Running CACATOO on the command line

Requirements

1. You need to install Node.js and NPM and described here:https://nodejs.org/en/download
   To check if they are already installed, type in the command line `node -v` or `npm -v`
2. Use NPM to install yargs:

   `npm install yargs`
3. Download `cacatoo` repository from https://github.com/bramvandijk88/cacatoo/

Method 1: Moving from HTML version to .js file

This method helps you create a *clean* JavaScript file that is used for running on the command line without any unused functions. In general, I would recommend method 2.

1. Create a file with the `.js` extension within the `examples` folder of the downloaded Cacatoo repository and use a text editor such as VScode to write in it.
2. Copy the user-defined code, i.e. all of the content in the JavaScript panel on JSfiddle. (If you are converting a `.html` file to `.js`, this means copying the contents within `<spript>` and `</script>`)
3. At the beginning of the code, add these lines to import the requirements.

   ```
   Simulation = require("PATH_TO_CACATOO")
   ```

   If your `.js` file is in the `examples` folder, then the path is `"../dist/cacatoo.js"`. With each subfolder within the `examples` folder, add an additional `../` to the path.
4. Currently, the code in is wrapped inside a `function cacatoo()`. Since `cacatoo.js` is required in the project and it already recognizes everything

without needing the wrapping function, you can remove `function cacatoo()` entirely, i.e. just take the code inside the function and remove the lines `function cacatoo() {` and the `}` in the end, and remove the indentation for the previously wrapped code.

5. Remove the lines for creating display and interactive elements as these are only useful for the web version of the code. This includes `sim.createDisplay...`, plots, buttons as well as the display related option in `config`.
6. Add lines to compute statistic(s) that you want as output from the simulations.
7. Add lines for writing an output. You may use `sim.log(STRING)` to display the input on the command line while the code is running. To write into a file, use `sim.write_append(STRING, "PATH_TO_OUTPUT_FILE")` within `sim.model.update`. If the output file name doesn't exist, it will create a new file and write in it, but make sure the directories in the path exist. Within the string, use `\\t` to insert a horizontal tab and `\\n` to create a new line. If you want to write a line only every 100 time steps for example, use the writing function within the IF statement `if (sim.time%100 == 0)`.
8. To run the script, simply enter `node FILENAME` on the command line.

Method 2 (simpler): Prepare a .js file that can be run on Node as well as on the browser

Using this method, you retain the ability to run the same code on the browser as well.

1. Create a file with the `.js` extension within the `examples` folder of the downloaded Cacatoo repository and use a text editor such as VScode to write in it.
2. Copy the user-defined code, i.e. all of the content in the JavaScript panel on JSfiddle. (If you are converting a `.html` file to `.js`, this means copying the contents within `<spript>` and `</script>`)
3. At the beginning of the code, add these lines to import the requirements.

```
if (typeof window == "undefined")  {
    Simulation = require("PATH_TO_CACATOO")
}
```

If your `.js` file is in the `examples` folder, then the path is `"../dist/cacatoo.js"`. With each subfolder within the `examples` folder, add an additional `../` to the path.
At the end of the code, add the following line:

```
if (typeof window == "undefined") cacatoo()
```

4. As you cannot visualize graphs or grids when running the script on Node, you need to compute some output statistic(s). Add lines within the code to compute statistic(s) that you want to read from the simulations.
5. Add lines for writing the output. You may use `sim.log(STRING)` to display them on the command line while the code is running. To write into a file, use `sim.write_append(STRING, "PATH_TO_OUTPUT_FILE")` within `sim.model.update`. If the output file name doesn't exist, it will create a new file and write in it, but make sure the directories in the path exist. Within the string, use `\\t` to insert a horizontal tab and `\\n` to create a new line. If you want to write a line only every 100 time steps for example, use the writing function within the IF statement `if (sim.time%100 == 0)`.
6. To run this script on the command line, simply enter `node FILENAME`.
7. To run this script on the browser, create a `.html` file in the same directory and enter the following and replace `FILENAME` with the name of the `.js` file:

```
<html>
<script src="../dist/cacatoo.js"></script> <!-- Include
cacatoo library (compiled with rollup) -->
<script src="../lib/all.js"></script>        <!-- Load o
ther packages -->


<link rel="stylesheet" href="../style/cacatoo.css"> <!--
Set style sheet -->

<!-- Include your model code here -->
<script src=FILENAME> </script>

<body onload="cacatoo()">
    <div class="header" id="header">
```

```
        <h2>Cacatoo </h2>
    </div>
    <div class="content" id="canvas_holder"></div>
    <div class="content" id="form_holder"></div>
    <div class="content" id="graph_holder"></div>
    <div class="footer" id="footer"></div>
    <div class="output" id="output"></div>
</body>
</html>
```

