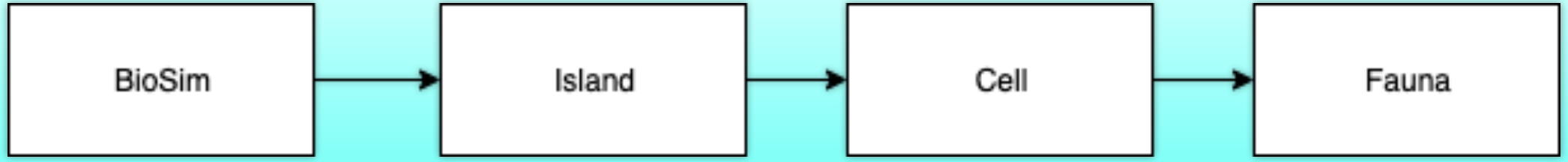
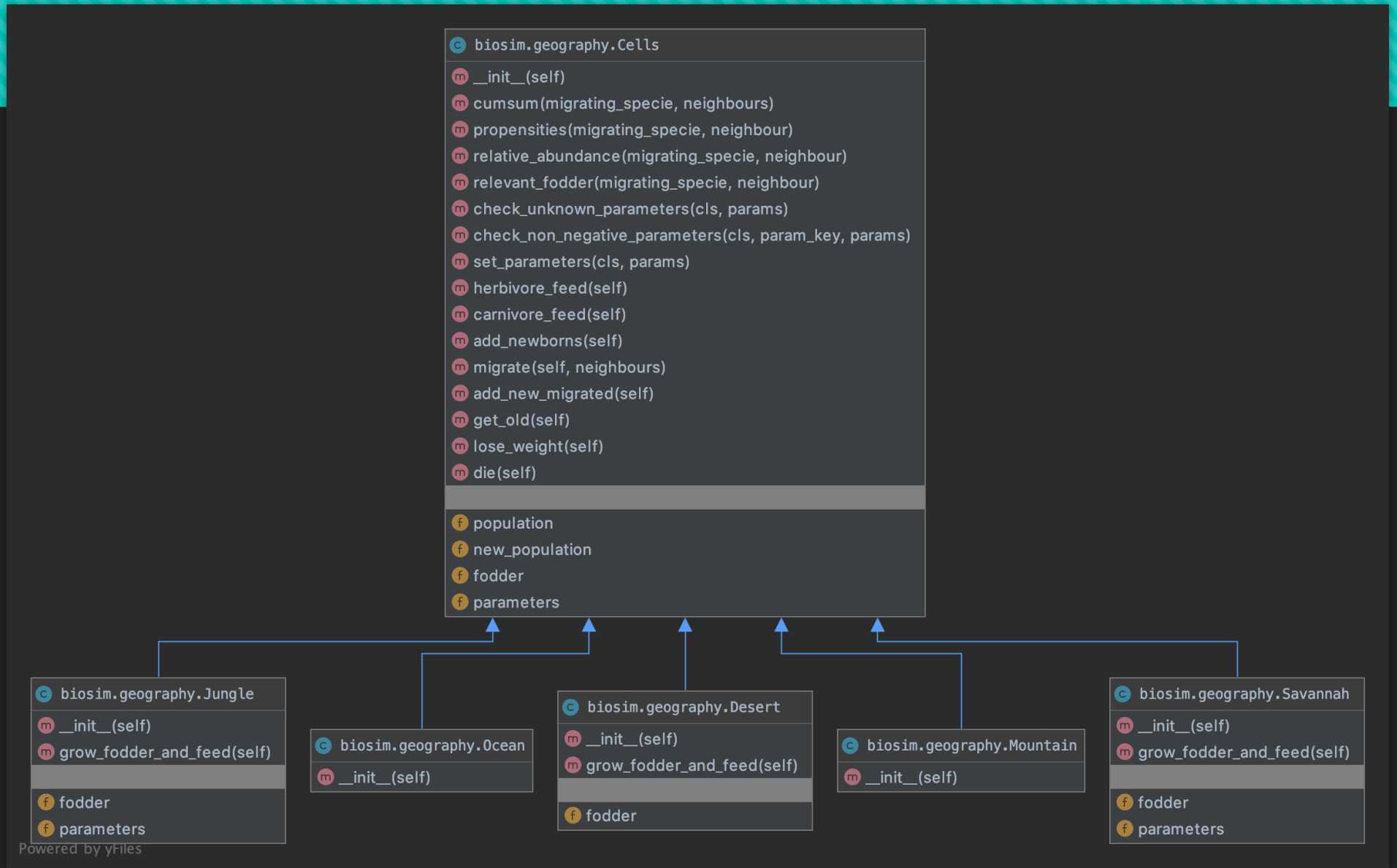
Modelling the Ecosystem of Rossumøya

