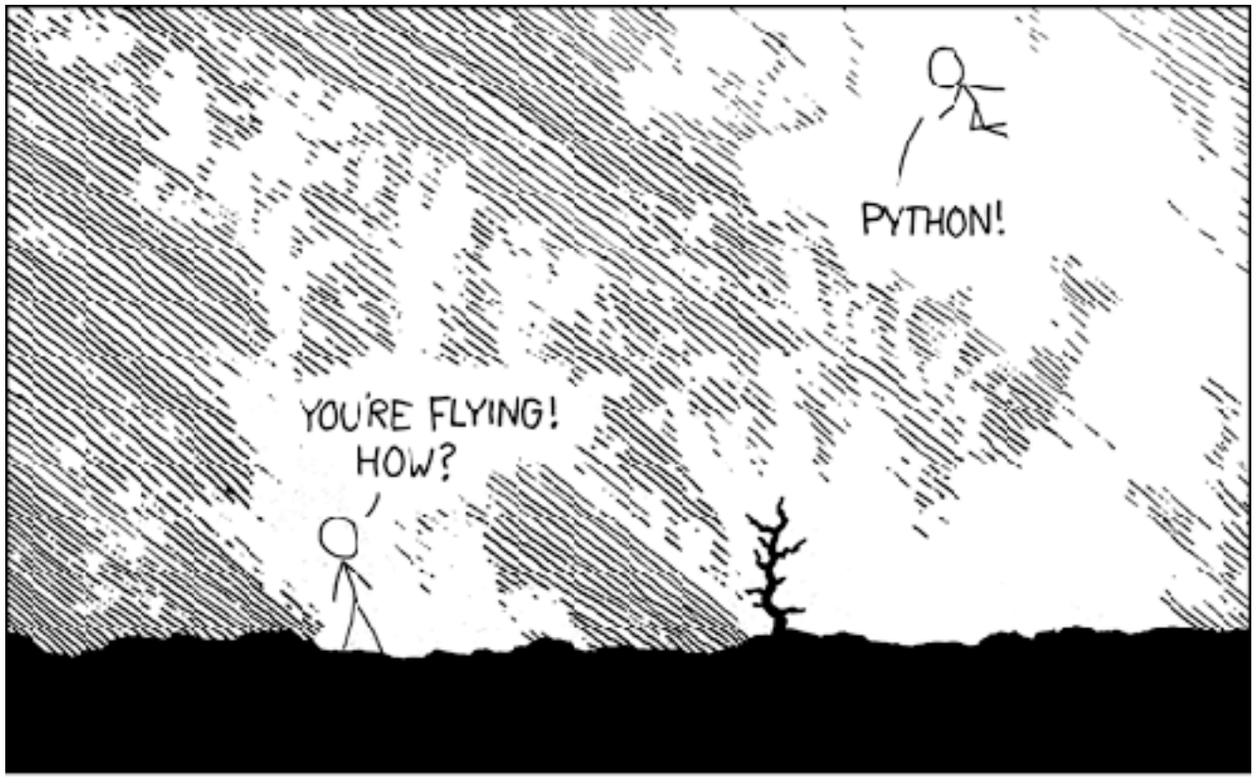
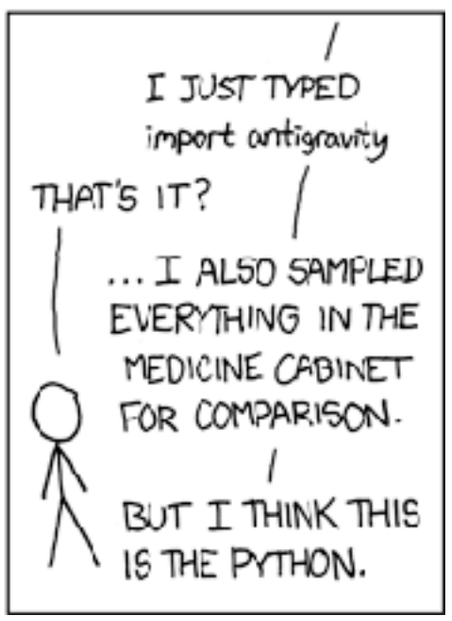
Introduction to python





14.5-16.5
@ UKE Hamburg

Jan Willem de Gee
(jwdegee@gmail.com)

Niklas Wilming
(nwilming@uke.de)

Easy syntax

Readability

High-level language

Object oriented

Why Python?

