

---

# **cellsium**

***Release 1.0.0***

**Christian C. Sachs**

**Jun 02, 2023**



## CONTENTS:

<b>1</b>	<b>CellSium Readme</b>	<b>1</b>
1.1	Front Matter . . . . .	1
1.2	Installation . . . . .	2
1.3	Usage . . . . .	2
1.4	Docker . . . . .	3
<b>2</b>	<b>License</b>	<b>5</b>
<b>3</b>	<b>Examples</b>	<b>7</b>
3.1	Creating random training data . . . . .	7
3.2	Creating a timelapse simulation . . . . .	8
3.3	Adding a custom cell model . . . . .	10
3.4	Jupyter Notebook Embedding Example . . . . .	11
<b>4</b>	<b>cellsium package</b>	<b>15</b>
4.1	cellsium.cli package . . . . .	15
4.2	cellsium.geometry package . . . . .	20
4.3	cellsium.model package . . . . .	22
4.4	cellsium.output package . . . . .	32
4.5	cellsium.simulation package . . . . .	42
4.6	cellsium.random module . . . . .	50
4.7	cellsium.parameters module . . . . .	51
4.8	cellsium.typing module . . . . .	55
<b>5</b>	<b>Indices and tables</b>	<b>57</b>
	<b>Python Module Index</b>	<b>59</b>
	<b>Index</b>	<b>61</b>



## CELLSIUM README

CellSium - *Cell Simulator for microfluidic microcolonies*

Fig. 1: CellSium example simulation result

### 1.1 Front Matter

CellSium is a cell simulator developed for the primary application of generating realistically looking images of bacterial microcolonies, which may serve as ground truth for machine learning training processes.

#### 1.1.1 Publication

If you use CellSium within scientific research, we ask you to cite our publication:

Sachs CC, Ruzaeva K, Seiffarth J, Wiechert W, Berkels B, Nöh K (2022) CellSium: versatile cell simulator for microcolony ground truth generation *Bioinformatics Advances*, Volume 2, Issue 1, 2022, vbac053, doi: 10.1093/bioadv/vbac053

It is available on the *Bioinformatics Advances* homepage at DOI: 10.1093/bioadv/vbac053 <<https://doi.org/10.1093/bioadv/vbac053>>.

## 1.1.2 Documentation

The documentation to CellSium can be built using [Sphinx](#), or be found readily built at [Read the Docs](#).

## 1.1.3 License

CellSium is available under the BSD license (see LICENSE.rst / license section).

# 1.2 Installation

## 1.2.1 Installation using pip

CellSium can be installed via pip, ideally create and activate an environment beforehand to install CellSium in.

```
> python -m pip install cellsium
```

## 1.2.2 Installation using conda

CellSium is available in the modsim Anaconda channel as well, using packages from the conda-forge channel. It can be installed with the following commands:

```
> conda install -c modsim -c conda-forge -y cellsium
```

# 1.3 Usage

Once installed, run CellSium via `python -m cellsium`, specifying the desired entrypoint and options, such as outputs. CellSium is built modular, various output modules can be activated simultaneously. To get an overview of the available options, use the `--help` switch. Furthermore, the main mode of setting tunable parameters are so called *tunables*, which can be set from the command line using the `-t` switches. A list of tunables can be shown using the `--tunables-show` argument.

```
> python -m cellsium --help
usage: __main__.py [-v] [-q] [-c CELL] [-p] [-w] [-o OUTPUT] [-h] [-m MODULE]
                  [--Output {COCOOutput,CsvOutput,FluorescenceRenderer,GenericMaskOutput,
↪JsonPickleSerializer,MeshOutput,NoisyUnevenIlluminationPhaseContrast,
↪PhaseContrastRenderer,PlainRenderer,PlotRenderer,QuickAndDirtyTableDumper,SvgRenderer,
↪TiffOutput,TrackMateXML,UnevenIlluminationPhaseContrast,YOLOOutput}]
                  [--PlacementSimulation {Box2D,Chipmunk,NoPlacement}] [-t TUNABLE] [--
↪tunables-show] [--tunables-load TUNABLES_LOAD] [--tunables-save TUNABLES_SAVE]

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output-file OUTPUT
  -w, --overwrite
  -p, --prefix
  -c CELL, --cell CELL
  -q, --quiet
```

(continues on next page)

(continued from previous page)

```

-v, --verbose
-m MODULE, --module MODULE
--Output {COCOOutput,CsvOutput,FluorescenceRenderer,GenericMaskOutput,
↪JsonPickleSerializer,MeshOutput,NoisyUnevenIlluminationPhaseContrast,
↪PhaseContrastRenderer,PlainRenderer,PlotRenderer,QuickAndDirtyTableDumper,SvgRenderer,
↪TiffOutput,TrackMateXML,UnevenIlluminationPhaseContrast,YOLOOutput}
--PlacementSimulation {Box2D,Chipmunk,NoPlacement}
-t TUNABLE, --tunable TUNABLE
--tunables-show
--tunables-load TUNABLES_LOAD
--tunables-save TUNABLES_SAVE

```

You can for example run a default simulation by just starting CellSium, the results will be shown interactively using matplotlib:

```
> python -m cellsium
```

For more in-depth usage examples, please see the examples section of the documentation.

## 1.4 Docker

An alternative to installing CellSium locally is running it via Docker. To run CellSium without interactive (GUI) elements, the following Docker command can be used, with parameters to CellSium being appended.

```

> docker run --tty --interactive --rm --volume `pwd`: /data --user `id -u` ghcr.io/modsim/
↪cellsium

```

To use interactive (GUI) elements such as the PlotRenderer, an X server must be reachable; under Linux the following command can be used:

```

> docker run --tty --interactive --rm --volume `pwd`: /data --user `id -u` --env DISPLAY=
↪$DISPLAY --volume /tmp/.X11-unix:/tmp/.X11-unix ghcr.io/modsim/cellsium

```





**LICENSE**

Copyright (c) 2015-2021 Christian C. Sachs, Forschungszentrum Jülich GmbH All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



## EXAMPLES

The following sections contain some usage examples of CellSium.

### 3.1 Creating random training data

The core mode of operation is the creation of ground truth data as training data for machine learning/deep learning applications. To this end, CellSium contains two output modes specifically tailored to produce outputs for common deep learning based object detectors/instance segmentation toolkits: The COCO and YOLO format. CellSium can as well just output binary masks along the images for use with other learning tools.

For example, the following command will random cell images, and output three datasets:

```
> python -m cellsium training \
  -t TrainingDataCount=64 \
  -t TrainingCellCount=512 \
  -t TrainingImageWidth=512 \
  -t TrainingImageHeight=512 \
  -t Calibration=0.0905158 \
  -t ChipmunkPlacementRadius=0.01 \
  -o training \
  --Output COCOOutput \
  --Output YOLOOutput \
  --Output GenericMaskOutput \
  -p
```

Note how the main mode of configuration of CellSium are *tunables*, these tunable parameters are set using the `-t` argument, followed by `Name=value`. The tunables are explained in the documentation, and can be listed via `--tunables-show` as well.

In this example, the output of 64 images of 512x512 size are requested, setting the pixel calibration to 0.0905158  $\mu\text{m}$  per pixel. `ChipmunkPlacementRadius` configures the physical placement and yields denser colonies. The name of the output files/directories is specified using `-o`.

The outputs of CellSium are modular. In this example, the `COCOOutput`, `YOLOOutput`, and `GenericMaskOutput` are enabled. As to prevent name clashes and allow easy coexistence, the `-p:code:` switch enables prefixing of the output names with the name of the respective output module.

Once the example has run, the directories `COCOOutput-training`, `GenericMaskOutput-training`, and `YOLOOutput-training` have been created, with examples of the `GenericMaskOutput` shown.

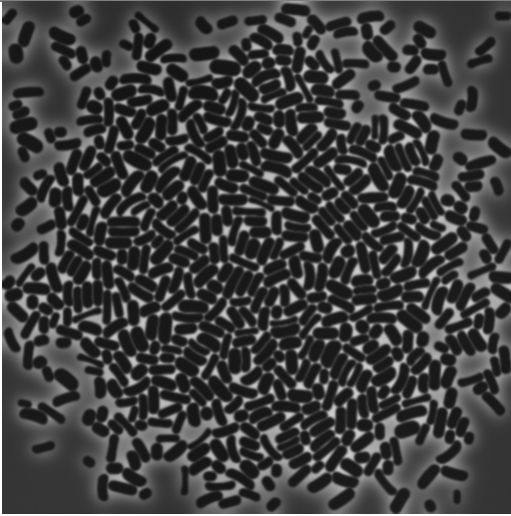


Fig. 1: GenericMaskOutput-training/images/00000000000000.png

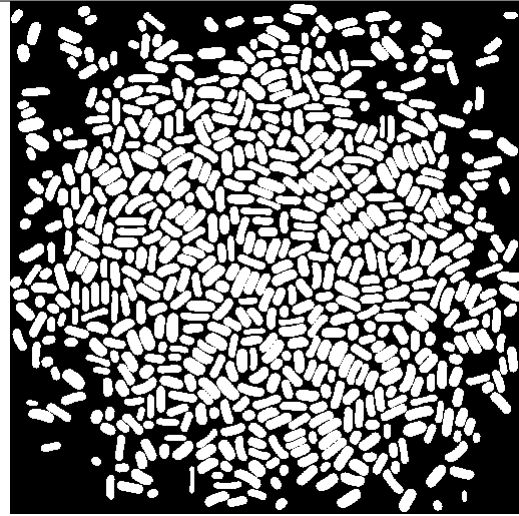


Fig. 2: GenericMaskOutput-training/masks/00000000000000.png

## 3.2 Creating a timelapse simulation

CellSium was originally developed to create time lapse simulations to create realistic microcolonies, which can serve as input data e.g., for simulations based on the geometry or the training and validation of tracking algorithms. To run a simulation, set up the desired outputs and tunables as explained, and use the `simulate` subcommand.

```
> python -m cellsium simulate \  
  -o simulate \  
  --Output GenericMaskOutput \  
  --Output TiffOutput \  
  -p
```

In this case, a microcolony will be simulated, a time lapse TIFF stack as well as mask output generated. Shown are three example images:

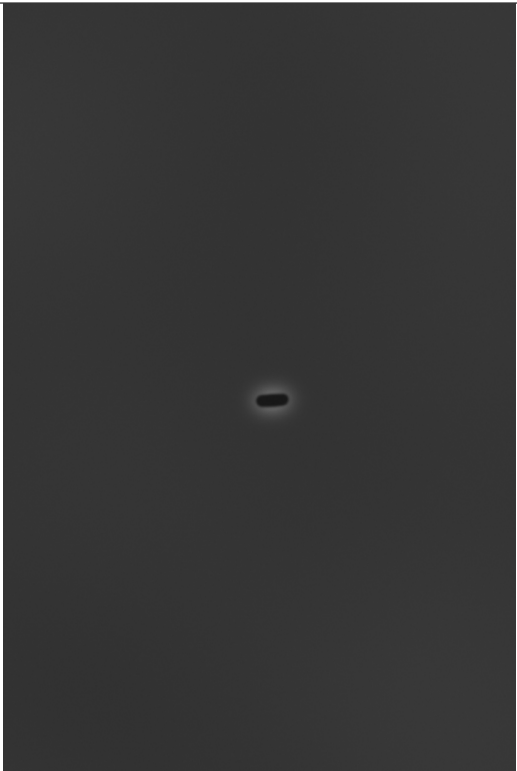


Fig. 3: GenericMaskOutput-training/images/00000000000000.png



Fig. 4: GenericMaskOutput-training/images/00000000000016.png

### 3.3 Adding a custom cell model

In the previous examples, the standard (sizer) cell model was used. However, the modular nature of CellSium makes it easy to integrate a custom cell model. In this example, the sizer model will be defined externally, so it can be more easily changed, and to showcase its difference, the easter egg square geometry will be applied:

Listing 1: square.py

```
from cellsium.model import assemble_cell, SimulatedCell, h_to_s, Square

class SquareCellModel(SimulatedCell):
    @staticmethod
    def random_sequences(sequence):
        return dict(elongation_rate=sequence.normal(1.5, 0.25)) #  $\mu\text{m}\cdot\text{h}^{-1}$ 

    def birth(self, parent=None, ts=None) -> None:
        self.elongation_rate = next(self.random.elongation_rate)
        self.division_time = h_to_s(1.0)

    def grow(self, ts) -> None:
        self.length += self.elongation_rate * ts.hours

        if ts.time > (self.birth_time + self.division_time):
            offspring_a, offspring_b = self.divide(ts)
            offspring_a.length = offspring_b.length = self.length / 2

Cell = assemble_cell(SquareCellModel, Square)
```

The custom model can be specified using the `-c` switch, specifying either an importable Python module, or the path of a Python file. If no class name is specified after the `:` colon, CellSium will attempt to import a class named `Cell` from the file/module.

```
> python -m cellsium simulate \
  -o square \
  --Output GenericMaskOutput \
  -c square.py:Cell \
  -p
```

CellSium cell objects are Python objects. They are built lending from OOP principles, using mixins in a very flexible way to join various properties. To gain deeper insights how to implement and alter cellular behavior or rendering, it is best to study the source code of CellSium.

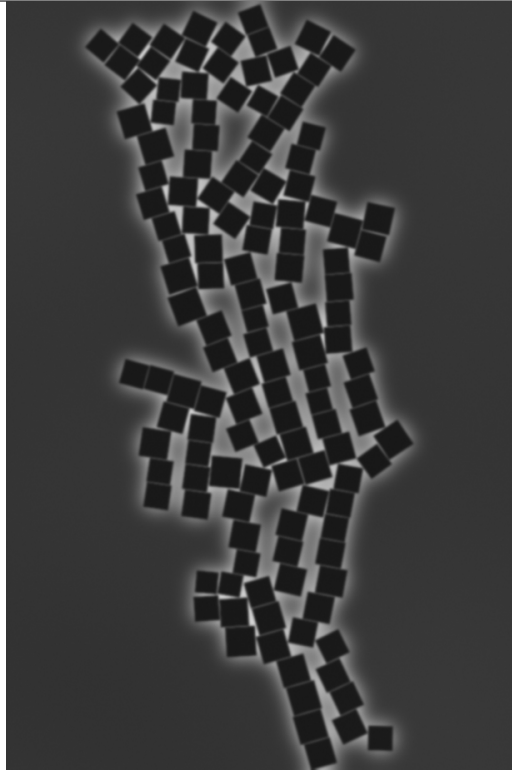


Fig. 6: GenericMaskOutput-square/images/0000000000033.png

### 3.4 Jupyter Notebook Embedding Example

In this example, CellSium is embedded in a Jupyter notebook to interactively run small simulations.

