

# BioSim

With Kim Son Ly and Mathias Mollatt

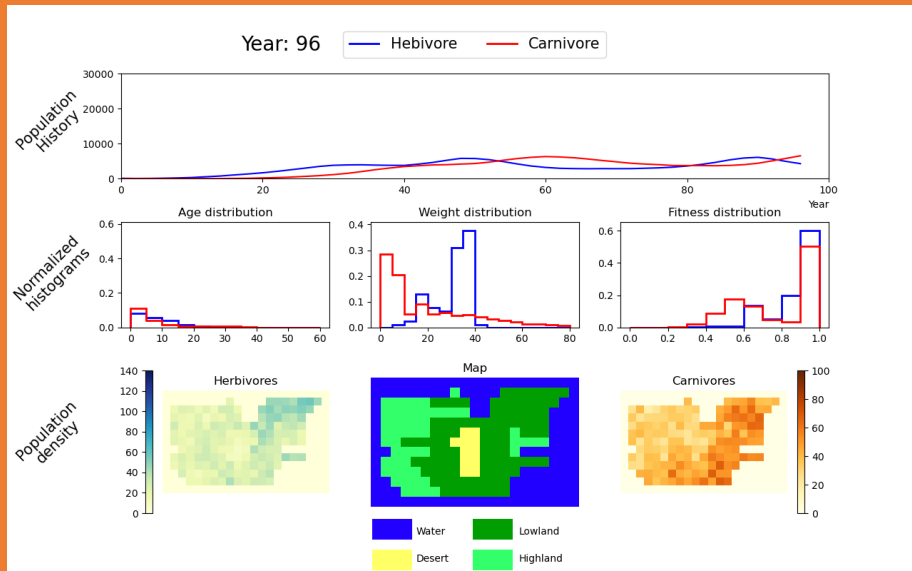
# Usage

```
13 # set up map
14 geogr = """\
15     WWW
16     WLW
17     WWW"""
18
19 # set up initial populations
20 ini_herbs = [{'loc': (2, 2),
21              'pop': [{'species': 'Herbivore',
22                      'age': 5,
23                      'weight': 20}
24                      for _ in range(50)]]]
25 ini_carns = [{'loc': (2, 2),
26              'pop': [{'species': 'Carnivore',
27                      'age': 5,
28                      'weight': 20}
29                      for _ in range(20)]]]
30
31
32 sim = BioSim(geogr, ini_herbs, img_dir='results') # Start instance of BioSim
33 sim.simulate(50) # Run simulation for 50 years with visualisation
34 sim.add_population(ini_carns) # Add carnivores
35 sim.simulate(250) # Run simulation for 250 years with visualisation
36
37 sim.make_movie() # Makes video of the visualisation
38
```

# Structure: Overview

## BioSim

### Graphics


















### Island

### Cell

### Animals

# Structure: Classes

## biosim.simulation.BioSim

```
 __init__(self, island_map=None, ini_pop=None)  
 set_animal_parameters(self, species, new_p  
 set_landscape_parameters(self, landscape, t  
 simulate(self, num_years)  
 update_graphics(self)  
 update_history_data(self)  
 update_island_data(self)  
 add_population(self, population)  
 save_log_file(self)  
 _save_simulation(self, file_name)  
 _load_simulation(file_name)  
 year(self)  
 num_animals(self)  
 num_animals_per_species(self)  
 make_movie(self, movie_fmt='mp4')
```

# Structure: Classes

## biosim.simulation.BioSim

```
__init__(self, island_map=None, ini_pop=None)
set_animal_parameters(self, species, new_p)
set_landscape_parameters(self, landscape, t)
simulate(self, num_years)
update_graphics(self)
update_history_data(self)
update_island_data(self)
add_population(self, population)
save_log_file(self)
_save_simulation(self, file_name)
_load_simulation(file_name)
year(self)
num_animals(self)
num_animals_per_species(self)
make_movie(self, movie_fmt='mp4')
```