Cross-platform

Free +
open source

“Batteries included”

Widely supported

Used by
industry

PsychoPy
Psychology software in Python

Nipype:
Neuroimaging in Python
Pipelines and Interfaces

MNE
MEG + EEG ANALYSIS & VISUALIZATION

SM StatsModels
Statistics in Python

pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

scikit learn

ni learn

PyMC3

NumPy

SciPy.org

matplotlib

python™

IP[y]: IPython
Interactive Computing

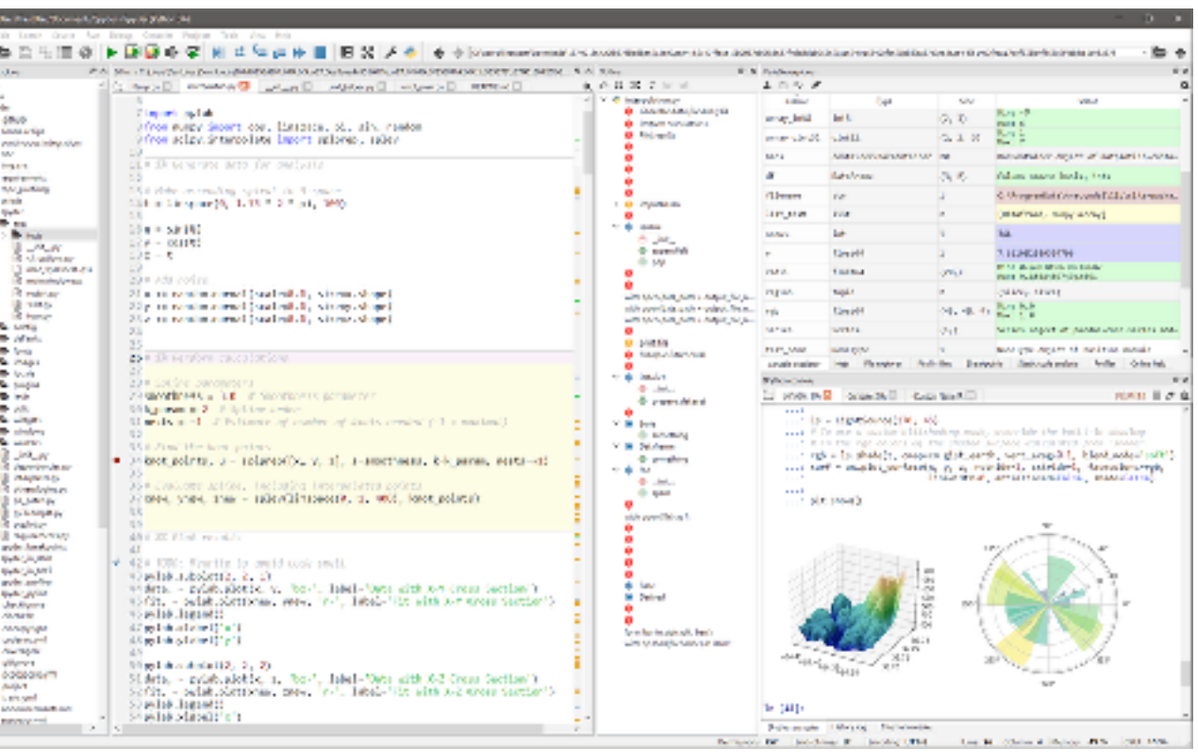
jupyter

IPython: Interactive Computing

```
~ ▷ ipython
Python 2.7.11 |Anaconda 4.0.0 (x86_64)| (default, Dec  6 2015, 18:57:58)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and ove
%quickref -> Quick reference.
help      -> Python's own help sy
object?   -> Details about 'objec

In [1]:
```

Jupyter QtConsole

```
Jupyter QtConsole 4.2.0
Python 2.7.11 |Anaconda 4.0.0 (x86_64)| (default, Dec  6 2015, 18:57:58)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help
object?   -> Details about 'ob

In [1]:
```



jupyter Grid cell analysis Last Checkpoint: 10/28/2016 (unsaved changes)

