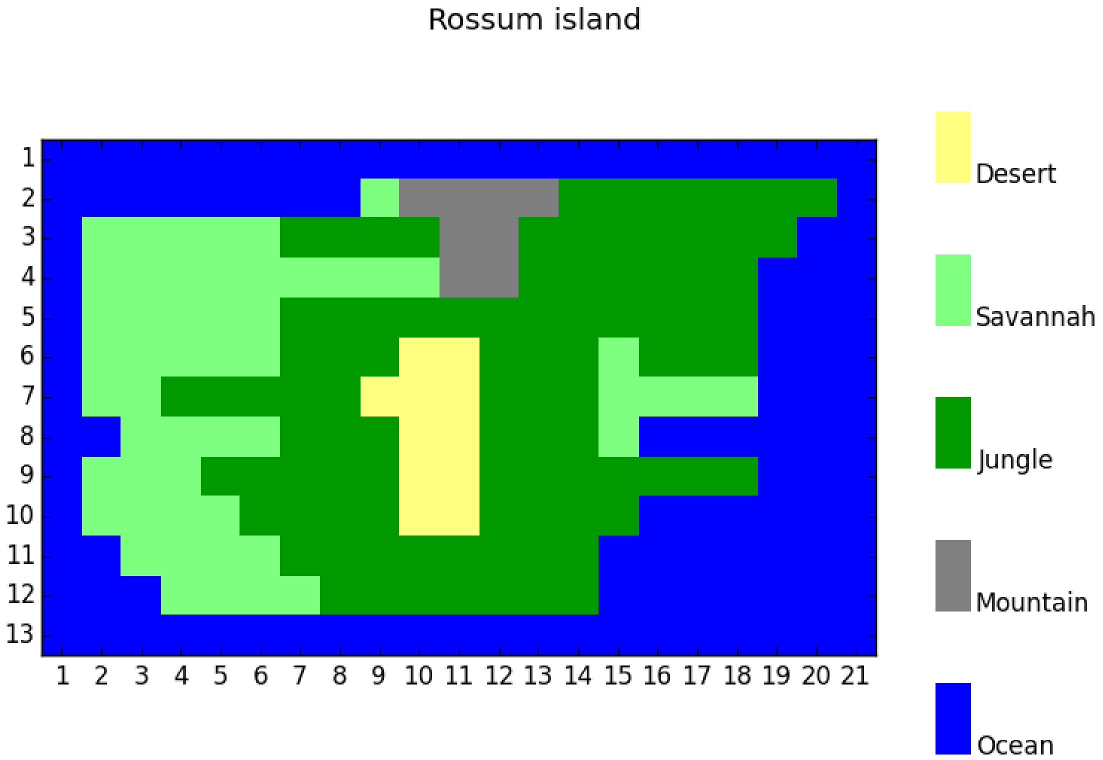
BioSim

Ecological simulation tool

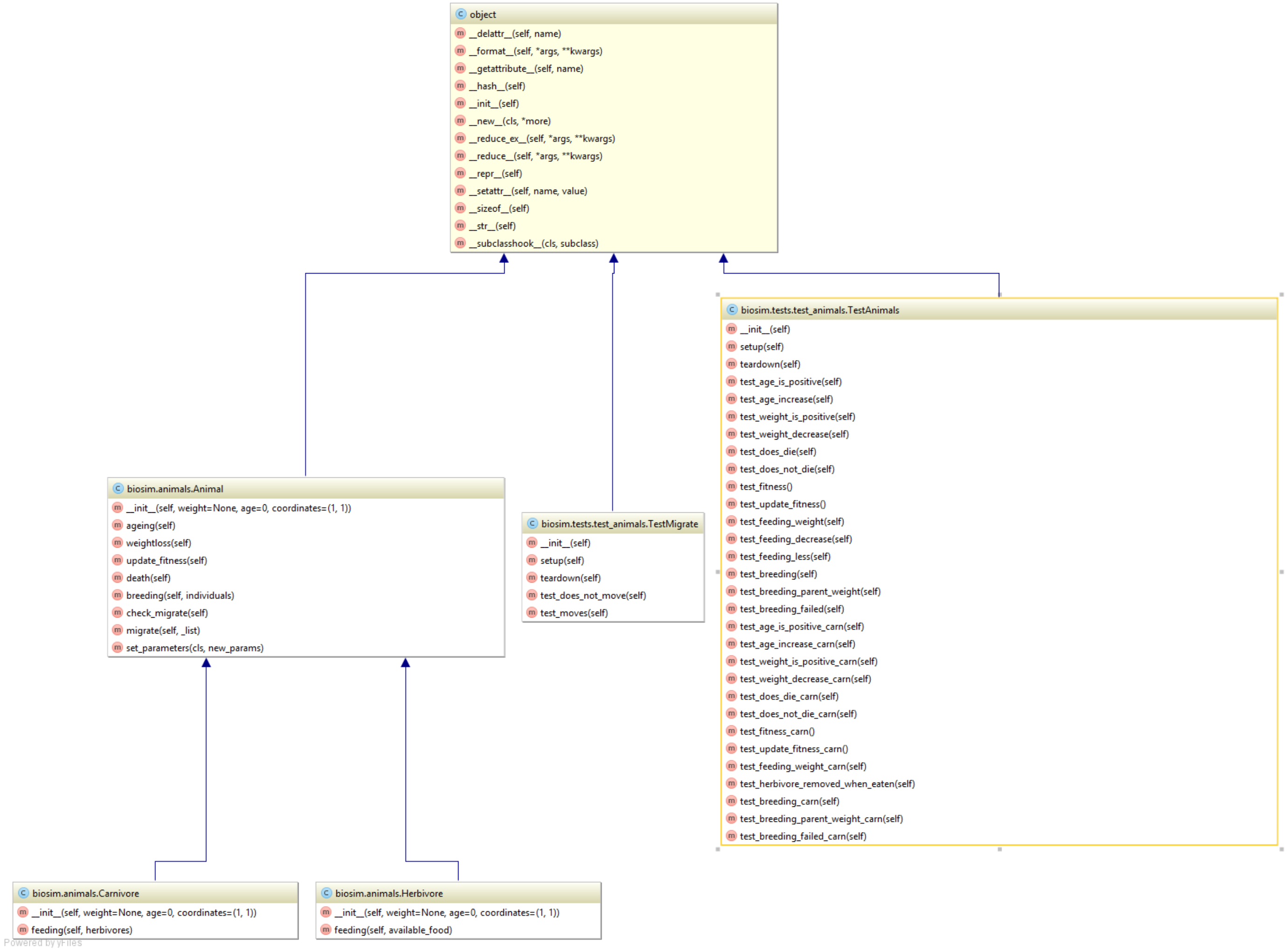
Project overview

- Create a simplified simulation.
- Two species
  - Customizable parameters
- Four biomes
  - Custom growth rates
- Plot results



