Manual of CACATOO functions you'll need

Ingredients of a CA model

It is good to realise that a model consists of a **configuration** (number of rows and columns, which states the model can have and their colours, random seed, etc.), an **initialisation** (setting up the model, with initial states, such as seeding the grid with 50% 0 and 50% 1; opening the displays to show the grid), a **next-state function**, which defines what *every grid cell* uses as its rule to update, and then finally the **update** function that governs how you apply the next-state function to every grid cell (synchronously or asynchronously) and what you do in-between updates with the next-state function (like mixing or diffusing). It is also the place to put code for plotting graphs. At the very bottom of the code file you can add interactive buttons with some easy functions, for example for pausing and unpausing the simulation. For most coding you'll do, you don't need to pay too much attention to this.

Configuring the simulation

You will mostly need to change the ncol and nrow of the system, along with the seed and perhaps the wrapping. For a full list of options, see here.

Setting up a simulation

Given your configuration, you can initialise a grid (or plane):

```
sim = new Simulation(config)

sim.makeGridmodel("modelname")

sim.initialGrid("modelname", "name of this plane", state_1,
fraction_of_squares_to_fill_with_state_1, state_2,
fraction_of_squares_to_fill_with_state_2, etc.) //you can keep on adding states
and fractions
```

You might also want to start with a spot of organisms, say bacteria. There is a handy function to initialise a circle with *n* individuals to a certain state:

```
sim.initialSpot(sim.YourGridNameHere, 'YourStateNameHere', 1, 30,
sim.YourGridNameHere.nc / 2, sim.YourGridNameHere.nr / 2)
```

This will make a circle with 30 individuals of state 1 at the centre of your simulation grid (sim.YourGridNameHere.nc and sim.YourGridNameHere.nr are the X and Y coordinates of the spot, which I here set to the number of columns and the number of rows divided by 2, to put the spot neatly in the middle!)

Selecting a random neighbour

We often want to select a random neighbouring square and check its state in our next state function. For that we use:

```
randomMoore8(model, col, row)
```

So, an example within a next state function will look like:

```
let randomNeighbour = this.randomMoore8(this, i, j) //give me a random
neighbour in the current simulation (this), at row i, at column j.Save in variable
randomNeighbour.
```

Then, you can access the *state* by doing `randomNeighbour.statename`. For example, colonisation with some chance:

```
If (randomNeighbour.statename == 1) { // if the randomNeighbour is alive (1)
        If (sim.rng.genrand_real1() <= 0.5) {//number between 0-1, so 50% chance of growth
                this.grid[i][j].statename = randomNeighbour.statename
        }
}
```

Counting neighbours
You might also want to *count* the number of neighbours of each state around a square. Let's say you want to know how *many* neighbouring grid points have state 1:

```
let sumOfOnes = countMoore8(modelname, col, row, statename, 1)
```

If you don't specify the final argument, you will get an array, like so:

```
let sumOfStates = countMoore8(modelname, col, row, statename) →
[amount_0; amount_1; amount_2 ; … ; amount_maxState]
```

Setting states in a certain part of the grid
If you want to have fine-grained control (say, to remove all organisms in a certain part of the field every X time steps), you can write your own loop. Below, let's assume there are 250 columns and 250 rows, and we want a 50 by 50 area to be set to 0 every 200 timesteps.

```
if(sim.time%200==0) { // every 200 time steps

for (let i = 100; i < sim.modelname.nc - 100; i++) // i are columns, sim.modelname.nc – 100 = 250- 100
        for (let j = 100; j < sim.modelname.nr - 100; j++) // j are rows, sim.modelname.nr – 100 = 250- 100
            sim.modelname.grid[i][j].statename = 0 // Set the state to 0

}
```

You would put this function in the update-function of the CA (the final function where `this.synchronous()` or `this.asynchronous` is called to update the CA)

Counting total populations
To know the total population of 0 and 1 in the system, use the following command:

```
let myPopSizes = sim.YourModelName.getPopsizes('YourStateName', [0, 1]));
```

This will return an output like [5002, 4998], meaning there's 5002 zeroes, and 4998 ones. If you want to work with the quantity of ones, and you just love to know how many dozens of ones you have (you quirky person!), use:

```
let quantityDozensOfOnes = myPopSizes[1]/12
```

Plotting population sizes and other quantities of interest
If you look at the default simulation configuration, you will see that `graph_interval` is set to 10, meaning that a datapoint is added to the graph every 10 simulation time steps. Meanwhile, `graph_update` is set to 50, meaning that the graph is only redrawn (with 5 new data points added

each time) every 50 time steps. You might want to see updates of your plots more often, if so, change these settings in the config at the top of the code file! **NB** use an underscore, not a dash! Use all plotting functions in the update function of the CA!

Plotting population sizes

If you want to show population sizes, simply use:

```
this.plotPopsizes('YourStateName', [0, 1])
```

This will draw a plot of the population sizes of zeroes and ones over time.

Plotting other quantities over time

If you want to draw certain custom-defined quantities over time, use:

```
this.plotArray(["This is a 5, so intense!", "An 8, daring today, aren't
we?"], [5, 8], ["gold", "#FF00AA"], "Plotting the unchanging numbers 5 and
8 over time, because this is what my life has come to")
```

As you can see, it takes in a list/array of the names of your lines, a list/array of the quantities in question, the a list/array of colours with which to plot the lines, and a plot title. The result will be a plot of 5 and 8 over time, which is sure to be mesmerising and informative.

Making plots with custom quantities on the X and Y axis

If you want to plot your own variables, say, per capita growth versus population size, you will need the function `plotXY()`. An example usage is shown below:

```
this.plotXY(["The value 3", "The value 5"], [3, 5], ["black"], "Title: 3
on the X-axis, 5 on the Y-axis; linecolour: black", { drawPoints: true,
strokeWidth: 1, pointSize: 2, strokePattern: [2, 2] })
```

As you can see, this function requires a list/array of titles for your quantities, a list/array of the quantities in question (here they are just two numbers, but these can be variables or calculations), a list/array of the colour(s) to use for plotting, a title for the plot, and optional other options to pass on to the plotting function (that you needn't concern yourself with).

Getting a random number

We're using stochastic cellular automata. That means that we use random numbers to compare against, for example, growth rates, to decide whether an organism grows or not. To get a random number between 0-1 use

`sim.rng.genrand_real1();` or equivalently: `sim.rng.random()`

Colours and state names
We won’t repeat CACATOO’s manual here. Please read this.

Other
Not in here? Look in the CACATOO documentation!

Written by Dieter Stoker, 2022. Last update: 15-02-2022. Based on Bram van Dijk’s excellent documentation.