Simulating Tundra environment.
OOP laboratory no. 2

Pijus Petkevičius

May 23, 2023

Contents

1. Task

Simulate the process until each of the prey colonies becomes extinct or the number of prey animals quadruples compared to its starting value. Print the data of each colony in each turn.

We are simulating the animals of the tundra. There are colonies of prey and predator animals. The number of animals in a colony affect the number of animals in other colonies. There are three predator species: the snowy owl, the arctic fox and the wolf. There are three kinds of prey: the lemming, the arctic hare and the gopher.

If the number of prey animals increase, predators can reproduce more quickly. If the number of prey is very large, most of them will wander away because they cannot find enough food. If the number of predators is large, the number of the prey decreases quicker as they are preyed upon.

Each colony has a name, a species, and the number of animals in the colony. The prey species are affected by the different predator species as follows. The number of animals in their own colony changes first, then they influence the predators.

Lemming: If they are preyed upon by a predator colony, the number of animals in their colony decreases by four times the number of animals in the predator colony. The number of animals in their colony doubles every second turn. If there are more than 200 animals in the colony, the number of animals in the colony decreases to 30.

Hare: If they are preyed upon by a predator colony, the number of animals in their colony decreases by double the number of animals in the predator colony. The number of animals in their colony grows by 50 percent (to one and a half times their previous number) every second turn. If there are more than 100 animals in the colony, the number of animals in the colony decreases to 20.

Gopher: If they are preyed upon by a predator colony, the number of animals in their colony decreases by double the number of animals in the predator colony. The number of animals in their colony doubles every fourth turn. If there are more than 200 animals in the colony, the number of animals in the colony decreases to 40.

Predators choose and attack a prey colony randomly in each turn. If there are not enough animals in the attacked colony (for example, there are not four times the number of predators in a lemming colony), the number of predators also decreases: every fourth predator out of the ones who didn't get prey perishes. Predators have offsprings every eighth turn. Normally, the snow owls have 1 offspring per 4 animals, the foxes have 3 offsprings per 4 animals, and the wolves have 2 offsprings per 4 animals.

The program should read the colonies from a text file. The first line contains the number of prey and predator colonies separated by a space. Each of the next lines contains the data of one colony separated by space: their name, their species, their starting number of animals. The species can be: o - owl, f - fox, w - wolf, l - lemming, h - hare, g - gopher.

2. Plan

To describe Colonies, We have 2 main classes: **Colony** and **Species**. **Species** abstract class has abstract method *Reproduce* and abstract property *MultiplicationFactor*. It has 2 children abstract classes: **Preys** and **Predators**. Both of them implement the *Reproduce* and *MultiplicationFactor* abstract method and properties.

The **Predator** class has predator specific properties: *_children*, *_perAnimal*. In **Predator** class we implement *Attack* method where the predator colony attacks the prey colony.

The **Prey** class has its specific properties *_maxColony*, *_resetColony*, *_multiplyFactor*, *_predatorFactor*.

- _maxColony - maximum number of colony animals, before it goes to _resetColony number
- _resetColony - number of animals after it reached _maxColony animals
- _multiplyFactor - how much the population of the Colony increases every _timeInterval years.
- _predatorFactory - how many animals of the Colony will be eater per 1 predator.

In *Prey* class we implement *Attacked* method where the number preys is decreased by the number of the predators.

The special prey types: Gopher, Hare and Lemming, are the children classes of the **Prey** which introduce special property values(seen in UML class diagram), passed using *base()* class. The same can be seen for the **Predator** children classe: Wolf, Owl, Fox.

All the classes of the **Species** children classes : Gopher, Owl, etc. are realized based on the Singleton design pattern, as it is enough to create one **Species** object for each class.

The abstract classes **Species**, **Prey**, **Predator** are using template design pattern, since the **Species** abstract class provides Reproduce(), IsPrey, MultiplicationFactor which are not implemented yet/virtual.