File Edit View Insert Cell Kernel Help

Code

```
plot(X,Y, color=(.8,.8,.8))
plot(X[:5000],Y[:5000], color='m')
plot(X[idx],Y[idx], 'ko')
xlim([-12.5, 12.5])
ylim([-12.5, 12.5])
grid('off')

subplot(1,4,2)
xlabel('Time')
ylabel('Firing rate')
plot(ratefct[:5000], 'm')
plot(idx, ratefct[idx], 'ko')
yticks(yticks()[0], [])

subplot(1,4,3)
xlabel('Time')
ylabel('Firing rate')
plot(ratefct[:5000], 'm')
plot(idx, ratefct[idx], 'ko')
```

OUTLINE

Monday

10:00 Introduction to python (we talk)

12:30: Lunch

14:00: 1st practical

Discuss solution (someones code)

Tuesday

10:00 Introduction to numpy/scipy (we talk)

Lunch

2nd practical: implement AI for bot

Wednesday

9:00: implement AI for bot

Tournament

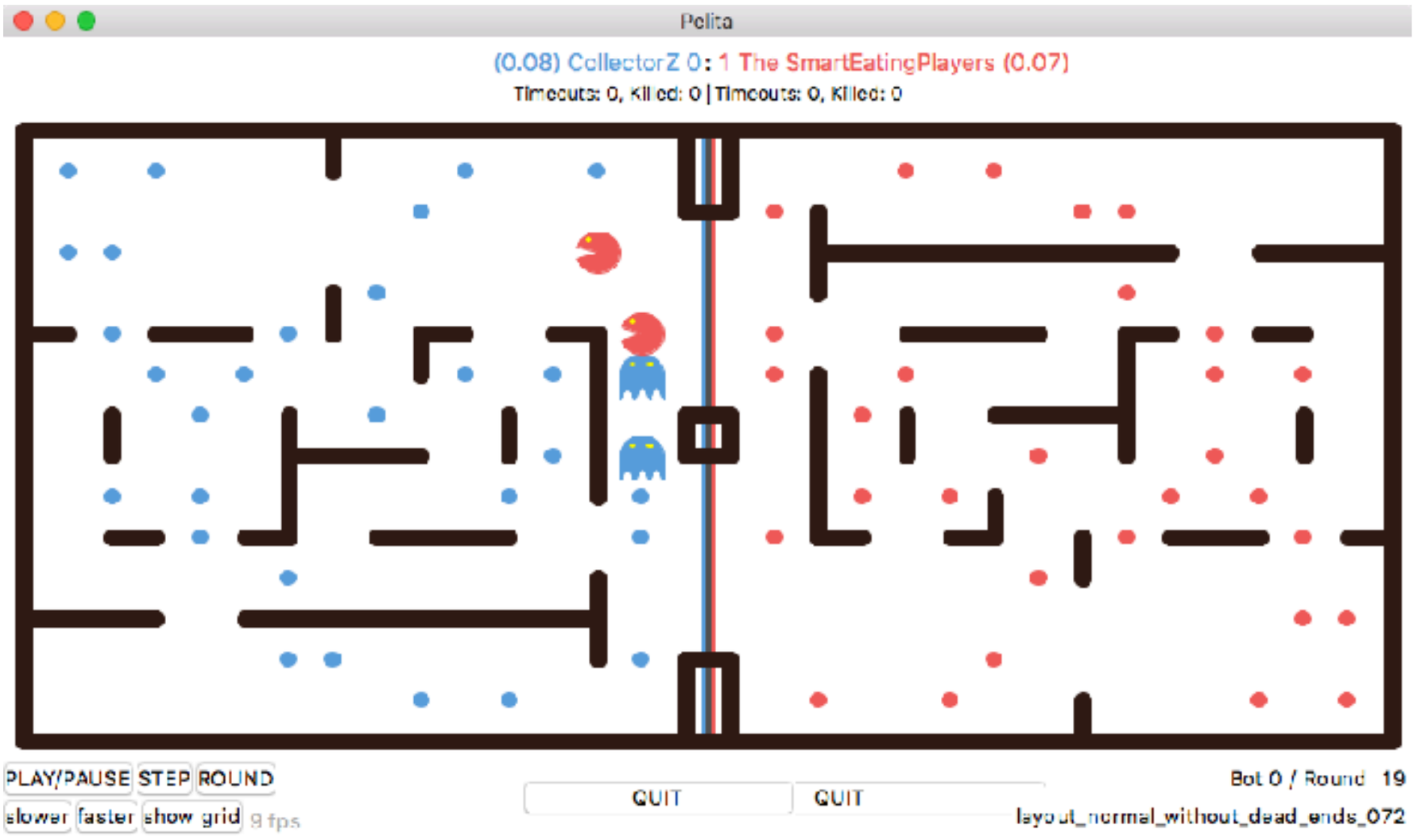
Outlook

13:00 End

During the practicals: ASK US!

