button and whose code can be executed independently at a time.

Each event should have one dominant function, e.g. initialization or the implementation of a specific algorithm.

One CARP program may contain up to 6 events with identifiers from "E0" to "E5" and, additionally, the special event SetUp.

Example

```
EVENT E0; (* initialization of cell plane *)
PlFillRandom (Dead,Alive);
ShowPlane;

EVENT E1;
```

Expressions

Expressions consist of operators and operands. These are the operands:

constants

A constant declaration (CONST) defines an identifier, which denotes a constant value within the block containing the declaration. A constant identifier cannot be included in its own declaration.

variables

A variable (VAR) declaration associates an identifier and a type with a location in the memory where values of that type can be stored.

procedures

A procedure may be either predefined or user-defined. User-defined procedures may be function procedures or procedures using a side effect.

operators

The different types of operators existing in CAT (arithmetic operators, logic operators, comparative operators, bit operators) allow to join operands.

Subexpressions can be enclosed in parentheses to change the order of precedence.

F

FOR ... TO ...BY ... DO ... OD

Syntax

```
FOR assignment TO expression [BY step] DO
    statement;
OD loop_variable;
```

Remarks

The FOR ... OD instruction causes the statement after DO to be executed once for each case the Boolean expression is true. The Boolean expression is checked **after** the first execution of statement sequence. So, statement sequence is executed at least one time.

The loop variable is implicitly defined and may not be defined at the top of your CARP program. The loop variable may be read inside a loop, but never be written to. After the loop is completed the content of the loop variable is not defined any more.

If the BY construct is used, you can change the interval by which the loop variable is incremented to the value which follows BY.

Example

```
FOR x := 1 TO x < 10 BY 2 DO
  WRITE (x);
  IF (x + 3) = 5
    THEN Brake
    ELSE y := x + 1
  FI;
OD x;
```

G

GetX

Syntax

```
GetX;
```

Remarks

Returns the current x-value of the treated cell inside a PARALLEL DO loop.

Example

```
EVENT E1;
  PARALLEL DO
    ...
    IF top > 0
      THEN
        WRITE ('','Current x value : ', GetX);
      FI;
  OD;
```

GetY

Syntax

```
GetY;
```

Remarks

Returns the current y-value of the treated cell inside a PARALLEL DO loop.

Example

```
EVENT E1;
  PARALLEL DO
    IF top = 1
      THEN
        WRITE (GetY);
      FI;
  OD;
ShowPlane;
```

I

Identifiers

Identifiers denote the following:

```
CONST(ants)
PROC(edures programs)
VAR(iables)
```

Identifiers can be formed of up to 31 characters.

- The first character of an identifier must be a letter. Upper or lower case letters are allowed at any place.
- The characters that follow the first one must be letters, digits, or underscores (no spaces).

Like reserved words, identifiers are **case-sensitive**. Identifier may not coincide with reserved words.

Examples

```
(* Identifiers *)
VAR Limit;
CONST A_State = 4;
      B_State = 8;
```

IF .. THEN .. ELSE .. FI

Syntax

```
IF expression THEN statement [ELSE statement] FI;
```

Remarks

IF, THEN and ELSE specify the conditions under which a statement will be executed.

If the Boolean expression after IF is true, the statement after THEN is executed. Otherwise, if the ELSE part is present, the statement after ELSE is executed.

Example

```
x := Random (1000);
IF (x > 995)
    THEN Self := Alive;
    ELSE Self := Dead;
FI;
```

INV

Syntax

```
INV (operand)
```

Remarks

The INV operator has an integer or bit operand and converts all its zeroes to ones and all ones to

zeroes. As operand is allowed: an integer constant, a variable, a procedure that returns an integer or a bit operand.

Example

```
VAR a, b;

EVENT E4;
a := %110; (* 6 *)
b := INV a;
WRITE ('', ' a: ', a);
WRITE ('', ' INV a: ', b); (* result : - 7 *)
END.
```

M

MOD

Syntax

(operand) MOD (operand)

Remarks

The MOD operator divides two operands and returns the remainder as the result. As operands are allowed: integer constants, variables or procedures that return an integer.