By Fábio Rodrigues Pereira and Rabin Senchuri

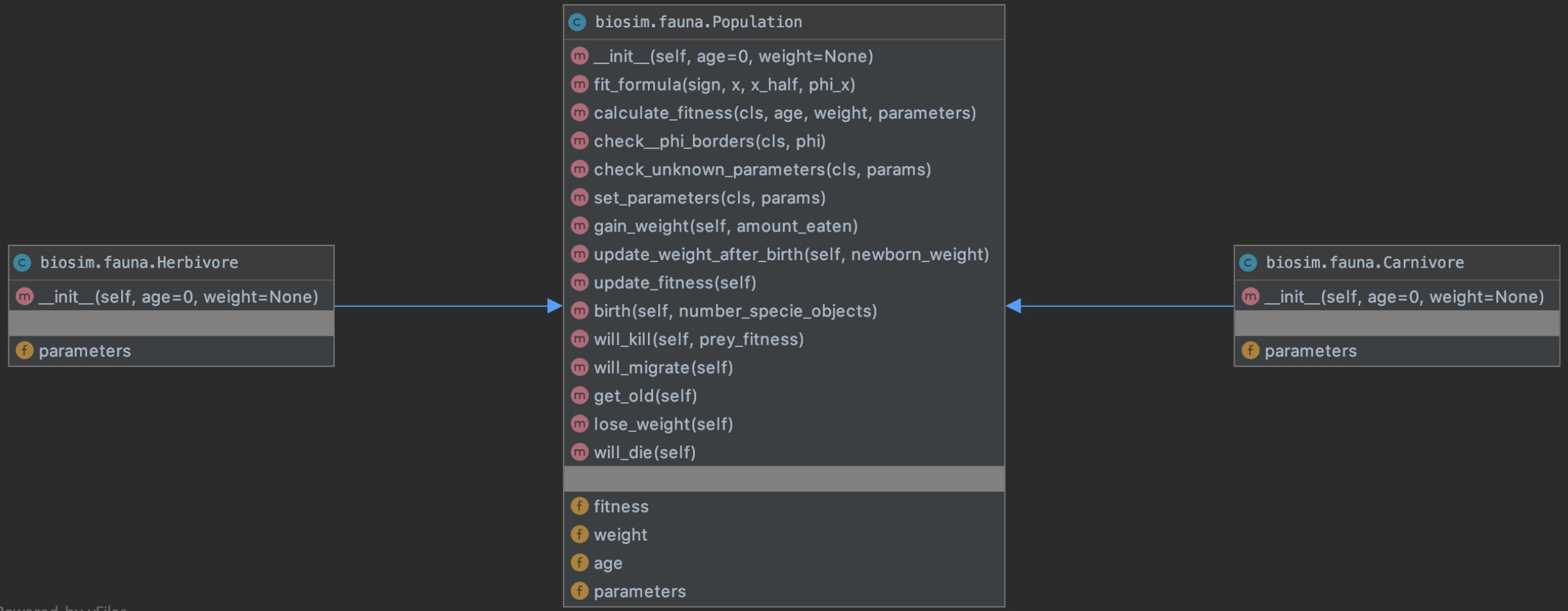
Classes



Geography Cell Class



Fauna Class


biosim.fauna.Herbivore
__init__(self, age=0, weight=None)
parameters
biosim.fauna.Population
__init__(self, age=0, weight=None)
fit_formula(sign, x, x_half, phi_x)
calculate_fitness(cls, age, weight, parameters)
check__phi_borders(cls, phi)
check_unknown_parameters(cls, params)
set_parameters(cls, params)
gain_weight(self, amount_eaten)
update_weight_after_birth(self, newborn_weight)
update_fitness(self)
birth(self, number_specie_objects)
will_kill(self, prey_fitness)
will_migrate(self)
get_old(self)
lose_weight(self)
will_die(self)
fitness
weight
age
parameters
biosim.fauna.Carnivore
__init__(self, age=0, weight=None)
parameters
Powered by yFiles

