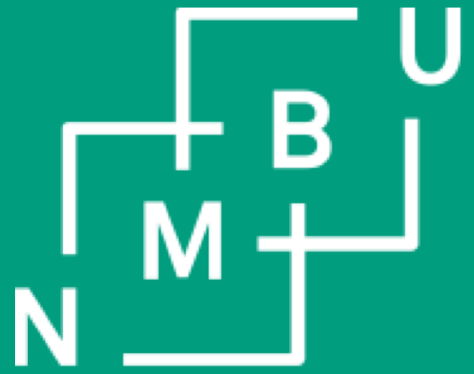
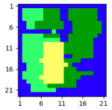


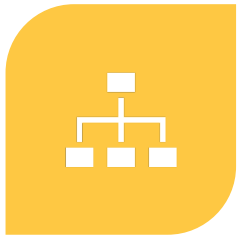
Norwegian University
of Life Sciences

Modeling of Rossumøya

Group 4
Anja Stene / Ghazal Azadi
Mon 22.06.2020



How we solved the task



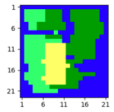
OVERALL STRUCTURE OF PROJECT WORK

OUR APPROACH TO TACKLE THE TASK

PLANNING PROGRESS

USING OUR STRENGTHS & PAIR PROGRAMMING

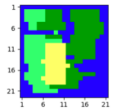
TIME USE & DEBUGGING



Examples of solutions

```python
def move_migrated_animals(self, migration_dct, x, y):
    """
    Moves the migrated animals by removing them from the current cell and adding to new cell.

    Parameters
    ----------
    migration_dct: dict
        of migrated animals and their new location cell
    x: int
        current x-coordinate
    y: int
        current y-coordinate
    """

    herbi_to_remove = []
    carni_to_remove = []

    for cell in migration_dct.keys():
        if not type(cell).__name__ == "Water":
            for animal in migration_dct[cell]:

                if type(animal).__name__ == "Herbivore":
                    cell.herbivores_list.append(animal)
                    herbi_to_remove.append(animal)
                else:
                    cell.carnivores_list.append(animal)
                    carni_to_remove.append(animal)

    self._cells[x, y].remove_migrated_animals(herbi_to_remove, carni_to_remove)
```

```python
def create_newborn(self, num_animals):
    """
    Creates a new animal based on several condition checks. If all checks passed:
    create a new animal of the same class as mother-animal, and return the newborn object.
    Parameters
    ----------
    num_animals: int
        Number of animals of same species in current cell

    Returns
    -------
    newborn: object
        if no newborn object is created, returns None

    """
    if self.check_mating_weight_conditions(num_animals):
        prob = min(1, self.params['gamma'] * self.calculate_fitness() * (num_animals - 1))
        creating = self.from_prob_to_binary(prob)
        if creating:
            newborn = self.__class__()
            if self.check_mother_minus_newborn_weight_conditions(newborn.weight):
                return newborn
    return None
```



Productive code

How to accomplish a productive code?