Development stages

- Simple models
- Testing done in parallel
- Optimization



```
object
__delattr__(self, name)
__format__(self, *args, **kwargs)
__getattribute__(self, name)
__hash__(self)
__init__(self)
__new__(cls, *more)
__reduce_ex__(self, *args, **kwargs)
__reduce__(self, *args, **kwargs)
__repr__(self)
__setattr__(self, name, value)
__sizeof__(self)
__str__(self)
__subclasshook__(cls, subclass)
```

```
biosim.animals.Animal
__init__(self, weight=None, age=0, coordinates=(1, 1))
ageing(self)
weightloss(self)
update_fitness(self)
death(self)
breeding(self, individuals)
check_migrate(self)
migrate(self, _list)
set_parameters(cls, new_params)
```

```
biosim.tests.test_animals.TestMigrate
__init__(self)
setup(self)
teardown(self)
test_does_not_move(self)
test_moves(self)
```

```
biosim.tests.test_animals.TestAnimals
__init__(self)
setup(self)
teardown(self)
test_age_is_positive(self)
test_age_increase(self)
test_weight_is_positive(self)
test_weight_decrease(self)
test_does_die(self)
test_does_not_die(self)
test_fitness()
test_update_fitness()
test_feeding_weight(self)
test_feeding_decrease(self)
test_feeding_less(self)
test_breeding(self)
test_breeding_parent_weight(self)
test_breeding_failed(self)
test_age_is_positive_carn(self)
test_age_increase_carn(self)
test_weight_is_positive_carn(self)
test_weight_decrease_carn(self)
test_does_die_carn(self)
test_does_not_die_carn(self)
test_fitness_carn()
test_update_fitness_carn()
test_feeding_weight_carn(self)
test_herbivore_removed_when_eaten(self)
test_breeding_carn(self)
test_breeding_parent_weight_carn(self)
test_breeding_failed_carn(self)
```

```
biosim.animals.Carnivore
__init__(self, weight=None, age=0, coordinates=(1, 1))
feeding(self, herbivores)
```

```
biosim.animals.Herbivore
__init__(self, weight=None, age=0, coordinates=(1, 1))
feeding(self, available_food)
```

Problems

- Combination of Herbivore and Carnivore
  - Superclass
- Implementation of migration method
  - Implemented late in development
  - Dependent on all other classes/files
- Translating written equations

From formula to code

An animal moves with probability $\mu\Phi$.

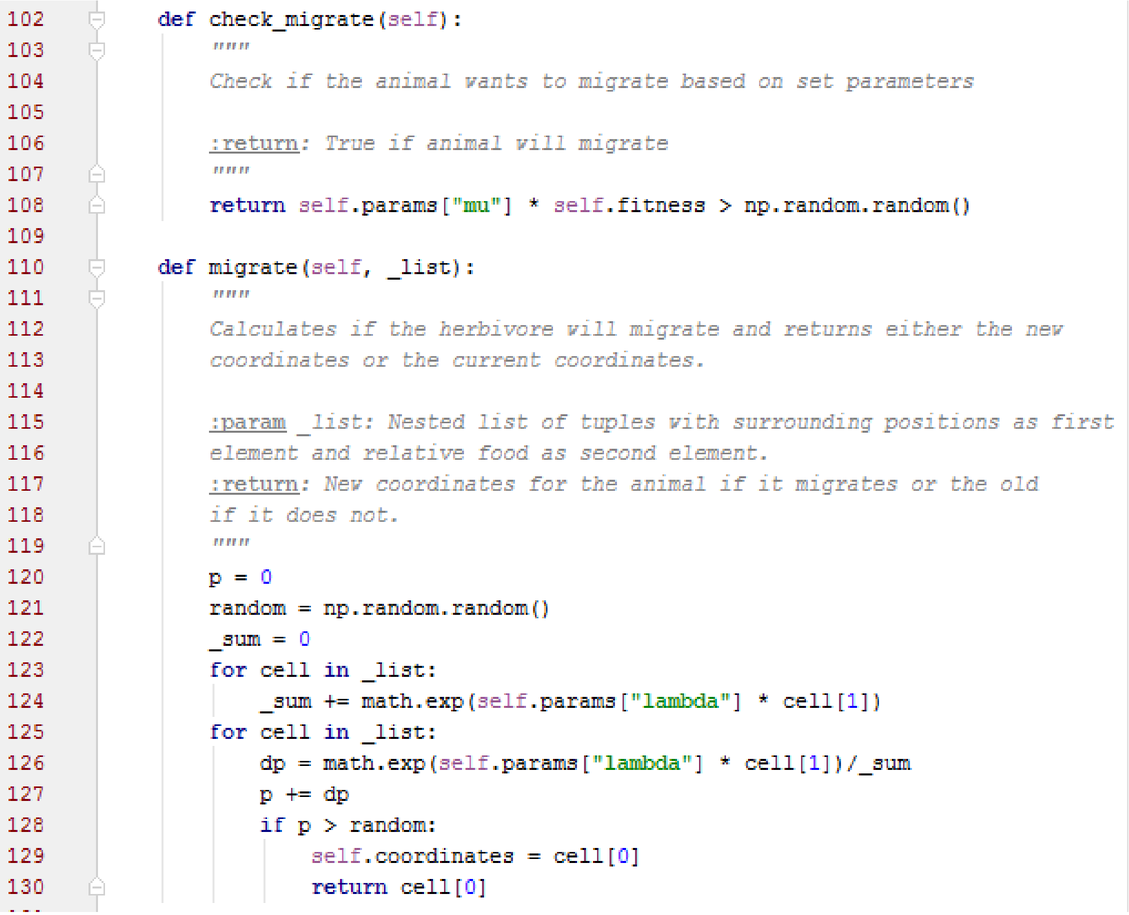
Then, the *propensity* to move from i to $j \in \mathcal{C}^{(i)}$ is given by