First, the necessary modules are imported:

```
# plotting
from matplotlib_inline.backend_inline import set_matplotlib_formats
set_matplotlib_formats('svg')
from matplotlib import pyplot

# general
from functools import partial

# RRF is the central random helper, used for seeding
from cellsium.random import RRF
# the model parts the new model is built upon
from cellsium.model import PlacedCell, SimulatedCell, assemble_cell
# the functions to actually perform the simulation
from cellsium.cli.simulate import perform_simulation, initialize_cells, h_to_s, s_to_h
# the PlotRenderer as it embeds nicely in Jupyter
from cellsium.output.plot import PlotRenderer
```

For the example, a model is defined directly within a Jupyter cell:

```

classSizerCell(SimulatedCell):
    @staticmethod
    def random_sequences(sequence):
        return dict(elongation_rate=sequence.normal(1.5, 0.25)) #  $\mu\text{m}\cdot\text{h}^{-1}$ 

    def birth(
        self, parent=None, ts=None
    ) -> None:
        self.elongation_rate = next(self.random.elongation_rate)
        self.division_time = h_to_s(0.5) # fast division rate

    def grow(self, ts):
        self.length += self.elongation_rate * ts.hours

        if ts.time > (self.birth_time + self.division_time):
            offspring_a, offspring_b = self.divide(ts)
            offspring_a.length = offspring_b.length = self.length / 2

```

```

# seed the random number generator
RRF.seed(1)

# perform_simulation returns an iterator which will indefinitely yield timesteps

simulation_iterator = perform_simulation(
    setup=partial(
        initialize_cells,
        count=1,
        cell_type=assemble_cell(SizerCell, placed_cell=PlacedCell)),
    time_step=30.0 * 60.0
)

# we step thru the first 5 of them ...
for _, ts in zip(range(5), simulation_iterator):
    # ... and plot them
    PlotRenderer().output(world=ts.world)
    pyplot.title("Simulation output at time=%.2fh" % (s_to_h(ts.time)))
    pyplot.show()
    # we have access to the Cell objects as well
    print(repr(ts.world.cells))

```

```

[Cell(angle=2.3294692309443428, bend_lower=-0.016261360844027475, bend_overall=-0.
↪04126808565523994, bend_upper=-0.09614075306364522, birth_time=0.0, division_time=1800.
↪0, elongation_rate=1.5981931788501658, id_=1, length=2.872894940610261, lineage_
↪history=[0], parent_id=0, position=[17.854225629902448, 28.67645476278087], width=0.
↪8960666826747447)]

```

```

[Cell(angle=2.3294692309443428, bend_lower=-0.016261360844027475, bend_overall=-0.
↪04126808565523994, bend_upper=-0.09614075306364522, birth_time=3600.0, division_
↪time=1800.0, elongation_rate=1.4017119040732795, id_=2, length=1.835995765017672,

```

(continues on next page)



(continued from previous page)

```

→ lineage_history=[0, 1], parent_id=1, position=[18.485770454254308, 28.010218132802386],
→ width=0.8960666826747447), Cell(angle=2.3294692309443428, bend_lower=-0.
→ 016261360844027475, bend_overall=-0.04126808565523994, bend_upper=-0.09614075306364522,
→ birth_time=3600.0, division_time=1800.0, elongation_rate=1.7743185975635118, id=3,
→ length=1.835995765017672, lineage_history=[0, 1], parent_id=1, position=[17.
→ 22268080555059, 29.342691392759356], width=0.8960666826747447))

```

```

[Cell(angle=2.3360626329715437, bend_lower=-0.016261360844027475, bend_overall=-0.
→ 04126808565523994, bend_upper=-0.09614075306364522, birth_time=3600.0, division_
→ time=1800.0, elongation_rate=1.4017119040732795, id=2, length=2.5368517170543115,
→ lineage_history=[0, 1], parent_id=1, position=[18.756872759489948, 27.72423023484169],
→ width=0.8960666826747447), Cell(angle=2.3228640880321927, bend_lower=-0.
→ 016261360844027475, bend_overall=-0.04126808565523994, bend_upper=-0.09614075306364522,
→ birth_time=3600.0, division_time=1800.0, elongation_rate=1.7743185975635118, id=3,
→ length=2.7231550637994277, lineage_history=[0, 1], parent_id=1, position=[16.
→ 951578500314948, 29.62867929072005], width=0.8960666826747447))

```

```

[Cell(angle=2.352798559960036, bend_lower=-0.016261360844027475, bend_overall=-0.
→ 04126808565523994, bend_upper=-0.09614075306364522, birth_time=7200.0, division_
→ time=1800.0, elongation_rate=0.8317980611709823, id=4, length=1.6188538345454755,
→ lineage_history=[0, 1, 2], parent_id=2, position=[19.593155424028517, 26.
→ 85358455987381], width=0.8960666826747447), Cell(angle=2.3416190585127383, bend_lower=-
→ 0.016261360844027475, bend_overall=-0.04126808565523994, bend_upper=-0.
→ 09614075306364522, birth_time=7200.0, division_time=1800.0, elongation_rate=1.
→ 2231841085165691, id=5, length=1.6188538345454755, lineage_history=[0, 1, 2], parent_
→ id=2, position=[18.471310193545275, 28.02153383690875], width=0.8960666826747447),
→ Cell(angle=2.314495314084782, bend_lower=-0.016261360844027475, bend_overall=-0.
→ 04126808565523994, bend_upper=-0.09614075306364522, birth_time=7200.0, division_
→ time=1800.0, elongation_rate=1.4247110511570817, id=6, length=1.8051571812905918,
→ lineage_history=[0, 1, 3], parent_id=3, position=[17.289158390942983, 29.
→ 25243321304336], width=0.8960666826747447), Cell(angle=2.304654468034007, bend_lower=-
→ 0.016261360844027475, bend_overall=-0.04126808565523994, bend_upper=-0.
→ 09614075306364522, birth_time=7200.0, division_time=1800.0, elongation_rate=1.
→ 8079038604401219, id=7, length=1.8051571812905918, lineage_history=[0, 1, 3], parent_
→ id=3, position=[16.06327851109302, 30.578267441297573], width=0.8960666826747447))

```

```

[Cell(angle=2.3820603181599442, bend_lower=-0.016261360844027475, bend_overall=-0.
→ 04126808565523994, bend_upper=-0.09614075306364522, birth_time=7200.0, division_
→ time=1800.0, elongation_rate=0.8317980611709823, id=4, length=2.0347528651309665,
→ lineage_history=[0, 1, 2], parent_id=2, position=[20.228643748180197, 26.
→ 199313752340714], width=0.8960666826747447), Cell(angle=2.3526475982043165, bend_
→ lower=-0.016261360844027475, bend_overall=-0.04126808565523994, bend_upper=-0.
→ 09614075306364522, birth_time=7200.0, division_time=1800.0, elongation_rate=1.
→ 2231841085165691, id=5, length=2.2304458880376, lineage_history=[0, 1, 2], parent_
→ id=2, position=[18.75284683312141, 27.731672298371393], width=0.8960666826747447),
→ Cell(angle=2.2968665429831496, bend_lower=-0.016261360844027475, bend_overall=-0.
→ 04126808565523994, bend_upper=-0.09614075306364522, birth_time=7200.0, division_

```

(continues on next page)

(continued from previous page)

```
→time=1800.0, elongation_rate=1.4247110511570817, id_=6, length=2.5175127068691325,
→lineage_history=[0, 1, 3], parent_id=3, position=[17.088954083136315, 29.
→417294148769457], width=0.8960666826747447), Cell(angle=2.2660140670444004, bend_
→lower=-0.016261360844027475, bend_overall=-0.04126808565523994, bend_upper=-0.
→09614075306364522, birth_time=7200.0, division_time=1800.0, elongation_rate=1.
→8079038604401219, id_=7, length=2.709109111510653, lineage_history=[0, 1, 3], parent_
→id=3, position=[15.346457855171874, 31.35753885164192], width=0.8960666826747447))
```

## CELLSIUM PACKAGE

CellSium - `_Cell_ _Si_mulator` for `_micro_fluidic _m_icrocolonies`

### 4.1 cellsium.cli package

CLI package, home to the individual entry points

`cellsium.cli.Cell`

alias of *SizerCell*

**class** `cellsium.cli.SizerCell(**kwargs)`

Bases: *PlacedCell, SizerCell*

Cell.

**class** `cellsium.cli.TimerCell(**kwargs)`

Bases: *PlacedCell, TimerCell*

Cell.

`cellsium.cli.add_output_prefix(output_name: str, output: Output) → str`

Adds an prefix to an output filename.

**Parameters**

- **output\_name** – Output name
- **output** – Output object

**Returns**

Name

`cellsium.cli.initialize_cells(simulator: Simulator, count: int = 1, cell_type: Optional[PlacedCell] = None, sequence: Optional[Any] = None) → Simulator`

Initialize cells and add them to a simulator.

**Parameters**

- **simulator** – Simulator to add cells to.
- **count** – Count of cells to generate
- **cell\_type** – cell type to use
- **sequence** – Random number sequence to use

**Returns**

Simulator

`cellsium.cli.initialize_simulator()` → *Simulator*

Constructor helper for a simulator.

**Returns**

Simulator instance

## 4.1.1 cellsium.cli.render package

Rendering CLI utility, render a simulation state saved using jsonpickle.

`cellsium.cli.render.subcommand_argparser(parser: ArgumentParser)` → None

Handle the argument parser for the 'render' subcommand.

**Parameters**

**parser** – Argument parser

**Returns**

None

`cellsium.cli.render.subcommand_main(args: Namespace)` → None

Entry point for the 'render' subcommand.

**Parameters**

**args** – Pre-parsed arguments

**Returns**

None

## 4.1.2 cellsium.cli.simulate package

Simulation CLI endpoint.

`class cellsium.cli.simulate.BoundariesFile(*args, **kwargs)`

Bases: Tunable

Boundaries file (in DXF format) to add boundaries/geometrical constraints

**default:** `str = ''`

**value** = ''

`class cellsium.cli.simulate.BoundariesScaleFactor(*args, **kwargs)`

Bases: Tunable

Scale factor for the boundaries

**default:** `float = 1.0`

**value** = 1.0

`class cellsium.cli.simulate.SimulationDuration(*args, **kwargs)`

Bases: Tunable

Time (simulated) the simulation should run

**default:** `float = 12.0`

**value** = 12.0

```
class cellsium.cli.simulate.SimulationOutputFirstState(*args, **kwargs)
```

Bases: Tunable

Whether to output the first state

**default:** bool = False

**value** = False

```
class cellsium.cli.simulate.SimulationOutputInterval(*args, **kwargs)
```

Bases: Tunable

Time intervals (simulated) at which an output should be written

**default:** float = 0.25

**value** = 0.25

```
class cellsium.cli.simulate.SimulationTimestep(*args, **kwargs)
```

Bases: Tunable

Time step at which the simulation state should be calculated

**default:** float = 0.016666666666666666

**value** = 0.016666666666666666

```
cellsium.cli.simulate.add_boundaries_from_dxf(file_name: str, simulator: Simulator, scale_factor: float
                                             = 1.0) → None
```

Add boundaries from a DXF file. Supported are LWPolyline and Polyline objects.

#### Parameters

- **file\_name** – dxf file name
- **simulator** – Simulator instance to add the boundaries to
- **scale\_factor** – Scale factor for the geometry

#### Returns

None

```
cellsium.cli.simulate.add_boundaries_from_tunables(simulator: Simulator) → Simulator
```

Add boundaries to a simulator if the appropriate tunables are set.

#### Parameters

**simulator** – Simulator

#### Returns

Simulator

```
cellsium.cli.simulate.compose(*args)
```

Compose callable args.

#### Parameters

**args** – Args

#### Returns

Callable

`cellsium.cli.simulate.initialize_output_times_from_tunables()`

Initialize the duration, output\_interval and last\_output variables, from tunables.

**Returns**

duration, output\_interval, last\_output

`cellsium.cli.simulate.measure_duration(iterator: Iterator[T]) → Iterator[Tuple[float, T]]`

Measure the (wall clock) time it takes to step forward the iterator, yielding the duration and iterator value.

**Parameters**

**iterator** – Iterator

**Returns**

Iterator of duration and iterator value

`cellsium.cli.simulate.perform_outputs(world: World, simulation_time: float, outputs: Iterable[Output],  
output_name: Optional[str] = None, overwrite: bool = False,  
prefix: bool = False, output_count: int = 0) → None`

Performs the output operations configured.

**Parameters**

- **world** – World to output
- **simulation\_time** – Simulation timepoint
- **outputs** – Outputs
- **output\_name** – Name to output to
- **overwrite** – Whether to overwrite
- **prefix** – Whether to prefix the outputs with the name of the Output type
- **output\_count** – The count of already outputted timesteps

**Returns**

None

`cellsium.cli.simulate.perform_simulation(simulator: Optional[Simulator] = None, setup:  
Optional[Callable[[Simulator], Simulator]] = None,  
time_step: float = 1.0) → Iterator[Timestep]`

Simple entrypoint to perform simulations. Will either accept a simulator instance or create one, call the setup callback on it if set, and then yield timesteps time\_step apart indefinitely.

**Parameters**

- **simulator** – Simulator
- **setup** – Setup callable
- **time\_step** – Time step

**Returns**

Iterator of Timestep instances

`cellsium.cli.simulate.prepare_output_name(output_name: str, output: Output, prefix: str) → str`

Prepare an output name.

**Parameters**

- **output\_name** – Output name
- **output** – Output object

- **prefix** – Prefix

#### Returns

Output name

`cellsium.cli.simulate.subcommand_main(args: Namespace) → None`

Entry point for the ‘simulate’ subcommand.

#### Parameters

**args** – pre-parsed arguments

#### Returns

None

### 4.1.3 cellsium.cli.training package

Training Generation CLI entypoint.

**class** `cellsium.cli.training.TrainingCellCount(*args, **kwargs)`

Bases: Tunable

Cells to add to training samples

**default:** `int = 32`

**value** = 32

**class** `cellsium.cli.training.TrainingDataCount(*args, **kwargs)`

Bases: Tunable

Training samples to generate

**default:** `int = 16`

**value** = 16

**class** `cellsium.cli.training.TrainingImageHeight(*args, **kwargs)`

Bases: Tunable

Image height in pixels of training images

**default:** `int = 128`

**value** = 128

**class** `cellsium.cli.training.TrainingImageWidth(*args, **kwargs)`

Bases: Tunable

Image width in pixels of training images

**default:** `int = 128`

**value** = 128

`cellsium.cli.training.subcommand_main(args: Namespace) → None`

Entry point for the ‘training’ subcommand.

#### Parameters

**args** – pre-parsed arguments

#### Returns

None

#### 4.1.4 cellsium.cli.cli module

CLI entrypoint.

