Menu button

The Menu button provides access to a menu that offers the same functions as the buttons left to the Menu button.

This way may be advisable for people that are used to menu driven programs.

Show Color
Show State
Color Mapping
Hex Display
Zoom
Repaint

Round counter

There are two counters that indicate the current generation of your cellular automaton model:
- a counter in a grey label to the left of the button panel of the state window
- a counter right to the program name 'CAT' label separated by a colon.

You can use both round counters particularly if color display of the cell matrix is switched off.

Palette customizing button

The above button provides four functions for manipulating the color palette and some others for loading, saving, displaying and printing a palette.

Overview on menu items and functions

item	effect
Initialize Palette	
B/W blend	Generates a regular scale of the colors black, white and grey with low differences between neighboring colors, but with remarkable differences between the most opposite colors.
Color blend	Generates a regular scale of colors with low diffences between neighbored colors, but with remarkable differences between the most opposed colors.
By parameters...	Initializes the STATE window by a blend of colors which can be controlled by certain parameters.
<user defined>	Allows to load a particular palette by means of this menu under a prefixed name.
Palette	
Load	Loads an existing palette as CAP file.
Save	Saves current palette as <name of model>.CAP file.
Save As	Saves current palette with free selectable name plus file extension .CAP.
Show	Shows the current color palette on the data display. Close the according window by doubleclicking the 'close' button in the top left corner.
Print	Prints the current color palette on your standard printer.

Corresponding CARP procedures:
RGBPalette, RGBBrush DelBrushes

Palette | B/W blend

Generates a regular pattern consisting of the colors black, white and grey with low diffences between neighbored colors, but with remarkable differences between the cells of the top left and the bottom right corner.

Click the Palette customizing button.
=> A menu pops up.
Select the item *B/W blend* in menu *Palette*.
=> The cell matrix will be initialized by a blend of black, white and gray cells with slight differences between adjacent cells, but with greater differences between distant cells.

On the actual Zet setting depends how rough or fine the grades between the different colors appears.

Palette | Color blend

Generates a regular color pattern with little differences between neighboring colors, but with remarkable differences between cells in the top left and the bottom right corner.

Click the Palette customizing button.
=> A menu pops up.
Select the item *Color blend* in menu *Palette*.
=> The cell matrix will be initialized by a color blend with slight differences between adjacent cells, but with great differences in colors between distant cells.

It depends on the actual Zet and Colors settings how rough or fine the grades between the different colors appear.

Palette | <user file>

The menu item <user file> serves under certain preconditions to load previously saved color palettes in a more rapid way.

Click the Palette customizing button.
=> A menu pops up.
Select the item <*file name*> in menu *Palette*.
=> The previous color palette will be overwritten by the selected one as can be noticed by a different appearance of the cell matrix.

To appear as a menu item, the corresponding file <user file.CAP> must be contained in the CAT.INI file and be a valid .CAP file.

Example for an corresponding entry in the file CAT.INI:

```
[Files]
LastProject=
UserPal1=USERFILE.CAP
UserPal2=
UserPal3=
UserPal4=

[Sounds]
ReadOnly=ding.wav
```

Note:
Up to four different palettes may be loaded by means of these menu items provided that corresponding names are contained in the CAT.INI file. To save an according .CAP file use Save or Save As...**.** To write the names of the .CAP files in the CAR.INI file, you can use any ASCII editor.

Palette | Load

Loads an existing color palette saved previously as a .CAP file.

Click the Palette customizing button.
=> A menu pops up.
Select the item *Load* in the menu *Palette*.
=> You will get a file selection box to choose a .CAP file from.
Select a palette (.CAP file) and click OK.
=> The previous color palette will be overwritten by the selected one as can be noticed by a different appearance of the cell matrix.

Note:
If you have changed your old color palette, you will not be prompted for saving these changes. The new color palette will overwrite the previous one without any further dialog. So: handle with care!

Palette | Save

Saves the actual state of the color palette as . CAP file.

Click the Palette customizing button.
=> A menu pops up.
Select the item *Save* in the menu *Palette*.
=> The current color palette will be saved under the prefixed name.

If there is no name assigned to the current color palette, you will be prompted for a name (cp. Save As).

Palette | Save As

Saves the current color palette under a free selectable name plus file extension .CAP.

Click the Palette customizing button.
=> A menu pops up.
Select the item *Save as* in the menu *Palette*.
=> You will get a file dialog box.
Type in a meaningful name and click OK.
=> The current color palette will be saved under the chosen name.

Palette | Show

Shows the current color palette immediately on the screen.

Click the Palette customizing button.
=> A menu pops up.
Select the item *Show* in the menu *Palette*.
=> You will get a window containing the current color palette similar to the one below.

The first figure of a column is the color mapping value (mostly connected with a state value), the second is the value for the color red, the third for the color green and the fourth for the color blue. Figures may also appear as hexadecimal figures. This depends on your current setting of the hexadecimal button.

Example of a color palette (compressed to two columns to save space):

```
000  000 000 000          015  150 000 000
001  010 000 000          016  160 000 000
002  020 000 000          017  170 000 000
003  030 000 000          018  180 000 000
004  040 000 000          019  190 000 000
005  050 000 000          020  200 000 000
006  060 000 000          021  210 000 000
007  070 000 000          022  220 000 000
008  080 000 000          023  230 000 000
009  090 000 000          024  240 000 000
010  100 000 000          025  250 000 000
011  110 000 000          026  255 010 000
012  120 000 000          027  255 020 000
013  130 000 000          028  255 030 000
014  140 000 000          029  255 040 000
```

Close the respective window by doubleclicking the 'close' button in the top left corner.

Palette | Print

Prints the current color palette on your standard printer.

Click the Palette customizing button.
=> A menu pops up.
Select the item *Print* in the menu *Palette*.
=> A print dialog box appears.
Confirm printing by clicking OK.
=> You will get a printout containing the current color palette similar to the one below.

The first figure of a column is the color mapping value (mostly connected with a state value), the second is the value for the color red, the third for the color green and the fourth for the color blue. Figures may also appear as hexadecimal figures. This depends on your current setting of the hexadecimal button.

Example of a printed color palette (compressed to two columns to save space):

```
000  000 000 000          015  150 000 000
001  010 000 000          016  160 000 000
002  020 000 000          017  170 000 000
003  030 000 000          018  180 000 000
004  040 000 000          019  190 000 000
005  050 000 000          020  200 000 000
006  060 000 000          021  210 000 000
007  070 000 000          022  220 000 000
008  080 000 000          023  230 000 000
009  090 000 000          024  240 000 000
010  100 000 000          025  250 000 000
011  110 000 000          026  255 010 000
012  120 000 000          027  255 020 000
013  130 000 000          028  255 030 000
014  140 000 000          029  255 040 000
```

State | Single Value 0

Initializes the cell matrix with the uniform value 0 and its assigned color.

Click the State control button.
=> A menu pops up.
Select the item *Single Value 0* in menu *Initialize States*.
=> The cell matrix will change to a single color representing a equal value for each cell.

Note:
This operation is advisable if you want to get a uniform cell matrix, that now can be changed on few positions by means of some left (ascending count) or right (descending count) mouse clicks on the corresponding cell position.

State | Single Value 1

Initializes the cell matrix with the uniform value 1 and its assigned color.

Click the State control button.
=> A menu pops up.
Select the item *Single Value 1* in menu *Initialize States*.
=> The cell matrix will change to a single color representing a equal value for each cell.

Note:
This operation is advisable if you want to get a uniform cell matrix, that now can be changed on few positions by aid of left (asending count) or right (descending count) mouse clicks on the corresponding cell position.

State | Single Value N

Initializes the cell matrix with the uniform value n to be entered by a dialog box.

Click the State control button.
=> A menu pops up.
Select the item *Single Value n...* in menu *Initialize States*.
A dialog box comes up.
Enter a figure inside the range of Zet and click OK:
=> The cell matrix will change to a single color representing the chosen value.

Note:
This operation is advisable if you want to get a uniform cell matrix, that now can be changed on few positions by aid of left (asending count) or right (descending count) mouse clicks on the corresponding cell position.
Entering values above Zet will **not** change the cell matrix.

State | Random Values

Initializes the cell matrix with the random values that are defined by .

Click the State control button.
=> A menu pops up.
Select the item *Random Values n* in menu *Initialize States*.
=> The cell matrix will adopt a random pattern, if every state is assigned to a different color. Otherwise, you will see the random dissemination of integer values at least by use of the Numeric state button.

Note:
This operation is advisable if you want to get random patterns on your cell matrix for e.g. a growth model.

State | Upstairs

Initializes the cell matrix with the pattern resembling a stair. The 'steps' of the stair vary with the prefixed Zet and Color value.

Click the State control button.
=> A menu pops up.
Select the item *Upstairs* in menu *Initialize States*.
=> The cell matrix will adopt a stair like pattern, beginning with the lowest value in the top left corner and continuing to the highest value on the right bottom corner.

The stair form results from the fact that the first step is smaller or greater than XYBound followed by XYBOUND * XYBOUD DIV Zet cells of the next value. If no stair is visible, you will despite see the stairlike dissemination of integer values by use of the Numeric state button.