$$\pi_{i\to j} = \begin{cases} 0 & \text{if } j \text{ is Mountain or Ocean} \\ e^{\lambda\epsilon_j} & \text{otherwise} \end{cases} \tag{6}$$

and the corresponding *probability* to move from i to j is given by

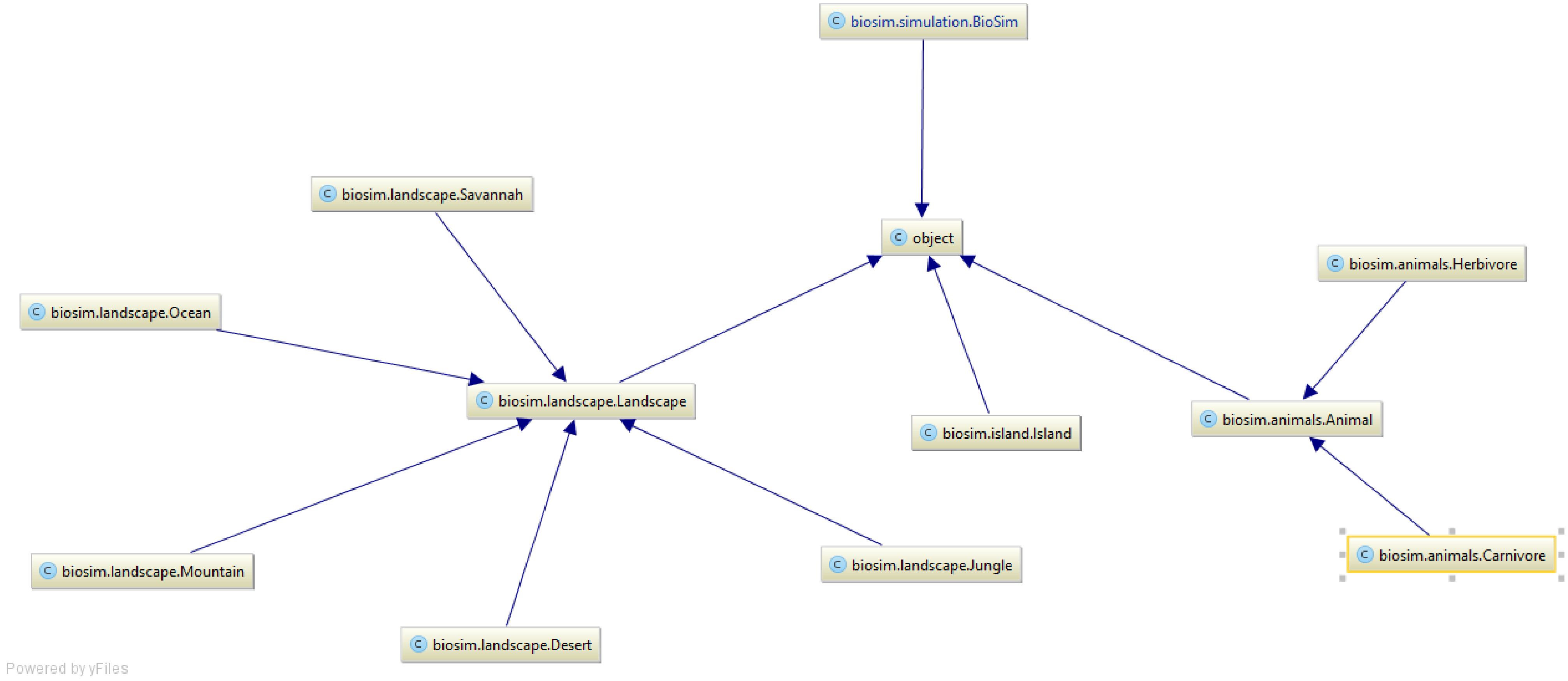
$$p_{i\to j} = \frac{\pi_{i\to j}}{\sum_{j\in\mathcal{C}^{(i)}} \pi_{i\to j}} \; . \tag{7}$$