OUTLINE

2nd practical: implement AI for pelita bot



During the practicals: ASK US!

Dynamic typing

No need to declare variable types.
But: Python keeps track of types.
Need explicit casts
(e.g. `int()` **or** `str()`**)**

Code Indentation

No `{}` **or** `end` **statements. Code is grouped by indentation. Use 4 spaces and no tabs.**

Operators

= assignment

== comparison

+-*/ math

Control flow

```
if statement:
    foo()
elif statement:
    bar()
else:
    foobar()
```

Loops

```
for i in collection:
    foobar()

while statement:
    foobar()
```

(remember `continue` **and** `break`**)**

Errors

```
try:
    foo()
except Exception as e:
    #fix error
finally:
    #cleanup
```

Miscellaneous

```
# Comments
"""Multi line strings"""
```

Indexing & Slicing

	-4	-3	-2	-1	
	0	1	2	3	
	A	**B**	**C**	**D**	

0 1 2 3 4

-4 -3 -2 -1

```
lst[0] = ‚A‘    lst[-3] = ‚B‘    lst[2:] = [‚C‘, ‚D‘]    lst[-1:] = ‚D‘

lst[::2] = [‚A‘, ‚C‘]    lst[::-2] = [‚D‘, ‚B‘]    lst[-1:-3] = []
```

Comprehensions

[statement loop conditional]

```
[['#' if x > y else '.' for x in range(10)] for y in range(10)]
```

```
lst = []
for y in range(10):
    for x in range(10):
        if x > y:
            lst.append(‚#‘)
        else:
            lst.append(‚.‘)
```

```
['#' for x in range(10) if x > 5]
```

```
lst = []
for x in range(10):
    if x > 5:
        lst.append(‚#‘)
```

Indexing

Starts at 0. Think interval. negative indices start from the end

Slicing

Slice and step through lists

Unpacking

Containers can be unpacked into variables:

```
a,b = [1,2]
```

References!

```
a = b = [1, 2]
a.extend([5])
b == [1,2]
```

Tuples

Immutable lists.

Dictionaries

Easy key-value store. (The thing that Matlab users didn’t know they were missing)

Comprehensions

Compact statement of simple for loops. Not shown: set comprehensions.

Anatomy of function!

```
def name(argument):
    '''
    Doc string
    '''
    body
    return value
```

Anatomy of function!

```
def name(a, *args, b=1, **kw):
    '''
    Doc string
    '''
    body
    return value1, value2, …
```

Return values

`return a, b, c` **-> tuple**

omitting a return = `return None`

Tuple unpacking

```
a, b, c = (1,2,3)
func(*(1,2,3), {,a‘:1})
```

Keyword arguments

```
def func(a=1, b=2, c=3)
```

Variable #arguments

```
def func(a, *args, b=1, **kw):
```

Functions are objects!

Have properties and can be assigned to other variables.

lambdas

Simple functions that map statement to output.

Decorators

Replace a function with a function that takes original function as input. Logging and Memoize/caching.

import / packages / modules

```
foobar/
    foobar/
        __init__.py    Tells python to treat this as a pkg.
        spirals.py     A module within the package foobar.
    setup.py           Install file
    README             Doc.
```

`__init__.py`: **Can be empty or provide global variables/functions.**

`setup.py`: **Contains logic to make package installable**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-n

from setuptools import setup, find_packages

setup(name='foobar',
      description='Foobar foobars foobar',
      author='Foo bar',
      packages=find_packages(),
      )
```

__init__.py

Turns folders into packages!

setup.py

```
python setup.py develop/install
```

imports

```
import x
import x as y
from x import y
from x import y as z, b
```

Variable #arguments

