Course Introduction and Setup

During the lab exercises of this course you will implement several models and use them to perform experiments. The first half of this course will focus on Cellular Automata (CA). The second half of the course will focus on Networking Models (NMs). This first introductional mini-exercise is focused on getting you set up and running. You will set up the environment in which you will work in the remainder of this course. This first document also specifies some rules and regulations for the remainder of this course. Unless otherwise stated these rules remain in effect for all assignments. **Please read this document carefully!**

Note: This assignment is **not** graded but it is important you get your environment up and running for the other assignments that are. If you require assistance, this will be provided during the work sessions in **the first week of this course**. Thereafter you yourself are responsible for having your working environment set up properly!

1 TicketVise and Contacting TAs

This year we'll be using both the discussion forum on Canvas and TicketVise. For general questions that might be relevant to other students, please use the discussion forum. Feel free to answer each other's questions as well! For individual questions, please use TicketVise. One of the TAs will pick up the ticket and get back to you as soon as possible.

On Tuesdays there is a session for general questions regarding the assignments, framework, or material of the course. On Thursdays there are individual work groups during which you can ask a TA more specific questions on your work. If you have a question we expect you to – if possible – see a TA at one of these sessions. The mailing list is **not** a substitute for these work sessions but rather a supplement for urgent or organisatory matters. We take attendance and may take this into consideration when deciding whether to answer questions sent to the mailing list.

2 Python Version

The assignments in this course are intended to be written in Python 3.8 or above. It is likely you already have a working version of Python 3 installed, which you can check by running `python3 -V` which should output your exact Python version. If you do not have it installed, you can follow the steps in appendix A. It will likely not be a problem if you have a slightly older version of Python 3 (Python 2 will **not** work!), but keep in mind that the TAs are grading your assignments in a Python 3.10 environment. If your program has major compatibility issues, you will **not** receive a grade for the submission.

2.1 Libraries

During this course you will use a number of Python libraries. You are by default allowed to use the libraries listed below unless otherwise specified. You should not need other libraries during the assignments and thus the usage of other libraries is in principle **not** permitted. If you have a question regarding a library please see a TA or send an e-mail to the mailing list.

- **MatPlotLib:** An extensive visualisation library. Can be used to plot graphs, images, etc. This course uses mainly the PyPlot interface which offers a number of functions that modify some aspect of an internal figure (i.e. the global state-machine). See https://matplotlib.org/ for more information.
- **NumPy**: Provides efficient matrices and other mathematical operations. Since Python is an interpreted language it cannot efficiently handle big arrays or matrices which you will need in this course. Since NumPy is implemented in C it is significantly faster than pure Python allowing for more complex computations in reasonable time-frames. Another advantage is that NumPy works well with MatPlotLib to visualise your findings. See https://numpy.org/ for more details.
- **NetworkX:** Allows for easy creation and modification of networks which will be the focus of the second part of this course. Like NumPy NetworkX works well with MatPlotLib such that you can visualise your networks easily. See https://networkx.github.io/ for the specifics.
- **PyICS:** A custom framework built for this course. PyICS makes it easy to create, visualise, and test models. For an overview of this library refer to section 3. For more details the documentation can be viewed using Python's help function: `help(pyics.Model)`. The library itself can be downloaded from Canvas.

3 PyICS

The PyICS framework can be downloaded from Canvas and can be extracted into a directory of your choosing. PyICS should run fine on Linux, Windows or macOS; but do note that as per programme guidelines your code is required to run properly on the latest LTS release of Ubuntu. We may refuse to grade your submission if it fails to run on Ubuntu.

3.1 Overview

PyICS makes it easier to create and test models. A **model** is a class which must contain at least three methods:

1. `reset`: The **reset** method is called every time the simulation should be (re)initialised. In the case of a CA this would include resetting the state of the grid to an empty one.
2. `step`: the **step** method should simulate a *single step* of the simulation. This method should also detect certain end-conditions for the simulations. It can tell the calling context whether to stop further simulation by returning the value `True` if the simulation is finished, or `False` otherwise.
3. `draw`: the **draw** method informs the simulation to draw its current state (i.e. with MatPlotLib). The reason that the `step` and `draw` methods are separate is twofold. First in some cases drawing might take a significant amount of time so only drawing every 100^{th} step might make more sense. Secondly you might desire to turn the visualisation off completely which is easy if the methods are separate.

A bare-bones model can be found on the 'Lab Tools' page under the name `ca.py`. You can use this file as a basis for the upcoming CA assignments.

3.2 The GUI

Visualising data often is easier to understand intuitively than an array of numbers. Hence the PyICS framework provides a GUI. All model files in the framework are set up in such a way that if the file is executed directly (and not imported from another file) it will start this GUI. The interface, as shown in Figure 1, contains three buttons to start/stop the simulation and to manually execute the step method a single time. The reset button will call the `reset` method.

By default the GUI will call the `draw` method after every step. This behaviour can be changed via the slides on the bottom labeled "Step Size". Setting this toa value of 100 means the `draw` method will be called once every 100 steps. The other sliders labeled "Step Visualisation Delay" controls the pause between each call to the step function (i.e. adds a time delay between steps). This allows you to observe each frame more carefully.

The interface allows a user to modify several *parameters* without restarting the application or modifying the code. Parameters are defined in the constructor of the model which in Python is called `__init__`. Parameters are defined and used in the code as follows:

```
1 class MyModel(Model):
2     def __init__(self):
3         # Call parent constructor
4         Model.__init__(self)
5         # Registers foo accessible via self.foo
6         self.make_param ('foo', 10)
7         # For example:
8         print self.foo # Prints "10"
```

After the call to `make_param` the parameter is accessible as any other variable and will automatically show up in the GUI. All parameters should have a default value. Specifying a default value will also specify the type. For more details about parameters in PyICS please refer to the documentation in `model.py`.

IMPORTANT: Any and all values you use in your models that you can or want to tweak should be accessible via the GUI. Values like the lambda value should be adjustable through the GUI and should **not** be hard-coded! Failure to do so will result in a significantly lower grade. Do note that only relevant variables ought to be included as a parameter and not simply all used variables in your code.

3.3 Lag Issues

If you experience a laggy PyICS GUI you are likely running an outdated version of Python/TK-inter. The framework can also struggle when running it on a VM. Make sure you update these and/or try running the PyICS framework on bare metal.

3.4 Potential Issues on macOS

When installing Python 3 using Homebrew, it is set up to use the system version of the Tk GUI framework. The version included in macOS, however, was last updated in 2014 and is by now plagued by myriads of bugs. The easiest solution is to reinstall Python 3 using the official distribution provided at python.org.

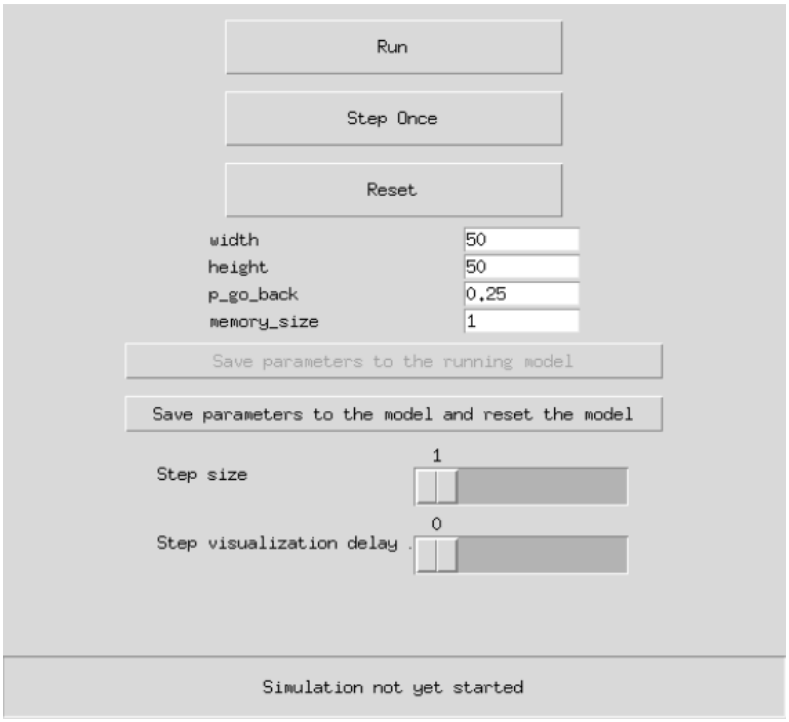

Figure 1: An example image of the interactive GUI provided with the PyICS framework.

A Python 3 installation

If the version of Python 3 installed on your system is very old, you can follow the instructions below to install the latest release.

A.1 Ubuntu 22.04/20.04

The easiest way to install Python 3.11 on Ubuntu 22.04 or 20.04 is from the ppa from deadsnakes. Note that this distribution is provided by he community, and not affiliated with Ubuntu or Python. Timely distribution of security updates is not guaranteed.

```
# Install Python 3.11
sudo add-apt-repository ppa:deadsnakes
sudo apt install python3.11 python3.11-dev python3.11-tk
# Install required packages
python3.11 -m pip install numpy matplotlib
```

Continue to appendix A.2.

A.2 After installation

If you want to run your code with Python 3.11, you will have to use `python3.11` instead of `python3`, because `python3` is still linked to your distributions primary Python 3.x release. For example to run `ca.py` you will need to execute:

```
python3.11 ca.py
```

Do not link python3 to python3.11. This can have unattended side effects, such as breaking apt!

If you need to install a pip package with Python 3.11, run `python3.11 -m pip install ...` instead of `pip3 install ...`.