Simulation

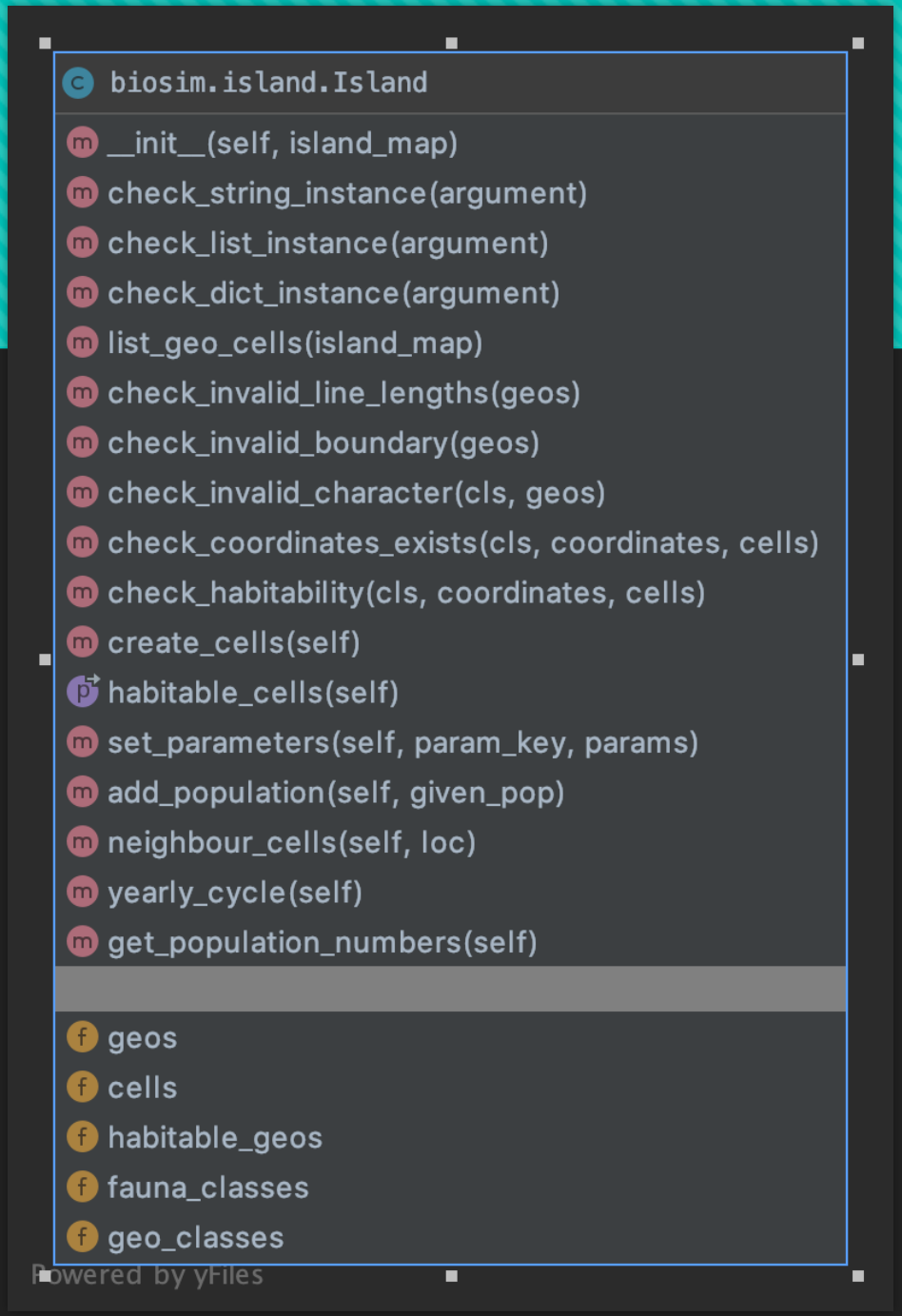
biosim.simulation.BioSim

- __init__(self, island_map, ini_pop, seed, ymax_animals=None, cmax_animals=None, img_base=None, img_fmt='png')
- generate_map_array(self)
- num_animals(self)
- year(self)
- num_animals_per_species(self)
- animal_distribution(self)
- set_animal_parameters(self, species, params)
- set_landscape_parameters(self, landscape, params)
- add_population(self, population)
- simulate(self, num_years, vis_years=1, img_years=None)
- save_figures(self)
- create_mp4(self, mov_fmt=DEFAULT_MOVIE_FORMAT)
- setup_graphics(self)
- herbivore_line(self)
- carnivore_line(self)
- static_map(self)
- update_counter_graph(self, pop_count)
- update_herb(self, pop)
- update_carn(self, distribution)
- update_graphics(self)

- img_no
- _carn_img_axis
- img_base
- last_year
- fig
- island
- _carnivore_line
- _mean_ax
- _island_map
- _img_axis
- cmax_animals
- _herbivore_line
- _map
- final_year
- herb_pop
- img_fmt
- _herb_img_axis
- year_num
- carn_pop
- ymax_animals
- map_colors
- map_labels

Powered by yFiles

Island Class



Minor changes on carnivore_feed method:

```python
def carnivore_feed(self):
    """This method organizes the population of carnivore in order
    of greatest fitness (those who eat first) to worst. Then,
    organizes the population of herbivore in order of worst fitness
    (those who are hunt first) to greatest. Then, per each animal,
    it is applied the carnivore eating rules, as following:

    Formula and conditions:
    ----------

        -> Carnivores prey on herbivores on Jungle, Savannah and
           Desert landscapes, and do not prey on each oder;
        -> A carnivore tries to kill a herbivore per time, beginning
           with the herbivore with worst fitness, and then to the
           next herbivore until has eaten an amount 'F' of
           herbivore weight;
        -> The probability to kill a herbivore is given by the method
           'will_kill()';
        -> The carnivore weight increases by the method
           'gain_weight()' which also updates its fitness;
        -> Every herbivore killed is removed from the population
           by the python's built-in method '.remove()'.
    """
```

-> We had a merge conflict with the files and it ended up that we pushed an old version of our carnivore_feed code for submission.