```
def func(a, *args, b=1, **kw):
```

Install packages

easy_install / pip / conda

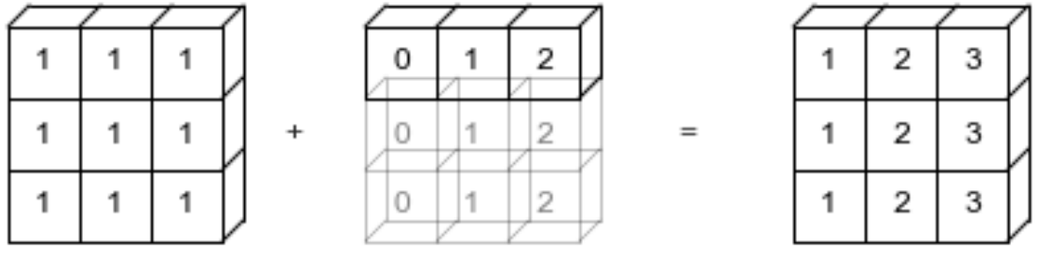
Roll your own

Doesn’t matter where you start python interpreter.

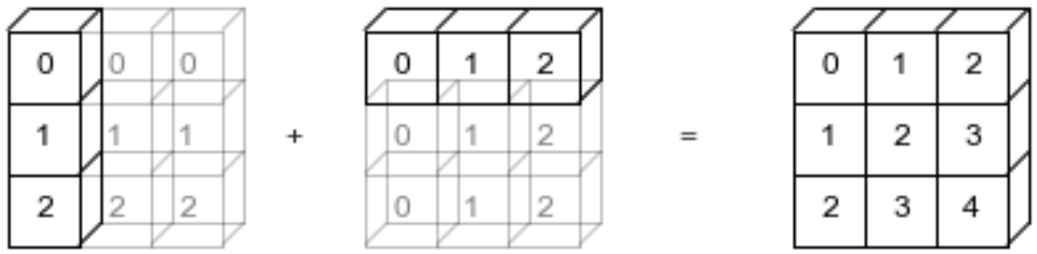
Broadcasting

$\texttt{np.arange}(3)+5$

$\texttt{np.ones}((3,3))+\texttt{np.arange}(3)$



$\texttt{np.arange}(3).\texttt{reshape}((3,1))+\texttt{np.arange}(3)$



When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when:

1. they are equal, or
2. one of them is 1

A	(1d array):	3
B	(1d array):	1
Result	(1d array):	3

A	(2d array):	3 x 3
B	(1d array):	3
Result	(2d array):	3 x 3

A	(2d array):	3 x 1
B	(1d array):	3
Result	(2d array):	3 x 3

Table 1

A	(2d array):	2 x 1
B	(3d array):	8 x 4 x 3

NumPy

Arrays

```
np.array([[1,2],[3,4]])
np.linspace(0, 10, 25)
np.ones((3,3), dtype=bool)
```

Matrix algebra