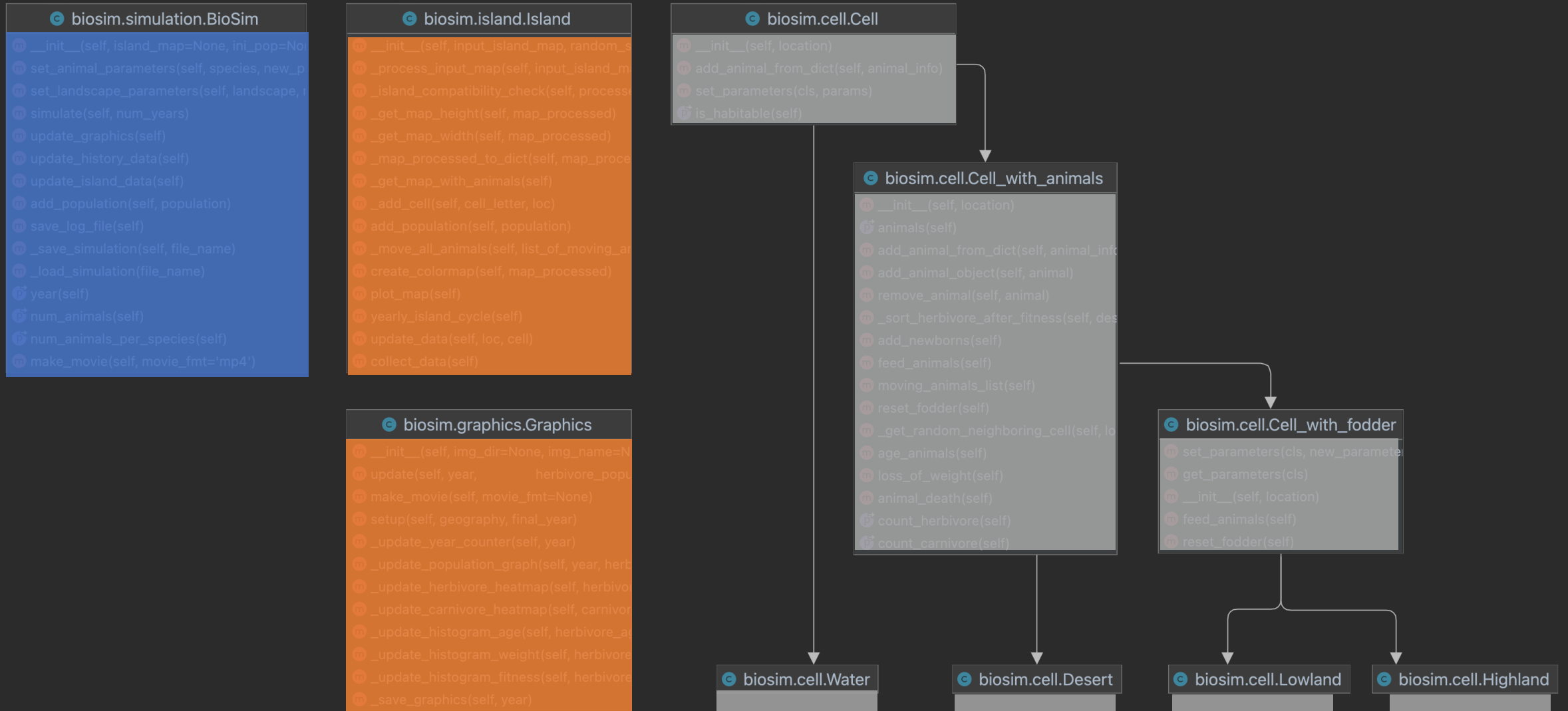
## biosim.island.Island

```
__init__(self, input_island_map, random_s)
_process_input_map(self, input_island_m)
_island_compatibility_check(self, process)
_get_map_height(self, map_processed)
_get_map_width(self, map_processed)
_map_processed_to_dict(self, map_proce)
_get_map_with_animals(self)
_add_cell(self, cell_letter, loc)
add_population(self, population)
_move_all_animals(self, list_of_moving_ar)
create_colormap(self, map_processed)
plot_map(self)
yearly_island_cycle(self)
update_data(self, loc, cell)
collect_data(self)
```