Before on geography.py:

```
258 263     for carnivore in self.population['Carnivore']:
259         self.population['Herbivore'] = carnivore.eat_herb(
260 +           self.population['Herbivore'])
```

And on fauna.py

```
236     def eat_herb(self, herbivores):
237         herbs_survived = []
238         feed = 0
239
240         for herbivore in herbivores[::-1]:
241             if feed < self.parameters["F"] \
242                     and self.will_kill(herbivore.fitness):
243                 if feed + herbivore.weight > self.parameters["F"]:
244                     herbivore.weight = self.parameters["F"] - feed
245                     self.weight += (
246                         self.parameters["beta"] * herbivore.weight
247 +                   )
248                     weight_eaten = self.parameters["F"]
249
250                 else:
251                     self.weight += (
252                         self.parameters["beta"] * herbivore.weight
253                     )
254                     feed += herbivore.weight
255             else:
256                 herbs_survived.append(herbivore)
257
258         return herbs_survived
259
```

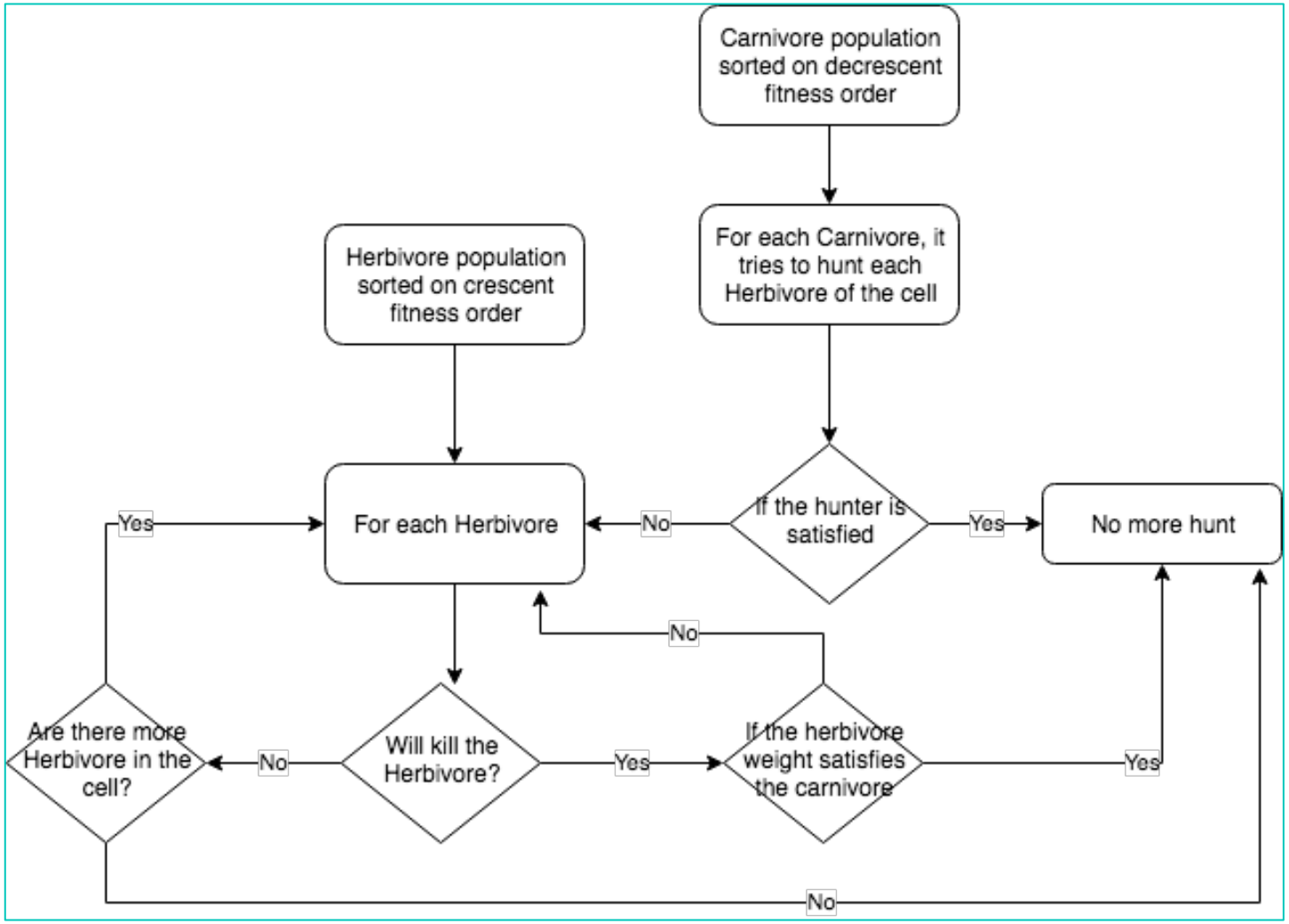
After:

```python
        self.population['Carnivore'].sort(key=lambda h: h.fitness,
                                          reverse=True)

        self.population['Herbivore'].sort(key=lambda h: h.fitness)

        for carnivore in self.population['Carnivore']:
            appetite = carnivore.parameters['F']
            amount_eaten = 0

            for herbivore in self.population['Herbivore']:

                if amount_eaten >= appetite:
                    break

                elif carnivore.will_kill(herbivore.fitness):
                    food_wanted = appetite - amount_eaten

                    if herbivore.weight <= food_wanted:
                        amount_eaten += herbivore.weight
                        self.population['Herbivore'].remove(herbivore)

                    elif herbivore.weight > food_wanted:
                        amount_eaten += food_wanted
                        self.population['Herbivore'].remove(herbivore)

            carnivore.gain_weight(amount_eaten)
```

