Elementary Cellular Automaton
Python Script User Manual

Lucas Werner

March 2019

Contents

1 Introduction

This user manual serves to help with functions found in version 2.3.6 of the Python 3.7 script "ECAutomaton.py". The script runs in the command line and keywords are used to operate it. Commands are not case-sensitive. Invalid commands will be ignored.

The Elementary Cellular Automaton (ECA) itself is the simplest kind of automaton, because every generation can be expressed as an array of bits, which makes it possible to visualise several generations and their development in a two-dimensional pattern. By specifying rules on how cells (bits) change from one generation (time step) to another, interesting systems can be created. In the elementary cellular automata, such rulebits are given by three bits, such that exactly $2^3 = 8$ different rulebits have to be specified for a complete rule.

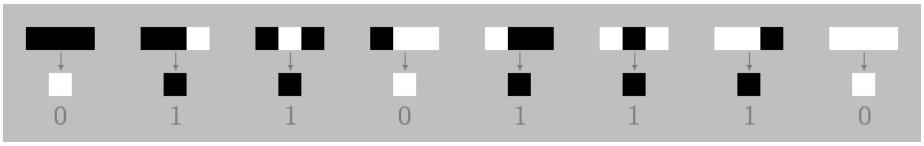

Figure 1: The rulebits of rule 110

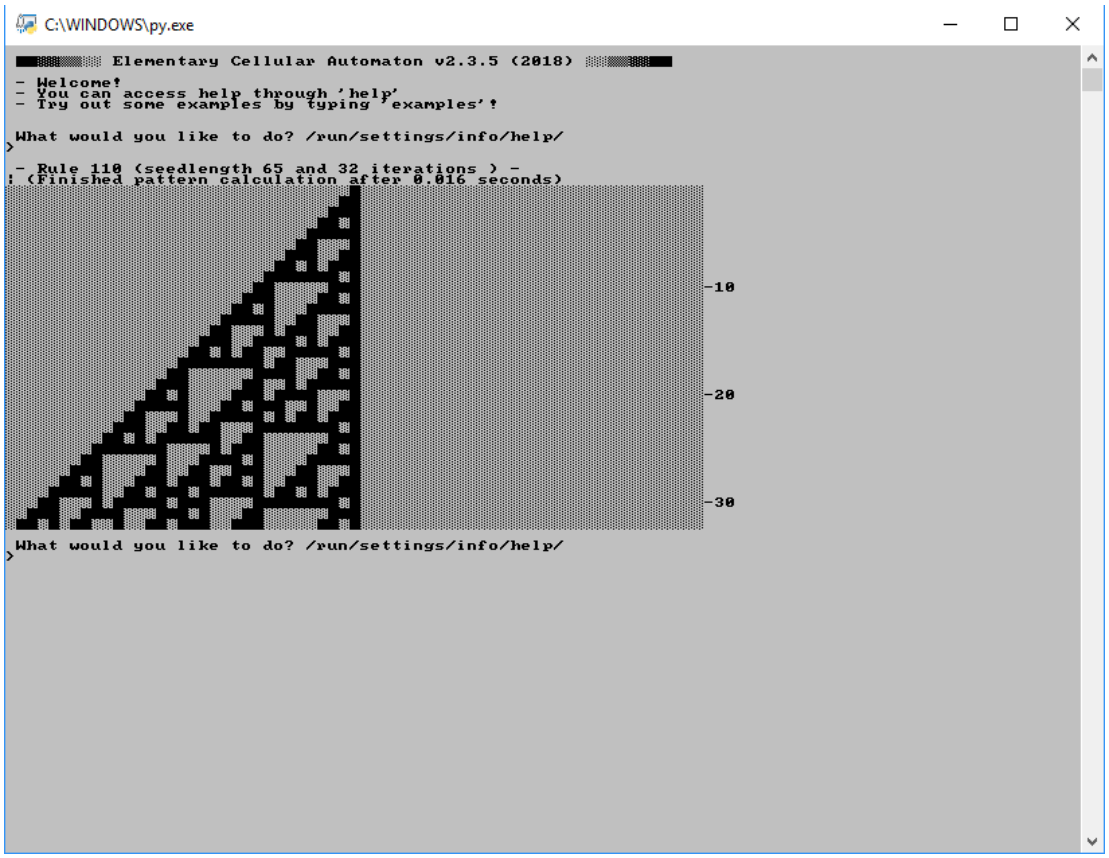

Figure 2: Rasterised square fonts work best for this script

2 Basic Functions

2.1 Getting Help with Commands

To access the built-in help menu while running the script enter "help" (or "?") into the command line, this will display 16 of the most common commands. For specific information on what a command does and how to use it, type "help [command]". When typing "commands" (or "command"/"cmd"/"cmds") all existing commands will be listed in the terminal, with their respective descriptions and usage.