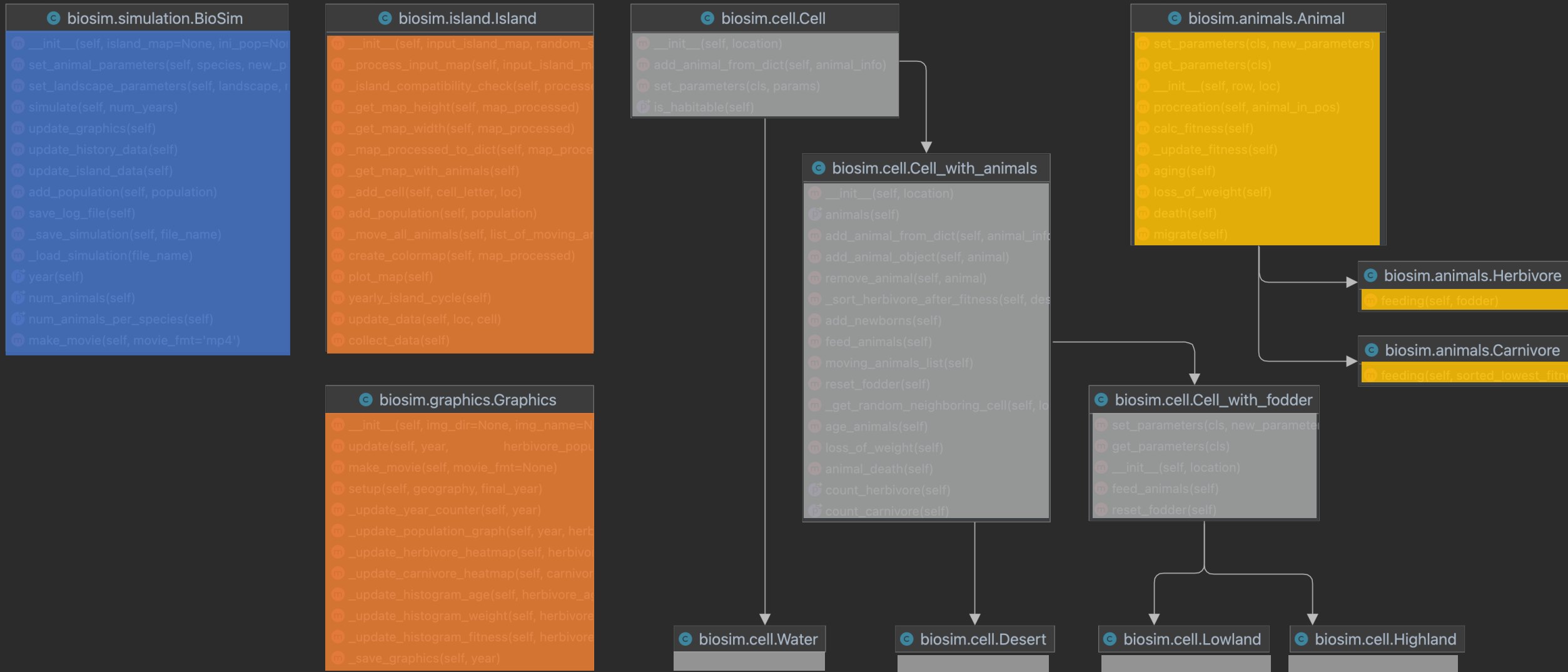
## biosim.graphics.Graphics

```
__init__(self, img_dir=None, img_name=N)
update(self, year, herbivore_popu)
make_movie(self, movie_fmt=None)
setup(self, geography, final_year)
_update_year_counter(self, year)
_update_population_graph(self, year, her)
_update_herbivore_heatmap(self, herbivo)
_update_carnivore_heatmap(self, carnivor)
_update_histogram_age(self, herbivore_ar)
_update_histogram_weight(self, herbivore)
_update_histogram_fitness(self, herbivore)
_save_graphics(self, year)
```