“Modelling the Ecosystem of Rossumøya” Dr. Hans Ekkehard Plesser, NMBU



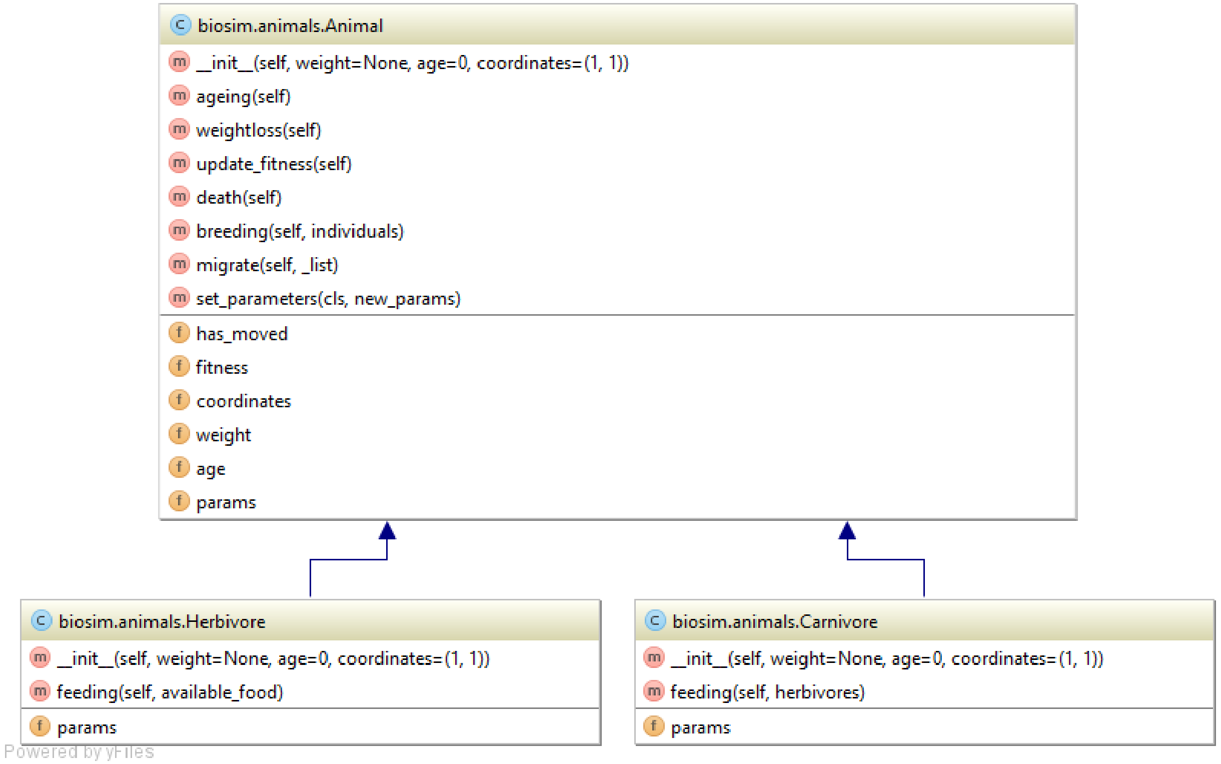
```
    def check_migrate(self):
        """
        Check if the animal wants to migrate based on set parameters

        :return: True if animal will migrate
        """
        return self.params["mu"] * self.fitness > np.random.random()

    def migrate(self, _list):
        """
        Calculates if the herbivore will migrate and returns either the new
        coordinates or the current coordinates.

        :param _list: Nested list of tuples with surrounding positions as first
        element and relative food as second element.
        :return: New coordinates for the animal if it migrates or the old
        if it does not.
        """
        p = 0
        random = np.random.random()
        _sum = 0
        for cell in _list:
            _sum += math.exp(self.params["lambda"] * cell[1])
        for cell in _list:
            dp = math.exp(self.params["lambda"] * cell[1])/_sum
            p += dp
            if p > random:
                self.coordinates = cell[0]
                return cell[0]
```