2.2 Viewing Current Settings

"Settings" (or "sett"/"current"/"options") is used to view some main settings (rule, rule in binary, iterations, seed, appearance of bits and order) To get more elaborated settings the command "data" (or "debug"/"scriptvalues") is advised, which will display all values of modifyable parameters in the script.

2.3 Information and Articles

Basic information is given by the command "information" (or "info"/"about"/"eca") and will read a description of the script, its functionalities and more. Sources are indexed under "links" (or "link"/"sources"/"articles") and will show further literature available online. The links are ordered by topics (General, Rule 110, "A New Kind of Science", Other Scripts, Wolfram demonstrations, Other articles and PIL help)

2.4 Quitting and Resetting

To exit the script, simply type "exit" (or "exit()"). "Reset" will reset all script parameters (cf. 2.2)

3 Changing Basic Properties

3.1 Rule

Changing rule can be done by entering "rule" and then the number, or directly with "rule [number]" (*Note: The Numbering System by Stephen Wolfram only allows for rules from 0 up to 255*).

3.1.1 Mirroring a Rule

To obtain a rule, which behaves identical but vertically mirrored to its original, the command "rule mirror" can be used. (*Note: To obtain a mirrored pattern, both rule and seed have to be mirrored, cf. 3.2.2*).

3.1.2 Randomising a Rule

To obtain a random rule from 0 up to 255 the command "rule random" can be used. To directly randomise the rule and print the resulting pattern, the command "rr" (or "randrule"/"randomrule" is available (cf. 3.2.1).

3.1.3 Inverting a Rule

To obtain the rule corresponding to its original, but with ones and zeroes switched the command "rule invert" exists.

3.1.4 Manually defining a rule

To manually define a rule by determining the outcome of each of the eight rulebits, the command "rule manual" exists.

3.1.5 Arbitrary Rules

With the command "rule bits", arbitrary rules are possible, such that other cell states than 0 and 1 can be used (*Note: Neither modification of the neighbourhood-radius (1) nor the implementation of probabilistic automata is available as of v2.3.6*).

3.1.6 Changing arbitrary Cell Appearance

To change the appearance of any cell state, the option "cells" (or "cell") is available.

3.2 Initial Condition

A new inital condition, commonly referred to as 'seed', can be set by typing "seed" (or "array"/"width"/"columns") and then entering the seed. (*Note: Seeds containing characters/cell states other than 0 and 1 may cause a tear of*

information in the pattern, as all possible rulebits containing that character have to be defined manually, cf. 3.1.5).

3.2.1 Randomising a Seed

To randomise a seed, the option "seed random" is available. A menu is opened, in which a probabilitypool can be entered (e.g. '10' and '1100' will both result in a 50% distribution of ones and zeroes. '11000' will result in 40% ones and 60% zeroes). To directly use the pre-set probabilitypool and print a new pattern with it, the command "rs" (or "randseed"/"randomseed") can be used (cf. 3.1.2

Advanced randomisation is provided by the command "seed random+", by which a menu similar to the aforementioned will be opened, in which a probabilitypool can be defined. However, in this probabilitypool the individual elements have to be separated by commas, such that longer sequences can be given as such an element (this also makes it possible to make a seed consisting of only one repeating sequence). (*Note: In the cases that the individual elements do not perfectly satisfy the length of the seed given before, the script will take as many elements as possible without crossing the limit given by previous length*). Similarly as to before, the command "rs+" (or "randseed+"/"randomseed+") can be invoked to print a newly randomised pattern.

3.2.2 Mirroring a Seed

To mirror a seed type "seed mirror".

3.2.3 Inverting a Seed

To invert the ones and zeroes in a seed, type "seed invert".