State | Load

Loads a state of a CAT model previously saved as .CAS file.

Click the State control button.
=> A menu pops up.
Select the item *Load* in menu *State*.
=> The cell matrix will adopt the state defined by the corresponding .CAS file.

Note:
If the loaded .CAS file is **greater** than the current cell matrix only those cells are loaded that fit into the current cell matrix. This process starts from the center and cuts eventually cell areas at the edges.
If the loaded .CAS file is **smaller** than the current cell matrix only those cells of the current cell matrix are affected that have counterparts in the loaded .CAS file. This process starts from the center and lets cell areas at the edges eventually unchanged.
If the to be loaded .CAS file contains more different states (Zet) than the current CAT model, the new state value will be computed as `high_state_value MOD Zet`.

State | Save

Saves the current state of the cell matrix under a prefixed name.

Click the State control button.
=> A menu pops up.
Select the item *Save* in menu *State*.
=> The cell matrix will be saved as <mod_name>.CAS file.

Note:
If no name is present, you will be prompted for a name by a Save as dialog box.
For taking a snapshot of a certain critical state of the cell matrix, the Save as function is recommended for less danger of overwriting important data.

State | SaveAs

Saves the current state of the cell matrix under a free selectable name.

Click the State control button.
=> A menu pops up.
Select the item *Save As* in menu *State*.
=> A Save As dialog box comes up and prompts for a file name.
Enter a meaninful file name.
=> The cell matrix will be saved as <filename>.CAS file.

Note:
This function is advisable if you want to keep snapshots of critical states of your cellular automaton model..

State | Show

Displays the current state of your cell matrix in numeric format on your data display. You will get an output simular to the one below.

Click the State control button.
=> A menu pops up.
Select the item *Show* in menu *State.*
=> You will receive a particular Plane window where each cell of your cell matrix is represented by its numeric value. Compare the example below.

If there are too many defined cells you can scroll the contents of this window to regard all cells.

Explanation of the sample Plane window:
The first two figures of the first line contain the number of XBound (size on the x axis) and YBound (size on the y axis) . The third figure in the second line contains the number of defined states (cp. Zet). All other figures represent the state of each cell according to their x and y coordinates. (The fourth value 010 in the fifth line represents the cell c (4|3)).

Figures may also appear as hexadecimal figures. This depends on your current setting of the hexadecimal button.

Example of a Plane window:

```
00000032  00000032
00000020
003 012 019 008 006 010 007 008 011 002 009 010 007 019 011 005 008 013 000 000 008 016 006
004 013 012 001 008 004 013 002 016
014 002 016 010 003 001 019 004 017 009 014 005 001 018 000 003 004 017 000 006 005 017 006
015 008 019 013 010 012 004 007 001
007 006 003 000 014 015 006 005 008 018 001 008 000 015 006 005 019 017 006 012 019 000 002
004 013 014 000 008 012 008 007 015
014 015 011 018 013 000 008 008 012 018 011 001 002 001 018 001 013 013 005 009 013 009 005
004 002 014 011 009 006 011 014 002
(abridged)
```

Close the according window by doubleclicking the close button in the top left corner.

State | Print

Prints the current state of your cell matrix on your standard printer.

Click the State control button.
=> A menu pops up.
Select the item *Print* in menu *State*.
=> You will get a printout simular to the one below of the current state of your cell matrix (.CAS file) on your printer.

The first two rows contain the number of XBound (size on the x axis) and YBound (size on the y axis) . The third figure in the second row contains the number of defined states (cp. Zet). All other figures represent the state of each cell according to its x and y coordinates. (The value 010 in the fifth line on the fourth place represents the cell c (4|3)).

Figures may also appear as hexadecimal figures. This depends on your current setting of the hexadecimal button.

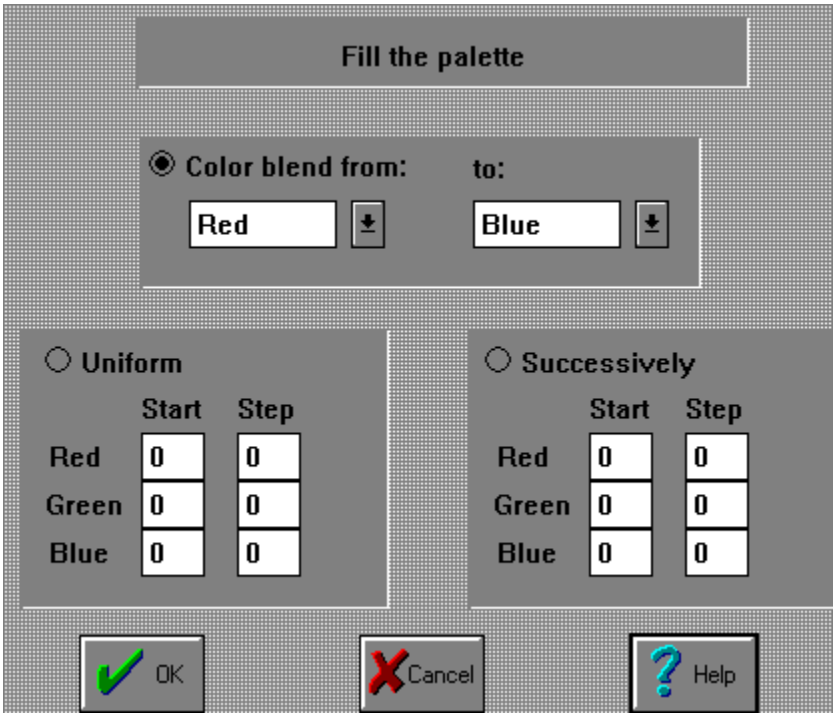
Example of a printed state window with XYBound 14:

```
0000001400000014
00000020
003 012 019 008 006 010 007 008 011 002 009 010 007 019
011 005 008 013 000 000 008 016 006 004 013 012 001 008
014 002 016 010 003 001 019 004 017 009 014 005 001 0180
000 006 005 017 006 015 008 019 013 010 012 004 007 001
007 006 003 000 014 015 006 005 008 018 001 008 000 015
006 005 019 017 006 012 019 000 002 004 013 014 000 008
014 015 011 018 013 000 008 008 012 018 011 001 002 001
018 001 013 013 005 009 013 009 005 004 002 014 011 009
017 012 001 011 013 016 014 016 005 000 011 015 015 011
009 003 006 014 010 011 005 017 006 016 010 002 006 015
012 017 001 000 010 009 017 009 011 012 011 014 019 019
016 006 002 006 016 006 003 014 011 016 019 011 016 011
010 019 012 017 018 002 004 012 013 017 012 014 018 019
007 000 008 018 019 018 018 014 012 019 010 002 000 005
```

Palette | By parameters

The item *By parameters* in the *Initialize Palette* menu initializes the cell matrix of the STATE window with a color blend which can be controlled by certain parameters by the user.

Click the State control button.
=> A menu pops up.
Click *By parameters...* in the *Initialize Palette* menu.
=> You are offered this dialog box:



You can change the color blend in two steps:

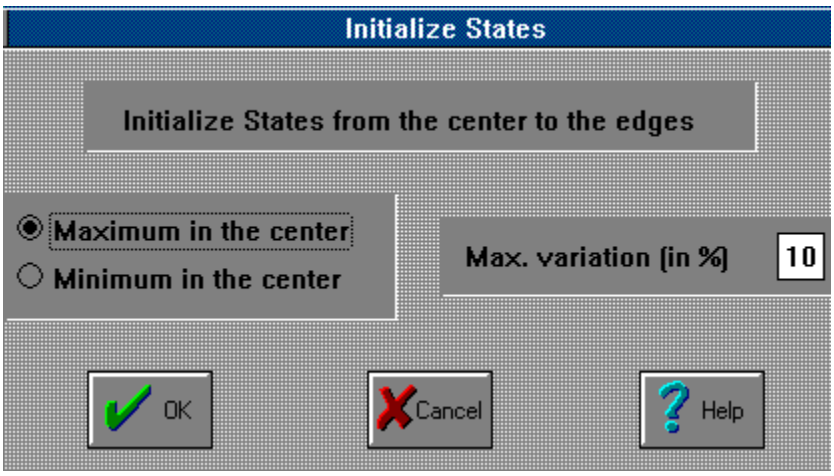
1. Change initial and terminal color for color blend by means of the drop down list box labeled 'Color blend from: to:':
Click on one of the scroll buttons with upward arrrows.
=> A box pops up to select a color from.

2. Change the scale (distribution) of used colors
To change the color scaling you can alternatively click 'Uniform' or 'Successively'.
=> A sample of the selected font will be shown in the bottom left corner.
Confirm your selection by clicking *OK*.

State | Centered

The item *Centered* in the *Initialize States* menu gives the cell matrix an onion-like shape with one color in the center surrounded by layers of other colors.

Click the State control button.
=> A menu pops up.
Click *Centered* in the *Initialize States* menu
=> You are offered the below dialog box.