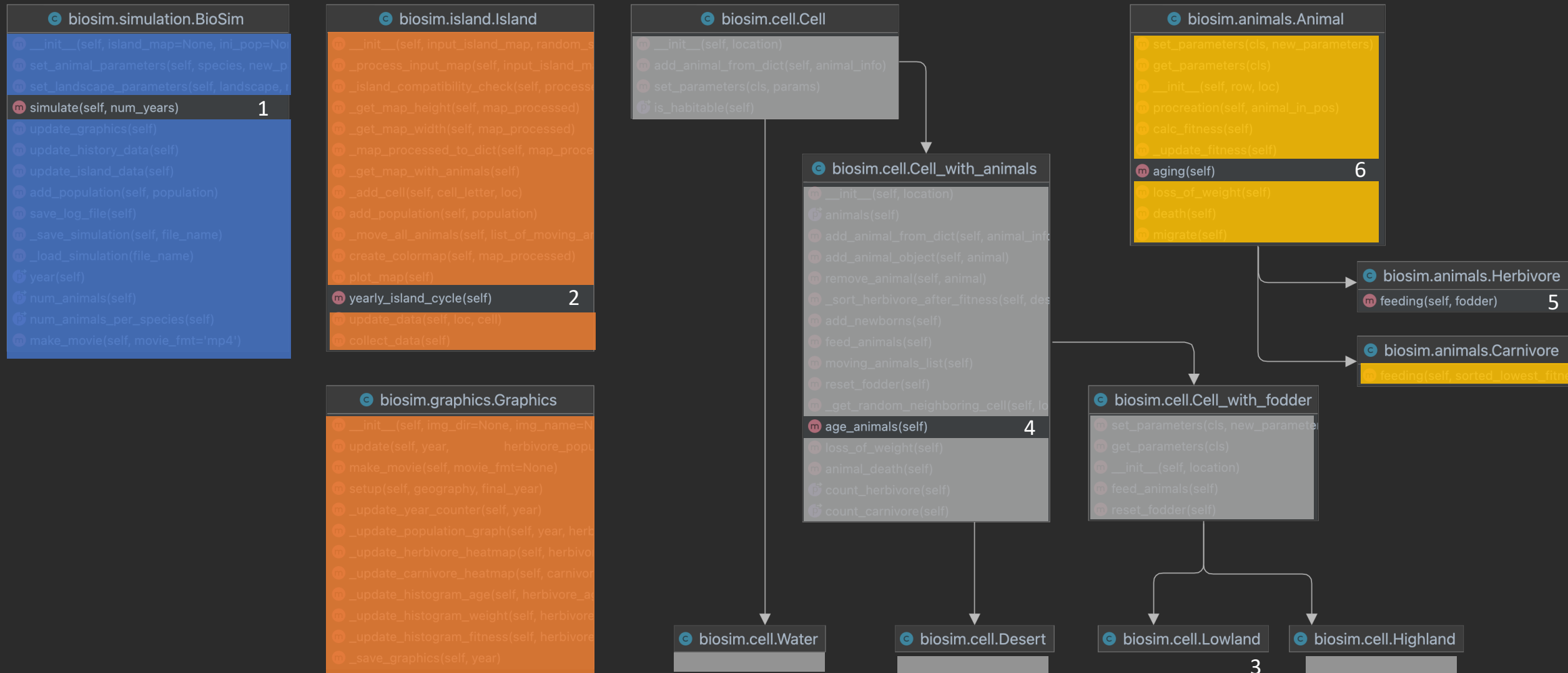
# Structure: Classes



# Structure: Classes

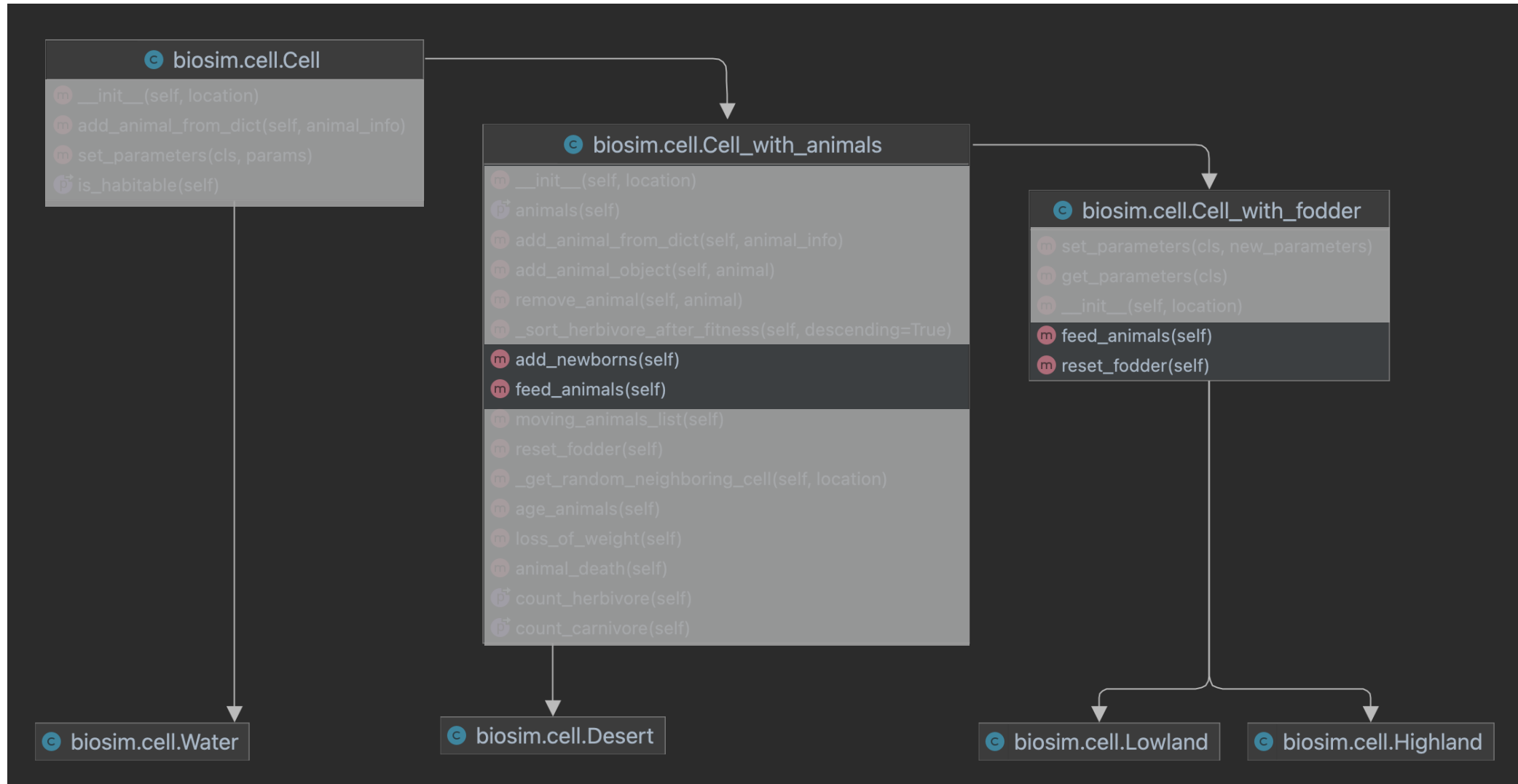


# Structure: Methods and fields





# Cell subclasses



# Cell subclasses

## Cell\_with\_animals

```
def feed_animals(self):  
    """..."""  
    # skip if there is no herbivores in the cell  
    if self.count_herbivore > 0:  
        self._sort_herbivore_after_fitness(descending=False)  
        random.shuffle(self.fauna["Carnivore"])  
        for animal in self.fauna["Carnivore"]:...
```