Example

```
VAR a, b;

EVENT 1;
b := 17;
a := b MOD 4;
```

MooreSum

Syntax

MooreSum

Remarks

MooreSum adds the state values of the northern, southern, western, eastern, northeastern, northwestern, southeastern and southwestern neighbors of Self.

```
m m m m m
m O O O m
m O S O m
m O O O m
m m m m m
```

MooreSum (O = evaluated cell)

Example

```
EVENT E1;
PARALLEL DO
  IF (MooreSum <> 4)
    Self := Ill;
FI;
```

```
OD;
ShowPlane;
```

N

NeumannSum

Syntax

```
NeumannSum
```

Remarks

NeumannSum adds the state values of the northern, southern, western and eastern neighbors of Self.

```
m m m m m
m m O m m
m O S O m
m m O m m
m m m m m
NeumannSum (O = evaluated cell)
```

Example

```
EVENT E1;
PARALLEL DO
  IF (NeumannSum > 4)
    Self := Red;
  FI;
OD;
ShowPlane;
```

NOT

Syntax

```
NOT (operand)
```

Remarks

The NOT operator negates the result of the Boolean expression or operand that follows.

If the cells of your CAT model may only have the state 0 or 1, you may also use a cell denoter as operand for NOT.

Example

```
IF NOT (a > limit)
  THEN a := Self
FI;
```

O

OddCell

Syntax

```
OddCell
```

Remarks

OddCell provides access only to those cells whose x-value in the matrix is odd. This effects a cell matrix resembling a chess-board.

The x-value is counted from the first top left cell to the last right bottom cell continuously. That means for example that for a matrix with XYSIZE 31 the first cell of the second row is considered even (i.e. 32nd cell).

Example

```
EVENT E1;  
PARALLEL DO  
    Self := OddCell;  
OD;  
ShowPlane;
```

OR

Syntax

```
(operand) OR (operand)
```

Remarks

The OR operator connects two operands and returns true, if one or both operands are true. The remaining case returns false.

If the cells of your CAT model may only have the state 0 or 1, you may treat cells with the OR operator, too.

Example

```
IF (a > limit) OR (Self = 9) THEN a := Self FI;
```

P

PARALLEL DO

```
PARALLEL DO  
    statement;  
    [statement;]  
OD;
```

Remarks

The PARALLEL DO executes the instructions of its body once for all cells of the cell matrix in parallel.

Internally, a copy of the present state of all cells at the beginning of the PARALLEL DO construct is made, so that all conditional instructions etc. take the value contained in this copy. At the end, the computed state of all cells is written back and kept for future evaluations.

Mostly, the Self procedure is used inside a PARALLEL DO construct as an important part of a cell-related algorithm.

Example

```
EVENT E1;
  PlClipActive;
  PARALLEL DO
    Self := North XOR South XOR East XOR West;
  OD;
  ShowPlane;
```

ParallelMethod

Syntax

```
ParallelMethod;
```

Remarks

Is now the default method and needs not to be particularly defined.

In a future version of CAT, there will also be a method SequentialMethod.

Example

```
- - -
```

PillowForm

Syntax

```
PillowForm;
```

Remarks

The topology PillowForm assumes an axis in the middle of the matrix. Cells of the edges that have the same distance to this axis are copied to their counterpart.

```

c4c3c2c1 | c1c2c3c4
c m m m | m m m c
c m m m | m m m c
c m m m | m m m c
c m m m | m m m c
c m m m | m m m c
c4c3c2c1 | c1c2c3c4
Topology PillowForm (c = copied cell)
```

Example

```
EVENT SetUp;
  PillowForm;
  PlClipActive;
  ShowPlane;
```

PipeForm

Syntax

```
PipeForm;
```

Remarks

The topology PipeForm forms a virtually pipe-shaped matrix (tube), i.e. the top and bottom edges of the cell matrix are mutually copied to their opposite edges.

```

c c c c c c c
m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
c c c c c c c
Topology PipeForm (c = copied cell)

```

Example

```

EVENT SetUp;
PipeForm;
PlClipActive;
ShowPlane;

```

PlClipActive

Syntax

```
PlClipActive;
```

Remarks

Restricts the effect of the subsequent instructions to the cell matrix without its border areas defined by an optional XYBound declaration. This setting is the default value.