Click either the button 'Minimum in the center' or let the default setting 'Maximum in the center' unchanged. Click in the white area on the right side if you want to change the granularity of colors and enter a value between 1 and 90. Click OK.
=> You will receive a cell matrix with one color in the center and other colors surrounding the center layer by layer.

This effect gets more visible with high Colors and Zet values.

Graphic window (STATE)

As the CARP program is the *formal* part of your cellular automaton model, so is the STATE window the *visual* and *apprehensible* part of it. It has to visualize the whole cell matrix and each cell state at a given time and in its historical development.

To make this possible the STATE window provides two kinds of representation of the cells' state:
- representation of the state of each cell by colors (graphic representation)
- representation of the state of each cell by decimal or hexadecimal numeric values (numeric representation).

Beyond, you can customize the appearance of this window in different ways by means of some of these buttons.

Overview on all functions:

A **Show / hide matrix button**
Switch to show or suppress display cell states by colors

B **Numeric state button**
Switch to display the state of each cell by figures

C **Color mapping button**
Displays the palette entry that is assigned to a single cell

D **Hexadecimal format button**
Switch to show numbers hexadecimal or decimal format

E **Magnifier button**
Switch to show or suppress border display

F **Color customizing button**
Changes the color that is assigned to a palette entry

G **Repaint button**
Paints the window again

H **State control button**
Opens the local menu: State (initialization and other functions)

I **Palette customizing button**
Opens the local menu: Palette (changing a whole palette)

J **Menu button**
Offers the same functions as provided by buttons

K Shows the number of executed rounds

L Shows the elapsed time

Note:
Many of these buttons have toggle switch logic. An activated toggle switch is indicated by a little red hook. Most of the functions executed by the above buttons can also be triggered by corresponding CARP procedures.

Show/hide matrix button

Normally, the state of each cell of your cellular automaton model is represented by a certain color depending on your current color settings. Switching off this color display is useful, if your model is very cpu time consuming and if you don't need to observe your model continously.

Click the above button to switch off color display.
The color display is switched off, the cell state window shows a grey area. Computation of each generation of the cellular automaton is continued and speeded up. The current generation is indicated by the round counters.

To return to the previous state click the above button again.
CAT will show the color reprensation of the matrix again.

Switching off color display assumes that the numeric state representation is disabled.

Corresponding CARP procedure:

Numeric state button

Your cellular automaton model may be represented in the state window in two kinds:

- graphic representation of the state of each cell by colors
- numeric representation of the state of each cell by decimal or hexadecimal numeric values.

To choose either form click the above button. The plane state window will switch from graphic to numeric representation and vice versa. The default setting is graphic representation.

To switch from decimal to hexadecimal number format use the hexadecimal format button.

Corresponding CARP procedure:

Color mapping button

Depending on your settings for defined colors and the optional use of the RGBPalette instruction the different states of the cell matrix are reflected by different color mappings. (Color mappings define the colors to be shown.)

Click the above button to switch on display of color mappings.
The values for the different color mappings will be shown in the cell matrix. To return to the previous state click the above button again.

This display may also be combined with the numeric state display. Mostly, these values will be identical, but for certain reasons they may differ.

Note:
If the window is too small or the XYSize for the matrix is to big to show the figures in the small cells, an according message is shown.
The color mappings may also be printed on your printer.

Related CARP procedure:
RGBBrush , RGBPalette

Hexadecimal format button

The above button provides switching from decimal to hexadecimal number format and vice versa.

Default setting is decimal format.

Magnifier button

The above button provides switching between two appearances of the cell matrix:
- showing the whole cell matrix with its border areas (cp. XYBound)
- showing the cell matrix without border areas

Default setting is display of the cell matrix with borders.

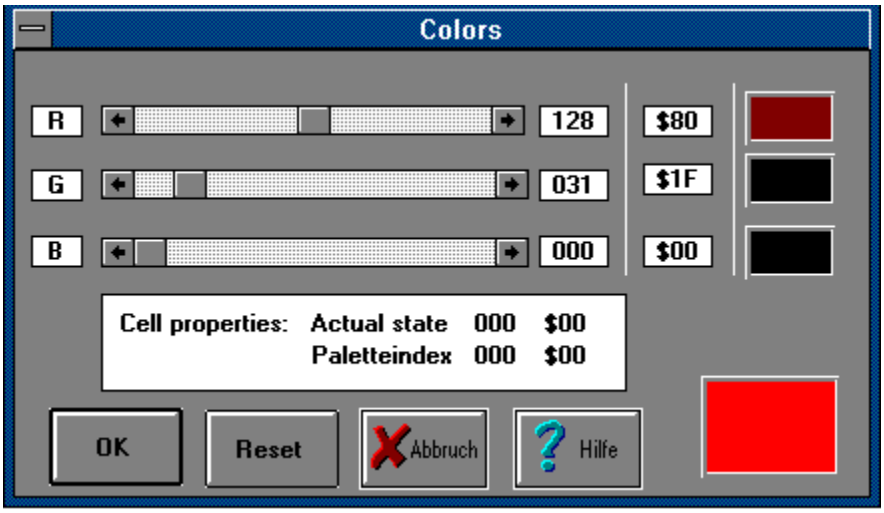
Clicking the above button enlarges the remaining cells more or less slightly. This button is effective regardless of color or numeric cell representation.

Corresponding CARP procedure: PlClipActive, PlClipAll

Color customizing button

The above button provides changing the color of a selected cell and all other cells of that kind.

Select a cell of the kind you want to change on the cell matrix. Click the above button. A dialog box with three sliders labeled "R"(ed), "G"(reen) and "R"(ed) will come up.



Drag one button of the red, green or blue sliders towards one end. The selected color will change heavily whereas the summirizing color panel in the right bottom corner will change slowly. This last panel represents the color the selected cell and all identical cells will adopt.

Click the OK button to confirm the new color selection or leave the dialog box by clicking CANCEL.

The color selection will override settings that are made by means of the RGBPalette instruction.

Corresponding CARP procedure:
RGBBrush , RGBPalette

Repaint button

The above button provides painting the active window again (Repaint). This is useful, if the active window is displayed uncompletely or with garbled colors.

Corresponding CARP procedure:
RePaint

State control button

Clicking the above button pops up a menu with these items:

item	short description
Initialize States	
Single Value 0	all cells initialized with value 0
Single Value 1	all cells initialized with value 1
Single Value n	all cells initialized with value n to be entered in a dialog box
Random Values	all cells initialized with random values. Range of values depends on Zet.
Centered	initializes the cell matrix with the highest value in the center descending to the lowest in the corners or vice versa. Range of values depends on Zet.
Upstairs	initializes the cell matrix with a stair-like pattern. Range of values depends on Zet
State	
Load	loads a previous saved state of a cell matrix (.CAS file).
Save	saves the current state of the cell matrix as .CAS file.
Save As	saves the current state of the cell matrix as .CAS file under a new name
Show	shows the current state of the cell matrix in a text winodw.
Print	prints the current state of the cell matrix (.CAS file).

CAT Help Index

The Index contains a list of all Help topics available for the Help Example. For information on how to use Help, press F1 or choose Using Help from the Help menu.

Color map

CAT starts with default values that provide to mark certain states of a cell by predefined colors immediately. On the other hand you have plenty of options for color manipulations that a summary may turn things clearer to you.

Color palette
CAT uses a palette of 230 colors that is structured like this:

map id	r-value	g-value	b-value
1	10	0	0
2	20	0	0
.....			
230	0	0	250

The default color palette starts with map id 0 for blue and ends with map id 230 for a light green. Each map id has assigned three values for the colors red, green and blue (socalled '**rgb triple**'). Each single setting for red, green or blue may vary from 0 to 255 (or $0 to $FF), where the defined color is mixed by the three component colors. (To get a impression of this use the Color customizing button)

The current color palette can be shown by means of the Palette | Show command. (There are also color values above 230, but these values are used by Windows and should not be manipulated by you. Otherwise, you may get garbled colors even in other applications.)

Color definition and manipulation: an introduction

Besides using reasonable default values to start modelling immediately, CAT allows customization of colors and assigning colors to cell states on three levels.

I. Palette customization
The color palette is a map of up to 230 colors that can
- be substituted by an existing color palette (file extension .CAP) by means of the Palette | Load command,
- be saved as .CAP file by means of the Palette | Save command or
- that can be newly created by use of the RGBPalette instruction.
Besides these facilities the palette can be manipulated by means of the Palette customizing button.

II. Color customization
A single color, however, can be created by means of the RGBBrush procedure or later be changed by means of the Color customizing button. Be careful using the DelBrush procedure unless you redefine a new color palette.

III. Assigning colors to cell states
The easiest thing is, if the values for Zet and Colors on the top of your CARP program are equal. Every state is then represented by a particular color. If one of the values is greater than the other, the range of states that are represented by the same color can be calculated as Zet DIV Colors. A sample: Zet = 240; Colors = 80; results in an interval of 3 different cell states represented by the same color. Which cell states correspond with a certain color can be easily observed if you activate the Numeric cell state button and the Color mapping button simultaneously.