`cellsium.cli.cli.load_class_from_module(module_class: str, default_class_name: str) → type`

`cellsium.cli.cli.main(args: Optional[Iterable[str]] = None) → Optional[int]`

Main entrypoint of the script, will redirect to various sub-scripts.

**Parameters**

**args** – arguments, if not specified will be taken from sys.argv

**Returns**

The return code of the individual subcommand

`cellsium.cli.cli.parse_arguments_and_init(args: Iterable[str], parser_callback: Optional[Callable[[ArgumentParser], None]] = None) → Namespace`

Basic setup (i.e. logging) and argument parsing.

**Parameters**

- **args** – Arguments
- **parser\_callback** – Additional callback to configure the argument parser

**Returns**

Parsed arguments

## 4.2 cellsium.geometry package

Various geometry handling functions.

`cellsium.geometry.add_empty_third_dimension(array: ndarray) → ndarray`

Adds an empty third dimension.

**Parameters**

**array** – 2D Array

**Returns**

3D Array

`cellsium.geometry.circle_segment(radius: float, start: ndarray, stop: ndarray, interval: float = 0.1, minimum_times: int = 10, times: Optional[int] = None) → ndarray`

Rasters a circle segment from start to stop with a radius radius.

**Parameters**

- **radius** – Radius
- **start** – Start point
- **stop** – Stop point
- **interval** – Interval
- **minimum\_times** – Minimal count of points to put between start and stop
- **times** – Alternatively: count of points to place



**Returns**

Coordinates

`cellsium.geometry.line(start: ndarray, stop: ndarray, interval: float = 0.1, minimum_times: int = 10, times: Optional[int] = None) → ndarray`

Rasters a line from start to stop with points at interval interval.

**Parameters**

- **start** – Start point
- **stop** – Stop point
- **interval** – Interval
- **minimum\_times** – Minimal count of points to put between start and stop
- **times** – Alternatively: count of points to place

**Returns**

Coordinates

`cellsium.geometry.parabolic_deformation(array: ndarray, factor: float) → ndarray`

Deform array by a parabola.

**Parameters**

- **array** – Coordinates
- **factor** – Factor

**Returns**

Deformed coordinates

`cellsium.geometry.rotate(data: ndarray, angle: float) → ndarray`

Rotates data by angle.

**Parameters**

- **data** – Coordinates
- **angle** – Angle

**Returns**

Rotated coordinates

`cellsium.geometry.rotate3d(data: ndarray, angle: float, axis_vector: ndarray) → ndarray`

Rotates data within 3D space around axis\_vector.

**Parameters**

- **data** – Coordinates
- **angle** – Angle
- **axis\_vector** – Axis vector of the rotation

**Returns**

Rotated points

`cellsium.geometry.rotate_and_mesh(points: ndarray, steps: int = 16, clean: bool = True, close_ends: bool = True) → Tuple[ndarray, ndarray]`

Produces a solid of revolution.

**Parameters**

- **points** – Coordinates
- **steps** – Steps of the revolution
- **clean** – Whether to clean the data beforehand
- **close\_ends** – Whether to close the ends

**Returns**

Tuple(Array of Points, Triangle Indices)

`cellsium.geometry.shift(data: ndarray, vector: ndarray) → ndarray`

Shifts coordinates.

**Parameters**

- **data** – Coordinates
- **vector** – Shift vector

**Returns**

Shifted coordinates

## 4.3 cellsium.model package

Cell model package.

**class** `cellsium.model.AutoMesh3D`

Bases: [\*Shape3D\*](#)

Mixin adding automatic solid-of-revolution generation.

**points3d\_on\_canvas**(steps: int = 16, simplify: bool = False) → Tuple[ndarray, ndarray]

**raw\_points3d**(steps: int = 16, simplify: bool = False) → Tuple[ndarray, ndarray]

**class** `cellsium.model.BentRod`

Bases: [\*RodShaped\*](#)

Bent rod shaped cell geometry.

**bend**(points: ndarray) → ndarray

**static defaults**() → Dict[str, Union[Callable[[], Any], float]]

**get\_approximation\_circles**() → Iterator[Tuple[float, Tuple[float, float]]]

**raw\_points**(simplify: bool = False) → ndarray

**raw\_points3d**(steps: int = 16, simplify: bool = False) → Tuple[ndarray, ndarray]

**class** `cellsium.model.CellGeometry`

Bases: [\*WithAngle\*](#), [\*WithPosition\*](#), [\*AutoMesh3D\*](#)

Cell geometry base by combining multiple mixins.

**points\_on\_canvas**() → ndarray

**class** cellsium.model.Coccoid

Bases: *Shape*

Coccoid (spherical) cell geometry.

**static defaults()** → Dict[str, Union[Callable[[], Any], float]]

**get\_approximation\_circles()** → Iterator[Tuple[float, Tuple[float, float]]]

**raw\_points**(*simplify: bool = False*) → ndarray

**class** cellsium.model.Copyable

Bases: object

Mixin for copyable objects.

**copy()** → *Copyable*

**class** cellsium.model.Ellipsoid

Bases: *Coccoid*

Ellipsoid cell geometry.

**static defaults()** → Dict[str, Union[Callable[[], Any], float]]

**raw\_points**(*simplify: bool = False*) → ndarray

**class** cellsium.model.IdCounter

Bases: object

Id provider singleton class.

**id\_counter:** int = 0

**classmethod next\_cell\_id()** → int

**classmethod reset()** → None

**class** cellsium.model.InitializeWithParameters(\*\*kwargs)

Bases: object

Mixin for objects with defaults.

**class** cellsium.model.PlacedCell(\*\*kwargs)

Bases: *WithLineageHistory*, *WithLineage*, *WithTemporalLineage*, *WithProperDivisionBehavior*, *InitializeWithParameters*, *Copyable*, *Representable*, *WithRandomSequences*, *RandomWidthLength*, *RandomBentRod*, *RandomPosition*, *RandomAngle*, *CellGeometry*, *BentRod*

**class** cellsium.model.Rectangle

Bases: *Shape*

Rectangular cell geometry.

**static defaults()** → Dict[str, Union[Callable[[], Any], float]]

**get\_approximation\_circles()** → Iterator[Tuple[float, Tuple[float, float]]]

**raw\_points**(*simplify: bool = False*) → Dict[str, Union[Callable[[], Any], float]]

**class** cellsium.model.Representable

Bases: object

Mixins for adding a repr implementation.

**class** cellsium.model.RodShaped

Bases: [Shape](#)

Rod shaped cell geometry.

**static defaults**() → Dict[str, Union[Callable[[], Any], float]]

**get\_approximation\_circles**() → Iterator[Tuple[float, Tuple[float, float]]]

**raw\_points**(*simplify: bool = False*) → ndarray

**rod\_raw\_points**(*simplify: bool = False*) → Tuple[ndarray, ndarray, ndarray, ndarray]

**class** cellsium.model.Shape

Bases: object

Base class for implementing cell shapes.

**static defaults**() → Dict[str, Union[Callable[[], Any], float]]

**get\_approximation\_circles**() → Iterator[Tuple[float, Tuple[float, float]]]

**raw\_points**(*simplify: bool = False*) → ndarray

**class** cellsium.model.Shape3D

Bases: [Shape](#)

Base class for implementing 3D cell shapes.

**raw\_points3d**(*steps: int = 32, simplify: bool = False*) → ndarray

**class** cellsium.model.SimulatedCell

Bases: object

Base class for simulated cells, allowing for division behavior.

**birth**(*parent: Optional[SimulatedCell] = None, ts: Optional[Timestep] = None*) → None

Called when a cell is “born”.

#### Parameters

- **parent** – Parent cell
- **ts** – Timestep

#### Returns

None

**divide**(*ts: Timestep*) → Iterable[[SimulatedCell](#)]

Called when a cell should divide, creates the daughter cells.

#### Parameters

**ts** – Timestep

#### Returns

None

**grow**(*ts*: [Timestep](#)) → None

Called each timestep to grow cell.

**Parameters**

**ts** – Timestep

**Returns**

None

**step**(*ts*: [Timestep](#)) → None

Timestep function of the cell object, called by the simulator.

**Parameters**

**ts** – Timestep

**Returns**

None

**class** `cellsium.model.SizerCell`

Bases: [SimulatedCell](#)

Example cell implementing a simple sizer growth mechanism.

**birth**(*parent*: *Optional*[[SizerCell](#)] = None, *ts*: *Optional*[[Timestep](#)] = None) → None

Called when a cell is “born”.

**Parameters**

- **parent** – Parent cell
- **ts** – Timestep

**Returns**

None

**grow**(*ts*: [Timestep](#)) → None

Called each timestep to grow cell.

**Parameters**

**ts** – Timestep

**Returns**

None

**static random\_sequences**(*sequence*: Any) → Mapping[str, Any]

**class** `cellsium.model.Square`

Bases: [Rectangle](#)

Square cell geometry.

**raw\_points**(*simplify*: bool = False) → ndarray

**class** `cellsium.model.TimerCell`

Bases: [SimulatedCell](#)

Example cell implementing a simple timer growth mechanism.

**birth**(*parent*: *Optional*[[TimerCell](#)] = None, *ts*: *Optional*[[Timestep](#)] = None) → None

Called when a cell is “born”.

**Parameters**

- **parent** – Parent cell

- **ts** – Timestep

**Returns**

None

**grow**(*ts*: Timestep) → None

Called each timestep to grow cell.

**Parameters**

**ts** – Timestep

**Returns**

None

**static random\_sequences**(*sequence*: Any) → Mapping[str, Any]

**class** cellsium.model.Timestep(*timestep*: float, *simulation*: Simulation, *simulator*: Simulator)

Bases: object

Timestep is an auxiliary class combining a certain timepoint, simulation and simulator.

**property hours**: float

The hours passed within this timestep.

**Returns**

Hours

**simulation**

**simulator**

**property time**: float

Total simulation time passed in seconds.

**Returns**

Seconds

**property time\_hours**: float

Total simulation time passed in hours.

**Returns**

Hours

**timestep**

**property world**: World

**class** cellsium.model.WithAngle

Bases: object

Mixin adding a cell angle.

**static defaults**() → Dict[str, Union[Callable[[], Any], float]]

**class** cellsium.model.WithFluorescence

Bases: object

Mixin adding a fluorescence value.

**static defaults**() → Dict[str, Union[Callable[[], Any], float]]

**class** cellsium.model.WithLineage

Bases: object

Mixin providing lineage tracking.

**copy()** → *WithLineage*

**static defaults()** → Dict[str, Union[Callable[[], Any], float]]

**next\_cell\_id()** → None

**class** cellsium.model.WithLineageHistory

Bases: object

Mixin providing lineage history.

**static defaults()** → Dict[str, Union[Callable[[], Any], float]]

**class** cellsium.model.WithPosition

Bases: object

Mixin adding a cell position.

**static defaults()** → Dict[str, Union[Callable[[], Any], float]]

**class** cellsium.model.WithProperDivisionBehavior

Bases: object

Mixin adding division angle calculation.

**get\_division\_positions**(*count: int = 2*) → List[List[float]]

**class** cellsium.model.WithRandomSequences

Bases: object

Mixin for objects with random sequences.

**all\_random\_sequences\_generated\_for** = {}

**classmethod** **get\_random\_sequences**(*sequence: Optional[Any] = None*) → Any

**property** **random**: Any

**class** cellsium.model.WithTemporalLineage

Bases: object

Mixing providing temporal lineage history.

**static defaults()** → Dict[str, Union[Callable[[], Any], float]]

**cellsium.model.assemble\_cell**(*simulated\_model: type, \*additional\_classes, placed\_cell: type = <class 'cellsium.model.PlacedCell'>, name: str = 'Cell'*)

Assembles a cell class from parent classes. Necessary as the cell class needs the right level of inheritance.

#### Parameters

- **simulated\_model** – Model class
- **additional\_classes** – Classes to create a cell type, or
- **placed\_cell** – A prepared cell
- **name** – Optional name

**Returns**

Cell class

`cellsium.model.generate_cell(*additional_classes: type, name: str = 'PlacedCell')`

Generates a cell class using the standard classes, and possible additional classes.

**Parameters**

- **additional\_classes** – Additional classes to inherit the cell from.
- **name** – Name of the class

**Returns**

Class

`cellsium.model.h_to_s(hours: Union[float, ndarray]) → Union[float, ndarray]`

Convert hours to seconds.

**Parameters**

**hours** – Hours

**Returns**

Seconds

`cellsium.model.s_to_h(seconds: Union[float, ndarray]) → Union[float, ndarray]`

Convert seconds to hours.

**Parameters**

**seconds** – Seconds

**Returns**

Hours

### 4.3.1 cellsium.model.initialization module

Cell parameter random initializations.

**class** `cellsium.model.initialization.RandomAngle`

Bases: object

Random initializations for cell angles.

**static random\_sequences**(sequence: RRF) → Dict[str, Any]

**class** `cellsium.model.initialization.RandomBentRod`

Bases: object

Random initializations for cell bent radii.

**static random\_sequences**(sequence: RRF) → Dict[str, Any]

**class** `cellsium.model.initialization.RandomFluorescence`

Bases: object

Random initializations for fluorescences.

**static random\_sequences**(sequence: RRF) → Dict[str, Any]



```

class cellsium.model.initialization.RandomPosition
    Bases: object
    Random initializations for cell positions.
    static random_sequences(sequence: RRF) → Dict[str, Any]

class cellsium.model.initialization.RandomWidthLength
    Bases: object
    Random initializations for cell width/lengths.
    static random_sequences(sequence: RRF) → Dict[str, Any]

```

### 4.3.2 cellsium.model.agent module

Cell model classes and routines, general.

```

class cellsium.model.agent.Copyable
    Bases: object
    Mixin for copyable objects.
    copy() → Copyable

class cellsium.model.agent.IdCounter
    Bases: object
    Id provider singleton class.
    id_counter: int = 0
    classmethod next_cell_id() → int
    classmethod reset() → None

class cellsium.model.agent.InitializeWithParameters(**kwargs)
    Bases: object
    Mixin for objects with defaults.

class cellsium.model.agent.Representable
    Bases: object
    Mixins for adding a repr implementation.

class cellsium.model.agent.WithLineage
    Bases: object
    Mixin providing lineage tracking.
    copy() → WithLineage
    static defaults() → Dict[str, Union[Callable[[], Any], float]]
    next_cell_id() → None

class cellsium.model.agent.WithLineageHistory
    Bases: object
    Mixin providing lineage history.

