Reference Manual

Generated by Doxygen 1.6.3

Wed Feb 23 14:18:45 2011

Contents

Chapter 1

Namespace Index

1.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 2

Class Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 Package CA.py

Provides us with classes for cellular automata.

4.1.1 Detailed Description

Provides us with classes for cellular automata. If a new automaton is to be implemented, it should find it's place here as well.

4.2 Package Display.py

Provides us with Displays to show the CA using colored squares (as for Sandpile) and images (as for vonNeumann).

4.2.1 Detailed Description

Provides us with Displays to show the CA using colored squares (as for Sandpile) and images (as for vonNeumann).

4.3 Package sim.py

sim.py is the central simulating unit so READ THIS! It handles displaying the simulated CA as well as userinput.

4.3.1 Detailed Description

sim.py is the central simulating unit so READ THIS! It handles displaying the simulated CA as well as userinput. The simulator is constructed in a model-view-controller fashion. The singleton Simulator manages multiple instances of class Display and of class CA. To reduce those numbers, for each used combination of CA-type and size (as in (sizeX,sizeY)), instances are kept in Simulator.CADict. When importing a CA, who's size and type were already in use, only the configuration is kept additionally. In other cases, a new CA and a new Display are created in Simulator.getNewCA().

There is a TODO-List:

- implement an event-check for console input events in the main execution loop by checking globalEventQueue. You can start the simulator with a python console by typing: $> python -i sim.py
- Histograms could be used, after all, they are written already, but no structure to switch them on and off is implemented; maybe their use should be memorized in Simulator.CADict as well?
- This is a development version of the whole project with nearly no faulttolerance and stuff implemented.
- I wasn't able to track the reason, why Sandpile-CAs aren't displayed in the defined colorset. This being colors from black, different shades of gray and white. Instead it's displayed in ugly green-ish colors. Other colors work and the principal procedure with overblitting and all that worked when i tested it in a minimal testcase. Somehow something in pygame stinks...
- The vonNeumann-CA is implemented using the original vonNeumann-transition-ruleset. There are other such as JvN-32, which unfortunately is used in the only provided configuration of the Pesavento-replicator. Hint: The difference is in the use of the C-states. More on this matter can be found here: http://www.pd.infn.it/~rnobili/au_cell/
- To improve performance, try not to blit the whole configuration that is showing on the screen but only those cells, that changed in the recent step. Of course when zooming in, only a section of the configuration is displayed already.
- Maybe don't use the usual array to hold the configuration but a more flexible one to hold configurations that contain a big amount of "dead cells" or "unused cells" like state U in vonNeumann.

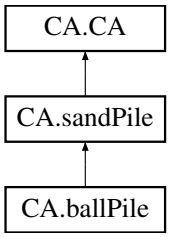
Chapter 5

Class Documentation

5.1 CA.ballPile Class Reference

Exactly the same as sandPile, but using images of colored footballs.

Inheritance diagram for CA.ballPile:



Public Member Functions

- def __init__
  The constructor.
- def getTitle
  Returns the cellular automaton's title.

Public Attributes

- palette
  Tells the Display module how to show each state.

Static Public Attributes

- list palette = [ ]
  Tells the Display module how to show each state.

5.1.1 Detailed Description

Exactly the same as sandPile, but using images of colored footballs.

5.1.2 Member Function Documentation

5.1.2.1 def CA.ballPile.getTitle (*self*)

Returns the cellular automaton's title.

It's a non-upper-cased version of CA.title, used to display it in the titlebar

Reimplemented from CA.CA.

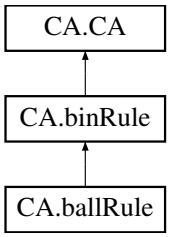
The documentation for this class was generated from the following file:

- CA.py

5.2 CA.ballRule Class Reference

Exactly the same as binRule, but with images of colored footballs instead of colored squares.

Inheritance diagram for CA.ballRule:



Public Member Functions

- def __init__
  constructor

Public Attributes

- title
  The title of this ca.

Static Public Attributes

- list palette = [ ]
  Tells the Display module how to show each state.

5.2.1 Detailed Description

Exactly the same as binRule, but with images of colored footballs instead of colored squares.

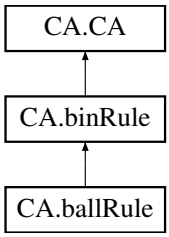
The documentation for this class was generated from the following file:

- CA.py

5.3 CA.binRule Class Reference

A cellular automaton that simulates all one dimensional binary rule cellular automaton, such as Rule 110.

Inheritance diagram for CA.binRule:



Public Member Functions

- def __init__
  constructor
- def getType
  Returns the type of the cellular automaton.
- def loopFunc
  Is called in every step of the simulation.
- def step
  What to do in every step.
- def updateAllCellsPy
  Updates all cells in plain python.
- def updateAllCellsWeaveInline
  Updates all cells using scipy.weave.inline for faster execution.

Public Attributes

- dim
  The dimension of binRule.
- ruleNr
  The rulenumber in decimal notation (in [0;255]).
- sizeX
  width of the ca
- sizeY
  height of the ca

- size

  the ca's size as (width,height)

- title

  The title of the ca.

- currConf

  The current configuration is stored here.

- nextConf

  The next step's configuration is stored here (ping-ponging!).

- ruleIdx

  An array that contains the value table for this particular binary transition rule.

Static Public Attributes

- list palette = [(0,0,0), (255,255,255)]

  Tells the Display module how to show each state.