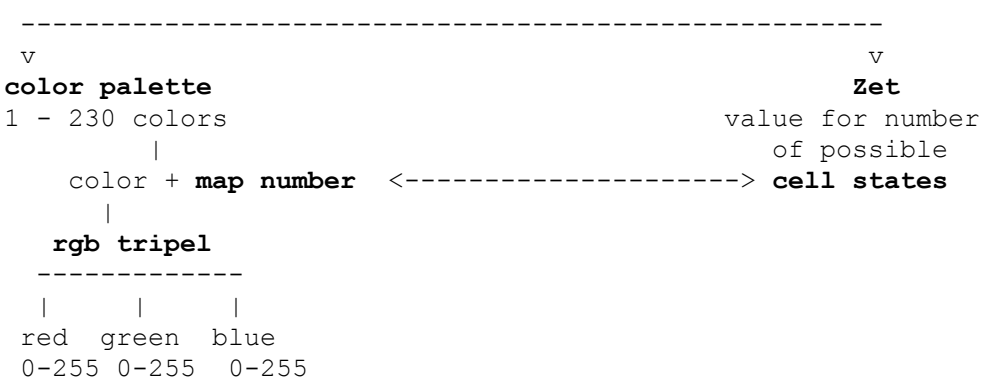


Figure: Relation between color palette, colors and cell states

As you can derive from the figure you assign a certain color to a particular cell state by its color mapping number. Each single red, green or blue value defines the component portion of this color to the merged color. Red set to 255, green to 0 and blue to 0 results for example in a deep red color.

Some samples with that you can get make experiments at your pc:

Sample a: Default color palette

```
(* program to demonstrate almost all available colors of the
  default color palette *)
```

```
RECIPE  XYSize  =  15;
        Zet     =  230;
        Colors   =  230;

EVENT E0;
     PARALLEL DO
     Self := GetX + (GetY * XYSize)
     OD;
     ShowPlane;
     END.
```

Sample b: Usage of the 'RGBBrush' procedure

```
(* program to demonstrate usage of 'RGBBrush *)
RECIPE  XYSize  =  15;
        Zet     =  2;
        Colors   = 2;

CONST Alive = 0;
      Dead = 1;

EVENT SetUp;
      RGBBrush (Dead, 0, 0, 255); (* assigning blue to Dead *)
      RGBBrush (Alive, 255, 0, 0); (* assigning red to Alive *)
      PlFillRandom (Alive, Dead);
      ShowPlane;
      END.
```

Sample c: Usage of the 'RGBPalette' procedure

```
(* program creates 10 colors and fills the STATE window
   by a random pattern  *)
ECIPE  XYSize  =  15;
        Zet     =  10;
        Colors  =  10;

CONST a_state = 0;
      b_state = 1;
      c_state = 2;
      d_state = 3;
      e_state = 4;
      f_state = 5;
      g_state = 6;
      h_state = 7;
      i_state = 8;
      k_state = 9;

EVENT SetUp;
      RGBPalette (Colors, 150, -20, 0, 20, 255, -20);
      PlFillRandom (a_state, k_state);
      ShowPlane;

     END.
```

Using 'RGBPalette', you may find it difficult to create 10 colors each differing suffiencently from another. If you find two colors too less differing, use the Color customizing button or use Shift + right mouse key to activate the Color dialog box for the corresponding cell.

Color palette

The color palette is a table of up to 230 entries. Each of these entries consists of an identifying number (color map) and three associated numbers representing the **portion** of the colors red, green and blue (a socalled RGB tripple) contained in merged 'summarized' color. The scale for each component color ranges from 0 to 255. A red value of 255 means for example means a deep red.

```
color mapping number (identifier)
|    red value
|     |  green value
|     |  |    blue value
V     V  V    V
000  000 000 000
001  010 000 000
002  020 000 000
003  030 000 000
004  040 000 000
005  050 000 000
006  060 000 000
007  070 000 000
008  080 000 000
009  090 000 000
```

Only those entries of the color palette actually defined by the RGBPalette or RGBBrush procedure overwrite existing entries of the old color palette. Those entries of the 230 accessible entries currently not defined remain unchanged.

You can display your current color palette on the screen or print it by means of the local menu.An example of such a printed color palette with 10 entries is shown above:

The number of lines in this example is the number of colors you use. If you modify this text, the palette you are using will be updated after you have closed this text window.

You may delete all of the text and write one or more lines in the above format. After closing the window only those colors will be updated.

Depending on the hexadecimal button figures are shown either as decimals or as hexadecimals.

Don't use color entries above 230! They are used by MS Windows and may impact your operating system or other Windows applications.

File Menu

The menu *File* collects a couple of items related to file handling and a menu item to quit the CAT environment.

New
Open
Save
Save As

Set Font
Print Text

Exit

Apart from the equivalent functions of the local menu the File menu functions have a global range. That means for example File | Save saves every file contribuing to a specific cellular automaton model, not only a particular file.

File Menu | New

The item *New* in the *File* menu provides an empty RECIPE window to write a new CARP program.

Click *New* in the *File* menu.

=> If there, a previous Recipe file (file extension .CAR) disappears from the window and a new empty RECIPE window is presented for writing a new CARP program. The corresponding Recipe file has the temporary file name NewFile.CAR until it will be saved the first time.

If the last .CAR file you worked with is not saved, you are asked whether you want to discard or save this file.

Compare also the equivalent, but local *New* function of the local menu.

File Menu | Open

The item *Open* in the *File* menu provides loading of an existing .CAT file as global workspace for a complete cellular automaton tool.

Click *Open* in the *File* menu.

=> A file selection box titled *Browse* comes up to choose a file name from. The file extension .CAT for a complete automaton tool model is preset, but may be changed by means of the control panel *List Files of Type.*

Compare also the simular, but local *Overload* function of the local menu.

File Menu | Save

The item *Save* in the *File* menu provides saving the current state of your whole cellular automaton model as .CAT file (workspace). However, this global *Save* not only saves the contents of the .CAT file, but of all files building up a CAT model (.CAR, .CAL, .CAS, .CAP, .CAT).

Click *Save* in the *File* menu.

=> The current model will be saved immediately. If your model has only the preset temporary name, you are prompted for a new name.

Compare also the equivalent, but local *Save* function of the local menu.

File Menu | Save As

The item *Save As* in the *File* menu provides saving the current state of your whole automaton tool model under a free selectable name (file extension .CAT).

Click *Save As* in the *File* menu
=> You are prompted for a new name with the preset file extension .CAT..
Type in a significant name and click *OK*.
=> The whole automaton tool is saved with its current state.

Compare also the equivalent, but local *Save As* function of the local menu.

File Menu | Set Font

The item *Set Font* in the *File* menu provides choosing an appropriate font for the text windows of CAT (LIST window, STATE window).

Click *Set Font* in the *File* menu
  => You are offered a dialog box to choose an appropiate font.
Click a font name.
  => A sample of the selected font will be shown in the bottom left corner.
Confirm your selection by clicking *OK*.

Note:
The selected font is only valid for for your current CAT model, not for the whole CAT environment.

Selecting a smaller font may be a means to have the whole CARP program present on your STATE window.

All fonts currently installed on your Windows machine are selectable.

Compare also the equivalent, but local *Set Font* function of the local menu.

File Menu | Print Text

The item *Print Text* in the *File* menu provides printing the contents of any CAT text window (LIST or STATE window).

Select *Print Text* in the *File* menu
=> You are offered the print dialog box.
Click *OK* to confirm printing.
=> The contents of the selected window will be printed on the preset printer.

Note:
To print the actual state of your STATE window (only text format) use the state control button.

To change the preset printer use the Setup option of the print dialog box.

The complete .CAR or .CAL file will be printed by *Print Text*.

Compare also the equivalent *Print* function of the local menu.

File Menu | Exit

The item *Exit* in the *File* menu provides leaving the whole CAT environment.

Click *Exit* in the *File* menu
=> You will return to the Windows environment.

Another way to leave CAT is to doubleclick the Close button in the upper left corner of your CAT window.

If any file of your current CAT model is not saved, you are prompted whether you want to save or discard the corresponding file.

Edit Menu

The *Edit* menu contains several items for text manipulation, particularly to move text to and from the Clipboard.

Command	Description
Undo	Undo last change
Cut	Moves selected text into Clipboard
Copy	Copies selected text into Clipboard
Paste	Inserts text from Clipboard
Delete	Deletes selected text without changing the clipboard
Clear	Deletes complete text of the current text window

Compare also the equivalent functions of the local menu.

Clipboard

A temporary storage location used to transfer data between applications, different text windows or on editing a single text (Cut, Paste).

Edit Menu | Undo

The item *Undo* in the *Edit* menu makes your last action undone.

Click *Undo* in the *Edit* menu
=> The state before you entered your last command will be restored.

CAT can undo the following edit commands:

Inserting a character
Deleting a character
Cutting a marked text portion
Pasting a marked text portion

A shortcut for *Edit / Undo* is to press CTRL+ z. Only the **last** action can be undone.

Compare also the equivalent *Undo* function of the local menu.