Example

```

EVENT SetUp;
  PlClipActive;
  RGBPalette (2, 20,10, 60,10, 80,10);

```

PlClipAll

Syntax

```
PlClipAll;
```

Remarks

Makes the whole cell matrix including the border areas available for subsequent instructions This procedure is only useful if you want to initialize the border areas.

Example

```

EVENT SetUp;
  PlClipAll;
  RGBPalette (2, 20,10, 60,10, 80,10);

```

PlClipXY

Syntax

```
PlClipXY ( x, y );
```

Remarks

Shows a central portion of the whole cell matrix with the corresponding x- and y-values. Thus, you can focus an area of special interest.

Example

```
EVENT E4;  
...  
PlClipXY (10,10);
```

PlFillRandom

Syntax

```
PlFillRandom ( Low, High );
```

Remarks

The procedure PlFillRandom initializes the cell matrix by random values ranging from parameter Low to High.

Keep in mind the range of states, which are defined by the Zet declaration. If the range of possible values produced by PlFillRandom exceeds the number of defined states, the range is restricted to the Zet value.

To take effect the parameters must be inside the range of defined colors and states (cp. Zet) and the smaller value **must** precede the greater one.

Example

```
EVENT E0;  
PlFillRandom (0,10);  
ShowPlane;
```

PlFillUni

Syntax

```
PlFillUni ( n );
```

Remarks

Gives the whole cell matrix a uniform state, which is defined by the n parameter, and via the associated color mapping value a uniform appearance.

To take effect the parameter must be inside the range of defined colors and states (cp. Zet).

Example

```
EVENT SetUp;  
PlClipActive;  
PlFillUni (32);  
ShowPlane;
```

PIFillUpStairs

Syntax

```
PIFillUpStairs ( Low, High, By );
```

Remarks

The procedure PIFillUpStairs creates a stair-like shape in the cell matrix. Thereby, the parameter Low gives the lower state and colormapping value , High the higher state and colormapping value and By the interval in which the range between Hi and Lo is filled. Useful for initialization purposes.

To take effect the parameters must be inside the range of defined colors and states (cp. Zet, Colors).

Example

```
EVENT SetUp;
  PClipAll;
  PIFillUpStairs (2, 20, 4);
  ShowPlane;
```

PROC

Syntax

```
PROC proc_identifier [VAR (identifier, identifier...)] ;;
[VAR identifier;]
[CONST identifier;]
BEGIN
  statement sequence;
[RETURN expression;]
END proc_identifier;
```

Remarks

A procedure is a program part, which performs a specific action, often based on a set of parameters. CAT provides both the function procedure that returns a value and the normal procedure that exchanges data with the CARP program via variables declared in the procedure head.

The procedure heading specifies the identifier for the procedure and the formal parameters (if any). A procedure is activated by a procedure call.

The procedure heading is followed by:

- a declaration part that declares local objects,
- the statements between BEGIN and END, which specify what is to be executed when the procedure is called.

A function procedure contains the keyword RETURN followed by an expression as last instruction.

Example

```
REF  knight_t_l    [-1, -2];
     knight_b_r    [1, 2];
     knight_mt_l   [-2, -1];
     knight_mb_r   [2, 1];
```

```
(* procedure adds four positions that might be reached by knight moves *)
PROC add_4_positions (VAR ret) ;;
```

```
BEGIN
```

```

ret := knight_t_l + knight_b_r + knight_mt_l + knight_mb_r;
END add_4_positions;

PROC add_4_pos;; (* the same more briefly and the      *)
BEGIN          (* procedure returning the value itself *)
RETURN knight_t_l + knight_b_r + knight_mt_l + knight_mb_r;
END add_4_pos;

EVENT E3;
  PARALLEL DO
    WRITE ('', 'Value : ', add_4_pos);
  OD;
  ShowPlane;

```

R

Random

Syntax

```
Random ( n );
```

Remarks

Returns a random number between 0 and n. Negative n values are not allowed.

Every call of a loop that contains the Random instruction produces the same sequence of results for internal reasons. If you want to avoid this effect, use Randomize additionally.

Example

```

VAR x;

EVENT E0;
...
x := Random (1000);
IF x > 950 THEN add_four_positions FI;

```

Randomize

Syntax

```
Randomize;
```

Remarks

Creates a new base number for the random number generator.