5.3.1 Detailed Description

A cellular automaton that simulates all one dimensional binary rule cellular automaton, such as Rule 110. CA.binRule handles all one dimenasional binary cellular automaton with the neighbourhood (-1,0,1).

5.3.2 Member Function Documentation

5.3.2.1 def CA.binRule.getType (*self*)

Returns the type of the cellular automaton.

It's a upper-cased version of CA.title, used to identify it internally as one or another cellular automaton.

Reimplemented from CA.CA.

5.3.2.2 def CA.binRule.loopFunc (*self*)

Is called in every step of the simulation.

Reimplemented from CA.CA.

5.3.2.3 def CA.binRule.step (*self*)

What to do in every step.

Calls binRule.updateAllCellsWeaveInline, that uses scipy.weave.inline

5.3.2.4 def CA.binRule.updateAllCellsPy (*self*)

Updates all cells in plain python.

5.3.3 Member Data Documentation

5.3.3.1 CA.binRule.nextConf

The next step's configuration is stored here (ping-ponging!).

Reimplemented from CA.CA.

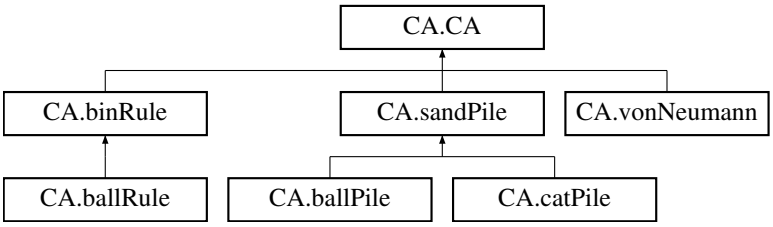
The documentation for this class was generated from the following file:

- CA.py

5.4 CA.CA Class Reference

Provides stuff like import/export functions, getter methods and resizing.

Inheritance diagram for CA.CA:



Public Member Functions

- def eventFunc

  Is executed when any events, such as mouseclicks or keyboardhits, are recorded and relayed to the cellular automaton.

- def exportConf

  Exports the current configuration to a file in XASIM-format.

- def exportConfAnnotatedLines

  Since we want to be compatible to XASIM, this file format is deprecated.

- def getType

  Returns the type of the cellular automaton.

- def getConf

  Returns the current configuration.

- def getDim

  Returns the cellular automaton's dimension.

- def getSize

  Returns the cellular automaton's size as (height,width).

- def getTitle

  Returns the cellular automaton's title.

- def importConf

  Imports a configuration from a file in XASIM-Format.

- def importConfAnnotatedLines

  Deprecated.

- def loopFunc

  Is called in every step of the simulation.

- def quit

  Prototype.

- def resize

  Resizing the cellular automaton.

- def setConf

  Set the current configuration to conf.

Public Attributes

- sizeX

  width of the ca

- sizeY

  height of the ca

- size

  the ca's size as (width,height)

- currConf

  The current configuration is stored here.

- nextConf

  The next step's configuration is stored here (ping-ponging!).

Static Public Attributes

- int INIT_ZERO = 0

  When passed as parameter, the configuration is filled with zeros.

- int INIT_ONES = 1

  When passed as parameter, the configuration is filled with ones.

- int INIT_RAND = 2

  When passed as parameter, the configuration is filled randomly.

- int INIT_FILE = 3

  When passed as parameter, the configuration is imported from a given file.

- int IMPORTOK = 0

  Returnflag when importing a configuration from file.

- int SIZECHANGED = 1

  Returnflag when importing a configuration from file.

- int WRONGCA = 2

  Returnflag when importing a configuration from file.

- int IMPORTNOTOK = 3

  Returnflag when importing a configuration from file.

- list palette = [ ]

  Tells the Display module how to show each state.

5.4.1 Detailed Description

Provides stuff like import/export functions, getter methods and resizing.

5.4.2 Member Function Documentation

5.4.2.1 def CA.CA.eventFunc (*self*, *event*)

Is executed when any events, such as mouseclicks or keyboardhits, are recorded and relayed to the cellular automaton.

Reimplemented in CA.sandPile, and CA.vonNeumann.

5.4.2.2 def CA.CA.exportConf (*self*, *filename*)

Exports the current configuration to a file in XASIM-format.

The first lines states the width, the next line the height, if the cellular automaton is 2 dimensional, the configuration itself beginning after that, consists all cell's states in a row, row by row, so the shape of the configuration file is the same as the configuration itself. Notice the ghostcells at the borders!

```
Example:
4                 // width
5                 // height
(0)(0)(0)(0)(0)   // here starts the configuration
(0)(2)(0)(3)(0)   // number of cols is witdh
(0)(2)(1)(2)(0)   // number of lines is height
(0)(0)(0)(0)(0)
SANDPILE
```

5.4.2.3 def CA.CA.exportConfAnnotatedLines (*self*, *filename*)

Since we want to be compatible to XASIM, this file format is deprecated.

5.4.2.4 def CA.CA.getTitle (*self*)

Returns the cellular automaton's title.

It's a non-upper-cased version of CA.title, used to display it in the titlebar

Reimplemented in CA.catPile, and CA.ballPile.

5.4.2.5 def CA.CA.getType (*self*)

Returns the type of the cellular automaton.

It's a upper-cased version of CA.title, used to identify it internally as one or another cellular automaton.

Reimplemented in CA.binRule.

5.4.2.6 def CA.CA.importConf (*self*, *filename*)

Imports a configuration from a file in XASIM-Format.