Edit Menu | Cut

The item *Cut* in the *Edit* deletes marked text portions and keeps them in the Clipboard for further actions.

Mark a part of the text by means of the mouse.
=> The corresponding text will be displayed inverse.
Click *Cut* in the *Edit* menu
=> The marked text will be deleted and stored temporarily in the Clipboard.

You can then Paste the contents of the clipboard at another position in the document, into a new document, or into another Windows application.

Shortcut is Crtl + x.

Compare also the equivalent *Cut* function of the local menu.

Edit Menu | Copy

The item *Copy* in the *Edit* menu copies marked text portions temporarily in the Clipboard without changing the text.

Mark a part of the text by means of the mouse.
=> The corresponding text will be displayed inverse.
Click *Copy* in the *Edit* menu
=> The marked text will be stored at the Clipboard. The inverse display of the text and the text itself remains unchanged.

You can then paste the contents of the clipboard at at another position in the document, into a new document or into another Windows application.

Shortcut is Crtl + c.

Compare also the equivalent *Copy* function of the local menu.

Edit Menu | Paste

The item *Paste* in the *Edit* menu inserts previously saved text from the Clipboard at another position in the document, into a new document or into another Windows application.

Click *Paste* in the *Edit* menu
=> Text contained in the Clipboard will be inserted at the current insertion point.

Precondition for using *Paste* is that you have stored any text in the clipboard. Otherwise, the *Paste* menu item is dimmed.

Shortcut is Crtl + v.

Compare also the equivalent *Paste* function of the local menu.

Edit Menu | Delete

The item *Delete* in the *File* menu deletes the marked text or a single character.

Click *Delete* in the *File* menu
=> The marked text (if any) or the character to the right of the insertion point will be deleted.

To undo the deleting use Undo.

Shortcut is Del.

Compare also the equivalent *Delete* function of the local menu.

Edit Menu | Clear All

The item *Clear* in the *Edit* menu removes the complete document without changing the contents of the clipboard.

Click *Clear* in the *Edit* menu
=> You will receive an empty text window.

Be careful with this command: there is **no** Undo**.**

Shortcut is Crtl + Del.

Search Menu

The *Search* menu contains functions to search and exchange text patterns in a text window.

Command	Description
Find	Specify a pattern and search for it
Replace	Specify one pattern and replace it by another
Next	Find next occurence

Search Menu | Find

The item *Find* in the *Search* menu searches for a pattern in a text window.

Click *Find* in the *Search* menu.
=> A dialog box prompts you for the pattern you want to search for.
Type in a text pattern and click *Find Next*.
=> In case of success, the text pattern will be highlighted in the text window.
Otherwise, you will get a message box '"pattern" not found'.

You can specify these options:

* Find — Type the text you want to find.
* Match Upper/Lower Case — Select this box to match the upper and lower case exactly.
* Match Whole Word Only — Select this box to find only entire words.
* Forward — Search forward in the document starting at the insertion point.
* Backward — Search backward in the document starting at the insertion point.

Search Menu | Replace

The item *Replace* in the *Search* menu searches for a certain pattern and substitutes it by a second one.

Click *Replace* in the *Search* menu.
=> A dialog box prompts you for the pattern you want to search for and the substitute pattern.
Type in the two patterns and click *Replace*.
=> In case of success, the next occurence of the text pattern will be replaced by the second pattern. In case of no success, you will get a message box '"pattern" not found'.

You can specify the following options:

* Replace with	Type the text you want to insert in place of the found text.
* Find Next	Only finds the next occurence of the search pattern.
* Replace All	Exchanges all text patterns found without any further dialog.
* Close	Closes the Replace dialog box without any action.
* Match case	Select this box to match the upper and lower case exactly.
* Match Whole Word Only	Select this box to find only entire words..

Search Menu | Next

The item *Next* in the *Search* menu searches for the next occurence of the pattern prreviously entered, without opening the Find dialog box again.

Click *Next* in the *Search* menu.

=> In case of success, the next occurence of the text pattern will be highlighted in the text window. In case of no success, you will get a message box '"pattern" not found'.

The shortcut is F3.

Window Menu

The *Window* menu contains functions to organize the CAT windows (RECIPE, STATE, LIST) according to different modes and purposes..

Command	Description
Tile	Puts all three windows side by side with equal space for each
Cascade	Resizes and layers the three windows
Arrange Icons	Arranges window icons displaced on the workspace

Additionally, the Window menu offers by a corresponding menu item to bring up any open CAT window (RECIPE, STATE, LIST, PALETTE (CAP) or PLANE window (CAS)).

Compare also the *Hide Window* function of the local menu.

Window Menu | Tile

The item *Tile* in the *Window* menu arranges all of the open windows on the screen so that a portion of each windows can be seen.

Click *Tile* in the *Window* menu.

=> The three CAT windows (LIST, RECIPE, STATE) are arranged side by side so that each window shares the same space on the CAT main window.

The shortcut is Alt w + t.

Window Menu | Cascade

The item *Cascade* in the *Window* menu arranges all open windows in a stack one window overlapping the other. When this is done the title bar of each window is visible so that the window can be made active by clicking on the title bar

Click *Cascade* in the *Window* menu.
=> The three windows will be put on a stack so that some part of each window is visible.

The shortcut is ALT w + c.

Window Menu | Arrange Icons

The item *Arrange Icons* in the *Window* menu places document window icons (if any) along the bottom of the window in a row left to right.

Click *Arrange Icons* in the *Window* menu.

=> CAT windows that have been minimized appear at the bottom of the screen aligned.

The shortcut is ALT w + i.

Main window

The CAT main window offers several pulldown menues ('File', 'Edit', 'Search', 'Window'), access to the help system you are now in and a toolbar of buttons to control your CARP program.

These control buttons are:

Compile	Compiles the program
Setup	Executes Event Setup
Event 0	Executes Event 0 in single step mode
Event 1	Executes Event 1 in single step mode
Event 2	Executes Event 2 in single step mode
Event 3	Executes Event 3 in single step mode
Event 4	Executes Event 4 in single step mode
Event 5	Executes Event 5 in single step mode

(bottom line of buttons)

Event 0 + arrow	Executes Event 0 in run mode
Event 1 + arrow	Executes Event 1 in run mode
Event 2 + arrow	Executes Event 2 in run mode
Event 3 + arrow	Executes Event 3 in run mode
Event 4 + arrow	Executes Event 4 in run mode
Event 5 + arrow	Executes Event 5 in run mode
Stop	Stops execution
Done	Executes Event Done

Note:
In a large CAT window, the above buttons will appear aligned.

Text Window

A text window displays and offers in some cases to edit text. Text windows are controlled by keyboard commands and keyboard input in contrast to graphic windows. Information on the current state of a text window is shown in the status line. There you find the a counter referring the current position (column : row) of the text cursor and a label "Modified", if a previously saved text has changed.

All text windows offer a local menu with these items accessible by clicking the right mouse button or by holding the shift key pressed while clicking the right mouse key.

Text windows are the LIST and RECIPE window, but also the color palette window (CAP), the CAT state window (CAS) or any other CAT file window.

Local menu

There are two kinds of local menus offered on any CAT text window (LIST, STATE, PALETTE, PLANE...):

- local menu invoked by clicking the right mouse button

This file related local menu differs from the File menu in the main window, its actions only refer to the currently selected text window.

New	Creates a new blank text window with NewFile.xxx file name.
Overload	Loads a text file into the current text window disregarding file extensions. Current text file will be overwritten.
Save	Saves current text window. If no user defined name is available, you are prompted for a file name.
Save As	Saves current text file under a free selectable name.
Insert	Inserts a file at current cursor position without overwriting old text file.
Set Font	Sets font for the current text window. Other text windows keep their old fonts.
Print	Prints current text window on the standard printer.
Hide Window	Puts the current text window on the last place of the window stack.

- local menu invoked by clicking the right mouse button plus shift key

The edit related local menu behaves the same like the Edit menu of the main window

Undo	Reverts the last editing function like Copy, Cut or Delete.
Cut	Deletes a marked text portion and saves it in the clipboard.
Copy	Copies a marked text portion and saves it in the clipboard.
Paste	Takes a text portion saved it in the clipboard and inserts it at the current cursor position
Delete	Deletes a marked text portion or one key left to the current cursor position.

On the non-editable LIST window only the Copy item is applicable.

RECIPE window

The RECIPE window contains the CARP program you write to program a particular cellular automaton model. The 'RECIPE window is a editable text window. Text operations like Cut, Copy and Paste can be executed as in Windows.

Some more functions are available if you use the local menu.

Window functions are as in Windows except closing. The RECIPE window can only be covered by other CAT windows or be iconified but not closed.

LIST window

The LIST window is the console window on which your CARP program, the CAT compiler or the CAT runtime system writes its output. The LIST window is a non-editable text window.

Some more functions are available if you use the local menu.

Window functions are as in Windows except closing. The LIST window can only be covered by other CAT windows or can be iconified, but not closed.

STATE window

The STATE window is the most important user interface to visualize and control your current CARP program. In contrast to the text windows LIST and RECIPE it is a graphic window (click keyword to get informations on its handling).