```

```
    static defaults() → Dict[str, Union[Callable[[], Any], float]]
```

```
class cellsium.model.agent.WithRandomSequences
```

```
    Bases: object
```

```
    Mixin for objects with random sequences.
```

```
    all_random_sequences_generated_for = {}
```

```
    classmethod get_random_sequences(sequence: Optional[Any] = None) → Any
```

```
    property random: Any
```

```
class cellsium.model.agent.WithTemporalLineage
```

```
    Bases: object
```

```
    Mixing providing temporal lineage history.
```

```
    static defaults() → Dict[str, Union[Callable[[], Any], float]]
```

### 4.3.3 cellsium.model.geometry module

Cell geometry model classes and routines.

```
class cellsium.model.geometry.AutoMesh3D
```

```
    Bases: Shape3D
```

```
    Mixin adding automatic solid-of-revolution generation.
```

```
    points3d_on_canvas(steps: int = 16, simplify: bool = False) → Tuple[ndarray, ndarray]
```

```
    raw_points3d(steps: int = 16, simplify: bool = False) → Tuple[ndarray, ndarray]
```

```
class cellsium.model.geometry.BentRod
```

```
    Bases: RodShaped
```

```
    Bent rod shaped cell geometry.
```

```
    bend(points: ndarray) → ndarray
```

```
    static defaults() → Dict[str, Union[Callable[[], Any], float]]
```

```
    get_approximation_circles() → Iterator[Tuple[float, Tuple[float, float]]]
```

```
    raw_points(simplify: bool = False) → ndarray
```

```
    raw_points3d(steps: int = 16, simplify: bool = False) → Tuple[ndarray, ndarray]
```

```
class cellsium.model.geometry.CellGeometry
```

```
    Bases: WithAngle, WithPosition, AutoMesh3D
```

```
    Cell geometry base by combining multiple mixins.
```

```
    points_on_canvas() → ndarray
```

```
class cellsium.model.geometry.Coccoid
```

```
    Bases: Shape
```

```
    Coccoid (spherical) cell geometry.
```

```

    static defaults() → Dict[str, Union[Callable[[], Any], float]]

    get_approximation_circles() → Iterator[Tuple[float, Tuple[float, float]]]

    raw_points(simplify: bool = False) → ndarray

class cellsium.model.geometry.Ellipsoid
    Bases: Cocoid
    Ellipsoid cell geometry.

    static defaults() → Dict[str, Union[Callable[[], Any], float]]

    raw_points(simplify: bool = False) → ndarray

class cellsium.model.geometry.Rectangle
    Bases: Shape
    Rectangular cell geometry.

    static defaults() → Dict[str, Union[Callable[[], Any], float]]

    get_approximation_circles() → Iterator[Tuple[float, Tuple[float, float]]]

    raw_points(simplify: bool = False) → Dict[str, Union[Callable[[], Any], float]]

class cellsium.model.geometry.RodShaped
    Bases: Shape
    Rod shaped cell geometry.

    static defaults() → Dict[str, Union[Callable[[], Any], float]]

    get_approximation_circles() → Iterator[Tuple[float, Tuple[float, float]]]

    raw_points(simplify: bool = False) → ndarray

    rod_raw_points(simplify: bool = False) → Tuple[ndarray, ndarray, ndarray, ndarray]

class cellsium.model.geometry.Shape
    Bases: object
    Base class for implementing cell shapes.

    static defaults() → Dict[str, Union[Callable[[], Any], float]]

    get_approximation_circles() → Iterator[Tuple[float, Tuple[float, float]]]

    raw_points(simplify: bool = False) → ndarray

class cellsium.model.geometry.Shape3D
    Bases: Shape
    Base class for implementing 3D cell shapes.

    raw_points3d(steps: int = 32, simplify: bool = False) → ndarray

class cellsium.model.geometry.Square
    Bases: Rectangle
    Square cell geometry.

```

**raw\_points**(*simplify: bool = False*) → ndarray

**class** cellsium.model.geometry.**WithAngle**

Bases: object

Mixin adding a cell angle.

**static defaults**() → Dict[str, Union[Callable[[], Any], float]]

**class** cellsium.model.geometry.**WithFluorescence**

Bases: object

Mixin adding a fluorescence value.

**static defaults**() → Dict[str, Union[Callable[[], Any], float]]

**class** cellsium.model.geometry.**WithPosition**

Bases: object

Mixin adding a cell position.

**static defaults**() → Dict[str, Union[Callable[[], Any], float]]

**class** cellsium.model.geometry.**WithProperDivisionBehavior**

Bases: object

Mixin adding division angle calculation.

**get\_division\_positions**(*count: int = 2*) → List[List[float]]

## 4.4 cellsium.output package

The output package contains the various output modules.

**class** cellsium.output.**Output**(\*args, \*\*kwargs)

Bases: Selectable, Multiple

Base class of the Output classes.

**display**(*world: World*, \*\*kwargs) → None

Output and display the World, e.g. via a GUI window.

### Parameters

- **world** – World
- **kwargs** – Additional arguments

### Returns

**output**(*world: World*, \*\*kwargs) → Optional[Any]

Outputs the World, this function is usually called by either write or display.

### Parameters

- **world** – World
- **kwargs** – Additional arguments

### Returns

**write**(world: [World](#), file\_name: str, \*\*kwargs) → None

Output and write the World to file\_name.

#### Parameters

- **world** – World
- **file\_name** – Filename to write output to
- **kwargs** – Additional arguments

#### Returns

**class** cellsium.output.**OutputIndividualFiles**(\*args, \*\*kwargs)

Bases: [Tunable](#)

Output individual files

**default:** bool = True

**value** = True

**class** cellsium.output.**OutputIndividualFilesWildcard**(\*args, \*\*kwargs)

Bases: [Tunable](#)

Pattern for individual file names

**default:** str = '{}'

**value** = '{}'

**class** cellsium.output.**OutputIndividualFilesZeros**(\*args, \*\*kwargs)

Bases: [Tunable](#)

Amount of digits used for outputting the frame number of individual file names

**default:** int = 3

**value** = 3

**class** cellsium.output.**OutputReproducibleFiles**(\*args, \*\*kwargs)

Bases: [Tunable](#)

Output files in a reproducible manner

**default:** bool = True

**value** = True

cellsium.output.**check\_overwrite**(path: str, overwrite: bool = False) → str

Check if a path exists, if so raising a RuntimeError if overwriting is disabled.

#### Parameters

- **path** – Path
- **overwrite** – Whether to overwrite

#### Returns

Path

`cellsium.output.ensure_extension(path: str, extension: str) → str`

Ensures that the path ends with extension, possibly adding it.

**Parameters**

- **path** – Path
- **extension** – Extension

**Returns**

Final path

`cellsium.output.ensure_number(path: str, number: int, disable_individual: bool = False) → str`

Depending on configuration, add a number to the path for consecutive output files.

**Parameters**

- **path** – Path
- **number** – Number
- **disable\_individual** – Possibility to disable adding of a number

**Returns**

Path with number

`cellsium.output.ensure_path(path: str) → str`

Ensures that the parent directory to the to path exists.

**Parameters**

**path** – Path

**Returns**

the path

`cellsium.output.ensure_path_and_extension(path: str, extension: str) → str`

Ensures that the parent directory to path exists, and it has extension, possibly by adding it.

**Parameters**

- **path** – Path
- **extension** – Extension

**Returns**

Final path

`cellsium.output.ensure_path_and_extension_and_number(path: str, extension: str, number: int, disable_individual: bool = False) → str`

Ensures that a path exists, has an extension and a number.

**Parameters**

- **path** – Path
- **extension** – Extension
- **number** – Number
- **disable\_individual** – Whether to disable adding of number

**Returns**

Final path

#### 4.4.1 cellsium.output.xml module

Output as Trackmate XML lineage files, compatible with the JuNGLE extensions.

**class** cellsium.output.xml.**TrackMateXML**(\*args, \*\*kwargs)

Bases: [Output](#)

**display**(world: [World](#), \*\*kwargs) → None

Output and display the World, e.g. via a GUI window.

##### Parameters

- **world** – World
- **kwargs** – Additional arguments

##### Returns

**output**(world: [World](#), time: float = 0.0, \*\*kwargs) → None

Outputs the World, this function is usually called by either write or display.

##### Parameters

- **world** – World
- **kwargs** – Additional arguments

##### Returns

**write**(world: [World](#), file\_name: str, time: float = 0.0, overwrite: bool = False, \*\*kwargs) → None

Output and write the World to file\_name.

##### Parameters

- **world** – World
- **file\_name** – Filename to write output to
- **kwargs** – Additional arguments

##### Returns

#### 4.4.2 cellsium.output.plot module

Output using matplotlib.

**class** cellsium.output.plot.**MicrometerPerCm**(\*args, \*\*kwargs)

Bases: [Tunable](#)

**default:** float = 2.5

**value** = 2.5

**class** cellsium.output.plot.**PlotRenderer**(\*args, \*\*kwargs)

Bases: [Output](#), [Default](#)

Output using matplotlib.

**display**(world: [World](#), \*\*kwargs) → None

Output and display the World, e.g. via a GUI window.

##### Parameters

- **world** – World
- **kwargs** – Additional arguments

**Returns**

**output**(*world*: [World](#), *\*\*kwargs*) → Tuple[Figure, Axes]

Outputs the World, this function is usually called by either write or display.

**Parameters**

- **world** – World
- **kwargs** – Additional arguments

**Returns**

**write**(*world*: [World](#), *file\_name*: str, *output\_count*: int = 0, *overwrite*: bool = False, *\*\*kwargs*) → None

Output and write the World to file\_name.

**Parameters**

- **world** – World
- **file\_name** – Filename to write output to
- **kwargs** – Additional arguments

**Returns**

### 4.4.3 cellsium.output.mesh module

Mesh output in the STL format.

**class** cellsium.output.mesh.**MeshOutput**(*\*args*, *\*\*kwargs*)

Bases: [Output](#)

Mesh output in the STL format.

**display**(*world*: [World](#), *\*\*kwargs*) → None

Output and display the World, e.g. via a GUI window.

**Parameters**

- **world** – World
- **kwargs** – Additional arguments

**Returns**

**output**(*world*: [World](#), *\*\*kwargs*) → List[Dict[str, ndarray]]

Outputs the World, this function is usually called by either write or display.

**Parameters**

- **world** – World
- **kwargs** – Additional arguments

**Returns**



**write**(world: [World](#), file\_name: str, overwrite: bool = False, output\_count: int = 0, \*\*kwargs) → None

Output and write the World to file\_name.

#### Parameters

- **world** – World
- **file\_name** – Filename to write output to
- **kwargs** – Additional arguments

#### Returns

### 4.4.4 cellsium.output.svg module

Output as SVG vector images.

**class** cellsium.output.svg.**SvgRenderer**(\*args, \*\*kwargs)

Bases: [Output](#)

**static** create\_xml()

**display**(world: [World](#), \*\*kwargs) → None

Output and display the World, e.g. via a GUI window.

#### Parameters

- **world** – World
- **kwargs** – Additional arguments

#### Returns

**output**(world: [World](#), \*\*kwargs) → None

Outputs the World, this function is usually called by either write or display.

#### Parameters

- **world** – World
- **kwargs** – Additional arguments

#### Returns

**static** points\_to\_path(points: ndarray) → str

**write**(world: [World](#), file\_name: str, overwrite: bool = False, output\_count: int = 0, \*\*kwargs) → None

Output and write the World to file\_name.

#### Parameters

- **world** – World
- **file\_name** – Filename to write output to
- **kwargs** – Additional arguments

#### Returns

### 4.4.5 cellsium.output.render module

Photorealistic rendered output.

**class** cellsium.output.render.FluorescenceRenderer(\*args, \*\*kwargs)

Bases: *PlainRenderer*

**channel:** int = 0

**output**(world: *World*, \*\*kwargs) → ndarray

Outputs the World, this function is usually called by either write or display.

**Parameters**

- **world** – World
- **kwargs** – Additional arguments

**Returns**

**class** cellsium.output.render.NoisyUnevenIlluminationPhaseContrast(\*args, \*\*kwargs)

Bases: *UnevenIlluminationPhaseContrast*

**output**(world: *World*, \*\*kwargs) → ndarray

Outputs the World, this function is usually called by either write or display.

**Parameters**

- **world** – World
- **kwargs** – Additional arguments

**Returns**

**class** cellsium.output.render.PhaseContrastRenderer(\*args, \*\*kwargs)

Bases: *PlainRenderer*

**output**(world: *World*, \*\*kwargs) → ndarray

Outputs the World, this function is usually called by either write or display.

**Parameters**

- **world** – World
- **kwargs** – Additional arguments

**Returns**

**class** cellsium.output.render.PlainRenderer(\*args, \*\*kwargs)

Bases: *Output*

**static convert**(image: ndarray, max\_value: int = 255) → ndarray

**debug\_output**(name: str, array: ndarray) → None

**display**(world: *World*, \*\*kwargs) → None

Output and display the World, e.g. via a GUI window.

**Parameters**

- **world** – World
- **kwargs** – Additional arguments

**Returns**

**static imwrite**(*name: str, img: ndarray, overwrite: bool = False, output\_count: Optional[int] = None*) → bool

**static new\_canvas**()

**output**(*world: World, \*\*kwargs*) → ndarray

Outputs the World, this function is usually called by either write or display.

#### Parameters

- **world** – World
- **kwargs** – Additional arguments

#### Returns

**static render\_cells**(*canvas: ndarray, array\_of\_points: ndarray, fast: bool = False*) → ndarray

**write**(*world: World, file\_name: str, overwrite: bool = False, output\_count: int = 0, \*\*kwargs*) → None

Output and write the World to file\_name.

#### Parameters

- **world** – World
- **file\_name** – Filename to write output to
- **kwargs** – Additional arguments

#### Returns

**write\_debug\_output** = False

**class** cellsium.output.render.**RenderChannels**(\*args, \*\*kwargs)

Bases: *Tunable*

Channels to render (i.e. output classes which produce rendered images)

**default** = 'NoisyUnevenIlluminationPhaseContrast'

**static get\_mapping**() → Dict[str, type]

**classmethod instantiate**() → *PlainRenderer*

**classmethod test**(*value: str*) → bool

**value** = 'NoisyUnevenIlluminationPhaseContrast'

**class** cellsium.output.render.**TiffOutput**(\*args, \*\*kwargs)

Bases: *Output*

**output**(*world: World, \*\*kwargs*) → List[ndarray]

Outputs the World, this function is usually called by either write or display.

#### Parameters

- **world** – World
- **kwargs** – Additional arguments

#### Returns

**output\_type**

alias of uint8

**write**(world: [World](#), file\_name: str, \*\*kwargs) → None

Output and write the World to file\_name.

**Parameters**

- **world** – World
- **file\_name** – Filename to write output to
- **kwargs** – Additional arguments

**Returns**

**class** cellsium.output.render.**UnevenIlluminationPhaseContrast**(\*args, \*\*kwargs)

Bases: [PhaseContrastRenderer](#)

**create\_uneven\_illumination**() → ndarray

**new\_uneven\_illumination**() → ndarray

**output**(world: [World](#), \*\*kwargs) → ndarray

Outputs the World, this function is usually called by either write or display.

**Parameters**

- **world** – World
- **kwargs** – Additional arguments

**Returns**

cellsium.output.render.**get\_canvas\_points\_for\_cell**(cell: [CellGeometry](#), image\_height: *Optional[Tuple[int, int]] = None*) → ndarray

cellsium.output.render.**new\_canvas**(dtype=<class 'numpy.float32'>) → ndarray

## 4.4.6 cellsium.output.serialization module

Serialization outputs.

**class** cellsium.output.serialization.**CsvOutput**(\*args, \*\*kwargs)

Bases: [Output](#)

CSV Tabular Output.

**output**(world: [World](#), time: *Optional[float] = None*, \*\*kwargs) → List[Dict[str, Any]]

Outputs the World, this function is usually called by either write or display.

**Parameters**

- **world** – World
- **kwargs** – Additional arguments

**Returns**

**write**(world: [World](#), file\_name: str, time: *Optional[float] = None*, overwrite: bool = False, output\_count: int = 0, \*\*kwargs)

Output and write the World to file\_name.

**Parameters**

- **world** – World

- **file\_name** – Filename to write output to
- **kwargs** – Additional arguments

#### Returns

**class** cellsium.output.serialization.JsonPickleSerializer(\*args, \*\*kwargs)

Bases: [Output](#)

Output as jsonpickle serialized files.

**display**(world: [World](#), \*\*kwargs) → None

Output and display the World, e.g. via a GUI window.

#### Parameters

- **world** – World
- **kwargs** – Additional arguments

#### Returns

**output**(world: [World](#), \*\*kwargs) → str

Outputs the World, this function is usually called by either write or display.

#### Parameters

- **world** – World
- **kwargs** – Additional arguments

#### Returns

**write**(world: [World](#), file\_name: str, overwrite: bool = False, output\_count: int = 0, \*\*kwargs) → None

Output and write the World to file\_name.

#### Parameters

- **world** – World
- **file\_name** – Filename to write output to
- **kwargs** – Additional arguments

#### Returns

**class** cellsium.output.serialization.QuickAndDirtyTableDumper(\*args, \*\*kwargs)

Bases: [Output](#)

Simple tabular output.

**output**(world: [World](#), \*\*kwargs) → ndarray

Outputs the World, this function is usually called by either write or display.

#### Parameters

- **world** – World
- **kwargs** – Additional arguments

#### Returns

**write**(world: [World](#), file\_name: str, time: Optional[float] = None, overwrite: bool = False, output\_count: int = 0, \*\*kwargs)

Output and write the World to file\_name.

#### Parameters

- **world** – World
- **file\_name** – Filename to write output to
- **kwargs** – Additional arguments

**Returns**

#### 4.4.7 cellsium.output.gt module

Ground truth outputs in COCO, YOLO and a generic mask format.

**class** cellsium.output.gt.COCOOutput(\*args, \*\*kwargs)

Bases: GroundTruthOutput

Output in the COCO format.

**static now()** → str

**class** cellsium.output.gt.GenericMaskOutput(\*args, \*\*kwargs)

Bases: GroundTruthOutput

Generic mask output (i.e. directories of files).

**static generate\_cells\_mask**(cells: Iterable[CellGeometry], cell\_value: int = 1, binary: bool = True) → ndarray

**static imwrite**(\*args, \*\*kwargs) → None

**class** cellsium.output.gt.YOLOOutput(\*args, \*\*kwargs)

Bases: GroundTruthOutput

Output in the YOLO format.

### 4.5 cellsium.simulation package

Simulation package contains the simulation/simulator-related classes.

**class** cellsium.simulation.BaseSimulator

Bases: object

**add**(cell: object) → None

Add a cell to the simulation.

**Parameters**

**cell** – Cell

**Returns**

None

**add\_boundary**(coordinates: ndarray) → None

Add a boundary to the simulation.

**Parameters**

**coordinates** – Coordinates of the boundary

**Returns**

None

**clear()** → None

Clear the (world of the) simulation.

**Returns**

None

**remove(*cell*)** → None

Remove a cell from the simulation.

**Parameters**

**cell** – Cell

**Returns**

None

**step(*timestep: float*)** → None

Advance the simulation by a timestep.

**Parameters**

**timestep** – Time passed in seconds

**Returns**

None

#### 4.5.1 cellsium.simulation.placement package

Placement simulation package, contains the placement simulators.

**class** cellsium.simulation.placement.**Box2D**(\*args, \*\*kwargs)

Bases: *PhysicalPlacement*, *PlacementSimulation*

**add(*cell: PlacedCell*)** → None

Add a cell to the simulation.

**Parameters**

**cell** – Cell

**Returns**

None

**add\_boundary(*coordinates: ndarray*)** → None

Add a boundary to the simulation.

**Parameters**

**coordinates** – Coordinates of the boundary

**Returns**

None

**remove(*cell: PlacedCell*)** → None

Remove a cell from the simulation.

**Parameters**

**cell** – Cell

**Returns**

None

**step**(*timestep: float*) → None

Advance the simulation by a timestep.

**Parameters**

**timestep** – Time passed in seconds

**Returns**

None

**verbose: bool = False**

**class** cellsium.simulation.placement.**Chipmunk**(\*args, \*\*kwargs)

Bases: [PhysicalPlacement](#), [PlacementSimulation](#), [Default](#)

**add**(*cell: PlacedCell*) → None

Add a cell to the simulation.

**Parameters**

**cell** – Cell

**Returns**

None

**add\_boundary**(*coordinates: ndarray*) → None

Add a boundary to the simulation.

**Parameters**

**coordinates** – Coordinates of the boundary

**Returns**

None

**clear**() → None

Clear the (world of the) simulation.

**Returns**

None

**convergence\_check\_interval: int = 15**

**inner\_step**(*time\_step: float = 0.1, iterations: int = 9999, converge: bool = True, epsilon: float = 0.1*) → float

**look\_back\_threshold: int = 5**

**remove**(*cell: PlacedCell*) → None

Remove a cell from the simulation.

**Parameters**

**cell** – Cell

**Returns**

None

**step**(*timestep: float*) → None

Advance the simulation by a timestep.

**Parameters**

**timestep** – Time passed in seconds

**Returns**

None



**verbose:** `bool = False`

**class** `cellsium.simulation.placement.PlacementSimulation(*args, **kwargs)`

Bases: [`BaseSimulator`](#), `Selectable`

### `cellsium.simulation.placement.pybox2d` module

Placement simulation using Box2D physics engine.

**class** `cellsium.simulation.placement.pybox2d.Box2D(*args, **kwargs)`

Bases: [`PhysicalPlacement`](#), [`PlacementSimulation`](#)

**add**(*cell*: [`PlacedCell`](#)) → `None`

Add a cell to the simulation.

**Parameters**

**cell** – `Cell`

**Returns**

`None`

**add\_boundary**(*coordinates*: `ndarray`) → `None`

Add a boundary to the simulation.

**Parameters**

**coordinates** – Coordinates of the boundary

**Returns**

`None`

**remove**(*cell*: [`PlacedCell`](#)) → `None`

Remove a cell from the simulation.

**Parameters**

**cell** – `Cell`

**Returns**

`None`

**step**(*timestep*: `float`) → `None`

Advance the simulation by a timestep.

**Parameters**

**timestep** – Time passed in seconds

**Returns**

`None`

**verbose:** `bool = False`

**cellsium.simulation.placement.base module**

**class** cellsium.simulation.placement.base.NoPlacement(\*args, \*\*kwargs)

Bases: *PlacementSimulation*

**class** cellsium.simulation.placement.base.PhysicalPlacement(\*args, \*\*kwargs)

Bases: *PlacementSimulation*, Virtual

**clear()** → None

Clear the (world of the) simulation.

**Returns**

None

**class** cellsium.simulation.placement.base.PlacementSimulation(\*args, \*\*kwargs)

Bases: *BaseSimulator*, Selectable

**class** cellsium.simulation.placement.base.PlacementSimulationSimplification(\*args, \*\*kwargs)

Bases: Tunable

How much the placement should be simplified, 0: use the normal shapes, 1: use simplified shapes, 2: use many-circle approximations

**default:** int = 0

**value** = 0

cellsium.simulation.placement.base.ensure\_python(value: Any) → Any

**cellsium.simulation.placement.pymunk module**

Placement simulation using Pymunk physics engine.

**class** cellsium.simulation.placement.pymunk.Chipmunk(\*args, \*\*kwargs)

Bases: *PhysicalPlacement*, *PlacementSimulation*, Default

**add(cell: PlacedCell)** → None

Add a cell to the simulation.

**Parameters**

**cell** – Cell

**Returns**

None

**add\_boundary(coordinates: ndarray)** → None

Add a boundary to the simulation.

**Parameters**

**coordinates** – Coordinates of the boundary

**Returns**

None

**clear()** → None

Clear the (world of the) simulation.

**Returns**

None

**convergence\_check\_interval:** `int = 15`

**inner\_step**(*time\_step: float = 0.1, iterations: int = 9999, converge: bool = True, epsilon: float = 0.1*) → float

**look\_back\_threshold:** `int = 5`

**remove**(*cell: PlacedCell*) → None

Remove a cell from the simulation.

**Parameters**

**cell** – Cell

**Returns**

None

**step**(*timestep: float*) → None

Advance the simulation by a timestep.

**Parameters**

**timestep** – Time passed in seconds

**Returns**

None

**verbose:** `bool = False`

**class** cellsium.simulation.placement.pymunk.**ChipmunkPlacementRadius**(\*args, \*\*kwargs)

Bases: Tunable

Chipmunk placement radius, additional radius objects will have around them

**default:** `float = 0.05`

**value** = `0.05`

## 4.5.2 cellsium.simulation.simulator module

Simulator base classes.

**class** cellsium.simulation.simulator.**Simulation**

Bases: object

Simulation represents the simulation state at a certain timepoint, i.e. a World and a time.

**class** cellsium.simulation.simulator.**Simulator**

Bases: *BaseSimulator*

Simulator class, a class serving as interface to World and sub-simulators (such as physical placement), as well as the caller of each cells step function.

**add**(*cell: object*) → None

Add a cell to the simulation.

**Parameters**

**cell** – Cell

**Returns**

None

**add\_boundary**(*coordinates: ndarray*) → None

Add a boundary to the simulation.

**Parameters**

**coordinates** – Coordinates of the boundary

**Returns**

None

**clear**() → None

Clear the (world of the) simulation.

**Returns**

None

**remove**(*cell: object*) → None

Remove a cell from the simulation.

**Parameters**

**cell** – Cell

**Returns**

None

**step**(*timestep: float = 0.0*) → *Timestep*

Advance the simulation by a timestep.

**Parameters**

**timestep** – Time passed in seconds

**Returns**

None

**class** cellsium.simulation.simulator.**Timestep**(*timestep: float, simulation: [Simulation](#), simulator: [Simulator](#)*)

Bases: object

Timestep is an auxiliary class combining a certain timepoint, simulation and simulator.

**property hours: float**

The hours passed within this timestep.

**Returns**

Hours

**simulation**

**simulator**

**property time: float**

Total simulation time passed in seconds.

**Returns**

Seconds

**property time\_hours: float**

Total simulation time passed in hours.

**Returns**

Hours

**timestep**

**property world:** *World*

**class** cellsium.simulation.simulator.**World**

Bases: object

The World class contains the cells and, if present, the boundaries.

**add**(*cell: object*) → None

Adds a cell to the World.

**Parameters**

**cell** – Cell

**Returns**

None

**add\_boundary**(*coordinates: ndarray*) → None

Add a boundary to the simulation.

**Parameters**

**coordinates** – Coordinates of the boundary.

**Returns**

None

**clear**() → None

Resets the World.

**Returns**

None

**commit**() → None

Commits a step. Cells to be added, and cells to be removed, will only be applied once commit is called.

**Returns**

None

**copy**() → *World*

Creates a copy of the World.

**Returns**

Copy of the World

**remove**(*cell: object*) → None

Removes a cell from the world.

**Parameters**

**cell** – Cell

**Returns**

None

## 4.6 cellsium.random module

Random number generation infrastructure.

**class** cellsium.random.RRF(*mode: str = 'callable'*)

Bases: object

Reproducible random function.

**classmethod** chain(*func: KwargFunction, \*\*kwargs*) → Iterator

Calls func with kwargs and yields from it.

**Parameters**

- **func** – Function to call
- **kwargs** – Kwargs to pass

**Returns**

Iterator of values

**classmethod** compose(*func: KwargFunction, \*\*kwargs*) → Iterator

Calls a function func with an element of the sequences from the kwargs as kwargs.

**Parameters**

- **func** – Function to be called
- **kwargs** – Kwargs of sequences, of which an element each will be used for each function call

**Returns**

Iterator of values

**generator** = <cellsium.random.RRF object>

**classmethod** seed(*seed: Optional[int] = None*) → int

Set the seed for the RRF.

**Parameters**

**seed** – Seed

**Returns**

Seed

**seed\_sequence** = SeedSequence( entropy=1, n\_children\_spawned=26, )

**seed\_value:** int = 1

**sequence** = <cellsium.random.RRF object>

**classmethod** spawn\_generator() → Generator

Generates a new np.random.Generator from the seed and the configured bitgenerator.

**Returns**

The Generator instance

**classmethod** wrap(*sequence: Iterable, func: AnyFunction*) → Iterator

Wraps the sequence with the function func so that each returned element x becomes func(x).

**Parameters**

- **sequence** – Input sequence

- **func** – Function to be called

#### Returns

Iterator of values

```
class cellsium.random.RandomNumberGenerator(*args, **kwargs)
```

Bases: Tunable

Random number generator to be used

```
classmethod available_rngs() → Dict[str, Type[BitGenerator]]
```

```
default: str = 'PCG64'
```

```
classmethod get() → Type[BitGenerator]
```

```
classmethod test(value: str) → bool
```

```
type_
```

alias of str

```
value = 'PCG64'
```

```
class cellsium.random.Seed(*args, **kwargs)
```

Bases: Tunable

Seed for the random number generator

```
default: int = 1
```

```
type_
```

alias of int

```
value = 1
```

```
cellsium.random.enforce_bounds(iterator: Iterator, minimum: float = -inf, maximum: float = inf) →  
    Iterator[Union[float, ndarray]]
```

Will iter thru an iterator til a value is within bounds. For arrays, all values will be considered.

#### Parameters

- **iterator** – Iterator
- **minimum** – Minimum value
- **maximum** – Maximum value

#### Returns

An iterator of values within bounds

## 4.7 cellsium.parameters module

Main set of parameters for the simulation.

```
class cellsium.parameters.Calibration(*args, **kwargs)
```

Bases: Tunable

Calibration for outputs, micrometer per pixel

```
default: float = 0.065
```

```
    value = 0.065

class cellsium.parameters.Height(*args, **kwargs)
    Bases: Tunable
    Height of the (outputted) simulation
    default: float = 60.0
    value = 60.0

class cellsium.parameters.NewCellBendLowerLower(*args, **kwargs)
    Bases: RandomlyDistributed
    Bend factor minimum for the lower part of new new cells
    default: float = -0.1
    value = -0.1

class cellsium.parameters.NewCellBendLowerUpper(*args, **kwargs)
    Bases: RandomlyDistributed
    Bend factor maximum for the lower part of new new cells
    default: float = 0.1
    value = 0.1

class cellsium.parameters.NewCellBendOverallLower(*args, **kwargs)
    Bases: RandomlyDistributed
    Bend factor minimum for new new cells
    default: float = -0.1
    value = -0.1

class cellsium.parameters.NewCellBendOverallUpper(*args, **kwargs)
    Bases: RandomlyDistributed
    Bend factor maximum for new new cells
    default: float = 0.1
    value = 0.1

class cellsium.parameters.NewCellBendUpperLower(*args, **kwargs)
    Bases: RandomlyDistributed
    Bend factor minimum for the upper part of new new cells
    default: float = -0.1
    value = -0.1

class cellsium.parameters.NewCellBendUpperUpper(*args, **kwargs)
    Bases: RandomlyDistributed
    Bend factor maximum for the upper part of new new cells
    default: float = 0.1
```



```

    value = 0.1

class cellsium.parameters.NewCellCount(*args, **kwargs)
    Bases: Tunable
    New cells to add to the simulation
    default: int = 1
    value = 1

class cellsium.parameters.NewCellLength1Mean(*args, **kwargs)
    Bases: RandomlyDistributed
    Mean cell length, subtype one
    default: float = 2.5
    value = 2.5

class cellsium.parameters.NewCellLength1Std(*args, **kwargs)
    Bases: RandomlyDistributed
    Standard deviation of the cell length, subtype one
    default: float = 0.15
    value = 0.15

class cellsium.parameters.NewCellLength2Mean(*args, **kwargs)
    Bases: RandomlyDistributed
    Mean cell length, subtype two
    default: float = 1.25
    value = 1.25

class cellsium.parameters.NewCellLength2Std(*args, **kwargs)
    Bases: RandomlyDistributed
    Standard deviation of the cell length, subtype one
    default: float = 0.15
    value = 0.15

class cellsium.parameters.NewCellLengthAbsoluteMax(*args, **kwargs)
    Bases: Tunable
    Absolute maximum length of new cells
    default: float = 3.5
    value = 3.5

class cellsium.parameters.NewCellLengthAbsoluteMin(*args, **kwargs)
    Bases: Tunable
    Absolute minimum length of new cells
    default: float = 0.8

```

**value = 0.8**

**class** cellsium.parameters.**NewCellRadiusFromCenter**(\*args, \*\*kwargs)

Bases: *RandomlyDistributed*

Maximum radius for new cells to be spawned from the origin

**default: float = 5.0**

**value = 5.0**

**class** cellsium.parameters.**NewCellWidthAbsoluteMax**(\*args, \*\*kwargs)

Bases: Tunable

Absolute maximum width of new cells

**default: float = 1.5**

**value = 1.5**

**class** cellsium.parameters.**NewCellWidthAbsoluteMin**(\*args, \*\*kwargs)

Bases: Tunable

Absolute minimum width of new cells

**default: float = 0.75**

**value = 0.75**

**class** cellsium.parameters.**NewCellWidthMean**(\*args, \*\*kwargs)

Bases: *RandomlyDistributed*

Mean cell width for new cells

**default: float = 1.0**

**value = 1.0**

**class** cellsium.parameters.**NewCellWidthStd**(\*args, \*\*kwargs)

Bases: *RandomlyDistributed*

Standard deviation of the width of new cells

**default: float = 0.1**

**value = 0.1**

**class** cellsium.parameters.**RandomlyDistributed**(\*args, \*\*kwargs)

Bases: Tunable

Parent class for randomly distributed tunables

**class** cellsium.parameters.**Width**(\*args, \*\*kwargs)

Bases: Tunable

Width of the (outputted) simulation

**default: float = 40.0**

**value = 40.0**

`cellsium.parameters.h_to_s(hours: Union[float, ndarray]) → Union[float, ndarray]`

Convert hours to seconds.

**Parameters**

**hours** – Hours

**Returns**

Seconds

`cellsium.parameters.pixel_to_um(pix: Union[float, ndarray]) → Union[float, ndarray]`

Convert pixel to micrometer.

**Parameters**

**pix** – Pixel value

**Returns**

Micrometer value

`cellsium.parameters.s_to_h(seconds: Union[float, ndarray]) → Union[float, ndarray]`

Convert seconds to hours.

**Parameters**

**seconds** – Seconds

**Returns**

Hours

`cellsium.parameters.um_to_pixel(um: Union[float, ndarray]) → Union[float, ndarray]`

Convert micrometer to pixel.

**Parameters**

**um** – Micrometer value

**Returns**

Pixel value

## 4.8 cellsium.typing module

Type hints definitions.

**class** `cellsium.typing.AnyFunction`

Bases: Protocol

**class** `cellsium.typing.KwargFunction`

Bases: Protocol



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### C

- `cellsium`, 15
- `cellsium.cli`, 15
  - `cellsium.cli.cli`, 20
  - `cellsium.cli.render`, 16
  - `cellsium.cli.simulate`, 16
  - `cellsium.cli.training`, 19
- `cellsium.geometry`, 20
- `cellsium.model`, 22
  - `cellsium.model.agent`, 29
  - `cellsium.model.geometry`, 30
  - `cellsium.model.initialization`, 28
- `cellsium.output`, 32
  - `cellsium.output.gt`, 42
  - `cellsium.output.mesh`, 36
  - `cellsium.output.plot`, 35
  - `cellsium.output.render`, 38
  - `cellsium.output.serialization`, 40
  - `cellsium.output.svg`, 37
  - `cellsium.output.xml`, 35
- `cellsium.parameters`, 51
- `cellsium.random`, 50
- `cellsium.simulation`, 42
  - `cellsium.simulation.placement`, 43
    - `cellsium.simulation.placement.base`, 46
    - `cellsium.simulation.placement.pybox2d`, 45
    - `cellsium.simulation.placement.pymunk`, 46
  - `cellsium.simulation.simulator`, 47
- `cellsium.typing`, 55





## A

add() (*cellsium.simulation.BaseSimulator* method), 42  
 add() (*cellsium.simulation.placement.Box2D* method), 43  
 add() (*cellsium.simulation.placement.Chipmunk* method), 44  
 add() (*cellsium.simulation.placement.pybox2d.Box2D* method), 45  
 add() (*cellsium.simulation.placement.pymunk.Chipmunk* method), 46  
 add() (*cellsium.simulation.simulator.Simulator* method), 47  
 add() (*cellsium.simulation.simulator.World* method), 49  
 add\_boundaries\_from\_dxf() (in module *cellsium.cli.simulate*), 17  
 add\_boundaries\_from\_tunables() (in module *cellsium.cli.simulate*), 17  
 add\_boundary() (*cellsium.simulation.BaseSimulator* method), 42  
 add\_boundary() (*cellsium.simulation.placement.Box2D* method), 43  
 add\_boundary() (*cellsium.simulation.placement.Chipmunk* method), 44  
 add\_boundary() (*cellsium.simulation.placement.pybox2d.Box2D* method), 45  
 add\_boundary() (*cellsium.simulation.placement.pymunk.Chipmunk* method), 46  
 add\_boundary() (*cellsium.simulation.simulator.Simulator* method), 47  
 add\_boundary() (*cellsium.simulation.simulator.World* method), 49  
 add\_empty\_third\_dimension() (in module *cellsium.geometry*), 20  
 add\_output\_prefix() (in module *cellsium.cli*), 15  
 all\_random\_sequences\_generated\_for (*cellsium.model.agent.WithRandomSequences* attribute), 30

all\_random\_sequences\_generated\_for (*cellsium.model.WithRandomSequences* attribute), 27  
 AnyFunction (class in *cellsium.typing*), 55  
 assemble\_cell() (in module *cellsium.model*), 27  
 AutoMesh3D (class in *cellsium.model*), 22  
 AutoMesh3D (class in *cellsium.model.geometry*), 30  
 available\_rngs() (*cellsium.random.RandomNumberGenerator* class method), 51

## B

BaseSimulator (class in *cellsium.simulation*), 42  
 bend() (*cellsium.model.BentRod* method), 22  
 bend() (*cellsium.model.geometry.BentRod* method), 30  
 BentRod (class in *cellsium.model*), 22  
 BentRod (class in *cellsium.model.geometry*), 30  
 birth() (*cellsium.model.SimulatedCell* method), 24  
 birth() (*cellsium.model.SizerCell* method), 25  
 birth() (*cellsium.model.TimerCell* method), 25  
 BoundariesFile (class in *cellsium.cli.simulate*), 16  
 BoundariesScaleFactor (class in *cellsium.cli.simulate*), 16  
 Box2D (class in *cellsium.simulation.placement*), 43  
 Box2D (class in *cellsium.simulation.placement.pybox2d*), 45

## C

Calibration (class in *cellsium.parameters*), 51  
 Cell (in module *cellsium.cli*), 15  
 CellGeometry (class in *cellsium.model*), 22  
 CellGeometry (class in *cellsium.model.geometry*), 30  
 cellsium  
   module, 15  
 cellsium.cli  
   module, 15  
   cellsium.cli.cli  
     module, 20  
   cellsium.cli.render  
     module, 16  
   cellsium.cli.simulate  
     module, 16

cellsium.cli.training  
     module, 19  
 cellsium.geometry  
     module, 20  
 cellsium.model  
     module, 22  
 cellsium.model.agent  
     module, 29  
 cellsium.model.geometry  
     module, 30  
 cellsium.model.initialization  
     module, 28  
 cellsium.output  
     module, 32  
 cellsium.output.gt  
     module, 42  
 cellsium.output.mesh  
     module, 36  
 cellsium.output.plot  
     module, 35  
 cellsium.output.render  
     module, 38  
 cellsium.output.serialization  
     module, 40  
 cellsium.output.svg  
     module, 37  
 cellsium.output.xml  
     module, 35  
 cellsium.parameters  
     module, 51  
 cellsium.random  
     module, 50  
 cellsium.simulation  
     module, 42  
 cellsium.simulation.placement  
     module, 43  
 cellsium.simulation.placement.base  
     module, 46  
 cellsium.simulation.placement.pybox2d  
     module, 45  
 cellsium.simulation.placement.pymunk  
     module, 46  
 cellsium.simulation.simulator  
     module, 47  
 cellsium.typing  
     module, 55  
 chain() (cellsium.random.RRF class method), 50  
 channel (cellsium.output.render.FluorescenceRenderer attribute), 38  
 check\_overwrite() (in module cellsium.output), 33  
 Chipmunk (class in cellsium.simulation.placement), 44  
 Chipmunk (class in cellsium.simulation.placement.pymunk), 46  
 ChipmunkPlacementRadius (class in cellsium.simulation.placement.pymunk), 47  
 circle\_segment() (in module cellsium.geometry), 20  
 clear() (cellsium.simulation.BaseSimulator method), 42  
 clear() (cellsium.simulation.placement.base.PhysicalPlacement method), 46  
 clear() (cellsium.simulation.placement.Chipmunk method), 44  
 clear() (cellsium.simulation.placement.pymunk.Chipmunk method), 46  
 clear() (cellsium.simulation.simulator.Simulator method), 48  
 clear() (cellsium.simulation.simulator.World method), 49  
 Coccoid (class in cellsium.model), 22  
 Coccoid (class in cellsium.model.geometry), 30  
 COC00Output (class in cellsium.output.gt), 42  
 commit() (cellsium.simulation.simulator.World method), 49  
 compose() (cellsium.random.RRF class method), 50  
 compose() (in module cellsium.cli.simulate), 17  
 convergence\_check\_interval (cellsium.simulation.placement.Chipmunk attribute), 44  
 convergence\_check\_interval (cellsium.simulation.placement.pymunk.Chipmunk attribute), 46  
 convert() (cellsium.output.render.PlainRenderer static method), 38  
 copy() (cellsium.model.agent.Copyable method), 29  
 copy() (cellsium.model.agent.WithLineage method), 29  
 copy() (cellsium.model.Copyable method), 23  
 copy() (cellsium.model.WithLineage method), 27  
 copy() (cellsium.simulation.simulator.World method), 49  
 Copyable (class in cellsium.model), 23  
 Copyable (class in cellsium.model.agent), 29  
 create\_uneven\_illumination() (cellsium.output.render.UnevenIlluminationPhaseContrast method), 40  
 create\_xml() (cellsium.output.svg.SvgRenderer static method), 37  
 CsvOutput (class in cellsium.output.serialization), 40  
  