For XASIM-Format, see CA.exportConf. For vonNeumann cellular automaton, RLE-files are supported as well (see vonNeumann.importConf).

Reimplemented in CA.vonNeumann.

5.4.2.7 def CA.CA.importConfAnnotatedLines (*self*, *filename*)

Deprecated.

Importing own, non-XASIM-compatible format.

5.4.2.8 def CA.CA.loopFunc (*self*)

Is called in every step of the simulation.

Reimplemented in CA.binRule, CA.sandPile, and CA.vonNeumann.

5.4.2.9 def CA.CA.quit (*self*)

Prototype.

.. Not really implemented yet

Reimplemented in CA.sandPile.

5.4.2.10 def CA.CA.resize (*self*, *sizeX*, *sizeY* = `None`)

Resizing the cellular automaton.

Not in use right now,

Reimplemented in CA.vonNeumann.

5.4.2.11 def CA.CA.setConf (*self*, *conf*)

Set the current configuration to conf.

This is used when switching between marked configurations and cellular automaton types.

Parameters

conf The configuration to load.

Reimplemented in CA.vonNeumann.

5.4.3 Member Data Documentation

5.4.3.1 CA.CA.nextConf

The next step's configuration is stored here (ping-ponging!).

Reimplemented in CA.binRule, CA.sandPile, CA.catPile, and CA.vonNeumann.

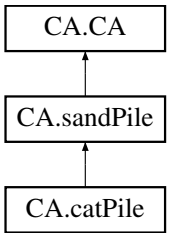
The documentation for this class was generated from the following file:

- CA.py

5.5 CA.catPile Class Reference

Exactly the same as sandPile, but has an image of a cat as starting configuration.

Inheritance diagram for CA.catPile:



Public Member Functions

- def __init__

  The constructor.

- def readImage

  Reads the image of a cat that is being used as starting configuration.

- def getTitle

  Returns the cellular automaton's title.

Public Attributes

- currConf

  The current configuration is stored here.

- nextConf

  The next step's configuration is stored here (ping-ponging!).

5.5.1 Detailed Description

Exactly the same as sandPile, but has an image of a cat as starting configuration.

5.5.2 Member Function Documentation

5.5.2.1 def CA.catPile.getTitle (*self*)

Returns the cellular automaton's title.

It's a non-upper-cased version of CA.title, used to display it in the titlebar

Reimplemented from CA.CA.

5.5.3 Member Data Documentation

5.5.3.1 CA.catPile.nextConf

The next step's configuration is stored here (ping-ponging!).

Reimplemented from CA.sandPile.

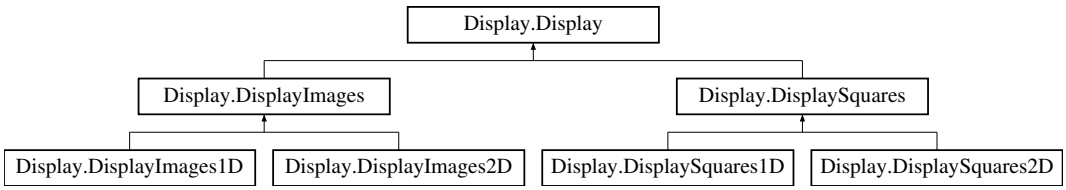
The documentation for this class was generated from the following file:

- CA.py

5.6 Display.Display Class Reference

class Display() handles everything connected to displaying the configuration of a CA.

Inheritance diagram for Display.Display:



Public Member Functions

- def __init__

  Constructor, initializes pretty everything.

- def getCACoordinates

  Get the coordinate of the cell that is clicked on in the display window.

- def getSize

  Get the size of the CA being displayed.

- def getUserInputKey

  A kind of commandline that is displayed as HUD.

- def getViewSize

  Get the actual size of display Used to get the count of currently displayed cells.

- def quit

  Well.

- def setText

  Setting up text to show a message.

- def showCounter

  Display a step counter Counter is displayed at the bottom left corner.

- def showText

  Display messages If textAlive > 0, i.e.

- def update

  Updating the pygame display, setting window caption.

Public Attributes

- sizeY

  Height and width of the displayed CA.

- size

  Size of the showing CA.

- scale

  Factor by which a cell is scaled to show it bigger than one pixel on the screen.

- screenSize

  Actual size of the simulation window.

- palette

  Colors (DisplaySquares) or images (DisplayImages) representing all states.

- simScreen

  Pygame display object.

- clock

  Clock used to calculate fps rate.

- screenXMin

  Index of the first cell displayed on the left.

- screenYMin

  Index of the first cell displayed on the top.

- zoomIdx

  Iterator for zoomSize selection (see zoomSizes).

- zoomSizes

  Fixed number of cells displayed (width, height) e.g.

- oneLiner

  Whether the CA to display will be a 1 or 2 dimensional one.

- surface

  Toplevel pygame surface.

- myfontSize

  Fontsize used to display messages.

- myfont

  Font used to display messages.

- newTextLive

  Number of iterations of the main loop in Simulator.start() for how long the messages are kept visible.

- textAlive

  How many iterations of the main loop left where message is visible.

- HUDText

  'HeadsUpDisplay', the message that is being shown

- counterPos

  where to display the step counter while simulating

- textPos

  where to display text while simulating

5.6.1 Detailed Description

class Display() handles everything connected to displaying the configuration of a CA. It handles zooming, resizing, scrolling, handling 1D and 2D CA, the colors used for different states of a cell, user input like file names and displaying short info messages and updating everything over and over again.