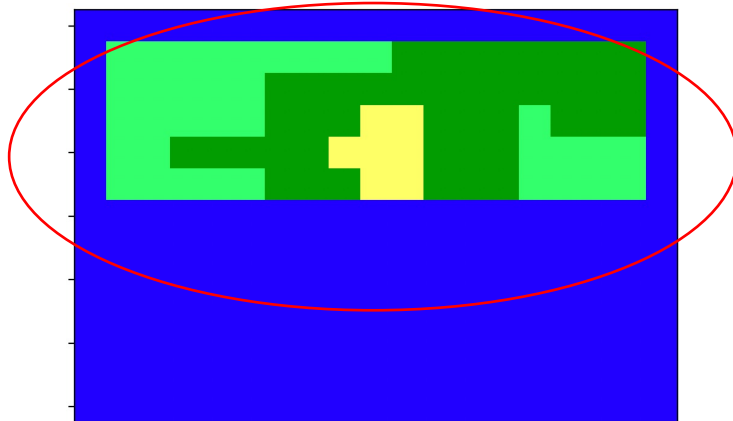
## Cell\_with\_fodder

```
def feed_animals(self):  
    """..."""  
    self._sort_herbivore_after_fitness()  
    for animal in self.fauna["Herbivore"]:...  
  
    super().feed_animals()
```

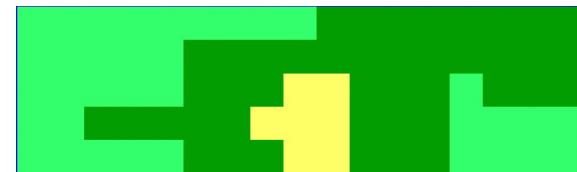
# Solutions: Optimisation

```
def _get_map_with_animals(self):  
    """  
    Creates a map(dict) with only habitable cells  
  
    Returns  
    -----  
    dict: map with animals  
    """  
    map_with_animals = {loc: cell for loc, cell in self.map.items() if cell.is_habitable}  
  
    return map_with_animals
```

Original map



Map that is used



# Solutions: Animal removal

```
def death(self):  
    r"""..."""  
    probability_of_death = self.params["omega"] * (1 - self.fitness)  
    if self.weight <= 0:  
        self.alive = False  
    elif random.random() < probability_of_death:  
        self.alive = False
```

```
def animal_death(self):  
    """..."""  
    for species, animal_list in self.fauna.items():  
        for animal in animal_list:  
            animal.death()  
        # Remove dead animals  
        self.fauna[species] = [animal for animal in animal_list if animal.alive]
```

# Error handling: Invalid border


```
geography = """\nWDW\nWLW\nWWW"""\n\nsim = BioSim(geography)
```

error1 x

File /Users/Kimley/Desktop/Repo/BioSim-022-Kim-Machias/src/bi...  
raise ValueError("Geography must be surrounded by water.")  
ValueError: Geography must be surrounded by water.

# Error handling: Invalid param

```
25
26 sim = BioSim("W", vis_years=1.5)
```

Run:  error2 x

[File](#) [/Users/Kimrcy/Desktop/Repo/biosim-022-Kim-machias/STC](#)

```
raise ValueError(f'vis_years must be a positive integer,
ValueError: vis_years must be a positive integer, not 1.5
```

Process finished with exit code 1

# Error handling: Invalid animal placement

```
ini_herbs = [{'loc': (1, 1),  
              'pop': [{'species': 'Herbivore',  
                       'age': 5.5,  
                       'weight': 20}]]]  
  
sim = BioSim("W", ini_pop=ini_herbs, vis_years=0)
```

error3 x

File /Users/Kimley/Desktop/Repo/biosim-022-Kim-Machias/src/biosim/cell.py, line  
raise ValueError(f"Cannot add animal to {type(self)} cell at loc: {self.location}")  
ValueError: Cannot add animal to <class 'biosim.cell.Water'> cell at loc: (1, 1)

# Unit tests: Probability of procreation

```
@pytest.mark.parametrize("animal", [std_herb(), std_carn()])
def test_procreation_prob(animal):
    """
    The outcome of procreation follows a binomial distribution. This test if the p-value of the binomial test is
    larger than 0.01, which is the significance level.
    """
    num_trials = 10000
    animal_in_cell = 2
    probability_of_procreation = min(1, animal.params["gamma"] * animal.fitness * animal_in_cell)

    num_babies = 0
    for i in range(num_trials):
        baby = animal.procreation(animal_in_cell)
        animal.weight = 60
        if baby is not None:
            num_babies += 1

    assert stats.binom_test(num_babies, num_trials, probability_of_procreation) > .01
```