Code structure



Code structure

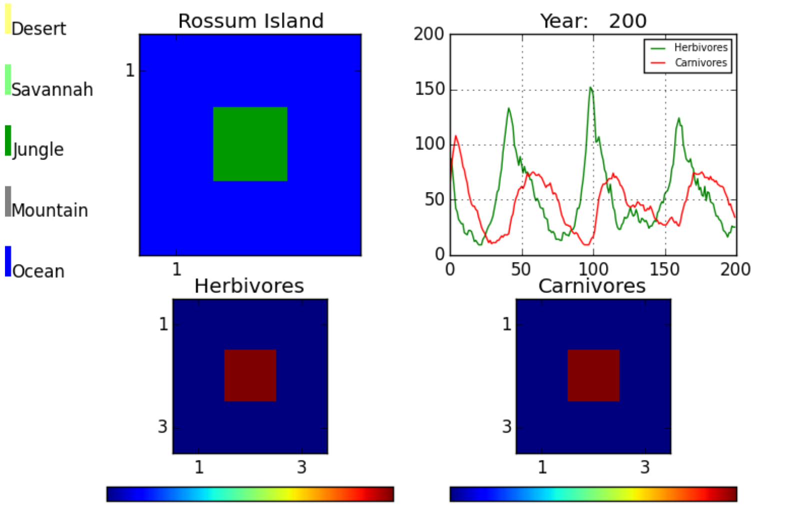
- Performance
- Included example scripts
- Redistribution of zip/tar.giz files
    - Complete package with documentation and example files


biosim.animals.Animal
__init__(self, weight=None, age=0, coordinates=(1, 1))
ageing(self)
weightloss(self)
update_fitness(self)
death(self)
breeding(self, individuals)
migrate(self, _list)
set_parameters(cls, new_params)
has_moved
fitness
coordinates
weight
age
params
biosim.animals.Herbivore
__init__(self, weight=None, age=0, coordinates=(1, 1))
feeding(self, available_food)
params
biosim.animals.Carnivore
__init__(self, weight=None, age=0, coordinates=(1, 1))
feeding(self, herbivores)
params
Powered by yFiles

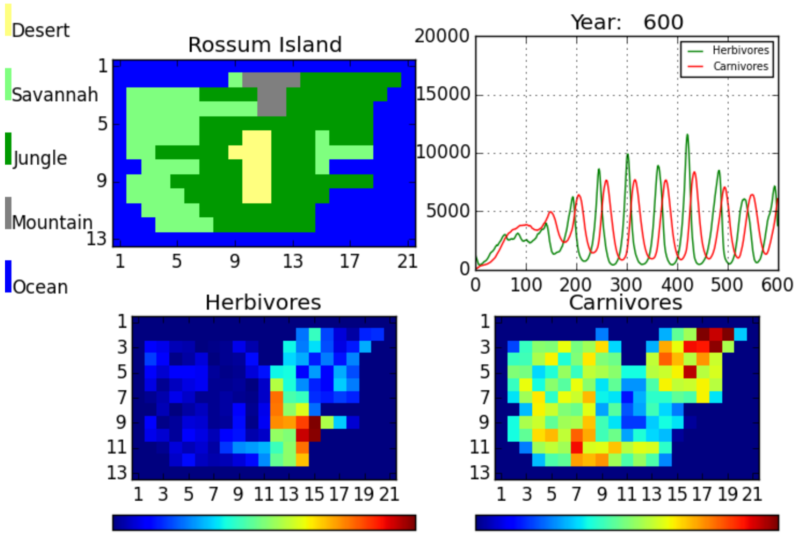
Example: Superclass «Animal»