It comprises

- facilities to control the execution of the CARP program
- status information (time, rounds) about the execution of the CARP program
- facilities to switch between the color or the figure representation of the cells' state various facilities to customize the display

Event 0 - 5

Executes Event 0 for one time


Executes Event 0 until STOP

Clicking one of the buttons from E0 to E5 performs **one** execution of the code of the according event. This mode is advisable for test purposes or to control a certain section of your CAT model's development very closely.

Clicking one of the circleform buttons below the buttons E0 to E5 performs execution of the code of the according event in run mode. The corresponding event is only stopped by clicking the STOP button, by runtime limits of the PD version of CAT or by limits of the available memory.

Compile button

Clicking this button compiles the current CARP program of the RECIPE window.

If the compiler finds any errors, you are prompted whether you want to continue compilation or to cancel compilation immediately. Error messages can afterwards be found in the LIST window.

Compilation is done implicetely if you click any event button including the Setup button.

Setup button

Clicking this button executes the code of the special event SetUp.

CARP instructions contained in this event should make global settings as for example for the size of the matrix, the maximal evaluated neighborhood (XYBound), the color palette or the topology.

This event is implicitely executed if you trigger any other event from E0 to E5.

Done button

Sorry! Not yet implemented.

Stop button

Clicking this button executes the code of the special event Stop.

Any other event currently executed will be stopped then. Be patient, if the actions on the screen don't stop immediately. The previously activated event has to be completed, before the Stop event can take effect.

Files of the CAT environment

A couple of files provide different functions inside the CAT environment. They can easily be identified by their file extension e.g. filename.**CAP.** The first two letters of the file extension signify that all these files belong to CAT, the third, however, is related to their function (In order of importance).

name	resolved acronyms	function
CAR	**CA**T **R**ecipe file	contains your CARP programming instructions for your cellular automaton model.
CAL	**CA**T **L**ist file	contains messages written either by WRITE instructions of your CARP program or by the compiler or the runtime system.
CAS	**CA**T **S**tatus file	contains the last state of your CAT state window in form of figures representing each cell. If you want to keep a certain selected state, use the Save as function to prevent the elected CAS file from being overwritten.
CAP	**CA**T **P**alette file	contains as decimal or hexadecimal values RGB tripels with an identifying mapping number that characterize a certain color. Not needed for normal use. Related CARP procedure RGBPalette.
CAT	**CAT** environment file	contains settings of the whole CAT environment specific to a certain cellular automaton model. May be used in conjunction with the Save as function as snapshot facility for important states of your model. Resembles structure of a MS Windows .INI file.

Remark:
All files are printable.

Copyright

License

Limited Warranty on Media and Manual
GMD gives no warranty, explicit or implied, with respect to this software, its quality, performance or suitability for a particular purpose. This software and manual are supplied "as is", and you, the user, are assuming the entire risk as to its quality and performance.

In no way will GMD be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect in the software or its documentation. In particular, GMD shall have no liability for any damage to programs or data used with this product, including the cost of recovering or repairing such programs or data.

The warranty and remedies set forth above are exclusive and in lieu of all others, oral or written, expressed or implied.

Hardware and software requirements

CAT can be run on any 100 % compatible PC with an 386- or 486 CPU with MS Windows 3.1, MS Windows / NT and OS/2 2.1 in the windows box.

4 Megabytes RAM are the absolute minimum required, 8, better 12 or 16 Megabytes advisable. High CPU speed (> 25 MHz) and an accelerated (S)VGA card are a good means to speed up CAT's graphic output.

Restrictions of the Public Domain version

There are restrictions if you run the public domain version of CAT: Clicking the Run button in your CAT environment admits a certain number of executions of the event´s code. After that, a message will come up "Runtime limit exceeded".
You then have to click the Run button again.

The regular version of CAT, however, doesn't contain this restriction.

Where to get a printed manual

A printed manual with the latest version of CAT is available at

German Nation Center for Computer Science (GMD)

PD service departement, ISAR

P.B. 13 16

D-53731 Sankt Augustin

Germany

for DM 35.-. It contains the last PD version of CAT, a couple of sample programs and a reference guide on the CARP language.

Goals of CAT

The Cellular Automaton Tool (CAT) is a tool to build and visualize cellular automatons in an easy-to-use way. CAT is headed to point out two concepts:
- the concept of cellular automatons
- the concept of parallel computing. Then, each cell stands for a CPU of a parallel computer that takes a part of a common task.

CAT can be used to illustrate the concepts of cellular automatons and parallel computing at introductory courses on one of these items or for self-instruction. When doing self-instruction, it is advisable to consult the printed CAT manual and a tutorial book.

Properties of CAT

The Cellular Automaton Tool (CAT) is a tool

- to make cellular automaton models and
- to learn about the parallel programming paradigm taking each cell as a CPU.

CAT offers:

- facilities to visualize the state of each cell of a cellular automaton with options for output on colored cells or figures representing the cell state,
- an easy to learn and pascal-like programming language to program cellular automatons named CARP.,
- many facilities to customize the appearance of your cellular automaton model,
- options to control the appearance of your cellular automaton model interactively by a quite intuitive user interface or by CARP instructions,
- an online help system you are in now and
- a printed manual.

CAT runs under MS Windows 3.1, MS Windows / NT and OS/2 2.1 in the windows box.

Local menu | New

Creates a new blank text window.

Click the right mouse button on a text window.
=> A menu pops up.
Click the item *New.*
=> You will get a new empty text window, any text of the text window will be deleted..

Note:
If you want to undo this action, click the local menu item overload to load the previous contents of your text window.

Local menu | Overload

Exchanges the contents of the current text window by a selected text file disregarding file extensions.

Click the right mouse button on a text window.
=> A menu pops up.
Select the item *Overload.*
=> You will get a dialog box to select a file name from.
Doubleclick any file name in the left hand File box and click OK to confirm your selection.
=> The text of the current text window will be substituted by the text of the selected file.

Note:
You may also overload e.g. a .CAP file by a .CAS file or even a non-text file. Keep in mind that this might damage your CAT environment.

Local menu | Save

Saves the actual state of the current text window.

Click the right mouse button on a text window.
=> A menu pops up.
Select the item *Save*.
=> The current text window will be saved under the prefixed name.

If there is no name assigned to the current text window, you will be prompted for a name (cp. Save As).

Local menu | Save As

Saves the actual state of the current text window under a user defined name.

Click the right mouse button on a text window.
=> A menu pops up.
Select the item *Save as.*
=> You will get a file dialog box.
Type in a meaningful name and click OK.
=> The current color palette will be saved under the chosen name.

If there is no name assigned to the current text window, you will be prompted for a name (cp. Save As).

Local menu | Insert File

Inserts a file at the current cursor position in the current text window without deleting the old text.

Click the right mouse button on a text window.
=> A menu pops up.
Select the item *Insert File.*
=> You will get a file dialog box.
Select a name in the left hand File box and click OK.
=> The selected text file will be inserted into your current text file at the current cursor position.

Be careful not to insert non-text files!

Local menu | Set Font

Changes the preset font of the current text window to a user defined font.

Click the right mouse button on a text window.
=> A menu pops up.
Select the item *Set Font.*
=> You will get a dialog box to select a font name from.
Doubleclick any font name in the left hand Font box.
=> The text in the Sample box changes to reflect the font you select.
Click OK to confirm your selection.
=> The font will be changed for the current text window.

Note:
The selected font is only valid for your current text window, not for the whole CAT environment.

All fonts currently installed on your Windows computer are selectable.

You may also change the font style and font size.

Local menu | Hide Window

Hides the current text window and makes the last activated window visible.

Click the right mouse button on a text window.
=> A menu pops up.
Select the item *Hide Window.*
=> The current text window will be hidden by other CAT windows and the last activated CAT window will be shown.

Concepts of CARP

CARP is an acronym and stands for **C**ellul**ar** Automaton **P**rogramming Language. It is the programming interface by which you can define your cellular automaton models within the CAT environment.

CARP is a high-level language related to Modula or Pascal with some additional constructs for parallel programming, e.g. the PARALLEL DO construct.

CARP procedures may define the visible matrix in the STATE window as main user interface of your cellular automaton model in different ways:
- Via customization procedures, you may define the size of the matrix and the size of the evaluated neighborhood of each cell. Beyond that, you may change settings for the color representation of each cell, for color palettes etc.
- Via the procedure Self, you may change the state of all cells simultaneously. You can combine a Self procedure with an evaluation of the state of the cell´s neighbors (referred to by the construct REF) within the limits of the general XYBound values. Keep in mind that this evaluation is really done *parallelly.* Example:

```
IF (north_east = 1)
    THEN Self := 0
FI
```

- Via control structures such as IF..THEN..ELSE..FI or WHILE..DO..OD, you may transform your algorithm to CARP programs like every ordinary programming language would allow.
- Via WRITE procedures, you may write certain important values of variables to the LIST window or a file, which may be used for controlling your automaton model.
- Via the SHL/SHR operators, you are able to mask bits and hide there informations about the previous state of a cell. You may thus take a historical perspective on your cellular automaton model.