3.2.4 Single-cell seed

To create a seed with a single cell, type "seed single", after which the three options "left" (or "l"), "center" (or "c) and "right" (or "r") will appear. Choosing one of the options will result in a single cell (1) being placed in the respective location on an array of the same length as the one before (Whereas arrays with even length will be extended by one cell).

3.2.5 Length of a Seed

To crop or extend the length of the current seed, the command "seed length" stands as an option". Entering a length shorter than the current one will result in the seed being cut off and a length greater will result in a seed being extended by zeroes to the appropriate length. As expected, identical length changes nothing.

3.2.6 Using an arbitrary binary Seed

With the function "seed decode" an arbitrary seed with only two distinct characters can be translated into a binary array (e.g. '22233' becomes '11100' or '00011' accordingly).

3.2.7 Centering a seed

After entering the command "seed center", one can enter a seed shorter than the current seed, the script will then try to center the entered sequence into the middle of an empty seed (zeroes) of length equal to the previous one.

3.3 Iterations

Using the command "iterations" (or "iter"/"generations"/"gen"/"height"/"rows"), the amount of iterations can be set. Alternatively, the number can be entered directly using "iterations [number]".

4 Viewing a Pattern

4.1 Running the ECA

The standard way to calculate the pattern resulting from the current settings is pressing enter without any command given (This is identical to typing the command "run").

4.1.1 Continuing a Pattern

The command "run continue" will automatically take the last generation of the last generated pattern and yield a pattern using it as seed.

4.1.2 Reversing a Pattern

The command "run reverse" will take the last generation of the last generated pattern and yield a pattern using it as a seed, and additionally take the generation *after* the last generation as so-called *history* (cf. 8.1.1. This will result in a reversed pattern if and only if the automaton state is set to 2nd-order (in all other cases it will behave identical to continuing the pattern, cf. 4.1.1)

4.2 Printing in command-line

The command "print" allows the user to (re-)print the last generated pattern (instead of calculating it again).

4.2.1 Small Print

The special sub-commands "print small" and "print small2" enable it to use special Unicode-characters (Block characters and quadrants) to display the last generated pattern (will only work for patterns with normal binary cell stated, i.e. 1 and 0). "Print small" will print a pattern, such that 4 cells are expressed by one character, whereas "print small2" will use characters displaying exactly two cells.

4.2.2 Running all rules

The special command "print_all" will apply the current settings to all rules from 0 up to 255 and try to print them. Additionally, with the command "print_all +save" all the generated patterns can be saved as images (cr. 4.3.2)

4.3 Images

The command "image" (or "png") saves and shows the last generated pattern. This function will only work if the Python Imaging Library (PIL or its fork Pillow) is installed (This can be done with pip, i.e. 'pip install Pillow' or 'python -m pip install –user Pillow'). Similarly to "save" (4.3.2), when adding an argument, i.e. "image [name]", the image will be saved under the name given.

4.3.1 Showing Images

The command "show" will show a the image corresponding to the last generated pattern (The image will be opened in an appropriate external program of choice).

4.3.2 Saving Images

The function "save [filename]" will save the last pattern with a name and file-type of choice (most common are: png/jpg/bmp). If no filename is given, the image will be saved as a png with nomenclature as follows: Rule-[rule][2nd-order]_([script id]-[No. of runs]).png

4.3.3 Changing the Saving Path

The standard location, where generated image files are saved, is in the same folder as the script itself. To make images save in a sub-folder, the command "imagepath" (or "path") can be used (Alternatively the more direct command "imagepath [path]" also works). The entered path hereby is a folder located in the same place as the script, and must be given as follows: '[folder]/' (subfolders can be added) (*Note: incorrect paths will result in images not being saved at all*)

5 Built-in Examples

As of version 2.3.6, there are 17 example patterns that can be accessed through the command "examples [number]" (or "example"/"ex"/"eg"). The following is a listing of such:

1. Slanted Sierpinski Fractal (rule 60)
2. Pascal's triangle modulo 2 (rule 90)
3. Particle collision simulation (rule 184)
4. Triangular fractal pattern (rule 150)
5. Chaos even from simple conditions (rule 30)
6. Rule with growth behaviour of sqrt(x) (rule 106)
7. Rule 110 example; type 'rule110 ether' to hide repeating tiles! (rule 110)
8. Example of a 2nd-order, reversible seed (rule 214R)
9. Interesting looking rule with individual compartments (rule 73R)
10. Carpet-pattern-rule (rule 150R)
11. Chaotic rule distantly resembling organic tissue (rule 105R)
12. Rule that turn out to be example 4 but rotated 90 degrees (rule 60R)
13. 'Inverted'-rule-version of example 9 (rule 146)
14. An interesting 2nd-order rule (rule 210R)
15. Another interesting 2nd-order rule (rule 202R)
16. And another one (rule 218R)
17. Pattern with horizontal symmetry axis (rule 90R)

6 The Preset Function

There are several sub-functions gathered under the command "preset". The latter by itself will open a menu from which to choose one of the former.