The general difference between DisplaySquares and DisplayImages is, that squares are uni-colored and can be blitted using only 1x1 pixel per cell and afterwards scaled up to the actual screensize, while images have to be scaled to the maximum size s, so that (s $*$ cells) doesn't exceed the screensize, and afterwards be scaled up a little to fit the screen exactly.

5.6.2 Member Function Documentation

5.6.2.1 def Display.Display.__init__ (*self*, *size*, *scale*, *palette*, *dim*, *oneLiner* = `False`)

Constructor, initializes pretty everything.

Parameters

size The size of the display in cells$*$cells

scale how big should a display be displayed at first

palette How to display a state

dim Dimension of CA

oneLiner 2 or 3 dimensional?

Reimplemented in Display.DisplaySquares, Display.DisplaySquares1D, Display.DisplaySquares2D, Display.DisplayImages, Display.DisplayImages1D, and Display.DisplayImages2D.

5.6.2.2 def Display.Display.quit (*self*)

Well.

..

5.6.2.3 def Display.Display.setText (*self*, *text*)

Setting up text to show a message.

Parameters

text The message that is going to be displayed

5.6.2.4 def Display.Display.showCounter (*self*, *c*)

Display a step counter Counter is displayed at the bottom left corner.

It could have been displayed at the top left corner but it would interfere with messages displayed by Display.showText()

Parameters

c The stepcount

5.6.2.5 def Display.Display.showText (*self*)

Display messages If textAlive $>$ 0, i.e.

if the message hasn't been visible for it's maximum time of visibility, the message is being displayed in the top left corner

5.6.3 Member Data Documentation

5.6.3.1 Display.Display.surface

Toplevel pygame surface.

All subsurfaces are children of this.

5.6.3.2 Display.Display.zoomSizes

Fixed number of cells displayed (width, height) e.g.

: [(20.0,20.0), (15.0,15,0), ...]

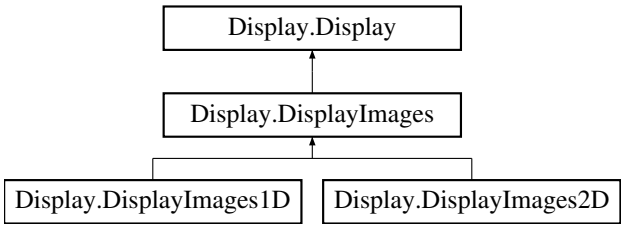
The documentation for this class was generated from the following file:

- Display.py

5.7 Display.DisplayImages Class Reference

The superclass for displayimagesXd.

Inheritance diagram for Display.DisplayImages:



Public Member Functions

- def __init__

  Constructor, initializes pretty everything.

- def drawConf

  Draws the blitted data to the screen.

- def getSize

  Get the size of the CA being displayed.

- def rescaleImages

  Rescale images when zooming in the simulator-window.

- def resize

  Make the display bigger or smaller.

Public Attributes

- screenSize

  Actual size of the simulation window.

- palette

  Images used to display the cell's different states.

- dim

  A quick way to remember whether a 1D or 2D CA is simulated.

- stateImages

  An array that holds the images used for all possible states in the currently displayed size.

- stateImageDict

  A map for easy access to different sizes of the images.

5.7.1 Detailed Description

The superclass for displayimagesXd. While DisplaySquare objects simply draw colored squares for each state, DisplayImages has to handle different versions of the used images for each state in different sizes, to ensure a constant imagequality when zooming in and out. These are stored in DisplayImages.stateImageDict, where the used scale (use the pair (scale,scale) when displaying quadratic shape) is the key to the array of actual scaled images

5.7.2 Member Function Documentation

5.7.2.1 def Display.DisplayImages.resize (*self*, *f*)

Make the display bigger or smaller.

Parameters

f Factor by which the size is scaled

Reimplemented in Display.DisplayImages1D, and Display.DisplayImages2D.

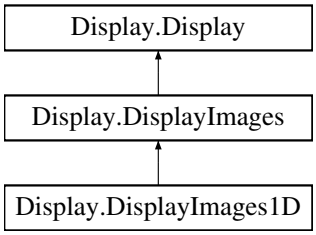
The documentation for this class was generated from the following file:

- Display.py

5.8 Display.DisplayImages1D Class Reference

Displayes 1D-CA with images for all states.

Inheritance diagram for Display.DisplayImages1D:



Public Member Functions

- def __init__
  Constructor, initializes pretty everything.
- def blitImages
  blit images to the pygame screen
- def resize
  resize screen
- def scroll
  When zoomed, scrolling to the left and to the right in 1D CA.
- def zoom
  Zooming into a 1D CA.

Public Attributes

- newlineSurface
  the subsurface onto which the new states are blitted
- zoomIdx
  Iterator for zoomSize selection (see zoomSizes).
- screenXMin
  Index of the first cell displayed on the left.

5.8.1 Detailed Description

Displayes 1D-CA with images for all states. While DisplayImages2D and DisplaySquares2D just overblit the whole display, DisplayImages1D (and, btw, DisplaySquares1D, too) shifts the displayed configuration of earlier steps up and displayes the new configuration in the bottom row (see DisplayImages1D.newlineSurface), creating a space-time-diagram, the downwards directed Y-Axis representing time.

5.8.2 Member Function Documentation

5.8.2.1 def Display.DisplayImages1D.scroll (*self*, *key*)

When zoomed, scrolling to the left and to the right in 1D CA.