NOTE: The `sim.write_append` function does not work when the script is run in the browser mode. You could instead use `sim.log` to display the output on the browser page as the simulation runs.

Read parameter value from the command line

If you wish to input parameter values using the command line, enter on the command line:

```
node FILENAME --parameter PARAMETER_VALUE
```

Now the `.js` file needs to read the parameter value from the command line.If you used Method 1 to create the `.js` file, type the following at the start of the code where you define the simulation parameters. Instead of entering the parameter value in the script, we will import the parameter value from the command line:

```
let yargs = require('yargs')
let cmd_params = yargs.argv

var parameter = cmd_params.parameter;
```

If you used Method 2, you still want to be able to use default parameter values for the browser version. Therefore, we define parameters as earlier and reassign parameter values if we are running it from the command line. Here is what the beginning of the code will look like:

```
var parameter = 2; //value here will be used for the browse
r version
```

```
if (typeof window == "undefined")  {
    Simulation = require('../dist/cacatoo.js')
    let yargs = require('yargs')
    let cmd_params = yargs.argv

    var parameter = cmd_params.parameter;
}
```

Parameter sweep

Before proceeding, make sure you have modified the code to read parameter values from the command line as described above. Typically, you might want to run the script for many values of the parameter. You can do this by running a loop as follows

```
for i in {1..N}; do echo node FILENAME --parameter $i; done
```

where the parameter takes integer values from 1 to N. Now, the run for each parameter value happens one after the other. To parallelize these runs, use the `parallel` command as

```
for i in {1..N}; do echo node FILENAME --parameter $i; done
| parallel -j ncores
```

where `ncores` is the number of CPUs you want to use to run the scripts parallelly. Now, the script will run for the first `ncores` parameter values while the remaining are queued to start running when one of the CPUs becomes available. Make sure `parallel` is installed. Check by running `parallel --version` on the command line.

Example implementation

Given below is the .js implementation of this JSfiddle code (https://jsfiddle.net/sultannazir/3rzugydo/22/) that can be run on the command line with parameter `species` which is the number of neutral strain species and `iter` which is simply an integer to mark the iteration of the stochastic run. For

example, if it is saved in the same folder as `cacatoo.js` with the filename `neutral_species.js`, then the command line reads

```
for i in {1..3}; do for j in {1..5}; do echo node neutral_species.js --species $i --iter $j; done; done | parallel -j 3
```

Here is full script for `neutral_species.js` using Method 1:

```
Simulation = require('../dist/cacatoo')
let yargs = require('yargs')
let cmd_params = yargs.argv

let sim;

let n_species = cmd_params.species;
let iter = cmd_params.iter;

let config = {
    title: "Neutral strains",
    description: "Observe",
    maxtime: 10000,
    ncol: 200,
    nrow: 200, // dimensions of the grid to build
    wrap: [true, true] // Wrap boundary [COLS, ROWS]
}

sim = new Simulation(config) // Initialise a new sim instance with configuration given above

sim.makeGridmodel("neutral");

sim.initialGrid(sim.neutral, "species", 0, 1.0)

for (let i = sim.neutral.nc / 2 - 100; i < sim.neutral.nc / 2 + 100; i++) // i are columns
    for (let j = sim.neutral.nr / 2 - 100; j < sim.neutral.nr / 2 + 100; j++) // j are rows
```

```
        sim.neutral.grid[i][j].species = Math.ceil(sim.rng.
genrand_real1() * n_species)


sim.neutral.nextState = function(i, j) {
    let randomNeighbour = this.randomMoore8(this, i, j)
    if (this.grid[i][j].species == 0) {
        if (randomNeighbour.species != 0) {
            if (sim.rng.genrand_real1() <= 0.6) {
                this.grid[i][j].species = randomNeighbour.s
pecies
            }
        }
    } else {
        if (sim.rng.genrand_real1() <= 0.3) {
            this.grid[i][j].species = 0
        }
    }
}

sim.neutral.update = function() {
    this.synchronous()

    // Computing statistics at each time step to write to o
utput
    var sp1 = 0
    var tot_sp = 0
    for (let i = 0; i < sim.neutral.nc; i++) for (let j =
0; j < sim.neutral.nr; j++) {
        if (this.grid[i][j].species != 0) tot_sp ++
        if (this.grid[i][j].species == 1) sp1 ++
    }

    var time = sim.time

    if(time%50 == 0) // write output every 50 time steps
        sim.write_append(time+'\t'+n_species+'\t'+iter
+'\t'+sp1+'\t'+tot_sp+'\n', 'output_neutral.dat')
```

```
}

sim.start()
```