Ease of use

Ease of manipulation

Speed

computational efficiency

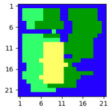
What can be improved

Task description requests

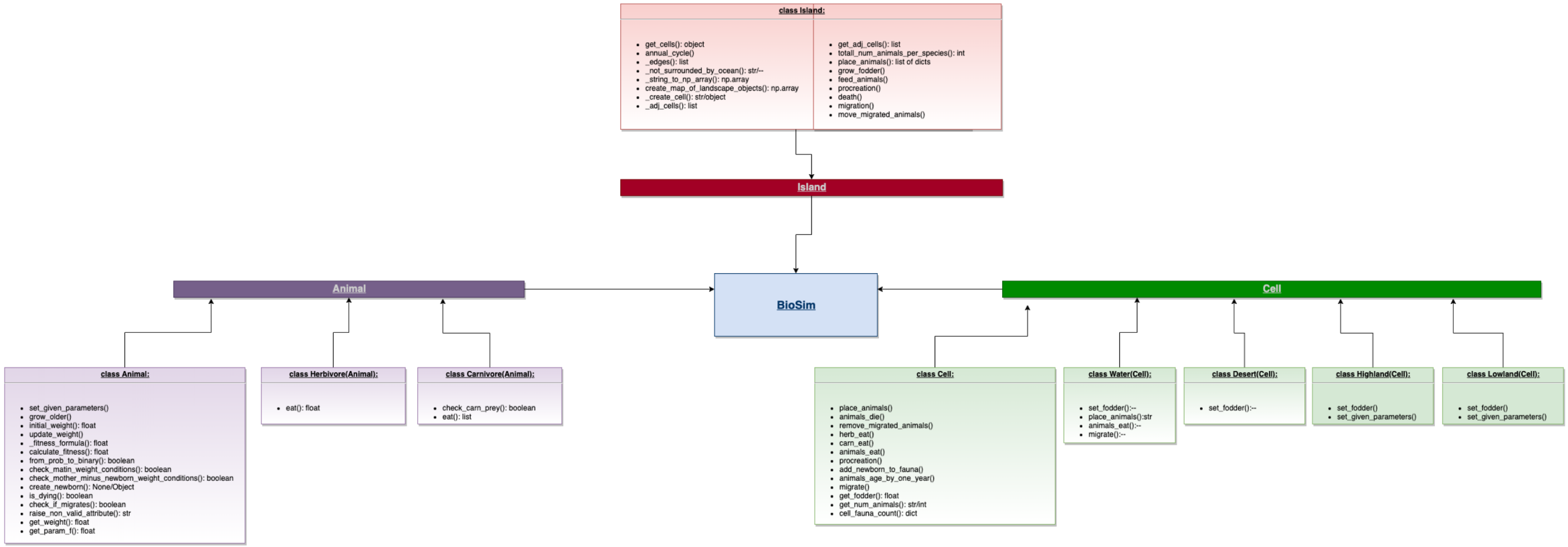
Test coverage

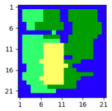
Readability

Performance optimization



Project Implementation at a Glance





Quality Insurance

Tests

- Normal tests

```
def test_calculate_fitness(self):
    """
    Firstly: To test if the weight is 0 the fitness would be also zero
    Secondly: To see if the fitness is a number between 0 and 1
    Thirdly: To see if the fitness increases by increment in weight
    Lastly: If the fitness decreases by aging
    """
    animal_zero_weight = [Herbivore(weight=0), Carnivore(weight=0)]
    for species in range(2):
        assert animal_zero_weight[species].calculate_fitness() == 0
    for species in range(2):
        assert 0 < self.animal[species].calculate_fitness() < 1
        fitness_before_weighting = self.animal[species].calculate_fitness()
        self.animal[1].eat(self.herb_list)  # Feeding the carnivore
        self.animal[0].eat()  # Feeding the herbivore
        fitness_after_weighting_but_before_aging = self.animal[species].calculate_fitness(
        if self.animal[1].check_carn_prey(self.animal[0].calculate_fitness(),
                                          self.animal[1].calculate_fitness()):
            assert fitness_after_weighting_but_before_aging - fitness_before_weighting > 0
        self.animal[species].grow_older()  # aging the animals
        fitness_after_aging = self.animal[species].calculate_fitness()
        assert fitness_after_aging - fitness_after_weighting_but_before_aging < 0
```

- Mocker tests

```
def test_prob_to_die(self, mocker):
    """
    To test if the random number is low enough the animal would definitely die
    """
    for species in range(2):
        mocker.patch('numpy.random.random', return_value=0)
        assert self.animal[species].is_dying() is True
```

- Statistical tests

```
def test_death(self, mocker):
    """
    Firstly: To test if the animal dies when its weight is less / equal to zero
    Secondly: Since there's a reverse relationship between death and fitness, animals with highe
    fitness should have lower probability to die. Existence of this relationship is being tested
    here
    """
    animal_zero_weight = [Herbivore(weight=0), Carnivore(weight=0)]
    die_low_weight = []
    die_high_weight = []
    sum_die_low_weight = []
    sum_die_high_weight = []
    self.animal[1].eat(self.herb_list)  # Feeding the carnivore for one time
    self.animal[0].eat()  # Feeding the herbivore for one time
    for species in range(2):
        assert animal_zero_weight[species].is_dying() is True

        for _ in range(500):
            for _ in range(100):
                die_low_weight.append(self.animal[species].is_dying())
            sum_die_low_weight.append(sum(die_low_weight))

        for _ in range(50):  # Feeding animals for 50 times leading to fitness increment
            self.animal[1].eat(self.herb_list)
            self.animal[0].eat()

        for _ in range(500):
            for _ in range(100):
                die_high_weight.append(self.animal[species].is_dying())
            sum_die_high_weight.append(sum(die_high_weight))

    contingency_table = np.array([sum_die_high_weight, sum_die_low_weight])
    chi2_stat, p_val, dof, ex = stats.chi2_contingency(contingency_table)
    assert p_val < ALPHA
```

```
def test_initial_weight_gaussian_dist(self):
    """
    To test if the initial weight of the animals come from a gaussian distribution
    """
    for species in range(2):
        for _ in range(2000):
            self.init_weight.append(self.animal[species].weight)
            ks_statistic, p_value = kstest(self.init_weight, 'norm')
            assert p_value < ALPHA
```

Documentation

- Doc strings

```
def check_carn_prey(self, herb_fitness, carn_fitness):
    """
    Check assessing if a carnivore can prey on a herbivore.
    Condition is based on both animals fitness.
    Probability of carnivore hunting the herbivore is:
    p = (carnivore_fitness - herbivore_fitness ) DeltaPhiMax,
    where deltaPhiMax is given in the Params dictionary

    Parameters
    ----------
    herb_fitness: float
    carn_fitness: float

    Returns
    -------
    binary: boolean
        Returns True if carnivore can prey on herbivore, False if not.
    """
```

- Sphinx

check_carn_prey(*herb_fitness*, *carn_fitness*) [source]

Check assessing if a carnivore can prey on a herbivore. Condition is based on both animals fitness. Probability of carnivore hunting the herbivore is: p = (carnivore_fitness - herbivore_fitness) DeltaPhiMax, where deltaPhiMax is given in the Params dictionary

Parameters:
- **herb_fitness** (*float*) –
- **carn_fitness** (*float*) –

Returns: **binary** – Returns True if carnivore can prey on herbivore, False if not.