In 1D CA scrolling up and down is not supported yet

Parameters

key Pygame.Key object containing the pressed key

5.8.2.2 def Display.DisplayImages1D.zoom (*self*, *c*)

Zooming into a 1D CA.

Here the initialized fixed zoomSizes from init() are used

Parameters

c User input indicating zooming in or zooming out

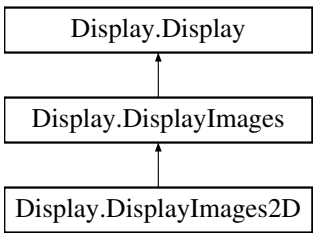
The documentation for this class was generated from the following file:

- Display.py

5.9 Display.DisplayImages2D Class Reference

Displayes 2D-CA with Images.

Inheritance diagram for Display.DisplayImages2D:



Public Member Functions

- def __init__

  Constructor, initializes pretty everything.

- def blitImages

  blit images to the pygame screen

- def resize

  resize the subsurface that is going to be blitted upon

- def scroll

  When zoomed, scrolling left, right, up and down in 2D CA.

- def zoom

  Zooming into a 2D CA Here the initialized fixed zoomSizes from init() are used.

Public Attributes

- subSurf

  Subsurface that is used to blit only the section of the CA that is actually displayed at the moment.

- counterPos

  where to display the step counter while simulating

- zoomIdx

  Iterator for zoomSize selection (see zoomSizes).

- screenXMin

  Index of the first cell displayed on the left.

- screenYMin

  Index of the first cell displayed on the top.

5.9.1 Detailed Description

Displayes 2D-CA with Images.

5.9.2 Member Function Documentation

5.9.2.1 def Display.DisplayImages2D.scroll (*self*, *key*)

When zoomed, scrolling left, right, up and down in 2D CA.

Parameters

key Pygame.Key object containing the pressed key

5.9.2.2 def Display.DisplayImages2D.zoom (*self*, *c*)

Zooming into a 2D CA Here the initialized fixed zoomSizes from init() are used.

Parameters

c User input indicating zooming in or zooming out

5.9.3 Member Data Documentation

5.9.3.1 Display.DisplayImages2D.subSurf

Subsurface that is used to blit only the section of the CA that is actually displayed at the moment.

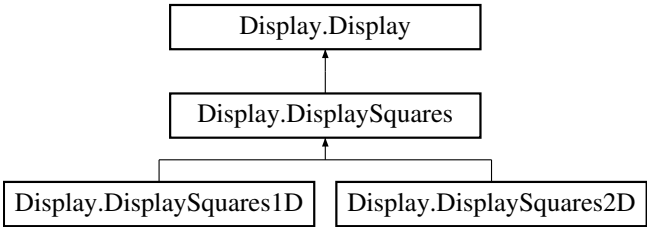
The documentation for this class was generated from the following file:

- Display.py

5.10 Display.DisplaySquares Class Reference

Displayes states using simple one-colored squares, using DisplaySquares.palette for colors.

Inheritance diagram for Display.DisplaySquares:



Public Member Functions

- def __init__

  Constructor, initializes pretty everything.

- def drawConf

  blit the data onto the screen

- def getSize

  Get the size of the CA being displayed.

- def resize

  Make the display bigger or smaller.

Public Attributes

- palette

  Colors for each state.

- dim

  the CA's dimension

- screenSize

  Actual size of the simulation window.

5.10.1 Detailed Description

Displayes states using simple one-colored squares, using DisplaySquares.palette for colors. DisplaySquares and children always blit the whole CA with squares of size (1,1) in the top left corner of the display and magnify the section that is being actually displayed

5.10.2 Member Function Documentation

5.10.2.1 def Display.DisplaySquares.resize (*self*, *f*)

Make the display bigger or smaller.

Parameters

f Factor by which the size is scaled

Reimplemented in Display.DisplaySquares2D.

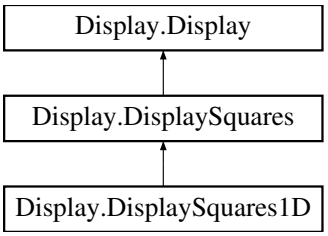
The documentation for this class was generated from the following file:

- Display.py

5.11 Display.DisplaySquares1D Class Reference

Displayes 1D-CA with colored squares for all states.

Inheritance diagram for Display.DisplaySquares1D:



Public Member Functions

- def __init__

  Constructor, initializes pretty everything.

- def blitArray

  Blitting function used for 1D CA.

- def scroll

  When zoomed, scrolling to the left and to the right in 1D CA In 1D CA scrolling up and down is not supported yet.

- def zoom

  Zooming into a 1D CA Here the initialized fixed zoomSizes from init() are used.

Public Attributes

- newlineSurface

  New configurations of a 1D CA are displayed only in the bottom line.

- displaySurface

  the section of the ca that is actually displayed

- zoomIdx

  Iterator for zoomSize selection (see zoomSizes).

- screenXMin

  Index of the first cell displayed on the left.

5.11.1 Detailed Description

Displayes 1D-CA with colored squares for all states. While DisplayImages2D and DisplaySquares2D just over-blit the whole display, DisplaySquares1D (and btw DisplayImages1D, too) shifts the displayed configuration of earlier steps up and displayes the new configuration in the bottom row (see DisplaySquares1D.newlineSurface), creating a space-time-diagram, the downwards directed Y-Axis representing time.