**D**  
 debug\_output() (cellsium.output.render.PlainRenderer method), 38  
 default (cellsium.cli.simulate.BoundariesFile attribute), 16  
 default (cellsium.cli.simulate.BoundariesScaleFactor attribute), 16  
 default (cellsium.cli.simulate.SimulationDuration attribute), 16

default (cellsium.cli.simulate.SimulationOutputFirstState attribute), 17

default (cellsium.cli.simulate.SimulationOutputInterval attribute), 17

default (cellsium.cli.simulate.SimulationTimestep attribute), 17

default (cellsium.cli.training.TrainingCellCount attribute), 19

default (cellsium.cli.training.TrainingDataCount attribute), 19

default (cellsium.cli.training.TrainingImageHeight attribute), 19

default (cellsium.cli.training.TrainingImageWidth attribute), 19

default (cellsium.output.OutputIndividualFiles attribute), 33

default (cellsium.output.OutputIndividualFilesWildcard attribute), 33

default (cellsium.output.OutputIndividualFilesZeros attribute), 33

default (cellsium.output.OutputReproducibleFiles attribute), 33

default (cellsium.output.plot.MicrometerPerCm attribute), 35

default (cellsium.output.render.RenderChannels attribute), 39

default (cellsium.parameters.Calibration attribute), 51

default (cellsium.parameters.Height attribute), 52

default (cellsium.parameters.NewCellBendLowerLower attribute), 52

default (cellsium.parameters.NewCellBendLowerUpper attribute), 52

default (cellsium.parameters.NewCellBendOverallLower attribute), 52

default (cellsium.parameters.NewCellBendOverallUpper attribute), 52

default (cellsium.parameters.NewCellBendUpperLower attribute), 52

default (cellsium.parameters.NewCellBendUpperUpper attribute), 52

default (cellsium.parameters.NewCellCount attribute), 53

default (cellsium.parameters.NewCellLength1Mean attribute), 53

default (cellsium.parameters.NewCellLength1Std attribute), 53

default (cellsium.parameters.NewCellLength2Mean attribute), 53

default (cellsium.parameters.NewCellLength2Std attribute), 53

default (cellsium.parameters.NewCellLengthAbsoluteMax attribute), 53

default (cellsium.parameters.NewCellLengthAbsoluteMin attribute), 53

default (cellsium.parameters.NewCellRadiusFromCenter attribute), 54

default (cellsium.parameters.NewCellWidthAbsoluteMax attribute), 54

default (cellsium.parameters.NewCellWidthAbsoluteMin attribute), 54

default (cellsium.parameters.NewCellWidthMean attribute), 54

default (cellsium.parameters.NewCellWidthStd attribute), 54

default (cellsium.parameters.Width attribute), 54

default (cellsium.random.RandomNumberGenerator attribute), 51

default (cellsium.random.Seed attribute), 51

default (cellsium.simulation.placement.base.PlacementSimulationSimplification attribute), 46

default (cellsium.simulation.placement.pymunk.ChipmunkPlacementRadius attribute), 47

defaults() (cellsium.model.agent.WithLineage static method), 29

defaults() (cellsium.model.agent.WithLineageHistory static method), 29

defaults() (cellsium.model.agent.WithTemporalLineage static method), 30

defaults() (cellsium.model.BentRod static method), 22

defaults() (cellsium.model.Coccioid static method), 23

defaults() (cellsium.model.Ellipsoid static method), 23

defaults() (cellsium.model.geometry.BentRod static method), 30

defaults() (cellsium.model.geometry.Coccioid static method), 30

defaults() (cellsium.model.geometry.Ellipsoid static method), 31

defaults() (cellsium.model.geometry.Rectangle static method), 31

defaults() (cellsium.model.geometry.RodShaped static method), 31

defaults() (cellsium.model.geometry.Shape static method), 31

defaults() (cellsium.model.geometry.WithAngle static method), 32

defaults() (cellsium.model.geometry.WithFluorescence static method), 32

defaults() (cellsium.model.geometry.WithPosition static method), 32

defaults() (cellsium.model.Rectangle static method), 23

defaults() (cellsium.model.RodShaped static method), 24

defaults() (cellsium.model.Shape static method), 24

defaults() (cellsium.model.WithAngle static method), 26

defaults() (cellsium.model.WithFluorescence static method), 26

[defaults\(\)](#) (*cellsium.model.WithLineage static method*), 27  
[defaults\(\)](#) (*cellsium.model.WithLineageHistory static method*), 27  
[defaults\(\)](#) (*cellsium.model.WithPosition static method*), 27  
[defaults\(\)](#) (*cellsium.model.WithTemporalLineage static method*), 27  
[display\(\)](#) (*cellsium.output.mesh.MeshOutput method*), 36  
[display\(\)](#) (*cellsium.output.Output method*), 32  
[display\(\)](#) (*cellsium.output.plot.PlotRenderer method*), 35  
[display\(\)](#) (*cellsium.output.render.PlainRenderer method*), 38  
[display\(\)](#) (*cellsium.output.serialization.JsonPickleSerializer method*), 41  
[display\(\)](#) (*cellsium.output.svg.SvgRenderer method*), 37  
[display\(\)](#) (*cellsium.output.xml.TrackMateXML method*), 35  
[divide\(\)](#) (*cellsium.model.SimulatedCell method*), 24

## E

[Ellipsoid](#) (*class in cellsium.model*), 23  
[Ellipsoid](#) (*class in cellsium.model.geometry*), 31  
[enforce\\_bounds\(\)](#) (*in module cellsium.random*), 51  
[ensure\\_extension\(\)](#) (*in module cellsium.output*), 33  
[ensure\\_number\(\)](#) (*in module cellsium.output*), 34  
[ensure\\_path\(\)](#) (*in module cellsium.output*), 34  
[ensure\\_path\\_and\\_extension\(\)](#) (*in module cellsium.output*), 34  
[ensure\\_path\\_and\\_extension\\_and\\_number\(\)](#) (*in module cellsium.output*), 34  
[ensure\\_python\(\)](#) (*in module cellsium.simulation.placement.base*), 46

## F

[FluorescenceRenderer](#) (*class in cellsium.output.render*), 38

## G

[generate\\_cell\(\)](#) (*in module cellsium.model*), 28  
[generate\\_cells\\_mask\(\)](#) (*cellsium.output.gt.GenericMaskOutput static method*), 42  
[generator](#) (*cellsium.random.RRF attribute*), 50  
[GenericMaskOutput](#) (*class in cellsium.output.gt*), 42  
[get\(\)](#) (*cellsium.random.RandomNumberGenerator class method*), 51  
[get\\_approximation\\_circles\(\)](#) (*cellsium.model.BentRod method*), 22  
[get\\_approximation\\_circles\(\)](#) (*cellsium.model.Cocoid method*), 23  
[get\\_approximation\\_circles\(\)](#) (*cellsium.model.geometry.BentRod method*), 30  
[get\\_approximation\\_circles\(\)](#) (*cellsium.model.geometry.Cocoid method*), 31  
[get\\_approximation\\_circles\(\)](#) (*cellsium.model.geometry.Rectangle method*), 31  
[get\\_approximation\\_circles\(\)](#) (*cellsium.model.geometry.RodShaped method*), 31  
[get\\_approximation\\_circles\(\)](#) (*cellsium.model.geometry.Shape method*), 31  
[get\\_approximation\\_circles\(\)](#) (*cellsium.model.Rectangle method*), 23  
[get\\_approximation\\_circles\(\)](#) (*cellsium.model.RodShaped method*), 24  
[get\\_approximation\\_circles\(\)](#) (*cellsium.model.Shape method*), 24  
[get\\_canvas\\_points\\_for\\_cell\(\)](#) (*in module cellsium.output.render*), 40  
[get\\_division\\_positions\(\)](#) (*cellsium.model.geometry.WithProperDivisionBehavior method*), 32  
[get\\_division\\_positions\(\)](#) (*cellsium.model.WithProperDivisionBehavior method*), 27  
[get\\_mapping\(\)](#) (*cellsium.output.render.RenderChannels static method*), 39  
[get\\_random\\_sequences\(\)](#) (*cellsium.model.agent.WithRandomSequences class method*), 30  
[get\\_random\\_sequences\(\)](#) (*cellsium.model.WithRandomSequences class method*), 27  
[grow\(\)](#) (*cellsium.model.SimulatedCell method*), 24  
[grow\(\)](#) (*cellsium.model.SizerCell method*), 25  
[grow\(\)](#) (*cellsium.model.TimerCell method*), 26

## H

[h\\_to\\_s\(\)](#) (*in module cellsium.model*), 28  
[h\\_to\\_s\(\)](#) (*in module cellsium.parameters*), 54  
[Height](#) (*class in cellsium.parameters*), 52  
[hours](#) (*cellsium.model.Timestep property*), 26  
[hours](#) (*cellsium.simulation.simulator.Timestep property*), 48

## I

[id\\_counter](#) (*cellsium.model.agent.IdCounter attribute*), 29  
[id\\_counter](#) (*cellsium.model.IdCounter attribute*), 23  
[IdCounter](#) (*class in cellsium.model*), 23  
[IdCounter](#) (*class in cellsium.model.agent*), 29

imwrite() (*cellsium.output.gt.GenericMaskOutput static method*), 42  
 imwrite() (*cellsium.output.render.PlainRenderer static method*), 38  
 initialize\_cells() (*in module cellsium.cli*), 15  
 initialize\_output\_times\_from\_tunables() (*in module cellsium.cli.simulate*), 17  
 initialize\_simulator() (*in module cellsium.cli*), 15  
 InitializeWithParameters (*class in cellsium.model*), 23  
 InitializeWithParameters (*class in cellsium.model.agent*), 29  
 inner\_step() (*cellsium.simulation.placement.Chipmunk method*), 44  
 inner\_step() (*cellsium.simulation.placement.pymunk.Chipmunk method*), 47  
 instantiate() (*cellsium.output.render.RenderChannels class method*), 39

## J

JsonPickleSerializer (*class in cellsium.output.serialization*), 41

## K

KwargsFunction (*class in cellsium.typing*), 55

## L

line() (*in module cellsium.geometry*), 21  
 load\_class\_from\_module() (*in module cellsium.cli.cli*), 20  
 look\_back\_threshold (*cellsium.simulation.placement.Chipmunk attribute*), 44  
 look\_back\_threshold (*cellsium.simulation.placement.pymunk.Chipmunk attribute*), 47

## M

main() (*in module cellsium.cli.cli*), 20  
 measure\_duration() (*in module cellsium.cli.simulate*), 18  
 MeshOutput (*class in cellsium.output.mesh*), 36  
 MicrometerPerCm (*class in cellsium.output.plot*), 35  
 module  
   cellsium, 15  
   cellsium.cli, 15  
   cellsium.cli.cli, 20  
   cellsium.cli.render, 16  
   cellsium.cli.simulate, 16  
   cellsium.cli.training, 19  
   cellsium.geometry, 20  
   cellsium.model, 22  
   cellsium.model.agent, 29  
   cellsium.model.geometry, 30  
   cellsium.model.initialization, 28  
   cellsium.output, 32  
   cellsium.output.gt, 42  
   cellsium.output.mesh, 36  
   cellsium.output.plot, 35  
   cellsium.output.render, 38  
   cellsium.output.serialization, 40  
   cellsium.output.svg, 37  
   cellsium.output.xml, 35  
   cellsium.parameters, 51  
   cellsium.random, 50  
   cellsium.simulation, 42  
   cellsium.simulation.placement, 43  
   cellsium.simulation.placement.base, 46  
   cellsium.simulation.placement.pybox2d, 45  
   cellsium.simulation.placement.pymunk, 46  
   cellsium.simulation.simulator, 47  
   cellsium.typing, 55

## N

new\_canvas() (*cellsium.output.render.PlainRenderer static method*), 39  
 new\_canvas() (*in module cellsium.output.render*), 40  
 new\_uneven\_illumination() (*cellsium.output.render.UnevenIlluminationPhaseContrast method*), 40  
 NewCellBendLowerLower (*class in cellsium.parameters*), 52  
 NewCellBendLowerUpper (*class in cellsium.parameters*), 52  
 NewCellBendOverallLower (*class in cellsium.parameters*), 52  
 NewCellBendOverallUpper (*class in cellsium.parameters*), 52  
 NewCellBendUpperLower (*class in cellsium.parameters*), 52  
 NewCellBendUpperUpper (*class in cellsium.parameters*), 52  
 NewCellCount (*class in cellsium.parameters*), 53  
 NewCellLength1Mean (*class in cellsium.parameters*), 53  
 NewCellLength1Std (*class in cellsium.parameters*), 53  
 NewCellLength2Mean (*class in cellsium.parameters*), 53  
 NewCellLength2Std (*class in cellsium.parameters*), 53  
 NewCellLengthAbsoluteMax (*class in cellsium.parameters*), 53  
 NewCellLengthAbsoluteMin (*class in cellsium.parameters*), 53  
 NewCellRadiusFromCenter (*class in cellsium.parameters*), 54  
 NewCellWidthAbsoluteMax (*class in cellsium.parameters*), 54  
 NewCellWidthAbsoluteMin (*class in cellsium.parameters*), 54



NewCellWidthMean (class in *cellsium.parameters*), 54  
 NewCellWidthStd (class in *cellsium.parameters*), 54  
 next\_cell\_id() (*cellsium.model.agent.IdCounter* class method), 29  
 next\_cell\_id() (*cellsium.model.agent.WithLineage* method), 29  
 next\_cell\_id() (*cellsium.model.IdCounter* class method), 23  
 next\_cell\_id() (*cellsium.model.WithLineage* method), 27  
 NoisyUnevenIlluminationPhaseContrast (class in *cellsium.output.render*), 38  
 NoPlacement (class in *cellsium.simulation.placement.base*), 46  
 now() (*cellsium.output.gt.COCOOutput* static method), 42  
**O**  
 Output (class in *cellsium.output*), 32  
 output() (*cellsium.output.mesh.MeshOutput* method), 36  
 output() (*cellsium.output.Output* method), 32  
 output() (*cellsium.output.plot.PlotRenderer* method), 36  
 output() (*cellsium.output.render.FluorescenceRenderer* method), 38  
 output() (*cellsium.output.render.NoisyUnevenIlluminationPhaseContrast* method), 38  
 output() (*cellsium.output.render.PhaseContrastRenderer* method), 38  
 output() (*cellsium.output.render.PlainRenderer* method), 39  
 output() (*cellsium.output.render.TiffOutput* method), 39  
 output() (*cellsium.output.render.UnevenIlluminationPhaseContrast* method), 40  
 output() (*cellsium.output.serialization.CsvOutput* method), 40  
 output() (*cellsium.output.serialization.JsonPickleSerializer* method), 41  
 output() (*cellsium.output.serialization.QuickAndDirtyTableDumper* method), 41  
 output() (*cellsium.output.svg.SvgRenderer* method), 37  
 output() (*cellsium.output.xml.TrackMateXML* method), 35  
 output\_type (*cellsium.output.render.TiffOutput* attribute), 39  
 OutputIndividualFiles (class in *cellsium.output*), 33  
 OutputIndividualFilesWildcard (class in *cellsium.output*), 33  
 OutputIndividualFilesZeros (class in *cellsium.output*), 33  
 OutputReproducibleFiles (class in *cellsium.output*), 33

## P

parabolic\_deformation() (in module *cellsium.geometry*), 21  
 parse\_arguments\_and\_init() (in module *cellsium.cli.cli*), 20  
 perform\_outputs() (in module *cellsium.cli.simulate*), 18  
 perform\_simulation() (in module *cellsium.cli.simulate*), 18  
 PhaseContrastRenderer (class in *cellsium.output.render*), 38  
 PhysicalPlacement (class in *cellsium.simulation.placement.base*), 46  
 pixel\_to\_um() (in module *cellsium.parameters*), 55  
 PlacedCell (class in *cellsium.model*), 23  
 PlacementSimulation (class in *cellsium.simulation.placement*), 45  
 PlacementSimulation (class in *cellsium.simulation.placement.base*), 46  
 PlacementSimulationSimplification (class in *cellsium.simulation.placement.base*), 46  
 PlainRenderer (class in *cellsium.output.render*), 38  
 PlotRenderer (class in *cellsium.output.plot*), 35  
 points3d\_on\_canvas() (*cellsium.model.AutoMesh3D* method), 22  
 points3d\_on\_canvas() (*cellsium.model.geometry.AutoMesh3D* method), 30  
 points\_on\_canvas() (*cellsium.model.CellGeometry* method), 22  
 points\_on\_canvas() (*cellsium.model.geometry.CellGeometry* method), 30  
 points\_to\_path() (*cellsium.output.svg.SvgRenderer* static method), 37  
 prepare\_output\_name() (in module *cellsium.cli.simulate*), 18