```
np.dot(M, v)
```

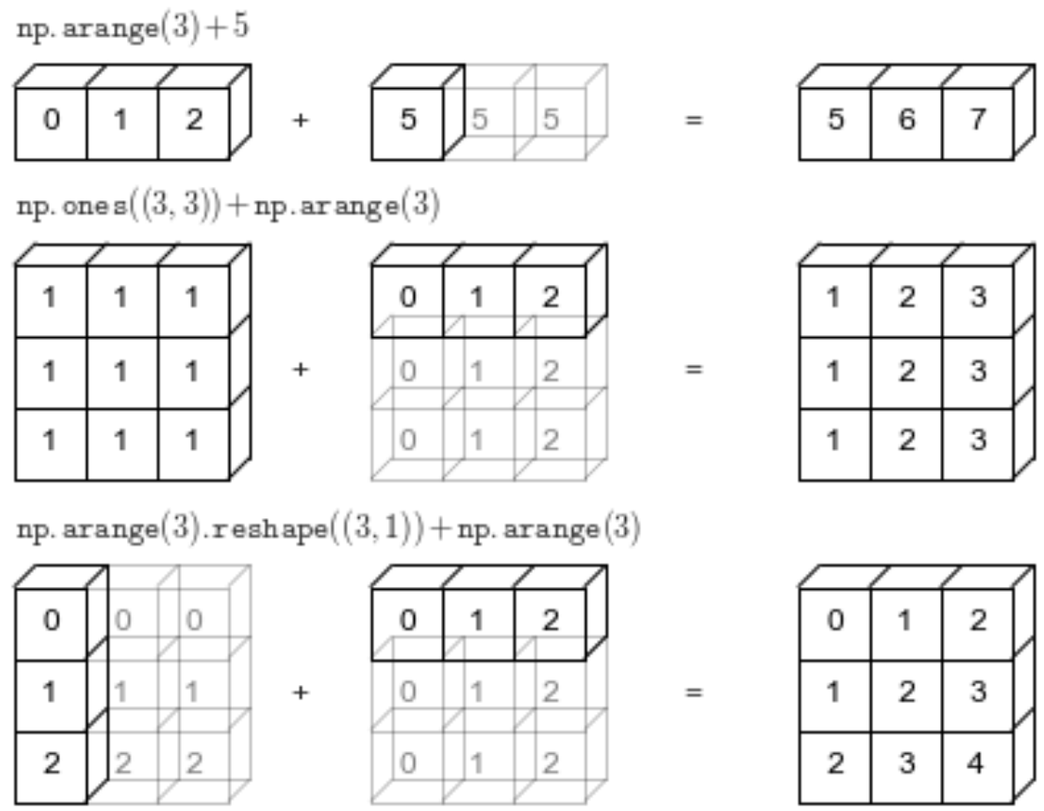
Data processing

```
np.mean(X, axis=0)
```

Reshaping & stacking

```
M.reshape((2,8))
np.vstack((M1, M2))
```

Broadcasting



Task one

The riddle: 100 prisoners are in solitary cells, unable to see, speak or communicate in any way from those solitary cells with each other. There's a central living room with one light bulb; the bulb is initially off. No prisoner can see the light bulb from his own cell. Everyday, the warden picks a prisoner at random, and that prisoner goes to the central living room. While there, the prisoner can toggle the bulb if he or she wishes. Also, the prisoner has the option of asserting the claim that all 100 prisoners have been to the living room. If this assertion is false (that is, some prisoners still haven't been to the living room), all 100 prisoners will be shot for their stupidity. However, if it is indeed true, all prisoners are set free. Thus, the assertion should only be made if the prisoner is 100 percent certain of its validity.

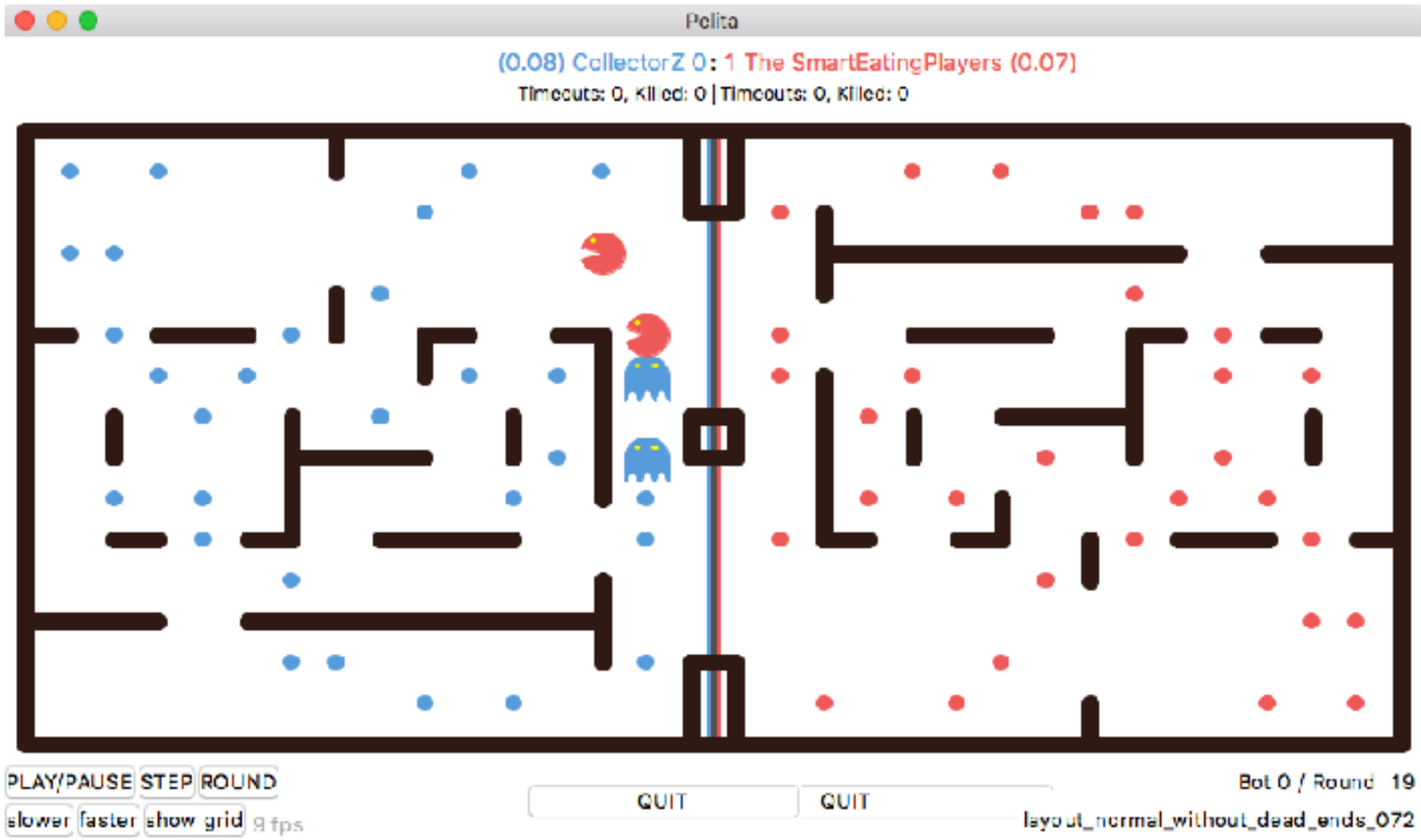
Before the random picking begins, the prisoners are allowed to get together to discuss a plan. So - what plan should they agree on, so that eventually, someone will make a correct assertion?

Question: How can the prisoners tell, with certainty, that all 100 of them have visited the central living room with the light bulb?

Task: Once you've decided on a strategy simulate how many turns the prisoners will have to take.