This procedure is advisable if you want to prevent that each loop (PARALLEL DO, WHILE), that contains a Random procedure produces the same sequence of random numbers. The Randomize procedure should be used in the event SetUp or in the event containing the Random procedure.

Randomize should **not** be used, if you are searching for a program error that is related to random numbers.

Example

```
EVENT SetUp;
  RGBPalette(Colors, $0, 10, $32,10, $B6,10,);
  Randomize;
```

RECIPE**Syntax**

```
RECIPE
[XYSize and/or XYBound declarations;]
[VAR declarations;]
[CONST declarations;]
[REF declarations;]
[PROC declarations;]
EVENT declarations;
statements;
END.
```

Remarks

A CARP program has to be started by the keyword "RECIPE" and terminates with the keyword "END." ("END" followed by a point). Between these delimiters, you can declare variables, constants, referred neighbors of a cell, user defined procedures and - as independently executable parts of a CARP program - events.

Normally, the keyword RECIPE is followed by settings for the size of the cell matrix and evaluated neighborhood, by definitions of constants (CONST) , variables (VAR) or referenced cells (REF) and by events (EVENT) that contain the proper program functions. A template for a program may look as follows:

Example

```
RECIPE XYSize = 50;
CONST ...;
VAR ...;
REF ...;

EVENT SetUp;
...

EVENT E0;
...

EVENT E1;...
...
END.
```

REF**Syntax**

```
REF identifier [xvalue,yvalue];      (read)
```

Remarks

The REF declaration assigns a name to specified neighboring cells of the cell Self and allows such

to refer to the value of these identified cells by their name. Precondition: The cell referred to may not exceed the limits set by XYBound.

To use the value of a certain reference cell you have to do two things:

- Define a referenced cell.
- Use the defined neighbors within the program by referring to their names. Compare the sample program part on the bottom:

Note:

- You may only read from referenced cells, not write to them. This is restricted to the procedure Self.
- X-values to the right of Self and Y-values on the bottom of Self have a positive value.

Example

```
REF   right_neighbor      [1,0];
      left_neighbor       [-1,0];
      top_neighbor        [0,-1];
      bottom_neighbor     [0,1];
      ...

EVENT E1;
  PARALLEL DO
    Self := top_neighbor OR left_neighbor OR Self OR
           right_neighbor OR bottom_neighbor;
  OD;
  ShowPlane;
END.
```

RePaint

Syntax

```
RePaint;
```

Remarks

Paints the graphic window again, if appearance or colors of the cell matrix are garbled. Scarcely useful inside a CARP program, compare instead the corresponding RePaint button.

Example

```
- - -
```

REPEAT .. UNTIL

Syntax

```
REPEAT
  statement;
  [statement;]
UNTIL expression;
```

Remarks

The statements between REPEAT and UNTIL are executed in sequence until, at the end of the loop body, the Boolean expression after UNTIL is true.

The sequence is executed at least once. The delimiter of the REPEAT ... UNTIL loop is a ';'.
Example

```

x := 1;
REPEAT
  IF (x + 3) = 8
    THEN WRITE (x);
    x := x + 1
  ELSE x := x + 1
  FI;
UNTIL x > 15;

```

RGBBrush

Syntax

```
RGBBrush ( n, r, g, b );
```

Remarks

Assigns the color mapping n the colors given by the parameters r(ed), g(reen) and b(blue).

This procedure is advisable, if you want to assign certain cell states to specific colors. (sample a)

This procedure may also be used if you want to change the previous overall color settings for a special color at a given time (sample b).

Example

(sample a)

```

RECIPE XYSize = 60;
      Zet     = 4;
      Colors  = 4;

CONST dead      = %00; (* bit sample 0000 *)
      just_died = %01; (* bit sample 0001 *)
      just_born = %10; (* bit sample 0010 *)
      alive     = %11; (* bit sample 0011 *)

EVENT E0;
  RGBBrush (dead, 0, 0, 0);           (* black *)
  RGBBrush (just_died, 152, 88, 46); (* brown *)
  RGBBrush (just_born, 74, 229, 3);  (* light green *)
  RGBBrush (alive, 50, 174, 30);     (* dark green *)