Here is full script for `neutral_species.js` using Method 2:

```
let sim;
var n_species = 2;
var iter = 2;

if (typeof window == "undefined")  {
    Simulation = require('../dist/cacatoo.js')
    let yargs = require('yargs')
    let cmd_params = yargs.argv
    var n_species = cmd_params.species;
    var iter = cmd_params.iter;
}

function cacatoo() {

  let config = {
    title: "Neutral strains",
    description: "Observe",
    maxtime: 100000,
    ncol: 200,
    nrow: 200, // dimensions of the grid to build
    sleep: 0,
    fastmode: true, // Ironically, to slow down the FPS, yo
u have to be in fast mode :D
    wrap: [true, true], // Wrap boundary [COLS, ROWS]
    scale: 2, // scale of the grid (nxn pixels per grid cel
l)
    graph_interval: 1,
    graph_update: 1,
    statecolours: {
      species: 'default'
```

```
    },
    num_colours: 9
  }

  sim = new Simulation(config) // Initialise a new sim inst
ance with configuration given above

  sim.makeGridmodel("neutral");
  sim.createDisplay("neutral", "species")
  sim.initialGrid(sim.neutral, "species", 0, 1.0)
  for (let i = sim.neutral.nc / 2 - 100; i < sim.neutral.nc
/ 2 + 100; i++) // i are columns
    for (let j = sim.neutral.nr / 2 - 100; j < sim.neutral.
nr / 2 + 100; j++) // j are rows
      sim.neutral.grid[i][j].species = Math.ceil(sim.rng.ge
nrand_real1() * n_species)
  sim.display()



  sim.neutral.nextState = function(i, j) {
    let randomNeighbour = this.randomMoore8(this, i, j)

    if (this.grid[i][j].species == 0) {
      if (randomNeighbour.species != 0) {
        if (sim.rng.genrand_real1() <= 0.6) {
          this.grid[i][j].species = randomNeighbour.species
        }
      }
    } else {
      if (sim.rng.genrand_real1() <= 0.3) {
        this.grid[i][j].species = 0
      }
    }

  }
```

```
  sim.neutral.update = function() {
    this.synchronous()
    this.plotPopsizes("species", [1, 2, 3, 4, 5, 6, 7, 8,
9])
    // Computing statistics at each time step to write to o
utput
    var sp1 = 0
    var tot_sp = 0
    for (let i = 0; i < sim.neutral.nc; i++) for (let j =
0; j < sim.neutral.nr; j++) {
      //sim.log(this.grid[i][j].species)
      if (this.grid[i][j].species != 0) tot_sp ++
      if (this.grid[i][j].species == 1) sp1 ++
    }
    var time = sim.time
    if(time%50 == 0){ // write output every 50 time steps
        sim.write_append(time+'\t'+n_species+'\t'+iter
+'\t'+sp1+'\t'+tot_sp+'\n', 'output_neutral.dat')
    }
  }

  // Adds a pattern load button
  sim.addButton("run/pause", function() {
    sim.toggle_play()
  })
  sim.addButton("update 1 step", function() {
    sim.step();
    sim.display()
  })


  sim.start()
  sim.toggle_play()
}

if (typeof window == "undefined") cacatoo()
```