```
from random import choice # Choose a random number

def prisonser(N=100):
    """
    Computes how many turns the prisoners need before being freed.

    Arguments:
        N : int, default=100

    Returns:
        The number of turns required by the prisoners.
    """

    # Implement your solution here
    return turns

if __name__ == '__main__':
    nr_prisoners = 100
    turns = prisonser(N=nr_prisoners)
    print( "total turns (days) required: {}".format(turns))
```

Task two



Use git to clone this repository:

```
git clone https://github.com/nwilming/pelita
cd pelita
python setup.py install
```

Download documentation:

Task two

Use git to clone this repository:

```
git clone https://github.com/nwilming/pelita
cd pelita
python setup.py install
```

Download documentation:

Clone the tournament repository and create a folder for your team_

```
git clone https://github.com/nwilming/NINPelitaTournament
cd NINPelitaTournament
mkdir groupN
cd groupN
… add your stuff …
git pull origin master
git add …
git commit
git push origin master
```

Task two

```
from pelita.datamodel import stop
from pelita.player import AbstractPlayer, SimpleTeam


class SmartRandomPlayer(AbstractPlayer):
    def get_move(self):

        dangerous_enemy_pos = [bot.current_pos
            for bot in self.enemy_bots if bot.is_destroyer]
        killable_enemy_pos = [bot.current_pos
            for bot in self.enemy_bots if bot.is_harvester]

        smart_moves = []
        for move, new_pos in list(self.legal_moves.items()):
            if (move == stop or
                new_pos in dangerous_enemy_pos):
                continue # bad idea
            elif (new_pos in killable_enemy_pos or
                  new_pos in self.enemy_food):
                return move # get it!
            else:
                smart_moves.append(move)

        if smart_moves:
            return self.rnd.choice(smart_moves)
        else:
            # we ran out of smart moves
            return stop

def factory():
    return SimpleTeam("The Smart Random Players", SmartRandomPlayer(), SmartRandomPlayer())
```