6.1 Pattern Presets

With "preset pattern", fast pattern sizes and types can be created. A menu will open, where such a preset can be entered with the following syntax: "[size] [type]", whereas the size will determine the seed length and the iterations and the type the kind of seed desired. For the type only four different options can be chosen, which are a single cell seed left, center or right or a random seed (using the random probabilitypool, cf. 3.2.1).

6.2 Changing Cell Appearance

"Preset cell" will open a menu from which to choose one of the standard ways to display cell states of a pattern printed in terminal (Such presets are 'standard'/'2'/'light'/'light2'/'literal').

6.3 Saving and loading patterns

It is possible to generate a string which wholly describes a unique pattern in this script. To get a corresponding string, the command "preset get" is to be used. Similarly, to load a pattern from such a string, "preset load" is used.

7 Other Functions and Properties

7.1 Edges

The "edges" (or "edge"/"boundary"/"boundaries") property describes how cells behave at the border. Normally, the sides are 'joined' ("wrap") and are treated as if they were connected. However this can be changed to simulate pillars of state cells ("1"/"0"/etc) at both ends.

7.2 Shift

For some patterns, it can be useful to actually slant the pattern to better visualise how it grows (e.g. rule 106). The command "shift [integer]" (or "slant") can be used for this. Positive numbers shift each generation to the right by the given amount, whereas negative numbers shift to the left.

7.3 Limit

The standard display limit is 40,000 cells (e.g. 200x200 pattern), if however desired, this can be adjusted using "limit [number]" (or "displaylimit"/"terminaldisplaylimit") to allow for larger command-line prints.

7.4 Linenumbers

When a pattern is printed, there will be line numbers indicating the generation of that line. The line numbers are normally given every 10 generations, it can be changed (or turned off, i.e. '0') using "linenumbers [number]" (or "rows"/"rownumbers"/"label"/"rowlabel").

7.5 Python

The command "python" (or "py"/"exec"/"execute"/"python3") will open a simulated python (3.7) interpreter, where code can be executed.

8 Advanced Functions

8.1 Changing Order

Using the command "order", the order of the automaton can be toggled (1st/2nd order). 2nd order automata are different, insofar as they take into consideration the cell state of the last generation in addition to the three bits. It will then carry out a XOR (exclusive or) operation on the previous cell state and the state that would have normally resolved from the three bits.

8.1.1 Advanced Initial Conditions

Because previous generations influence the outcome of an automaton, the function "history" (or "predecessor") provides the option to change the generation that would have been before the initial seed of a pattern. The history is normally just zeroes, but can be changed to ones, identical to the initial seed or a custom seed (which must be of exact same length).

8.2 Merging Patterns

The merging function can be used to compare two pattern of same size. First, a pattern has to be marked for comparison. For this, the command "merge set" (or "compare"/"comparison") is to be entered after having calculated the first pattern. After having generated the second pattern, either "merge", "merge image" or "merge save" have to be entered. "Merge" will print the comparison pattern (where changed cells are highlighted) like a normal pattern. "Merge image" will save and open the comparison pattern and an additional difference map (showing in black on white the cells that are different). "Merge save" will just save the two images and not try to open them in an external program.

8.3 Rule 110

Because of rule 110's universality (cf. 2.3), special functions have been dedicated to work with it. The first is "rule110" (or "r110"), which will switch the rule to 110 and allow the user to create a rule 110-specific seed consisting of gliders in letter codes. The user can also enter own sequences of cells, though they have to be in binary. An example of such a code is: 'e e B$\hat{0}$10100 C C2 Gun e'. Because so-called 'ether'-tiles are the medium through which information is transported in rule 110, the command "rule110 ether" will make ether tiles appear lighter than normal cells, making structures in specific patterns more visible.