Return type: boolean

eat(*herb_list*) [source]

Eat method for Carnivores. Loops through a list of all available herbivores, checks if the carnivore preys on it. If the carnivore eats the herbivore, its appended to the eaten_herbs list, to be returned. The weight and fitness is updates for every animal the carnivore eats, and the carnivore never eats more than the appetite parameter F, set in the Params dictionary.

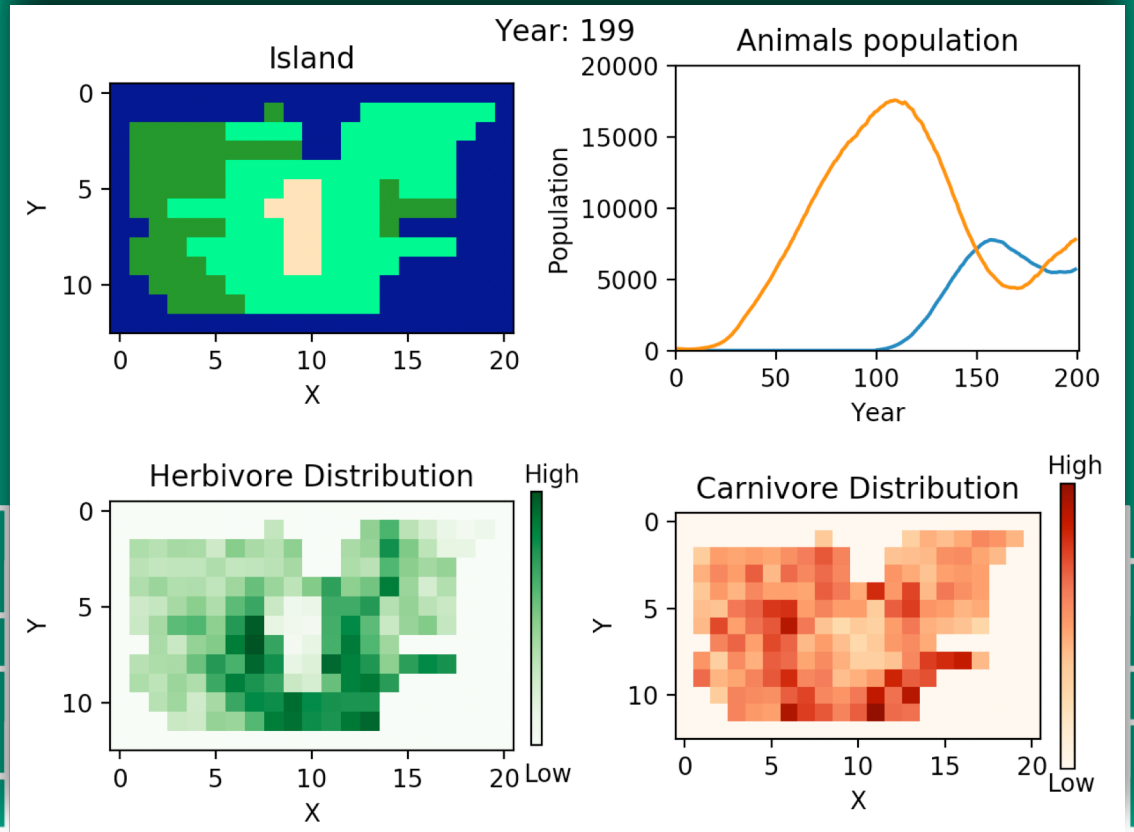
Parameters: **herb_list** (*list*) – list of all available herbivores in cell

Returns: **eaten_herbs** – List of all the eaten herbivores. List is returned empty if no Herbivore were eaten by the Carnivore

Return type: List

params = *{'DeltaPhiMax': 10.0, 'F': 50.0, 'a_half': 40.0, 'beta': 0.75, 'eta': 0.125, 'gamma': 0.8, 'mu': 0.4, 'omega': 0.8, 'phi_age': 0.3, 'phi_weight': 0.4, 'sigma_birth': 1.0, 'w_birth': 6.0, 'w_half': 4.0, 'xi': 1.1, 'zeta': 3.5}*

Simulation of 200 years



Thank you