BIOSIM ISLAND SIMULATION

Vegard Helland, Lydia Lindgren

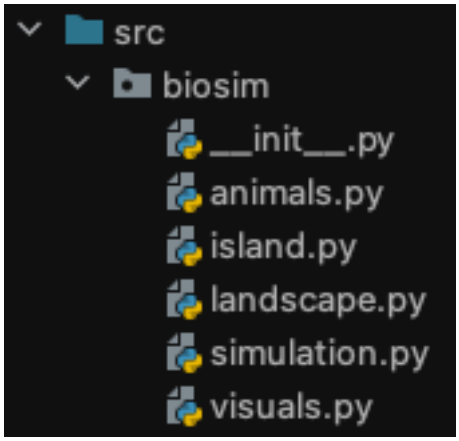
21. januar 2022

INTRO

The biosim package provides a simulation of the life-cycle of herbivores and carnivores on an imaginary island called Rossumøya.

THE STRUCTURE OF THE CODE

The code is structured in a way that is supposed to make it easy to navigate among the files.



CLASSES:

simulation.py

- **BioSim** — Class simulating a BioSim project.

animals.py

- **Animals** — Class containing the common behaviors of Herbivores and Carnivores.
- **Herbivore** — Subclass keeping track of the attributes of the herbivore class. Also contains methods specific to herbivore behaviors.
- **Carnivores** — Subclass keeping track of the attributes of the carnivore class. Also contains methods specific to carnivore behaviors.

island.py

- **Tile** — Class keeping track of all the herbivore and carnivore objects in each tile/cell on the island.
- **Island** — Subclass representing the island in the simulation, and goes through the yearly cycle for each year.

landscape.py

- **Lowland** — Class keeping track of the attributes of the lowland landscape type on the island.
- **Highland** — Class keeping track of the attributes of the highland landscape type on the island.
- **Water** — Class keeping track of the attributes of the water landscape type on the island.
- **Desert** — Class keeping track of the attributes of the desert landscape type on the island.

visuals.py

- **Visual** — Class providing the visualization of the BioSim project.

HOW TO RUN

- Used through the BioSim-class.

```
    def __init__(self, island_map, ini_pop, seed,
                 vis_years=1, ymax_animals=None, cmax_animals=None, hist_specs=None,
                 img_dir=None, img_base=None, img_fmt='png', img_years=None,
                 log_file=None):
```

- `island_map`, `ini_pop` and `seed` are mandatory inputs.
- The rest are optional inputs.
- This sets the parameters for the simulation.

```
    def simulate(self, num_years):
```

- `num_years` is the number of years of simulation.

PROBLEMS ENCOUNTERED AND HOW THEY WERE SOLVED

UPDATING FITNESS

```
    def update_fitness(self):
        self.fitness = 0 if self.weight <= 0 \
            else (1 / (1 + math.exp(self.phi_age * (self.age - self.a_half)))) * \
                (1 / (1 + math.exp(-self.phi_weight * (self.weight - self.w_half))))
```

```
    def update_age(self):
        self.age += 1
        self.update_fitness()
```

A MORE EFFECTIVE SOLUTION

```
    def update_fitness(self):
        if self.fitness_update:
            self.fitness = 0 if self.weight <= 0 \
                else (1 / (1 + math.exp(self.phi_age * (self.age - self.a_half)))) * \
                     (1 / (1 + math.exp(-self.phi_weight * (self.weight - self.w_half))))
            self.fitness_update = False
```

```
    def update_age(self):
        self.age += 1
        self.fitness_update = True
```

```
    def animals_age(self):
        if self.herbs:
            for herb in self.herbs:
                herb.update_age()
        if self.carns:
            for carn in self.carns:
                carn.update_age()
```

MIGRATION WITH FLAGS

```
    def __init__(self,weight,Age):
        self.count_animals()
        self.dead = False
        self.weight = weight
        self.age = age
        self.moved = False
```

```
    def Migrate(self)
        for herb in tile.herbs:
            if not self.moved:
                if herb.migrate():
                    new_loc = self.new_location(loc)
                if get_map[new_loc[0] - 1][new_loc[1] - 1].traversable:
                    herb.moved = True
                    get_map[new_loc[0] - 1][new_loc[1] - 1] = herb
                    del herb
            self.remove_herb()
```

```
    for tile in self.tiles:
        for herb in tile.herbs:
            herb.moved = False
```

MIGRATION WITH LISTS

```
def animals_migrate(self, loc, get_map):
    for herb in self.herbs:
        herb.update_fitness()
        if herb.migrate():
            new_loc = self.new_location(loc)
            if get_map[new_loc[0] - 1][new_loc[1] - 1].traversable:
                self.mig_h.append(herb)
                get_map[new_loc[0] - 1][new_loc[1] - 1].migrants_herbs(herb)
        self.remove_herb()
```

```
    def integrate(self):
        if self.new_h:
            self.herbs += self.new_h
            self.new_h = []
        if self.new_c:
            self.carns += self.new_c
            self.new_c = []
```

ENSURING QUALITY

USING TESTS

- Prioritized testing on the individual level

```python
def test_age(self):
    age0 = self.ani.age
    self.ani.update_age()
    assert self.ani.age == age0 + 1
```

- Tile-level

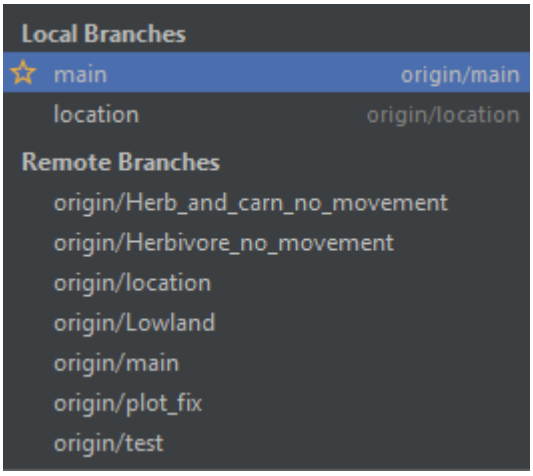
```python
def animals_age(self):
    if self.herbs:
        for herb in self.herbs:
        herb.update_age()

    if self.carns:
        for carn in self.carns:
            carn.update_age()
```

- Island-level

```python
def aging(self):
    for loc in self.tiles:
        loc.animals_age()
```

WORKING IN SEVERAL DIFFERENT BRANCHES



- Began creating herbivore objects with no migration.
- Continued with carnivore objects in a different branch.
- Worked on implementing the lowland landscape type.
- Started including locations and migration while working on the visualizations.
- Merged everything with the main branch once the code was finalized.

SOME INTERESTING RESULTS

- checkerboard.mp4
- Shows the migration-pattern if all animals moves once every year

FURTHER DEVELOPMENT

- Visual/graphics vs island/animal
- Docstrings and visuals