```

(sample b)

```

VAR cell_state;

EVENT E4;
IF generation_counter > 100
  THEN RGBBrush (cell_state, 24 ,30, 30)
FI;

```

RGBPalette

Syntax

```
RGBPalette ( n, r0, ri, g0, gi, b0, bi );
```

Remarks

The procedure RGBPalette allows to define a set of colors and their dissemination on the color palette. These parameters have to be defined:

- n Number of colors to define. Generally, this number should comply with the number of defined states (Zet).
- r0 Starting point for the red value.
- ri Increment value by which the red value increases. Values above 255 are corrected to a maximum value 255.
- g0 Starting point for the green value.
- gi Increment value by which the green value increases. Values above 255 are corrected to a maximum value 255.
- b0 Starting point for the blue value.
- bi Increment value by which the blue value increases. Values above 255 are corrected to a maximum value 255.

Some general remarks: each defined color is a set of three values for their portion of red, green and blue (rgb). Each of this component color has a definition range from 0 to 255 (hexadecimal \$0 to \$FF). Red, green and blue each set to 255 result in the color white, red, green and blue each set to 0 result in the color black. That is the area, in which you may select certain colors.

Note:

RGBPalette sets the colors for your automaton tool model in a global way. Besides, you may define a specific color by means of the RGBBrush procedure.

Colors defined either by RGBPalette or RGBBrush may be varied interactively later on by means of the color customizing button. To use this button for particular colors is most advisable, because it is very difficult to predict the resulting color only by defining the red, green and blue parameters.

Values for increments (ri, gi, bi) may also be negative. This makes sense together with high starting values for r0, g0 or b0.

Values may be given as decimal or hexadecimal figures with leading \$.

The example program below will generate this color palette:

| | r-value | g-value | b-value |
|---------|---------|---------|---------|
| color 1 | 30 | 40 | 50 |
| color 2 | 45 | 55 | 65 |
| color 3 | 60 | 70 | 80 |
| color 4 | 75 | 85 | 95 |
| color 5 | 90 | 100 | 110 |

Example

```
EVENT SetUp;
RGBPalette (5, 30,15, 40,15, 50,15);
```

RingForm

Syntax

```
RingForm;
```

Remarks

The topology RingForm forms a virtual endless matrix connecting at first two edges and then the edges of the built up pipe. This body is also known as torus. The RingForm topology is the **default setting** and therefore, the RingForm instruction may be omitted.

```

C C C C C C C
C m m m m m C
C m m m m m C
C m m m m m C
C m m m m m C
C m m m m m C
C m m m m m C
C C C C C C C

```

Topology RingForm (c = copied cell)

Example

```

EVENT SetUp;
RingForm;
PlClipActive;
ShowPlane;

```

S

Self

Syntax

```
Self      (read / write)
```

Remarks

The only instruction to change the state of a cell and thereby the whole cell matrix is Self. All other cell matrix-related procedures only allow reading of a cell state.

Self is strongly connected with the PARALLEL DO instruction. Inside a PARALLEL DO cycle, Self allows for each cell read or write access.

All instructions inside a PARALLEL DO and related to Self and other referred cells have to be thought of as actually happening **simultaneously**. (In fact, on a single CPU computer, a copy of the state of all cells will be made, and, depending on these values, the instructions for all cells will be of course carried out subsequently.) But focussing on CAT's concept, Self and PARALLEL DO are the decisive keys to leave array treatment and such things behind and turn to the new programming paradigm 'the cell in its environment'.

The effect of the sample instructions below (it implements Conveys Life program): For each cell of the cell matrix will be controlled as to whether Self is 'alive' (read access) and has two or three 'alive' neighbors ('alive' is assigned to the state 1 of a cell). If this is true, Self will be set to 'alive' (write access with the ':=' procedure). Otherwise, if Self is 'dead' and has three 'alive' neighbors, Self will be set again to 'alive'. In all other cases, Self will be considered as 'too lonely' or 'overcrowded' and therefore set to 'dead'.

Example

```

PARALLEL DO
  IF (Self = Alive) AND
    ((MooreSum = 3) OR (MooreSum = 2))
  THEN Self := Alive
  ELSE IF (Self = Dead) AND (MooreSum = 3)

```

```

        THEN Self := Alive
        ELSE Self := Dead
        FI
    FI;
OD;

```

SetLattice

Syntax

```
SetLattice (thickness, foregroundcolor, backgroundcolor);
```

Remarks

Returns a lattice pattern originating from the center of your cell matrix with free spaces of size 'thickness' and with the corresponding fore- and backgroundcolors.