# Tests: Migration

```
@pytest.mark.parametrize("species", ["Herbivore", "Carnivore"])
def test_migration_integration(species):
    """Test migration integration test.
    This forces the animals to move each year.
    The test is checking if the animals are following the checkerboard pattern."""
    geography = """..."""

    sim = BioSim(island_map=geography, seed=23456,
                  cmap_animals={"Herbivore": 1, 'Carnivore': 1},
                  ymax_animals=100, vis_years=0)
    sim.set_animal_parameters(species, {'mu': 100000, 'eta': 0}) # Ensures that th

    ini_pop = [{...}]

    sim.add_population(population=ini_pop)
    map = sim.island.habital_map
```



```
for year in range(1, 8):
    sim.simulate(num_years=1)
    for loc, cell in map.items():
        x = loc[0]
        y = loc[1]

        # Check if the animals are following the checkerboard pattern
        if year % 2 == 0:
            if x % 2 == 0:
                if y % 2 != 0:
                    assert cell.animals == []
            elif x % 2 != 0:
                if y % 2 == 0:
                    assert cell.animals == []
        elif year % 2 != 0:
            if x % 2 == 0:
                if y % 2 == 0:
                    assert cell.animals == []
            elif x % 2 != 0:
                if y % 2 != 0:
                    assert cell.animals == []
```

# Tests: yearly cycle

```
def test_yearly_cycle(mocker):  
    """  
    Test yearly cycle by choosing some methods in animal class and counting them manually.  
    Then comparing them to how many times the program calls the methods.  
    """  
  
    # Keep count of methods called  
    mocker.spy(Herbivore, 'calc_fitness')  
    mocker.spy(Herbivore, 'procreation')  
    mocker.spy(Herbivore, 'feeding')  
    mocker.spy(Herbivore, 'migrate')  
    mocker.spy(Herbivore, 'aging')  
    mocker.spy(Herbivore, 'loss_of_weight')  
    mocker.spy(Herbivore, 'death')  
  
    mocker.spy(Carnivore, 'calc_fitness')  
    mocker.spy(Carnivore, 'procreation')  
    mocker.spy(Carnivore, 'feeding')  
    mocker.spy(Carnivore, 'migrate')  
    mocker.spy(Carnivore, 'aging')  
    mocker.spy(Carnivore, 'loss_of_weight')  
    mocker.spy(Carnivore, 'death')  
  
    geogr = """..."""  
    sim = BioSim(island_map=geogr, vis_years=0)  
  
    # Adding 1 herbivore  
    ini_herbs = [{...}]  
    sim.add_population(ini_herbs)  
    sim.simulate(1)  
  
    # adding 1 carnivore  
    ini_carns = [{...}]  
    sim.add_population(ini_carns)  
  
    sim.simulate(1)
```

```
result = {"h_calc_fit": Herbivore.calc_fitness.call_count,  
          "h_procreation": Herbivore.procreation.call_count,  
          "h_feeding": Herbivore.feeding.call_count,  
          "h_migrate": Herbivore.migrate.call_count,  
          "h_aging": Herbivore.aging.call_count,  
          "h_loss_of_weight": Herbivore.loss_of_weight.call_count,  
          "h_death": Herbivore.death.call_count,  
          "c_calc_fit": Carnivore.calc_fitness.call_count,  
          "c_procreation": Carnivore.procreation.call_count,  
          "c_feeding": Carnivore.feeding.call_count,  
          "c_migrate": Carnivore.migrate.call_count,  
          "c_aging": Carnivore.aging.call_count,  
          "c_loss_of_weight": Carnivore.loss_of_weight.call_count,  
          "c_death": Carnivore.death.call_count}  
  
# Manually counted method calls:  
expect = {"h_calc_fit": 5,  
          "h_procreation": 2,  
          "h_feeding": 2,  
          "h_migrate": 2,  
          "h_aging": 2,  
          "h_loss_of_weight": 2,  
          "h_death": 2,  
          "c_calc_fit": 2,  
          "c_procreation": 1,  
          "c_feeding": 0,  
          "c_migrate": 1,  
          "c_aging": 1,  
          "c_loss_of_weight": 1,  
          "c_death": 1}  
  
assert result == expect
```