-> The method has been changed after submission, as soon as we got aware. But it is important to say that the previous code is not wrong and has not caused any difference on the simulation's results.

Flow chart of carnivore eating rules:

```
258         self.population['Carnivore'].sort(key=lambda h: h.fitness,
259                                           reverse=True)
260
261         self.population['Herbivore'].sort(key=lambda h: h.fitness)
262
263         for carnivore in self.population['Carnivore']:
264             appetite = carnivore.parameters['F']
265             amount_eaten = 0
266
267             for herbivore in self.population['Herbivore']:
268
269                 if amount_eaten >= appetite:
270                     break
271
272                 elif carnivore.will_kill(herbivore.fitness):
273                     food_wanted = appetite - amount_eaten
274
275                     if herbivore.weight <= food_wanted:
276                         amount_eaten += herbivore.weight
277                         self.population['Herbivore'].remove(herbivore)
278
279                     elif herbivore.weight > food_wanted:
280                         amount_eaten += food_wanted
281                         self.population['Herbivore'].remove(herbivore)
282
283             carnivore.gain_weight(amount_eaten)
284
```



Minor changes on migration method:

BEFORE

```
298         """
299         for migrating_specie, animals in self.population.items():
300             if len(neighbours) > 0 and len(animals) > 0:
301                 for animal in animals:
302                     cum_prob = self.cumsum(migrating_specie,
303                                            neighbours)
304                     if animal.will_migrate():
305                         n = 0
306                         while np.random.random() >= cum_prob[n]:
307                             n += 1
308                         neighbours[n].new_population[
309                             migrating_specie].append(animal)
310                         self.population[migrating_specie].remove(animal)
311
```

AFTER

```
322         np.random.shuffle(neighbours)
323         for migrating_specie, animals in self.population.items():
324             if len(neighbours) > 0 and len(animals) > 0:
325                 for animal in animals:
326                     cum_prob = self.cumsum(migrating_specie,
327                                            neighbours)
328                     if animal.will_migrate():
329                         n = 0
330                         while np.random.random() >= cum_prob[n]:
331                             n += 1
332                             n = 0 if n > 3 else n
333                         neighbours[n].new_population[migrating_specie].\
334                             append(animal)
335                         self.population[migrating_specie].remove(animal)
```

Changes:

1:

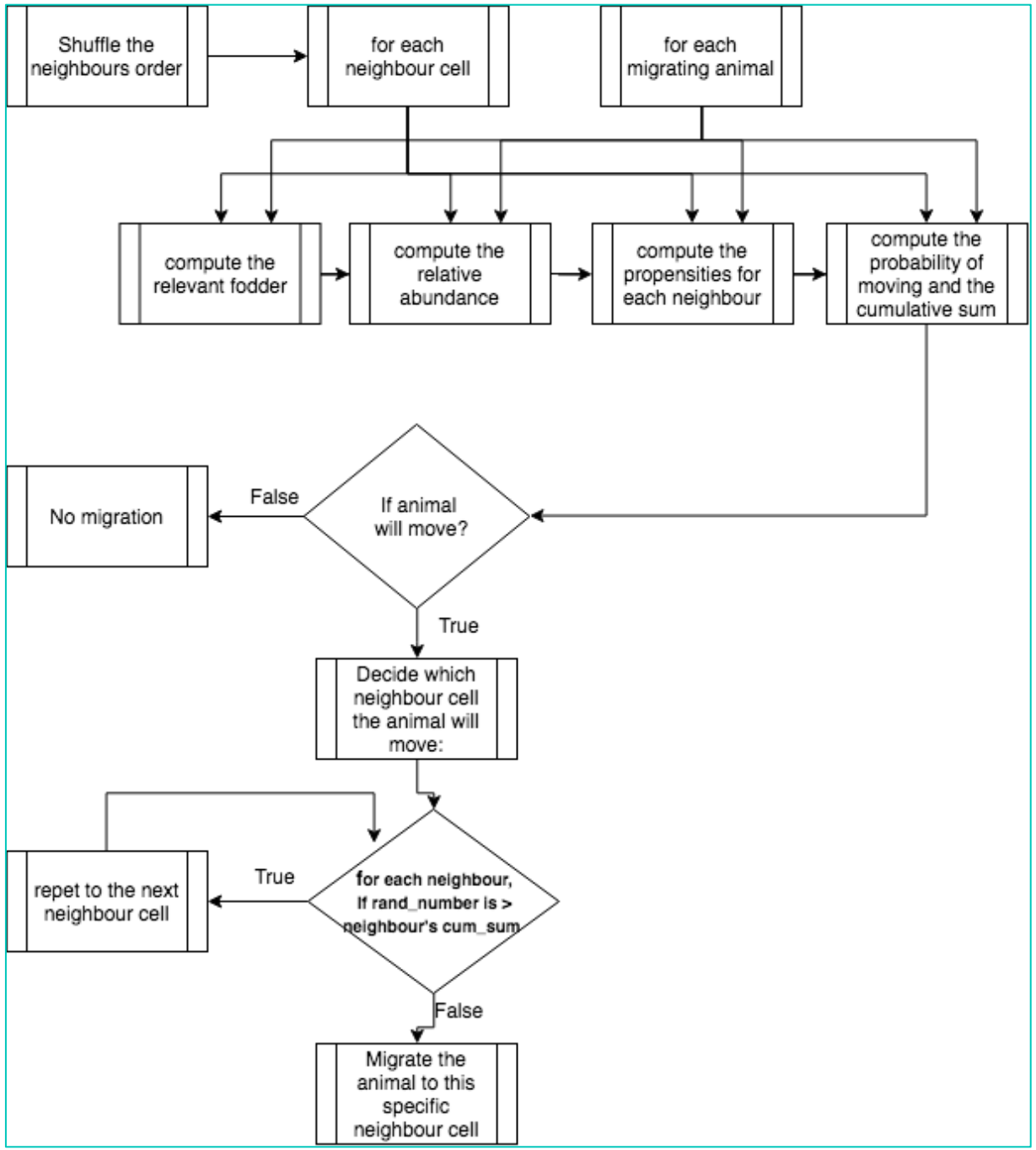
```
322         np.random.shuffle(neighbours)
```

2:

```
332                             n = 0 if n > 3 else n
```

Flow chart of migration method:

```
np.random.shuffle(neighbours)
for migrating_specie, animals in self.population.items():
    if len(neighbours) > 0 and len(animals) > 0:
        for animal in animals:
            cum_prob = self.cumsum(migrating_specie,
                                   neighbours)
            if animal.will_migrate():
                n = 0
                while np.random.random() >= cum_prob[n]:
                    n += 1
                    n = 0 if n > 3 else n
                neighbours[n].new_population[migrating_specie].\
                    append(animal)
                self.population[migrating_specie].remove(animal)
```



Minor changes on yearly_cycle method:

BEFORE

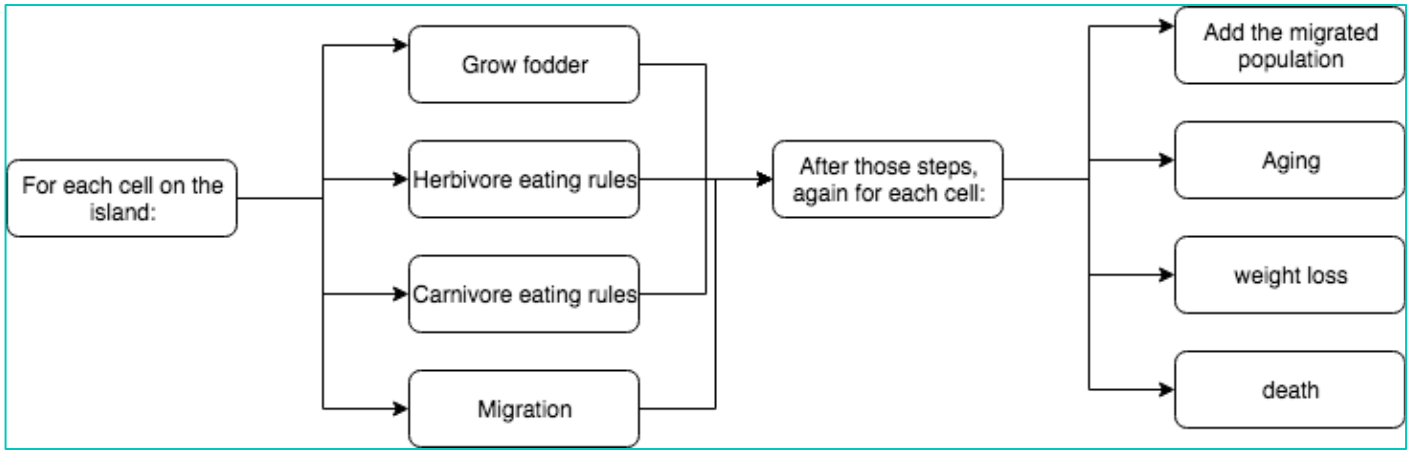
```
248     def yearly_cycle(self):
249         """This method calls, in order, the methods that compound
250         the yearly cycle dynamics of the island, such that:
251
252             1. Growing of fodder;
253             2. Animal's feeding;
254             3. Animals's birth;
255             4. Animal's migration;
256             5. Animal's aging;
257             6. Animal's weight loss;
258             7. Animal's death.
259         """
260         for coord, geo_object in self.habitable_cells.items():
261             geo_object.grow_fodder_and_feed()
262             geo_object.add_newborns()
263             geo_object.migrate(self.neighbour_cells(coord))
264             geo_object.add_new_migrated()
265             geo_object.lose_weight()
266             geo_object.get_old()
267             geo_object.die()
```