5.11.2 Member Function Documentation

5.11.2.1 def Display.DisplaySquares1D.scroll (*self*, *key*)

When zoomed, scrolling to the left and to the right in 1D CA In 1D CA scrolling up and down is not supported yet.

Parameters

key Pygame.Key object containing the pressed key

5.11.2.2 def Display.DisplaySquares1D.zoom (*self*, *c*)

Zooming into a 1D CA Here the initialized fixed zoomSizes from init() are used.

Parameters

c User input indicating zooming in or zooming out

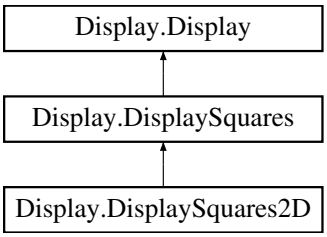
The documentation for this class was generated from the following file:

- Display.py

5.12 Display.DisplaySquares2D Class Reference

Displayes 2D-CA using colored squares.

Inheritance diagram for Display.DisplaySquares2D:



Public Member Functions

- def __init__

  Constructor, initializes pretty everything.

- def blitArray

  Blitting function used for 2D CA.

- def resize

  resize the images representing the states

- def scroll

  When zoomed, scrolling left, right, up and down in 2D CA.

- def zoom

  Zooming into a 2D CA Here the initialized fixed zoomSizes from init() are used.

Public Attributes

- subSurf

  the part of display on which the whole CA is blitted upon

- counterPos

  where to display the step counter while simulating

- zoomIdx

  Iterator for zoomSize selection (see zoomSizes).

- screenXMin

  Index of the first cell displayed on the left.

- screenYMin

  Index of the first cell displayed on the top.

5.12.1 Detailed Description

Displayes 2D-CA using colored squares.

5.12.2 Member Function Documentation

5.12.2.1 def Display.DisplaySquares2D.scroll (*self*, *key*)

When zoomed, scrolling left, right, up and down in 2D CA.

Parameters

key Pygame.Key object containing the pressed key

5.12.2.2 def Display.DisplaySquares2D.zoom (*self*, *c*)

Zooming into a 2D CA Here the initialized fixed zoomSizes from init() are used.

Parameters

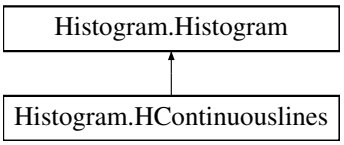
c User input indicating zooming in or zooming out

The documentation for this class was generated from the following file:

- Display.py

5.13 Histogram.HContinuouslines Class Reference

Inheritance diagram for Histogram.HContinuouslines:



Public Member Functions

- def **runProcess**

Public Attributes

- **histWindowSize**
- **offsetX**
- **offsetY**
- **surf**
- **scaleSubSurf**
- **histSubSurf**
- **infoSubSurf**

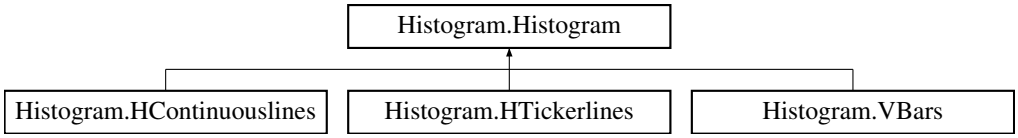
The documentation for this class was generated from the following file:

- Histogram.py

5.14 Histogram.Histogram Class Reference

abstract superclass, dont use this one ##

Inheritance diagram for Histogram.Histogram:



Public Member Functions

- def **__init__**
- def **show**
- def **hide**
- def **close**
- def **handleNonPygameEvents**
- def **set_title**
- def **update**
- def **getInfoPositions**

Public Attributes

- **N**
- **maxVal**
- **activePalette**
- **eventQueue**
- **QUIT**
- **SHOW**
- **HIDE**
- **info**
- **conn2**
- **p**
- **infoColorRectSize**

5.14.1 Detailed Description

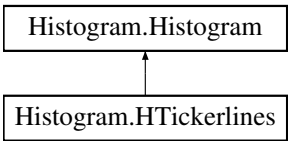
abstract superclass, dont use this one ##

The documentation for this class was generated from the following file:

- Histogram.py

5.15 Histogram.HTickerlines Class Reference

Inheritance diagram for Histogram.HTickerlines:



Public Member Functions

- def **runProcess**

Public Attributes

- **histWindowSize**
- **offsetX**
- **offsetY**
- **surf**
- **scaleSubSurf**
- **histSubSurf**
- **infoSubSurf**

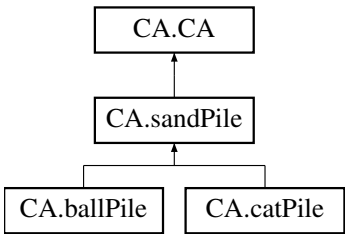
The documentation for this class was generated from the following file:

- Histogram.py

5.16 CA.sandPile Class Reference

The SandPile cellular automaton.

Inheritance diagram for CA.sandPile:



Public Member Functions

- def __init__

  The constructor.

- def addGrain

  Since a sandPile ca runs out of activity eventually if no grains are added, it has to happen once in a while.

- def addGrainRandomly

  Randomly throws a new grain into the configuration.

- def eventFunc

  What happens when a event occurs? When the left mousebutton is clicked on a cell, a grain is added to that, a right-click resets the cell's state to 0.

- def getHistogram

  Returns a histogram over the ca's states.

- def loopFunc

  Is called in every step of the simulation.

- def quit

  Prototype.

- def setState

  Set a cell's state to s.