The main interface of any cellular automaton model defined by its CARP program is the colored cell matrix of the STATE window. A certain color of a cell corresponds to a certain cell state, which may be expressed by integers as well.

That is why CARP knows integers as the only data types and, in a restricted way, bit data types.

The CARP language is case-sensitive. Thus, you have to write the keywords in the same way as they appear in the syntax descriptions.

CARP Overview

Assumption:
This online description of CARP assumes that you are familiar with the basic concepts of programming.

CARP is an acronym and stands for **C**ellul**ar** Automaton **P**rogramming Language. This modula- or pascal-like language is the tool by which you can define your cellular automaton models within the CAT environment.

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 *)
```

Program structure (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, refered 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, as essential elements, program parts that are called EVENT. A template for a program may look as follows:

Example

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

EVENT SetUp;
...

EVENT E0;
...

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

EVENT

Syntax

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

Remarks

An event is the program code 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 an identifier from "E0" to "E5" and additionally the special event SetUp.

Example

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

EVENT E1;
...
```

Declaration of constants and variables

Before any event declaration, you may declare variables (VAR), constants (CONST), user-defined procedures (PROC) or referenced cells (REF) in the head part of your CARP program.

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 (e.g. Self [Operator] [Operand]) or write access (Self := expression).

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 ('resurrection' rule), 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;
     ShowPlane
OD;
```

Control structures

Control structures provide to execute certain program parts repeatedly or depending on Boolean expressions.

CAT provides these control structures:

IF .. THEN .. ELSE .. FI
WHILE ..DO OD
REPEAT .. UNTIL
PARALLEL DO
FOR .. TO .. BY .. DO .. OD

Brake

Comparative operators

CAT provides the usual comparative operators known from e.g. Pascal.

Operator		Operand types	Result type
=	equal to	variables, constants	Boolean
<>	unequal	variables, constants	Boolean
<	less than	variables, constants	Boolean
>	greater than	variables, constants	Boolean
<=	less than or equal to	variables, constants	Boolean
>=	greater than or equal to	variables, constants	Boolean

Operators

Operators connect different operands and build expressions or compound instructions. CAT has four kinds of operators:

Arithmetic operators
Logic operators
Comparative operators
Bit operators

Any8Sum

Syntax

```
Any8Sum (op1, op2, op3, op4, op5, op6, op7, op8)
```

Remarks

Returns the sum of eight operands. As operands are allowed constants, variables, referenced cells, Self or procedures returning integer. All 8 parameters must be present. The same parameter may occur repeatedly.

Example

```
EVENT E0;
a := Any8Sum (neigh_l, neigh_r, neigh_t, neigh_b, neigh_tl, neigh_tr,
neigh_bl, neigh_br);
```

Customization procedures

Topology customization
There are five different topologies, by which you can direct how cells situated on the edges of the matrix behave. The default topology is RingForm. Note the interference of topology procedures with the XYBound setting.

- Sheetform
- BarrelForm
- PipeForm
- RingForm
- PillowForm

Matrix customization
Matrix customization procedures allow to define which part of the matrix is to be shown and provides different kinds of initialization.

- PlClipActive
- PlClipAll
- PlClipXY
- PlFillUni
- PlFillUpstairs
- PlFillRandom

Color customization
Color customization procedures provide different kinds of defining whole color palettes or single colors for identifying a certain kind of cells.

- DelBrushes
- RGBBrush
- RGBPalette

Compiler and runtime messages

During the compilation or execution of your CARP program, errors found or extraordinary events happening during program execution are reported to the LIST window. Two kinds of messages may occur:

Compiler error Messages
Runtime messages

Alphabetical index on CARP-related items

Usage hints

Assumption:
This description assumes that you are familiar with the basic concepts of programming.

CARP is an acronym and stands for **C**ellul**ar** Automaton **P**rogramming Language. This modula- or pascal-like language is the tool by which you can define your cellular automaton models within the CAT environment.

A good way to learn about this language could be to make a CARP program available on the screen or as a printed sheet, to distinguish between known and unknown matters and to have a look on those keywords you want to get more information for.

If you want to get a more structured view on CARP, use at least the keywords RECIPE, Events, Self, REF, VAR, CONST, Control structures and Predefined procedures. Whenever necessary, jump to those keywords that look somehow mysterious to you.

More advanced things are dealt with in Customization procedures and User-definable procedures.

If you want to learn about a specific CARP instruction, use the alphabetical index to jump to the corresponding explanation.

Other items of possible interest:

Declarations	Identifiers
Logic Operators	Arithmetic Operators
Error Messages	
Alphabetical index on CARP instructions	

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;
```

Logic Operators

CAT provides the usual Boolean operators for programming. AND, OR and XOR are operators with two operands, NOT, however, has only one operand. NOT binds its operand stronger than AND, OR and XOR, if operands of these two types are used side by side in one expression.

Operator	Operation	Operand types	Result type
NOT	negation	Boolean	Boolean
AND	logical and	Boolean	Boolean
OR	logical or	Boolean	Boolean
XOR	logical xor	Boolean	Boolean

Operand 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;
```

Operand 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;
```

Operand 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;
```

Operand 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;
```

Operand +

Syntax

```
(operand) + (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;
```

Operand -

Syntax

```
(operand) - (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;
```

Operator %

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 implicidly.

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;
```

Operator *

Syntax

```
(operand) * (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;
```

Operator MOD

Syntax

```
(operand) MOD (operand)
```

Remarks

The MOD operator devides 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;
```

Operator DIV

Syntax

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

Remarks

The DIV operator devides 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;
```

Operator 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 explicedly defined binary digits. In the following sample b returns the value 3 both times.

```
a := %110; (* 6 *)
b := a SHR 1;
WRITE ('',b);
```

OR

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

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.
```

Operator 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 explicedly defined binary digits. In the following sample the variable b returns the value 12 both times.

Example

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

OR

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

Operator INV

Syntax

```
INV (operand)
```

Remarks

The INV operator has an integer or bit operand and converts all its 0s to 1s and all 1s to 0s. 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.
```

Arithmetic Operators

CAT provides four operators for the fundamental operations of arithmetic and the modulo operator MOD. All operations and operators deal with and create integer values.

Operator	Operation	Operand types	Result type
+	Addition	integer type	integer type
-	Subtraction	integer type	integer type
*	Multiplication	integer type	integer type
DIV	Integer division	integer type	integer type
MOD	Remainder	integer type	integer type

Bit operators

CAT provides four operators for the bit handling. Integers are hereby regarded as hexadecimal values if they are introduced by a % operator. If you want to use these particular operators and instructions, you should be familiar with the basic rules of assembler programming.

Operator	Operation	Operand types	Result type
SHL	shift left	integer type	integer type (may be interpreted as bit sample)
SHR	shift right	integer type	integer type (may be interpreted as bit sample)
INV	invert	integer type	integer type (may be interpreted as bit sample)
%	turn to bit	0 and 1	bit pattern

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 implicedly 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;
```

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 executed at all.

Example

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

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 semicolon.

Example

```
x := 1;

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

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;
```

BEGIN ... END

Syntax

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

Remarks

Instructions bracketed by the keywords BEGIN and END may be used as an additonal 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;
```

CONST

Syntax

```
CONST
  identifier = expression;
  ...
  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 the WRITE procedure with a string parameter.

Examples

```
(* Constant Declarations *)
CONST
  limit   = 65000;
  KeyCode  = 943762;
```

VAR
Variable declarations

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;
```

User-definable procedures

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 prodecure that exchanges data with the CARP program it is in 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
                 procedure returning the value itself *)
BEGIN
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;
```

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 devided by a comma, strings must be included by ' (apostrophe).

Several facilities for formatting the output are provided:

[(var):n]	If the contents of the variable or the constanthas less than n digits, the output is indended accordingly.
''	Causes at the end or beginning of the parameter list a carriage return / linefeed (CR/LF) at the end or beginning of the output.

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;
```

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 referred 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.
```

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.

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

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;
```

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 YSize declaration.

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

Example

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

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;
```

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 PlClipAll procedure is used or at an according 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;
```

Zet

Syntax

```
Zet = n;
```

Remarks

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

Example

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

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;
```

ShowCell

Syntax

```
ShowCell( n );
```

Remarks

This procedure shows 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);
```

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;
```

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);
```

RePaint

Syntax

```
RePaint;
```

Remarks

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

Example

```
- - -
```

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;
```

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 portion of the whole cell matrix counted from its center with the size x and y. Thus, you focus on an area of special interest.

Example

```
EVENT E4;
PlFillRandom (1,4);
WinClipXY ( 5, 5);
```

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);
```

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);
```

PlClipXY

Syntax

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

Remarks

Shows a portion of the whole cell matrix counted from its center with the size x and y. Thus, you can focus on an area of special interest.