NOW

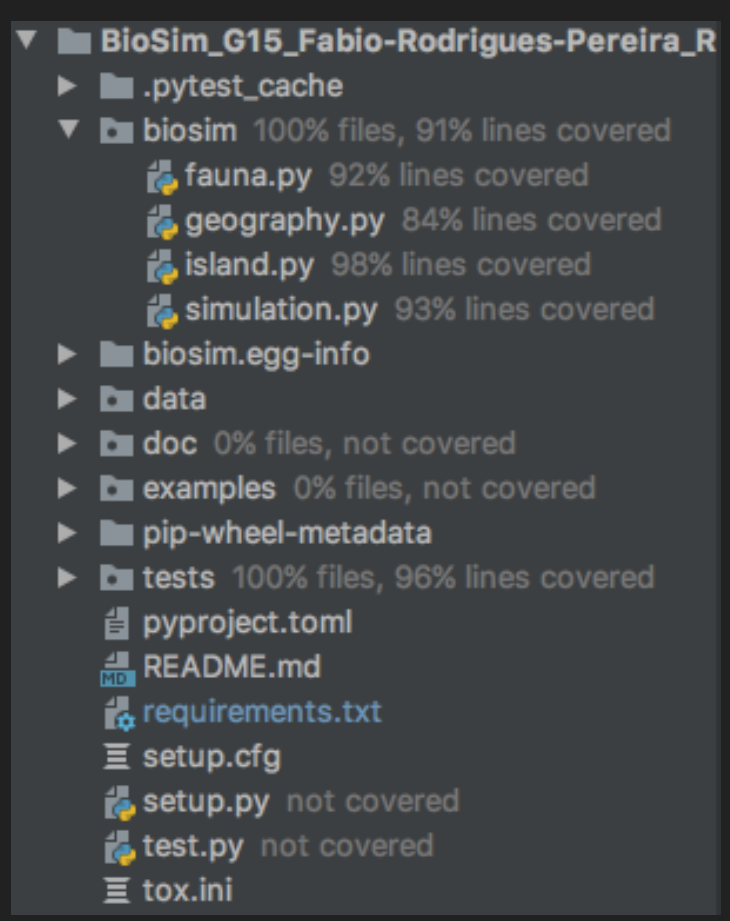
```
248     def yearly_cycle(self):
249         """This method calls, in order, the methods that compound
250         the yearly cycle dynamics of the island, such that:
251
252             1. Growing of fodder;
253             2. Animal's feeding;
254             3. Animals's birth;
255             4. Animal's migration;
256             5. Animal's aging;
257             6. Animal's weight loss;
258             7. Animal's death.
259         """
260         for coord, geo_object in self.habitable_cells.items():
261             geo_object.grow_fodder_and_feed()
262             geo_object.add_newborns()
263             geo_object.migrate(self.neighbour_cells(coord))
264
265         for coord, geo_object in self.habitable_cells.items():
266             geo_object.add_new_migrated()
267             geo_object.lose_weight()
268             geo_object.get_old()
269             geo_object.die()
```