- def step

  What to do in every step.

- def updateAllCellsPyHistImpl

  Updates all cells in plain python and calclulates the new histogram, too.

- def updateAllCellsPyHistExpl

  Updates all cells in plain python without calculating the new histogram.

- def updateAllCellsWeaveInline

  Updates all cells using scipy.weave.inline.

Public Attributes

- dim

  The ca's dimension.

- title

  The ca's title.

- size

  the ca's size as (width,height)

- sizeY

  height of the ca

- histogram

  The histogram over all states in the ca, as numpy.array.

- currConf

  The current configuration is stored here.

- nextConf

  The next step's configuration is stored here (ping-ponging!).

Static Public Attributes

- list palette

  Tells the Display module how to show each state.

- tuple info

  Histogram info for each state.

5.16.1 Detailed Description

The SandPile cellular automaton.

5.16.2 Member Function Documentation

5.16.2.1 def CA.sandPile.addGrain (*self*, *x*, *y*)

Since a sandPile ca runs out of activity eventually if no grains are added, it has to happen once in a while.

Parameters

- ***x*** X-coordinate where to add a grain
- ***y*** Y-coordinate where to add a grain

5.16.2.2 def CA.sandPile.eventFunc (*self*, *e*)

What happens when a event occurs? When the left mousebutton is clicked on a cell, a grain is added to that, a right-click resets the cell's state to 0.

Reimplemented from CA.CA.

5.16.2.3 def CA.sandPile.getHistogram (*self*)

Returns a histogram over the ca's states.

5.16.2.4 def CA.sandPile.loopFunc (*self*)

Is called in every step of the simulation.

Reimplemented from CA.CA.

5.16.2.5 def CA.sandPile.quit (*self*)

Prototype.

.. Not really implemented yet

Reimplemented from CA.CA.

5.16.2.6 def CA.sandPile.step (*self*)

What to do in every step.

Calls sandPile.updateAllCellsWeaveInline, that uses scipy.weave.inline

5.16.2.7 def CA.sandPile.updateAllCellsPyHistExpl (*self*)

Updates all cells in plain python without calculating the new histogram.

5.16.2.8 def CA.sandPile.updateAllCellsPyHistImpl (*self*)

Updates all cells in plain python and calclulates the new histogram, too.

5.16.3 Member Data Documentation

5.16.3.1 CA.sandPile.histogram

The histogram over all states in the ca, as numpy.array.

Containing the absolute frequency of each state

5.16.3.2 tuple CA.sandPile.info [static]

Initial value:

```
( "state 0", "state 1", "state 2", "state 3",
              "state 4", "state 5", "state 6", "state 7" )
```

Histogram info for each state.

5.16.3.3 CA.sandPile.nextConf

The next step's configuration is stored here (ping-ponging!).

Reimplemented from CA.CA.

Reimplemented in CA.catPile.

5.16.3.4 list CA.sandPile.palette [static]

Initial value:

```
[(0, 0, 0), (32, 32, 32), (64, 64, 64), (96, 96, 96),
                (128, 128, 128), (160, 160, 160), (192, 192, 192), (224, 224, 224)
       ]
```

Tells the Display module how to show each state.

Reimplemented from CA.CA.

Reimplemented in CA.ballPile, and CA.ballPile.

The documentation for this class was generated from the following file:

- CA.py

5.17 sim.Simulator Class Reference

Central simulation unit.

Public Member Functions

- def __init__

  The constructor.

- def getNewCA

  get new instance of a given CA since more than one CA-object is used to display different kinds and sizes of CA, more than one object is easier to handle than a single one

- def start

  The main function This function runs until the simulator ends.

- def step

  Wrapper a thin layer between caller and callee of the mighty step() functions of every CA.

- def stop

  Wrapper to stop simulation.

Public Attributes

- oneLiner

  a flag if the first CA is a 2 oder 3 dimensional CA

- scale

  scale by which a single cell is displayed (pixel)

- caDict

  a map where one can get the object to a given type and size of CA

- display

  references to the currently used CA and Display

- histograms

  a list of histograms that are used to visualize the distribution of different states in the displayed conf.

- caConfDict

  a map to the marked configurations that can be reloaded.

- delayGranularity

  delta between two different levels of delaying the simulation to slow it down and have a look

- currDelay

  the current level of delay

- caKeys

  a list of all the different sizes and kinds of CA the simulator has handled yet

5.17.1 Detailed Description

Central simulation unit. Handles user I/O, importing and exporting, marking, and everything else

5.17.2 Member Function Documentation

5.17.2.1 def sim.Simulator.start (*self*)

The main function This function runs until the simulator ends.

The main while(1)-loop is a pygameism, all keyboard- and mouse-events are cought here and relayed to Display- and CA-objects, respectively

5.17.2.2 def sim.Simulator.stop (*self*)

Wrapper to stop simulation.

Not in use right now (Jan 17th 2011)

5.17.3 Member Data Documentation

5.17.3.1 sim.Simulator.caConfDict

a map to the marked configurations that can be reloaded.

You can mark a configuration by hitting 'm' during the simulation

5.17.3.2 sim.Simulator.histograms

a list of histograms that are used to visualize the distribution of different states in the displayed conf.

Since this is not always useful (vonNeumann...?!), it's not always needed...