Functionality and results

- Single cell average herbivores:
  - 250-300 Herbivores
- Single cell average herb & carn:
  - 50-120 Herbivores
  - 40-70 Carnivores
- Different re-growth rates and animal params



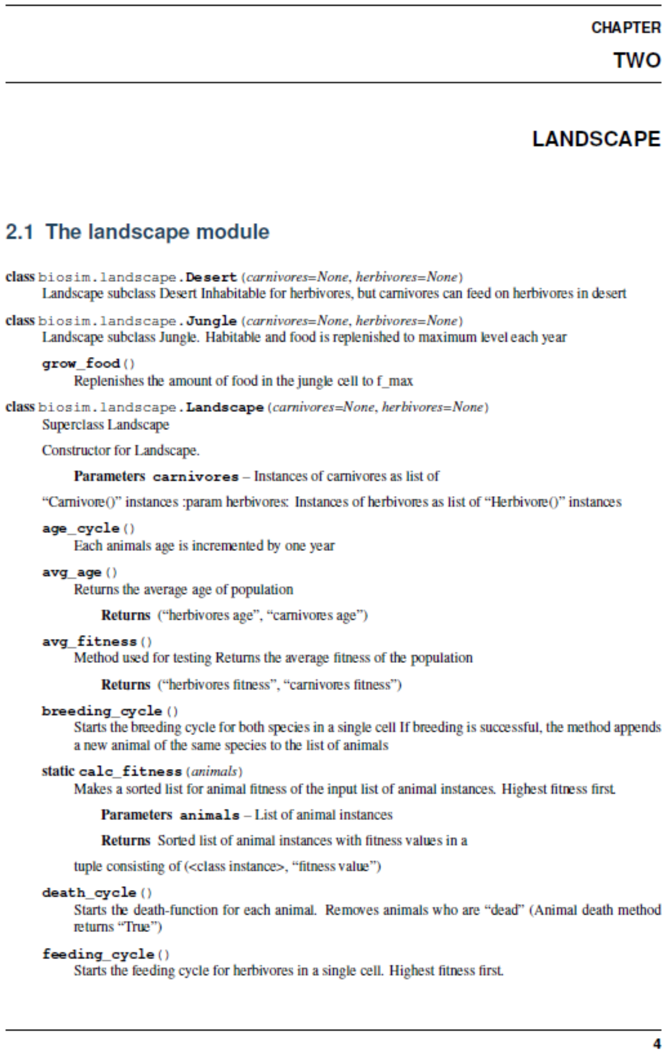
Single cell simulation, default params



Custom animal and island parameters

Documentation

- Documentation for use of BioSim generated using sphinx
- Explains all classes and methods in BioSim
- Further instructions in README



CHAPTER

TWO

LANDSCAPE

2.1 The landscape module

class `biosim.landscape.`**`Desert`** (*carnivores=None*, *herbivores=None*)
Landscape subclass Desert Inhabitable for herbivores, but carnivores can feed on herbivores in desert

class `biosim.landscape.`**`Jungle`** (*carnivores=None*, *herbivores=None*)
Landscape subclass Jungle. Habitable and food is replenished to maximum level each year

`grow_food` ()
Replenishes the amount of food in the jungle cell to f_max

class `biosim.landscape.`**`Landscape`** (*carnivores=None*, *herbivores=None*)
Superclass Landscape

Constructor for Landscape.