Example

```

EVENT SetUp;
PlClipActive;
SetLattice(3,1,19);

```

SheetForm

Syntax

```
SheetForm;
```

Remarks

The topology SheetForm makes the evaluation of algorithms end on the edges of the cell matrix without any further continuation on other edges.

```

m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
Topology SheetForm (m = normal cell,
no copied cell)

```

Example

```

EVENT SetUp;
SheetForm;
PlClipActive;
ShowPlane;

```

SHL

Syntax

```
(operand) SHL (operand)
```

Remarks

The SHL operator shifts all bits of a binary digit by the value of the second operand times to the left. Leading digits are filled by 0.

This works for integer variables, constants or referenced cells interpreted as binary values as well as for explicitly defined binary digits. In the following sample the variable b returns the value 12 both times.

Example

```
a := %110; (* 6 *) (* using bit operator *)
b := a SHL 1;
WRITE ('',b);

(* without bit operator *)

a := 6;
b := a SHL 1;
WRITE ('',b);
```

ShowCell

Syntax

```
ShowCell (n);
```

Remarks

This procedure shows the current state of the cell with the x-value n.

This value is computed as $n = x + (XSize * (y - 1))$. (An example: in a cell matrix with XYSIZE = 10 the first cell in the top left corner counts 0 and the last cell in the bottom right corner counts 99.)

This procedure is very CPU-time-consuming and should only be used if the focus is on a single cell.

Example

```
EVENT E1;
  PARALLEL DO
    IF x > delimiter
      THEN
        Self := (NeumannSum + Self) > 0
      FI;
  OD;
ShowCell (74);
ShowCell (75);
ShowCell (76);
```

ShowKind

Syntax

```
ShowKind (w);
```

Remarks

Shows the state and color mapping of a single cell.

Useful only if the focus is on these settings of a single cell. Can then be combined with the ShowCell procedure.

Example

```

EVENT E1;
  ShowKind (74);
  ShowKind (75);
  ShowKind (76);
  ShowCell (74);
  ShowCell (75);
  ShowCell (76);

```

ShowPlane

Syntax

```
ShowPlane;
```

Remarks

This function is necessary for showing the whole cell matrix in its current state. Should normally occur at the end of any event description for control purposes. If your cellular automaton model is very CPU time-consuming, you can order to display only every tenth or whatever generation of your CAT model.

Never use ShowPlane inside a PARALLEL DO instruction, for this might crash CAT.

Example

```

EVENT E1;
...
IF i < 50
  THEN ShowPlane
  ELSE IF i MOD 10 = 0
    THEN ShowPlane
  FI
FI;
i := i +1;

```

SHR

Syntax

```
(operand) SHR (operand)
```

Remarks

The SHR operator shifts all bits of a binary digit by the value of the second operand times to the right. Leading digits are filled by 0.

This works for integer variables, constants or referenced cells interpreted as binary values as well as for explicitly defined binary digits.

Example

```

(* life model (cp. Convey) with four different states: *)
(* life, just_born, dead, just_died *)

RECIPE XYBound = 60;

CONST dead      = %00; (* bit sample 0000 *)
  just_died     = %01; (* bit sample 0001 *)
  just_born     = %10; (* bit sample 0010 *)
  alive        = %11; (* bit sample 0011 *)

```

```

                (*                A *)
                (*                "alive bit" *)

REF   right_n   [1,0];
      left_n    [-1,0];
      top_n     [0,-1];
      bot_n     [0,1];

PROC add_second_bit;;
(* procedure evaluates second bit of neighbors that *)
(* indicate alive state and returns sum of found bits*)

BEGIN
                *)
RETURN ((right_n XOR %01) SHR 1) + ((left_n XOR %01) SHR 1) + ((top_n XOR
%01) SHR 1) + ((bot_n XOR %01) SHR 1)
END add_second_bit;

EVENT E0;
  RGBBrush (dead, 0, 0, 0);          (* black *)
  RGBBrush (just_died, 152, 88, 46); (* brown *)
  RGBBrush (just_born, 74, 229, 3);  (* light green *)
  RGBBrush (alive, 50, 174, 30);     (* dark green *)

EVENT E1;
PARALLEL DO
a := add_second_bit;

IF (a = 2) OR (a = 3)
  THEN IF (a = 3) AND ((Self = dead) OR (Self = just_died))
    THEN Self := just_born;
    ELSE Self := alive
    FI;
  ELSE IF (Self = alive) OR (Self = just_born)
    THEN Self := just_died;
    ELSE Self := dead;
    FI;
FI;
OD;
ShowPlane;

END.