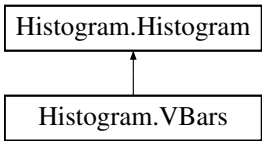
The documentation for this class was generated from the following file:

- sim.py

5.18 Histogram.VBars Class Reference

use these! ##

Inheritance diagram for Histogram.VBars:



Public Member Functions

- def **runProcess**

Public Attributes

- **histWindowSize**
- **offsetX**
- **offsetY**
- **surf**
- **scaleSubSurf**
- **histSubSurf**
- **infoSubSurf**

5.18.1 Detailed Description

use these! ##

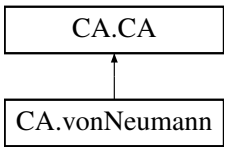
The documentation for this class was generated from the following file:

- Histogram.py

5.19 CA.vonNeumann Class Reference

The cellular automaton proposed by John von Neumann.

Inheritance diagram for CA.vonNeumann:



Public Member Functions

- def __init__

  The constructor.

- def enlist

  Used to append cells to the list of cells to handle in the next step.

- def eventFunc

  Is executed when any events, such as mouseclicks or keyboardhits, are recorded and relayed to the cellular automaton.

- def getConf

  Returns the current configuration.

- def importConf

  Imports a configuration from a file in XASIM-Format.

- def loopFunc

  Is called in every step of the simulation.

- def resize

  Resizing the cellular automaton.

- def setConf

  Set the current configuration to conf.

- def step

  Calls the actual function that is used to calculate the next configuration.

- def updateAllCellsWeaveInline

  Updates all cells using scipy.weave.inline.

- def updateAllCellsWeaveInlineFewStates

  Update cells, but only those that changed or are in the neighbourhood of one of those.

Public Attributes

- title

  The ca's title.

- dim

  The ca's dimension.

- size

  the ca's size as (width,height)

- sizeY

  height of the ca

- displayableStateDict

  A map from states according to the bitmask to pygame blittable states (between 0 and 28).

- nameStateDict

  A map from human readable vonNeumann states (such as 'U', 'T020' and 'C11') actual states calculated via bitmask.

- states

  An array containing all correct states (see vonNeumann).

- currConf

  The current configuration is held here as usual, these two arrays contain the real configuration, that is used in every step .

- nextConf

  The current configuration is held here.

- **cActArr**
- **nActArr**
- **cList**
- **nList**
- **cCounter**
- **nCounter**
- displayConf

  The configuration that is blittet.

Static Public Attributes

- list palette = [ ]

  Tells the Display module how to show each state.

5.19.1 Detailed Description

The cellular automaton proposed by John von Neumann.

```
All states are encoded in a bitmask:

   <--MSB                     10                              LSB
 ...... 0 0 0 0 0 0 0 0 0 X X u a1 a0 eps sc1 sc0 s2 s1 s0 e1 e0
                           | | | |  |  |    |   |   |  |  |  |  |-> current e
 XX = 00 -> U     <--------| | | |  |  |    |   |   |  |  |  |----> next e
 XX = 01 -> C     <----------| | |  |  |    |   |   |  |  |
 XX = 10 -> S                | | |  |  |    |   |   |  |  |-------> lsb on S
 XX = 11 -> T                | | |  |  |    |   |   |  |----------> ...
                             | | |  |  |    |   |   |-------------> msb on S
 S{} is encoded as SMASK_111 | | |  |  |    |   |-----------------> s-state counter
                             | | |  |  |    |---------------------> s-state counter
                             | | |  |  |
                             | | |  |  |--------------------------> excited
                             | | |  |-----------------------------> direction
                             | |-------------------------------------> direction
                             |---------------------------------------> special
```

5.19.2 Member Function Documentation

5.19.2.1 def CA.vonNeumann.eventFunc (*self*, *event*)

Is executed when any events, such as mouseclicks or keyboardhits, are recorded and relayed to the cellular automaton.

Reimplemented from CA.CA.

5.19.2.2 def CA.vonNeumann.importConf (*self*, *filename*)

Imports a configuration from a file in XASIM-Format.

For XASIM-Format, see CA.exportConf. For vonNeumann cellular automaton, RLE-files are supported as well (see vonNeumann.importConf).

Reimplemented from CA.CA.

5.19.2.3 def CA.vonNeumann.loopFunc (*self*)

Is called in every step of the simulation.

Reimplemented from CA.CA.

5.19.2.4 def CA.vonNeumann.resize (*self*, *sizeX*, *sizeY* = `None`)

Resizing the cellular automaton.

Not in use right now,

Reimplemented from CA.CA.

5.19.2.5 def CA.vonNeumann.setConf (*self*, *conf*)

Set the current configuration to conf.

This is used when switching between marked configurations and cellular automaton types.

Parameters

conf The configuration to load.

Reimplemented from CA.CA.

5.19.2.6 def CA.vonNeumann.updateAllCellsWeaveInlineFewStates (*self*)

Update cells, but only those that changed or are in the neighbourhood of one of those.

This is done via bitchecking, and hence admittedly difficult to read. Every subsection of the transitionfunction from von Neumann's paper is marked.

5.19.3 Member Data Documentation

5.19.3.1 CA.vonNeumann.currConf

The current configuration is held here as usual, these two arrays contain the real configuration, that is used in every step .

.. (see vonNeumann.displayConf)

Reimplemented from CA.CA.

5.19.3.2 CA.vonNeumann.displayConf

The configuration that is blittet.

.. But in this CA the states are not enumerable from 0..28, but scattered between 0 and ~2^13, so we need a dict (see vonNeumann.displayableStateDict) to map the states to 0..28, so the Display-module can display states without knowing the difference

The documentation for this class was generated from the following file:

- CA.py

Index