Parameters **`carnivores`** – Instances of carnivores as list of

“Carnivore()” instances :param herbivores: Instances of herbivores as list of “Herbivore()” instances

`age_cycle` ()
Each animals age is incremented by one year

`avg_age` ()
Returns the average age of population

Returns (“herbivores age”, “carnivores age”)

`avg_fitness` ()
Method used for testing Returns the average fitness of the population

Returns (“herbivores fitness”, “carnivores fitness”)

`breeding_cycle` ()
Starts the breeding cycle for both species in a single cell If breeding is successful, the method appends a new animal of the same species to the list of animals

static **`calc_fitness`** (*animals*)
Makes a sorted list for animal fitness of the input list of animal instances. Highest fitness first.

Parameters **`animals`** – List of animal instances

Returns Sorted list of animal instances with fitness values in a

tuple consisting of (<class instance>, “fitness value”)

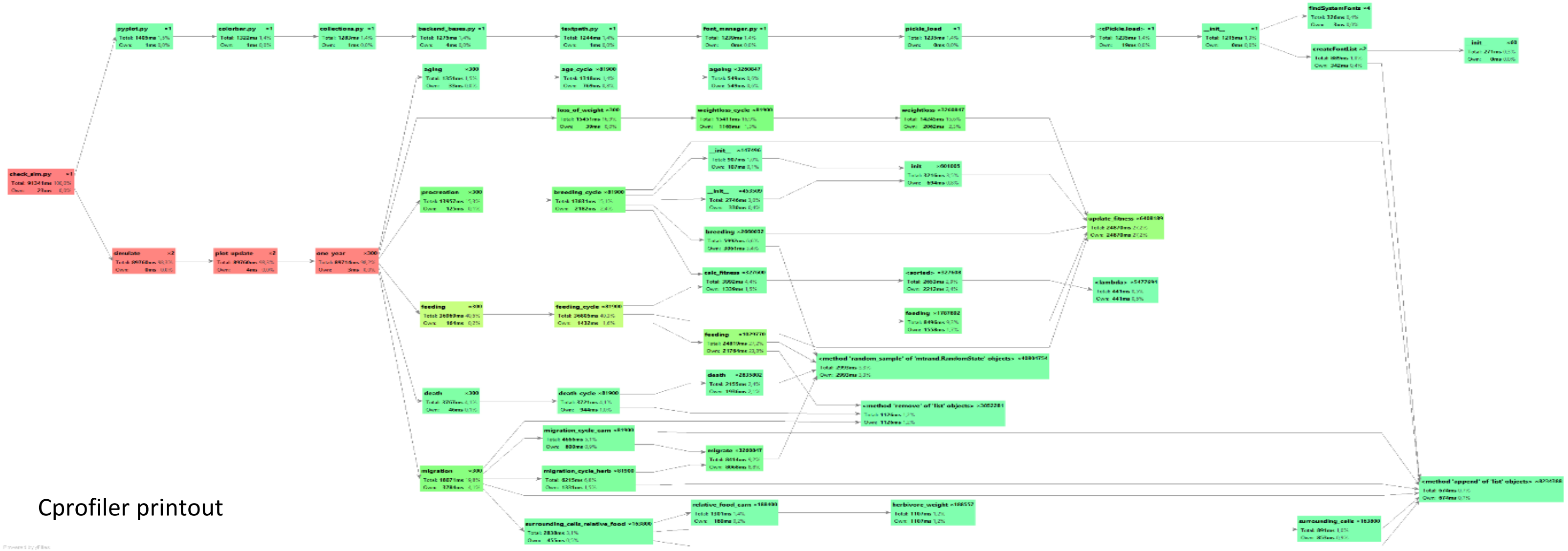
`death_cycle` ()
Starts the death-function for each animal. Removes animals who are “dead” (Animal death method returns “True”)

`feeding_cycle` ()
Starts the feeding cycle for herbivores in a single cell. Highest fitness first.

4

Further developments

- Revision II
  - With better class structure and more flexible code
  - Update fitness and carnivore feeding bottleneck
  - Use of comprehensions in list and dictionary generation
  - Implement use of cython-compiled code to speed up code execution



Cprofiler printout