```

Statement

A statement is one of the following:

```

assignment (:=)
BEGIN..END
FOR..TO..BY..DO..OD
PARALLEL...DO
IF..THEN..ELSE FI
PROC(edure)
REPEAT .. UNTIL
WHILE .. DO .. OD

```

V

VAR

Syntax

```
VAR
  identifier, ... identifier;
```

Remarks

A variable (VAR) declaration associates an identifier with a location in the memory where values can be stored.

You may not combine a declaration of a variable with an assignment like you might expect from the usage of constants. Assign a value to the variable inside an event.

Examples

```
(* Variable Declarations *)
VAR
  x , y , z;

x := 3;
```

W

WHILE ... DO ... OD

Syntax

```
WHILE expression DO statement OD;
```

Remarks

A WHILE statement contains an expression, which controls the repeated execution of one or several statements embraced by the keywords 'DO' and 'OD'. The statement after DO is executed repeatedly as long as the Boolean expression is true.

The expression is evaluated before the statement is executed, so if the expression is false at the beginning, the statement will not be executed at all.

Example

```
WHILE i < 20 DO
  Self := NeumannSum DIV 2;
  i := i + 1 OD;
```

WinClipActive

Syntax

```
WinClipActive;
```

Remarks

Shows only the active part of the cell matrix without any border areas. WinClipAll or WinClipActive are only relevant for the **appearance** of the STATE window.

The same can be achieved interactively by means of the magnifier button.

Example

```
EVENT SetUp;
```

```

    RGBPalette(Colors, $0, $FF, $32,0, $B6,0);
    WinClipActive;
    ShowPlane;

```

WinClipXY

Syntax

```
WinClipXY (x, y);
```

Remarks

Shows a central portion of the whole cell matrix with the corresponding x- and y-values. Thus, you can focus on an area of special interest.

Example

```

EVENT E4;
P1FillRandom (1,4);
WinClipXY ( 5, 5);

```

WinClipAll

Syntax

```
WinClipAll;
```

Remarks

Shows the whole cell matrix including the border areas. WinClipAll or WinClipActive are only relevant for the **appearance** of the STATE window.

The same can be done interactively by means of the magnifier button.

Example

```

EVENT SetUp;
    RGBPalette(Colors, $0, $FF, $32,0, $B6,0);
    WinClipAll;
    ShowPlane;

```

WrDCaps

Syntax

```
WrDCaps;
```

Remarks

Acronym for 'Write Display Capabilities'. The corresponding values specific to your data display are written into the LIST window. Useful for system administration and service purposes, especially on graphic resolution issues. A possible output in the LIST window:

```
Display capabilities
```

```

H/V Resolution :      1024      768
Pixel/Planes   :         8        1

```

```
Colors          :      20
Palette/reserv :    256    20
```

Example

```
EVENT SetUp;
  RGBPalette(Colors, 127, 2, 127,30, 127,30);
  ShowPlane;
  WrDCaps;
```

WRITE

Syntax

```
WRITE ([string] | [(VAR) identifier] | [(CONST) identifier] | [(PROC)
identifier] [:n] ['']);
```

Remarks

Writes the contents of variables, constants, values of function procedures or strings to the LIST window and the .CAL file. Different operands have to be divided by a comma, strings must be included by ' (apostrophe).

Several facilities are provided for formatting the output:

- [(var):n] If the contents of the variable or the constant has less than n digits, the output is indented accordingly.
- '' Put at the end or beginning of the parameter list, a carriage return / linefeed (CR/LF) at the end or beginning of the output is caused.