## Q

QuickAndDirtyTableDumper (class in *cellsium.output.serialization*), 41

## R

random (*cellsium.model.agent.WithRandomSequences* property), 30  
 random (*cellsium.model.WithRandomSequences* property), 27  
 random\_sequences() (*cellsium.model.initialization.RandomAngle* static method), 28  
 random\_sequences() (*cellsium.model.initialization.RandomBentRod* static method), 28

`random_sequences()` (*cellsium.model.initialization.RandomFluorescence static method*), 28  
`random_sequences()` (*cellsium.model.initialization.RandomPosition static method*), 29  
`random_sequences()` (*cellsium.model.initialization.RandomWidthLength static method*), 29  
`random_sequences()` (*cellsium.model.SizerCell static method*), 25  
`random_sequences()` (*cellsium.model.TimerCell static method*), 26  
`RandomAngle` (*class in cellsium.model.initialization*), 28  
`RandomBentRod` (*class in cellsium.model.initialization*), 28  
`RandomFluorescence` (*class in cellsium.model.initialization*), 28  
`RandomlyDistributed` (*class in cellsium.parameters*), 54  
`RandomNumberGenerator` (*class in cellsium.random*), 51  
`RandomPosition` (*class in cellsium.model.initialization*), 28  
`RandomWidthLength` (*class in cellsium.model.initialization*), 29  
`raw_points()` (*cellsium.model.BentRod method*), 22  
`raw_points()` (*cellsium.model.Coccioid method*), 23  
`raw_points()` (*cellsium.model.Ellipsoid method*), 23  
`raw_points()` (*cellsium.model.geometry.BentRod method*), 30  
`raw_points()` (*cellsium.model.geometry.Coccioid method*), 31  
`raw_points()` (*cellsium.model.geometry.Ellipsoid method*), 31  
`raw_points()` (*cellsium.model.geometry.Rectangle method*), 31  
`raw_points()` (*cellsium.model.geometry.RodShaped method*), 31  
`raw_points()` (*cellsium.model.geometry.Shape method*), 31  
`raw_points()` (*cellsium.model.geometry.Square method*), 31  
`raw_points()` (*cellsium.model.Rectangle method*), 23  
`raw_points()` (*cellsium.model.RodShaped method*), 24  
`raw_points()` (*cellsium.model.Shape method*), 24  
`raw_points()` (*cellsium.model.Square method*), 25  
`raw_points3d()` (*cellsium.model.AutoMesh3D method*), 22  
`raw_points3d()` (*cellsium.model.BentRod method*), 22  
`raw_points3d()` (*cellsium.model.geometry.AutoMesh3D method*), 30  
`raw_points3d()` (*cellsium.model.geometry.BentRod method*), 30  
`raw_points3d()` (*cellsium.model.geometry.Shape3D method*), 31  
`raw_points3d()` (*cellsium.model.Shape3D method*), 24  
`Rectangle` (*class in cellsium.model*), 23  
`Rectangle` (*class in cellsium.model.geometry*), 31  
`remove()` (*cellsium.simulation.BaseSimulator method*), 43  
`remove()` (*cellsium.simulation.placement.Box2D method*), 43  
`remove()` (*cellsium.simulation.placement.Chipmunk method*), 44  
`remove()` (*cellsium.simulation.placement.pybox2d.Box2D method*), 45  
`remove()` (*cellsium.simulation.placement.pymunk.Chipmunk method*), 47  
`remove()` (*cellsium.simulation.simulator.Simulator method*), 48  
`remove()` (*cellsium.simulation.simulator.World method*), 49  
`render_cells()` (*cellsium.output.render.PlainRenderer static method*), 39  
`RenderChannels` (*class in cellsium.output.render*), 39  
`Representable` (*class in cellsium.model*), 23  
`Representable` (*class in cellsium.model.agent*), 29  
`reset()` (*cellsium.model.agent.IdCounter class method*), 29  
`reset()` (*cellsium.model.IdCounter class method*), 23  
`rod_raw_points()` (*cellsium.model.geometry.RodShaped method*), 31  
`rod_raw_points()` (*cellsium.model.RodShaped method*), 24  
`RodShaped` (*class in cellsium.model*), 24  
`RodShaped` (*class in cellsium.model.geometry*), 31  
`rotate()` (*in module cellsium.geometry*), 21  
`rotate3d()` (*in module cellsium.geometry*), 21  
`rotate_and_mesh()` (*in module cellsium.geometry*), 21  
`RRF` (*class in cellsium.random*), 50

## S

`s_to_h()` (*in module cellsium.model*), 28  
`s_to_h()` (*in module cellsium.parameters*), 55  
`Seed` (*class in cellsium.random*), 51  
`seed()` (*cellsium.random.RRF class method*), 50  
`seed_sequence` (*cellsium.random.RRF attribute*), 50  
`seed_value` (*cellsium.random.RRF attribute*), 50  
`sequence` (*cellsium.random.RRF attribute*), 50  
`Shape` (*class in cellsium.model*), 24  
`Shape` (*class in cellsium.model.geometry*), 31  
`Shape3D` (*class in cellsium.model*), 24  
`Shape3D` (*class in cellsium.model.geometry*), 31  
`shift()` (*in module cellsium.geometry*), 22  
`SimulatedCell` (*class in cellsium.model*), 24

simulation (*cellsium.model.Timestep* attribute), 26  
simulation (*cellsium.simulation.simulator.Timestep* attribute), 48  
Simulation (class in *cellsium.simulation.simulator*), 47  
SimulationDuration (class in *cellsium.cli.simulate*), 16  
SimulationOutputFirstState (class in *cellsium.cli.simulate*), 16  
SimulationOutputInterval (class in *cellsium.cli.simulate*), 17  
SimulationTimestep (class in *cellsium.cli.simulate*), 17  
simulator (*cellsium.model.Timestep* attribute), 26  
simulator (*cellsium.simulation.simulator.Timestep* attribute), 48  
Simulator (class in *cellsium.simulation.simulator*), 47  
SizerCell (class in *cellsium.cli*), 15  
SizerCell (class in *cellsium.model*), 25  
spawn\_generator() (*cellsium.random.RRF* class method), 50  
Square (class in *cellsium.model*), 25  
Square (class in *cellsium.model.geometry*), 31  
step() (*cellsium.model.SimulatedCell* method), 25  
step() (*cellsium.simulation.BaseSimulator* method), 43  
step() (*cellsium.simulation.placement.Box2D* method), 43  
step() (*cellsium.simulation.placement.Chipmunk* method), 44  
step() (*cellsium.simulation.placement.pybox2d.Box2D* method), 45  
step() (*cellsium.simulation.placement.pymunk.Chipmunk* method), 47  
step() (*cellsium.simulation.simulator.Simulator* method), 48  
subcommand\_argparser() (in module *cellsium.cli.render*), 16  
subcommand\_main() (in module *cellsium.cli.render*), 16  
subcommand\_main() (in module *cellsium.cli.simulate*), 19  
subcommand\_main() (in module *cellsium.cli.training*), 19  
SvgRenderer (class in *cellsium.output.svg*), 37

## T

test() (*cellsium.output.render.RenderChannels* class method), 39  
test() (*cellsium.random.RandomNumberGenerator* class method), 51  
TiffOutput (class in *cellsium.output.render*), 39  
time (*cellsium.model.Timestep* property), 26  
time (*cellsium.simulation.simulator.Timestep* property), 48  
time\_hours (*cellsium.model.Timestep* property), 26

time\_hours (*cellsium.simulation.simulator.Timestep* property), 48  
TimerCell (class in *cellsium.cli*), 15  
TimerCell (class in *cellsium.model*), 25  
timestep (*cellsium.model.Timestep* attribute), 26  
timestep (*cellsium.simulation.simulator.Timestep* attribute), 48  
Timestep (class in *cellsium.model*), 26  
Timestep (class in *cellsium.simulation.simulator*), 48  
TrackMateXML (class in *cellsium.output.xml*), 35  
TrainingCellCount (class in *cellsium.cli.training*), 19  
TrainingDataCount (class in *cellsium.cli.training*), 19  
TrainingImageHeight (class in *cellsium.cli.training*), 19  
TrainingImageWidth (class in *cellsium.cli.training*), 19  
type\_ (*cellsium.random.RandomNumberGenerator* attribute), 51  
type\_ (*cellsium.random.Seed* attribute), 51

## U

um\_to\_pixel() (in module *cellsium.parameters*), 55  
UnevenIlluminationPhaseContrast (class in *cellsium.output.render*), 40

## V

value (*cellsium.cli.simulate.BoundariesFile* attribute), 16  
value (*cellsium.cli.simulate.BoundariesScaleFactor* attribute), 16  
value (*cellsium.cli.simulate.SimulationDuration* attribute), 16  
value (*cellsium.cli.simulate.SimulationOutputFirstState* attribute), 17  
value (*cellsium.cli.simulate.SimulationOutputInterval* attribute), 17  
value (*cellsium.cli.simulate.SimulationTimestep* attribute), 17  
value (*cellsium.cli.training.TrainingCellCount* attribute), 19  
value (*cellsium.cli.training.TrainingDataCount* attribute), 19  
value (*cellsium.cli.training.TrainingImageHeight* attribute), 19  
value (*cellsium.cli.training.TrainingImageWidth* attribute), 19  
value (*cellsium.output.OutputIndividualFiles* attribute), 33  
value (*cellsium.output.OutputIndividualFilesWildcard* attribute), 33  
value (*cellsium.output.OutputIndividualFilesZeros* attribute), 33  
value (*cellsium.output.OutputReproducibleFiles* attribute), 33



- value (*cellsium.output.plot.MicrometerPerCm* attribute), 35
  - value (*cellsium.output.render.RenderChannels* attribute), 39
  - value (*cellsium.parameters.Calibration* attribute), 51
  - value (*cellsium.parameters.Height* attribute), 52
  - value (*cellsium.parameters.NewCellBendLowerLower* attribute), 52
  - value (*cellsium.parameters.NewCellBendLowerUpper* attribute), 52
  - value (*cellsium.parameters.NewCellBendOverallLower* attribute), 52
  - value (*cellsium.parameters.NewCellBendOverallUpper* attribute), 52
  - value (*cellsium.parameters.NewCellBendUpperLower* attribute), 52
  - value (*cellsium.parameters.NewCellBendUpperUpper* attribute), 52
  - value (*cellsium.parameters.NewCellCount* attribute), 53
  - value (*cellsium.parameters.NewCellLength1Mean* attribute), 53
  - value (*cellsium.parameters.NewCellLength1Std* attribute), 53
  - value (*cellsium.parameters.NewCellLength2Mean* attribute), 53
  - value (*cellsium.parameters.NewCellLength2Std* attribute), 53
  - value (*cellsium.parameters.NewCellLengthAbsoluteMax* attribute), 53
  - value (*cellsium.parameters.NewCellLengthAbsoluteMin* attribute), 53
  - value (*cellsium.parameters.NewCellRadiusFromCenter* attribute), 54
  - value (*cellsium.parameters.NewCellWidthAbsoluteMax* attribute), 54
  - value (*cellsium.parameters.NewCellWidthAbsoluteMin* attribute), 54
  - value (*cellsium.parameters.NewCellWidthMean* attribute), 54
  - value (*cellsium.parameters.NewCellWidthStd* attribute), 54
  - value (*cellsium.parameters.Width* attribute), 54
  - value (*cellsium.random.RandomNumberGenerator* attribute), 51
  - value (*cellsium.random.Seed* attribute), 51
  - value (*cellsium.simulation.placement.base.PlacementSimulationSimplification* attribute), 46
  - value (*cellsium.simulation.placement.pymunk.ChipmunkPlacementRadius* attribute), 47
  - verbose (*cellsium.simulation.placement.Box2D* attribute), 44
  - verbose (*cellsium.simulation.placement.Chipmunk* attribute), 44
  - verbose (*cellsium.simulation.placement.pybox2d.Box2D* attribute), 45
  - verbose (*cellsium.simulation.placement.pymunk.Chipmunk* attribute), 47
- ## W
- Width (class in *cellsium.parameters*), 54
  - WithAngle (class in *cellsium.model*), 26
  - WithAngle (class in *cellsium.model.geometry*), 32
  - WithFluorescence (class in *cellsium.model*), 26
  - WithFluorescence (class in *cellsium.model.geometry*), 32
  - WithLineage (class in *cellsium.model*), 26
  - WithLineage (class in *cellsium.model.agent*), 29
  - WithLineageHistory (class in *cellsium.model*), 27
  - WithLineageHistory (class in *cellsium.model.agent*), 29
  - WithPosition (class in *cellsium.model*), 27
  - WithPosition (class in *cellsium.model.geometry*), 32
  - WithProperDivisionBehavior (class in *cellsium.model*), 27
  - WithProperDivisionBehavior (class in *cellsium.model.geometry*), 32
  - WithRandomSequences (class in *cellsium.model*), 27
  - WithRandomSequences (class in *cellsium.model.agent*), 30
  - WithTemporalLineage (class in *cellsium.model*), 27
  - WithTemporalLineage (class in *cellsium.model.agent*), 30
  - world (*cellsium.model.Timestep* property), 26
  - world (*cellsium.simulation.simulator.Timestep* property), 49
  - World (class in *cellsium.simulation.simulator*), 49
  - wrap() (*cellsium.random.RRF* class method), 50
  - write() (*cellsium.output.mesh.MeshOutput* method), 36
  - write() (*cellsium.output.Output* method), 32
  - write() (*cellsium.output.plot.PlotRenderer* method), 36
  - write() (*cellsium.output.render.PlainRenderer* method), 39
  - write() (*cellsium.output.render.TiffOutput* method), 39
  - write() (*cellsium.output.serialization.CsvOutput* method), 40
  - write() (*cellsium.output.serialization.JsonPickleSerializer* method), 41
  - write() (*cellsium.output.serialization.QuickAndDirtyTableDumper* method), 41
  - write() (*cellsium.output.svg.SvgRenderer* method), 37
  - write() (*cellsium.output.xml.TrackMateXML* method), 41
  - write\_debug\_output (*cellsium.output.render.PlainRenderer* attribute), 39
- ## Y
- YOL00output (class in *cellsium.output.gt*), 42