Example

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

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 *)
```

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 be used also 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 = 34;

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

Predefined procedures

CAT provides predefined procedures that differ not at least on the level of global or local range.

GLOBAL EFFECT

Evaluation and global settings

ParallelMethod

Topology-related procedures

Sheetform	BarrelForm
PillowForm	RingForm
PipeForm	

Matrix-related procedures

PlClipActive	WinClipActive
PlClipAll	WinClipAll
PlClipXY	SetLattice
PlFillUni	
PlFillUpstairs	ShowPlane
PlFillRandom	

Color-related procedures (global)

DelBrushes
RGBPalette
RePaint

LOCAL EFFECT

Color-related procedures (local)

RGBBrush

Control procedures

ShowKind	WrMCaps
ShowCell	WrDCaps
Beep	WrPPars
WRITE	

Cell-related procedures:

MooreSum	GetX
NeumannSum	GetY
Any8Sum	**:=**
Random	Randomize

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 corrrected 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 corrrected 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 corrrected 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 , in which you may select certain colors. You will find more informations about this topic at Color definition and manipulation or Color palette.

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 part 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);
```

SetLattice

Syntax

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

Remarks

Returns a lattice pattern 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);
```

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 : ');
            WRITE (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;
```

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;
```

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
```

Topology BarrelForm (c = copied cell)

Example

```
EVENT SetUp;
BarrelForm;
PlClipActive;
ShowPlane;
```

PipeForm

Syntax

```
PipeForm;
```

Remarks

The topology PipeForm forms a virtually pipeshaped matrix (tube), i.e. the top and bottom edges of the cell matrix are mutually copied to the 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
c c c c c c c
```

Topology PipeForm (c = copied cell)

Example

```
EVENT SetUp;
PipeForm;
PlClipActive;
ShowPlane;
```

RingForm

Syntax

```
RingForm;
```

Remarks

The topology RingForm forms a virtual endless matrix connecting at first two edges and then the edges of the build up pipe. This body is also known as thorus.

The RingForm topology is the **default setting** and may therefore 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 c c c c c c
```

Topology RingForm (c = copied cell)

Example

```
EVENT SetUp;
RingForm;
PlClipActive;
ShowPlane;
```

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;
```

ParallelMethod

Syntax

```
ParallelMethod;
```

Remarks

Is now the default method and doesn't need to be particularly defined.

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

Example

```
- - -
```

PlFillUni

Syntax

```
PlFillUni ( n );
```

Remarks

Gives the whole cell matrix a uniform color, which is defined by the n parameter and its according color palette entry.

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

Example

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

PlFillUpStairs

Syntax

```
PlFillUpStairs ( Lo, Hi, By );
```

Remarks

The procedure PlFillUpStairs creates a stair-like shape in the cell matrix. Thereby, the parameter *Lo* gives the lower colormapping value , *Hi* the higher 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;
   PlClipAll;
   PlFillUpStairs (2, 20, 4);
   ShowPlane;
```

PlFillRandom

Syntax

```
PlFillRandom ( Lo, Hi );
```

Remarks

The procedure PlFillRandom initializes the cell matrix by random values ranging from parameter Lo to Hi.

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;
```

WrPPars

Syntax

```
WrPPars;
```

Remarks

Acronym for 'Write Plane Parameters'. The according 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;
```

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;
```

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;
```

Brake

Syntax

```
Brake;
```

Remarks

Provides leaving a loop before the execution of all instructions is completed. Jumps to the end of a loop are only possible inside the current loop.

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;
```

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;
```

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;
```

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 result 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 ...
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;
```

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;
```

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;
```

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;
```

:= (assignment procedure)

Syntax

```
[VAR] identifier | Self := expression;
```

Remarks

The assignement 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;
```

Compiler messages

ERROR 110 : expression cannot begin with this Symbol
ERROR 140 : unknown identifier
ERROR 351 : '=' expected
ERROR 110 : unknow character or symbol
ERROR 140 : no source
ERROR 351 : 'end of program, period expected
ERROR 110 : end of comment "*)" without begin "(*"
ERROR 110 : 'too many "(*"; comment not closed '
ERROR 110 : 'too many digits in number '
ERROR 110 : 'number too large '
ERROR 110 : 'wrong chars in number '
ERROR 110 : 'too many digits in hex number '
ERROR 110 : 'hex number too large '
ERROR 110 : 'wrong chars in hex number '
ERROR 110 : 'too many digits in binary number '
ERROR 110 : 'binary number too large '
ERROR 110 : 'wrong chars in binary number '
ERROR 110 : 'identifier too long '
ERROR 110 : 'string too long '
ERROR 110 : 'expression expected '
ERROR 110 : 'factor expected '
ERROR 110 : 'cannot access procedure, within expression '
ERROR 110 : 'cannot access this object, within expression '
ERROR 110 : '")" expected '
ERROR 110 : an expression cannot begin with this Symbol '
ERROR 110 : 'expression cannot begin with this Symbol '
ERROR 110 : 'cannot evaluate this object '
ERROR 110 : '")" expected '
ERROR 110 : 'factor cannot begin with this Symbol '
ERROR 110 : 'assembler program too long '
ERROR 110 : 'number too large '
ERROR 110 : 'too many identifiers '
ERROR 110 : 'X-parameter out of range '
ERROR 110 : 'Y-parameter out of range '
ERROR 110 : 'duplicate identifier '
ERROR 110 : 'unknown identifier *** '
ERROR 110 : 'wrong symbol in Statement '
ERROR 110 : '";" expected '
ERROR 110 : 'END expected '
ERROR 110 : 'cannot assign, read only object '
ERROR 110 : '":=" expected '
ERROR 110 : 'only assignment to "Self" is feasible '
ERROR 110 : 'statement expected '
ERROR 110 : 'FOR variable expected '
ERROR 110 : '":=" expected '
ERROR 110 : 'TO expected '
ERROR 110 : '(FOR) DO expected '
ERROR 110 : '(FOR) OD expected '
ERROR 110 : 'FOR_identifier do not match '
ERROR 110 : 'EVENT identifier expected '
ERROR 110 : 'duplicate EVENT declaration ';

ERROR 110 : '";" expected '
ERROR 110 : 'END. of RECIPE expected '
ERROR 110 : 'DO expected '
ERROR 110 : 'OD expected '
ERROR 110 : 'UNTIL expected '
ERROR 110 : 'THEN expected '
ERROR 110 : 'ELSE or FI expected '
ERROR 110 : '"(" expected '
ERROR 110 : '")" expected '
ERROR 110 : 'too many actual parameters '
ERROR 110 : '"," or ")" expected '
ERROR 110 : '")" expected '
ERROR 110 : 'not enough actual parameters '
ERROR 110 : 'nested FORALL !!! '
ERROR 110 : '(FORALL) OD or expression expected '
ERROR 110 : '(FORALL) OD expected '
ERROR 110 : 'identifier expected, after CONST '
ERROR 110 : '"=" expected, after CONST identifier '
ERROR 110 : 'semicolon expected (in declaration) '
ERROR 110 : 'VAR_identifier expected '
ERROR 110 : 'REF_identifier expected '
ERROR 110 : '"[" expected, after REF identifier '
ERROR 110 : '"," expected ');
ERROR 110 : '"]" expected, in REF declaration '
ERROR 110 : 'identifier expected, after VAR '
ERROR 110 : '")" expected '
ERROR 110 : 'PROC expected '
ERROR 110 : 'maximum depth of block nesting exceeded '
ERROR 110 : 'cannot declare REF"s in procedure '
ERROR 110 : 'identifier expected, after PROC '
ERROR 110 : 'semicolon expected '
ERROR 110 : '"RECIPE" expected '
ERROR 110 : '"=" expected '
ERROR 110 : '";" expected '
ERROR 110 : '";" expected '
ERROR 110 : BEGIN, VAR, CONST or PROC expected '
ERROR 110 : 'END (of procedure) expected '
ERROR 110 : PROC_identifier expected '
ERROR 110 : 'procedure END_identifier expected '
ERROR 110 : 'procedure END_identifier do not match '

Error: Expression cannot begin with this Symbol

Error: Unknown identifier

Error: '=' expected

Known bugs

Events not defined in the CARP program are carried out
Workaround: Do not click the according event buttons.

Message "Window too small to display color mapping" comes late.
Workaround: Enlarge STATE window or CAT main window.

Erroneous display of cell matrix on high XYSize and high Zet values
Workaround: reduce the according values.

User defined CAP file doesn't occur in Palette menu
Workaround: Load the according color palette using the local menu.

Prompt for saving changed files before quitting CAT sometimes missing
Workaround: Leave and reenter CAT.

SetLattice procedure doesn't work properly
No workaround

Maximize / minimize button do not work allways properly
Workaround: Leave and reenter CAT.

Dialogbox 'Editor buffer is full' doesn't work
Workaround: Click the Close button of the dialogbox in the upper left corner. Bring up the LIST window and the local menu by clicking the right mouse button. Select 'New' to delete all contents of the LIST window. You are now able to continue work.

Minor trouble shooting items

Garbled color display
CAT buttons seem not to work
Plane state window not printable

Runtime messages

Runtime	: Runtime system is currently controlling the execution of an event - just a control message.
ShareWare	: Run limit of an event was exceeded due to the restrictions of the public domain version of CAT.
Plane too large	: Limitations of the program have been exceeded. Reduce XYSize and / or Zet value.