If the buffer to which all data is moved is full, you will get the message "Editor buffer is full" and will be prompted whether you want to overwrite the contents or stop writing. All written informations may later be inspected by means of the LIST window.

Example

```
IF (x > limit)
  THEN WRITE ('','Limit exceeded with value : ');
  WRITE (x : 8, '');
FI;
```

WrMCaps

Syntax

```
WrMCaps;
```

Remarks

Acronym for 'Write Memory Capabilities'. The corresponding values of RAM (Random Access Memory) usage specific to your hardware and operating system configuration are written into the LIST window. Useful for system administration and service purposes, especially if you are in doubt about sufficient memory (RAM).

A possible output in the LIST window may look as follows:

```
Memory and resources

Mem_free   KB :   47730
Mem_block  KB :   16320
Sys_Res    %  :     62
```

```
GDI_Res    %    :    62
Usr_Res    %    :    83
```

Example

```
EVENT SetUp;
  RGBPalette(Colors, 127, 2, 127,30, 127,30);
  ShowPlane;
  WrMCaps;
```

WrPPars

Syntax

```
WrPPars;
```

Remarks

Acronym for 'Write Plane Parameters'. The corresponding values for your actual CAT cell matrix configuration are written into the LIST Window. Useful for system administration and debugging purposes.

A possible output in the LIST window may look as follows:

```
CAT actual parameters

X/YSize   :      31      31
X/YBound  :       3       3
X/YTotal  :      37      37
Act/TotSz :     961    1369
Org/Skip  :     114       6
```

Example

```
EVENT SetUp;
  RGBPalette(Colors, 127, 2, 127,30, 127,30);
  ShowPlane;
  WrPPars;
```

X

XOR

Syntax

```
(operand) XOR (operand)
```

Remarks

The XOR operator adds two operands and returns true if one of the two operands returns true and the other false. If both the operands return true or false, the whole expression returns false. As operands are allowed: integer constants, variables, referenced cells or procedures that return an integer.

Example

```
b := 8;
IF (a > limit) XOR (b = 8)
  THEN a := Self
```

```
FI;
```

XYBound

Syntax

```
XYBound = n;
```

Remarks

Defines the range of the neighborhood of the cell Self (in x and y values) that can be evaluated by any instruction of your CARP program. Referenced cells (REF) must be inside the range of the XYBound.

XYBound defines moreover the width of the border area of the cell matrix that is shown if the PIClipAll procedure is used or that is effected by an equivalent setting of the magnifier button.

Example

```
RECIPE XYSize = 140;
      XYBound = 1 ;
```

```
REF  east      [1,0];
     west      [-1,0];
     north     [0,-1];
     south     [0,1];
     north_ea  [1,-1];
     north_we  [-1,-1];
     south_ea  [1,1];
     south_we  [-1,1];
```

```
PROC add_second_bit; (* procedure evaluates second bit of*)
                    (* neighbors indicating alive state *)
BEGIN              (* and returns sum of found bits      *)
RETURN ((east XOR %01) SHR 1) + ((west XOR %01) SHR 1) +
        ((north XOR %01) SHR 1) + ((south XOR %01) SHR 1) +
        ((north_ea XOR %01) SHR 1) + ((north_we XOR %01) SHR 1) +
        ((south_ea XOR %01) SHR 1) + ((south_we XOR %01) SHR 1)
END add_second_bit;
```

XYSize

Syntax

```
XYSize = n;
```

Remarks

Defines the horizontal (x) and vertical (y) size of a cell matrix. If you want to define a different YSize compared to XSize, you can use the YSizedeclaration.

Keep in mind that high XYSize values are very CPU time-consuming.

Example

```
RECIPE XYSize = 120;
      XYBound = 2;
```

Y

YSize

Syntax

```
YSize = n;
```

Remarks

Defines the vertical (y) size of a cell matrix

Keep in mind that high XYSIZE or YSIZE values are very CPU time-consuming

Example

```
RECIPE XYSIZE = 120;  
      YSIZE = 100;
```

Z

Zet

Syntax

```
Zet = n;
```

Remarks

Zet is the number of different states that can be adopted by any cell. The Zet value complies normally with the number of available Colors.

Example

```
RECIPE XYSIZE = 140;  
      XYBOUND = 3 ;  
      ZET = 20;  
      COLORS = 20;
```