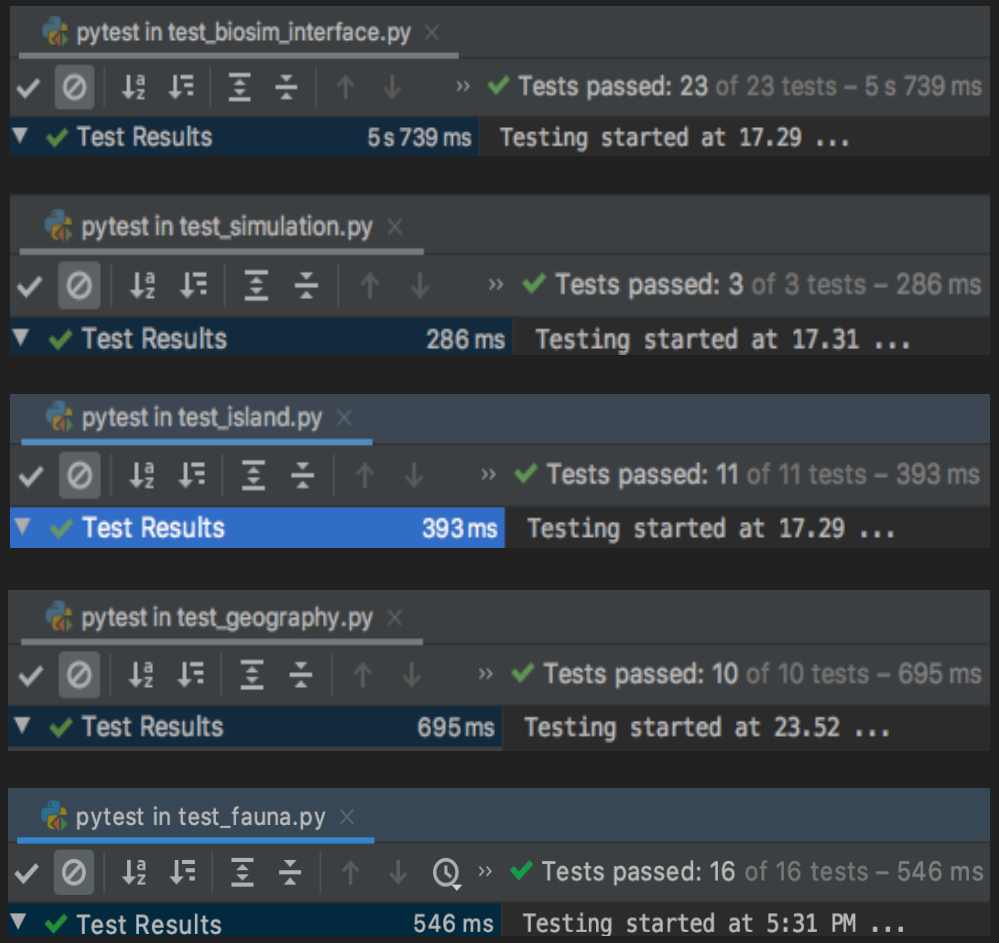
Flow chart of yearly cycle method:

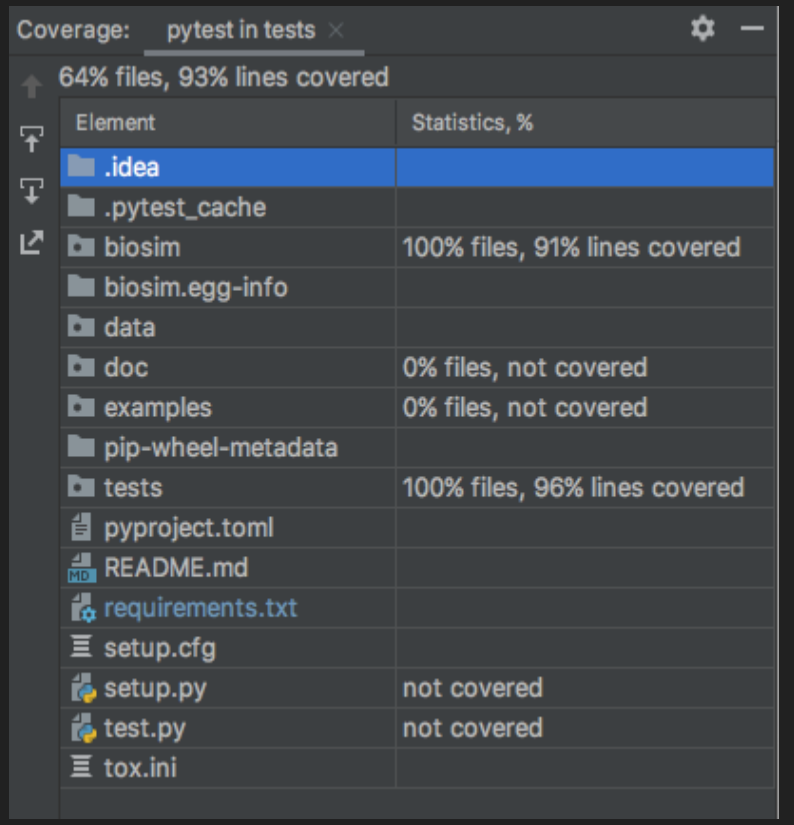
```
260         for coord, geo_object in self.habitable_cells.items():
261             geo_object.grow_fodder_and_feed()
262             geo_object.add_newborns()
263             geo_object.migrate(self.neighbour_cells(coord))
264
265         for coord, geo_object in self.habitable_cells.items():
266             geo_object.add_new_migrated()
267             geo_object.lose_weight()
268             geo_object.get_old()
269             geo_object.die()
```



Why our code is trustworthy?







Visualization features: