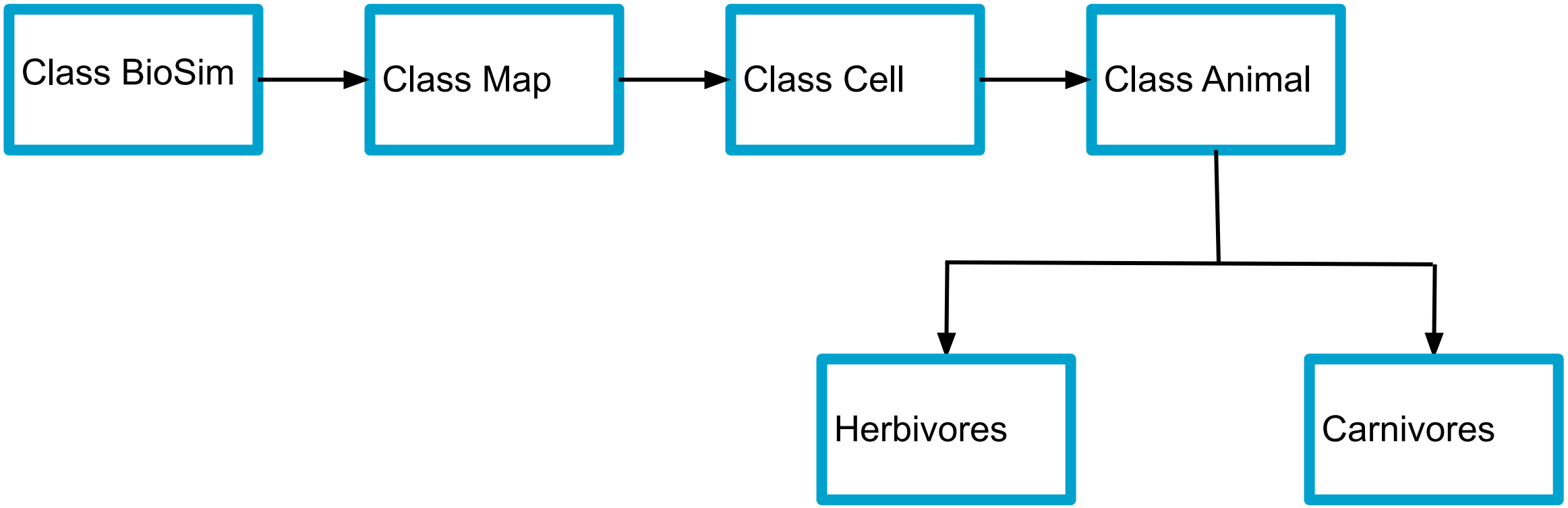
Bio-Simulation Model

Ecosystem in the Rossumøya island

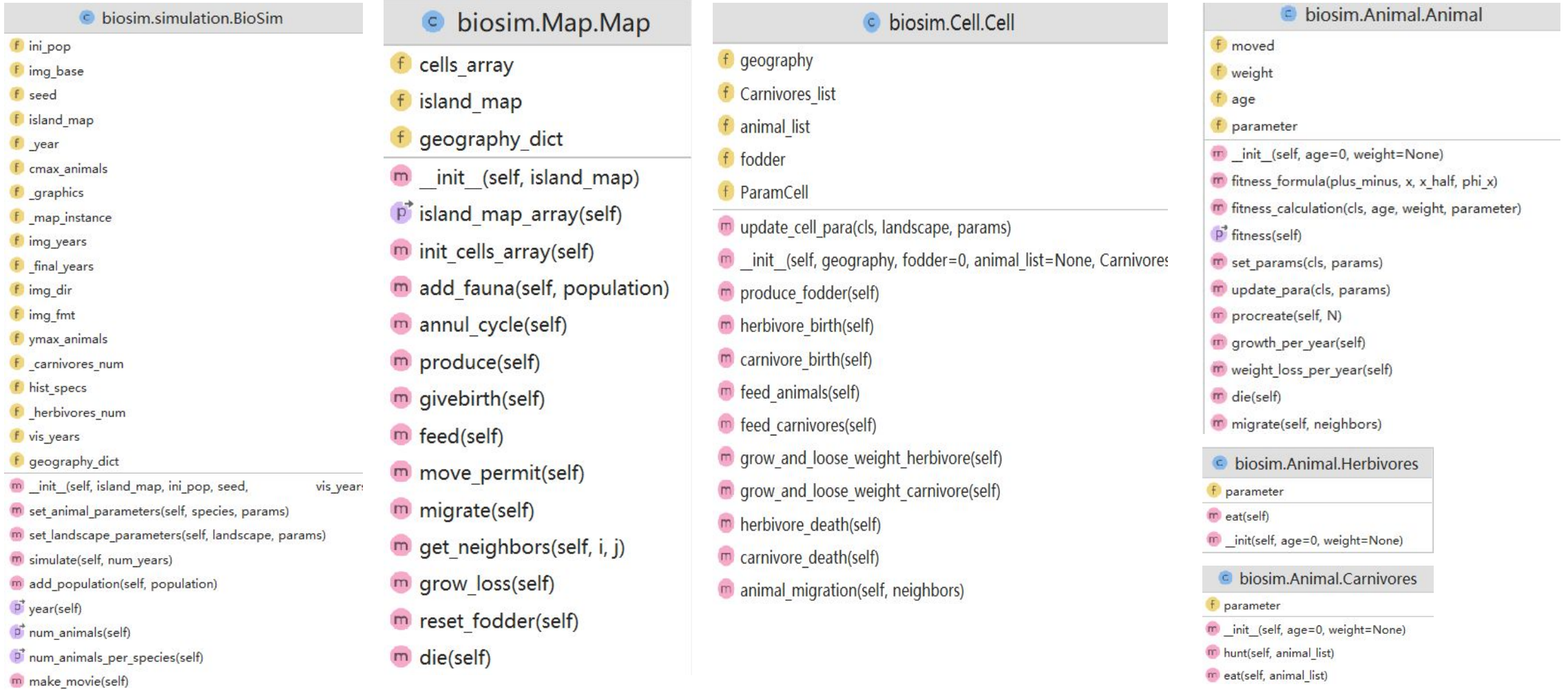
By Hongpeng Zhang & Sujan Devkota

2023- 01-26

Structure:


Class BioSim
Class Map
Class Cell
Class Animal
Herbivores
Carnivores

Structure:


biosim.simulation.BioSim
ini_pop
img_base
seed
island_map
_year
cmax_animals
_graphics
_map_instance
img_years
_final_years
img_dir
img_fmt
ymax_animals
_carnivores_num
hist_specs
_herbivores_num
vis_years
geography_dict
__init__(self, island_map, ini_pop, seed,
set_animal_parameters(self, species, params)
set_landscape_parameters(self, landscape, params)
simulate(self, num_years)
add_population(self, population)
year(self)
num_animals(self)
num_animals_per_species(self)
make_movie(self)
biosim.Map.Map
cells_array
island_map
geography_dict
__init__(self, island_map)
island_map_array(self)
init_cells_array(self)
add_fauna(self, population)
annul_cycle(self)
produce(self)
givebirth(self)
feed(self)
move_permit(self)
migrate(self)
get_neighbors(self, i, j)
grow_loss(self)
reset_fodder(self)
die(self)
biosim.Cell.Cell
geography
Carnivores_list
animal_list
fodder
ParamCell
update_cell_para(cls, landscape, params)
__init__(self, geography, fodder=0, animal_list=None, Carnivores
produce_fodder(self)
herbivore_birth(self)
carnivore_birth(self)
feed_animals(self)
feed_carnivores(self)
grow_and_loose_weight_herbivore(self)
grow_and_loose_weight_carnivore(self)
herbivore_death(self)
carnivore_death(self)
animal_migration(self, neighbors)
biosim.Animal.Animal
moved
weight
age
parameter
__init__(self, age=0, weight=None)
fitness_formula(plus_minus, x, x_half, phi_x)
fitness_calculation(cls, age, weight, parameter)
fitness(self)
set_params(cls, params)
update_para(cls, params)
procreate(self, N)
growth_per_year(self)
weight_loss_per_year(self)
die(self)
migrate(self, neighbors)
biosim.Animal.Herbivores
parameter
eat(self)
_init(self, age=0, weight=None)
biosim.Animal.Carnivores
parameter
__init__(self, age=0, weight=None)
hunt(self, animal_list)
eat(self, animal_list)

Problems and Solutions:(maybe change to “Problems”?)

Problems:
Animals:

- How animals procreate ?
- How carnivores hunt and eat ?

Cell:

- how to use cells in the map/ island properly ?

Migration:

- dynamics of the whole animal migration ?
- How to stop incoming animals to migrate ?

Solutions:

Specific Methods(maybe change to “Solutions”?):

Codes:

- Creating cells instances as numpy array
- carnivores hunt & eat method
- move permit

Tests:

- Used parametrization, fixtures

Documentation:

- Sphinx
  - Example Code, note boxes and figures

Creating cell instances:

```
@property
def island_map_array(self):
    """Turn island_map into a numpy array..."""
    return np.array([list(line) for line in self.island_map.split('\n')])

Hongpeng +1
def init_cells_array(self):
    """..."""
    cells_array = np.empty(self.island_map_array.shape, dtype=object)
    for geo in Map.geography_dict.keys():
        n_geo = len(cells_array[self.island_map_array == geo])
        cells_array[self.island_map_array == geo] = [Cell.Cell(geography=geo) for _ in
                                                     range(n_geo)]

    return cells_array
```

Specific Methods:

Fitness Calculation method:

```
@staticmethod
@numba.jit
def fitness_formula(plus_minus, x, x_half, phi_x):
    """Method to return the formula to be used in fitness calculation..."""
    formula = 1 / (1 + np.power(np.e, plus_minus * phi_x * (x - x_half)))
    return formula

Sujan Devkota
@classmethod
def fitness_calculation(cls, age, weight, parameter):
    """Method to calculate the fitness of the animals..."""
    fitness = cls.fitness_formula(1, age, parameter['a_half'], parameter['phi_age']) \
        * cls.fitness_formula(-1, weight, parameter['w_half'], parameter['phi_weight'])
    return fitness
```

Procreate method:

```
def procreate(self, N):
    """..."""
    log_mu = np.log(self.parameter['w_birth'] ** 2 / (
            self.parameter['w_birth'] ** 2 + self.parameter['sigma_birth'] ** 2) ** 0.5)
    log_sigma = np.log(1 + self.parameter['sigma_birth'] ** 2 / self.parameter['w_birth']
                       ** 2) ** 0.5

    prob = min(1, self.parameter['gamma'] * self.fitness * N)
    if (self.weight >= (self.parameter['w_birth'] + self.parameter['sigma_birth']) *
        self.parameter['zeta']) \
            & (self.age > 1) & (random.choices([False, True], [1 - prob, prob])[0]):
        baby_weight = random.lognormvariate(log_mu, log_sigma)
        if self.weight - self.parameter['xi'] * baby_weight >= 0:
            self.weight -= self.parameter['xi'] * baby_weight
            baby_animal = self.__class__(0, baby_weight)
            return True, baby_animal
        else:
            return False,
    else:
        return False,
```

Specific Methods:

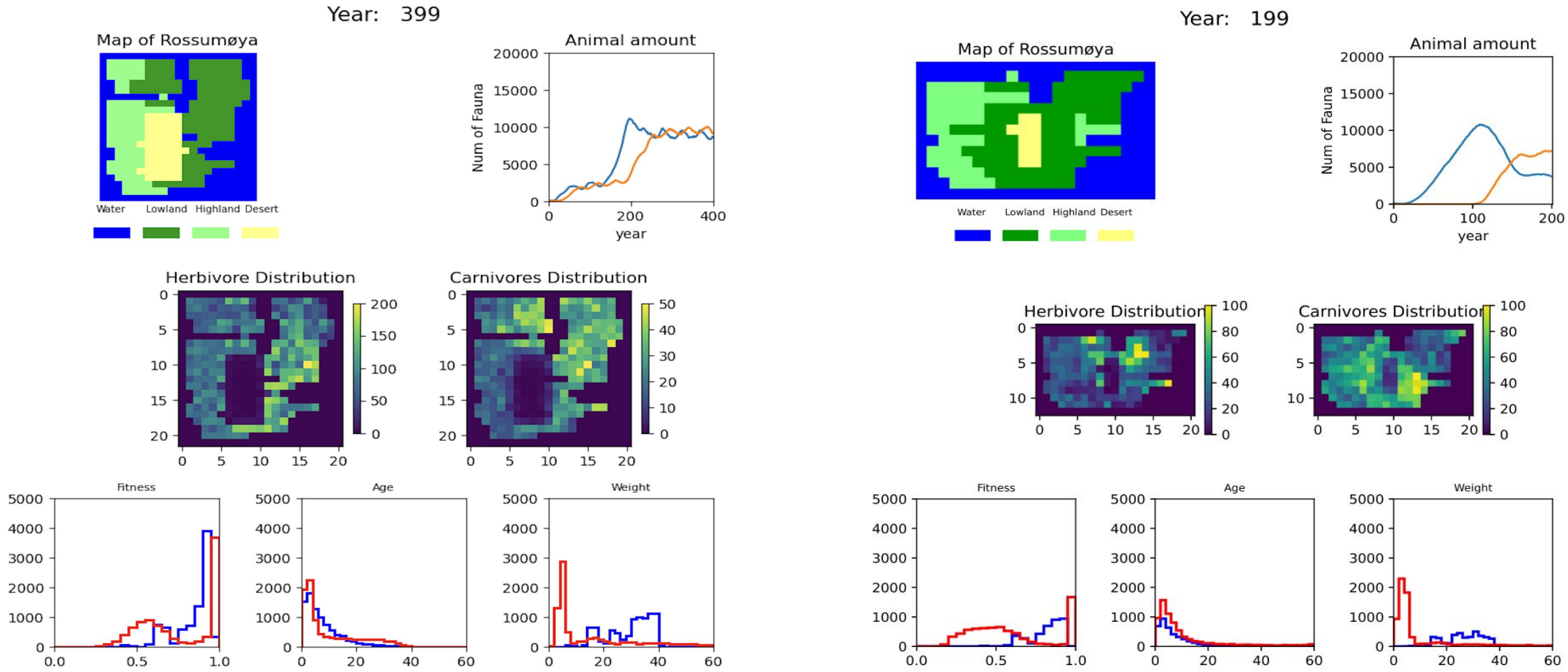
Hunt Method:

```
def hunt(self, animal_list):
    """..."""
    hunt_list = []
    fit = self.fitness
    for animal in animal_list:
        prob = 1
        animal_fitness = animal.fitness
        if fit <= animal_fitness:
            prob = 0
        elif 0 < fit - animal_fitness < self.parameter['DeltaPhiMax']:
            prob = (fit - animal_fitness) / self.parameter['DeltaPhiMax']
        if random.choices([False, True], [1 - prob, prob])[0]:
            hunt_list.append(animal)
```

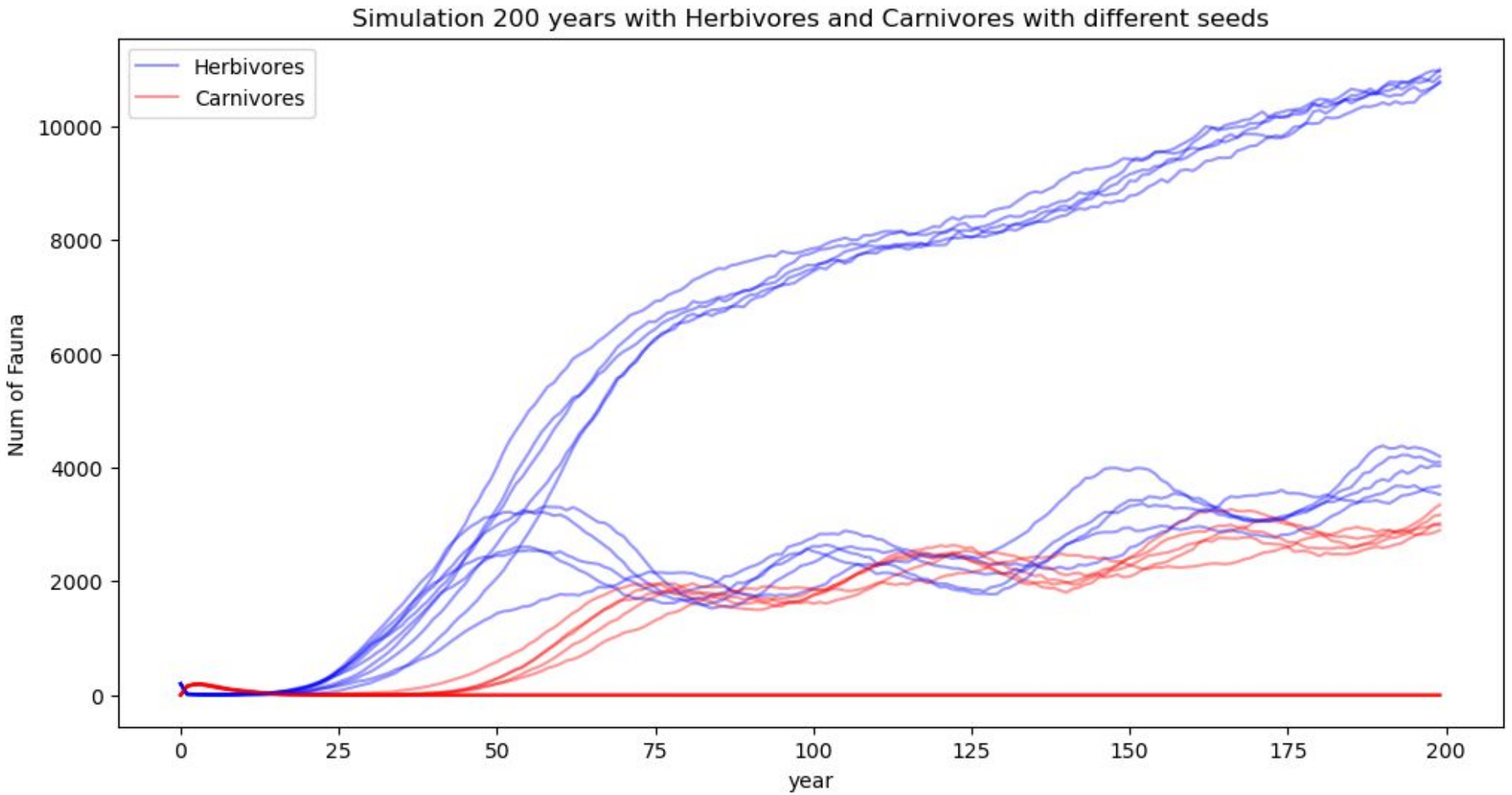
Eat Method:

```
def eat(self, animal_list):
    """..."""

    hunt_list = self.hunt(animal_list)
    hunt_weight_list = [animal.weight for animal in hunt_list]
    cum_weight_list = np.cumsum(hunt_weight_list)
    if len(cum_weight_list) < 2:
        eaten_animal_list = hunt_list
        self.weight += self.parameter['beta'] * sum(cum_weight_list)
    elif self.parameter['F'] > cum_weight_list[-2]:
        eaten_animal_list = hunt_list
        self.weight += self.parameter['beta'] * cum_weight_list[-1]
    else:
        stop_eat_index = np.searchsorted(cum_weight_list, self.parameter['F'], side='left')
        eaten_animal_list = hunt_list[:stop_eat_index + 1]
        self.weight += self.parameter['beta'] * cum_weight_list[stop_eat_index]

    return eaten_animal_list
```

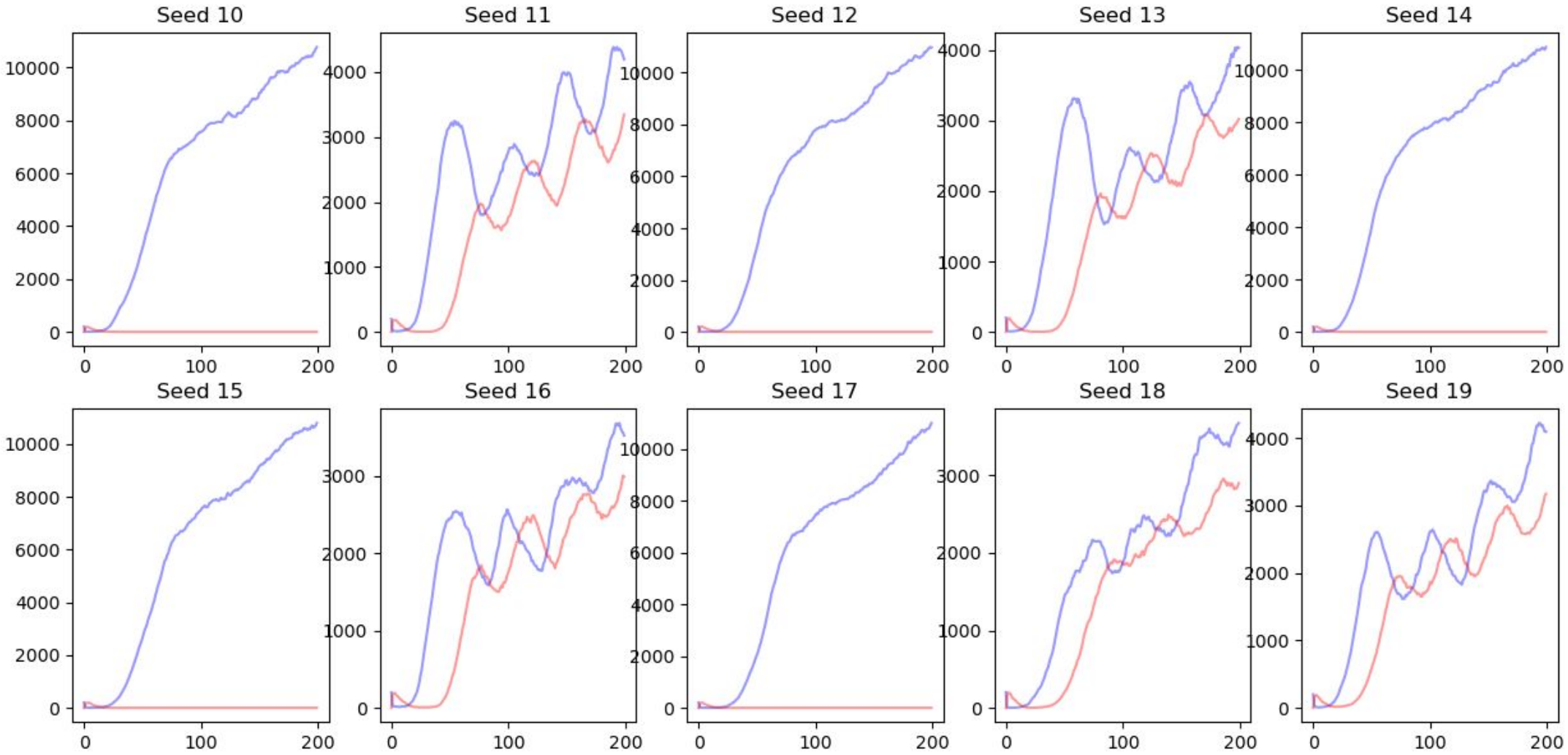
Results of sample sim and check sim:



Results of sample sim and check sim:



Results of sample sim and check sim:



Results of example simulation:

T-test

```
C:\Users\uranu\anaconda3\envs\inf200jan\python.exe C:/Users/uranu/INF200git/biosim-a25-hongpeng-sujan/examples/T-test.py
The init population of herbivores and carnivores, have normal weight distribution with mu=20, sigma=4.
After 300 years simulation
**********************************************************************************************
One sample T-test on herbivores and carnivores weight
The p_value of T-test herbivores weight is 1.4312453090737645e-158
The p_value of T-test carnivores weight is 0.00518138816444705 35
The p_value of normal distribution herbivores weight is 2.73002447541238e-13
The p_value of normal distribution carnivores weight is 2.5660369572935347e-17
```

Limitations:

- Some methods are not called level by level
- Statistical test result could be better
- Tests could be improved.
- Optimization could have been better - make it faster, fix weak warnings, fixing cross level calling.

Thank You!