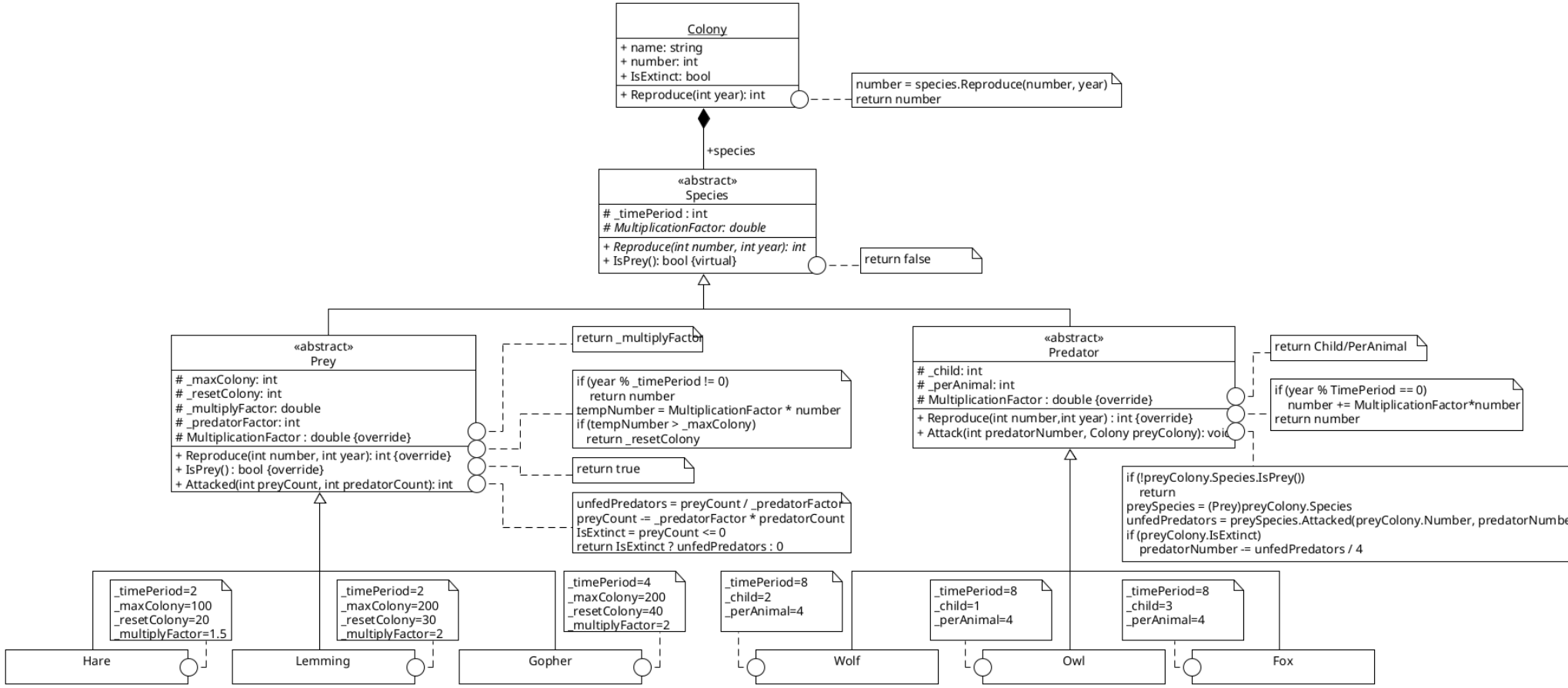


Figure 1: Uml class diagram

In the preys specification we give the initial Preys list, and variable count. We use traditional summation pattern to do that:

Specification:

$$A = (preys : Colony^*, count : N)$$

$$Pre = (preys = preys')$$

$$Post = (Pre \wedge count = \sum_{i=1}^{|preys|} preys[i].Number)$$

In the analogy we specify parts which were changed compared to the original algorithm.

Enor(E)	i=1..n
f(e)	preys[i].Number
s	creatures
H,+,0	$Preys^*, \sum, 0$

Structugram:

count:=0	
i=1..|preys|	
	count = count + preys[i].Number

For one iteration of the wildlife, we reproduce both of the preys and predator colonies, and then we iteratively go through all of the predator colonies, which attack random prey colony (denoted by index rand()).

Specification:

$$A = (preys : Colony^*, predators : Colony^*, year : int)$$

$$Pre = (preys = preys' \wedge predators = predators' \wedge year > 0)$$

$$Post = (\forall i \in [1..|preys|] : preys[i].Reproduce(year) \wedge \forall i \in [1..|predators|] : predators[i].Reproduce(year)$$

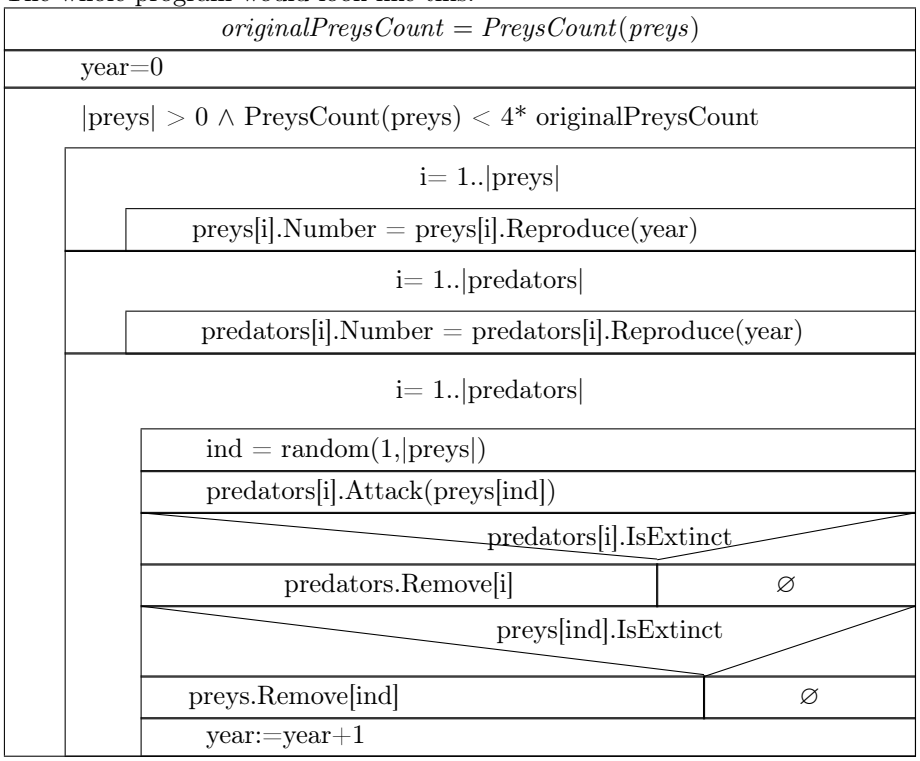
$$\wedge \forall i \in [1..|predators|] : preys[i], predators[i] = predators[i].Attack(prey[random()]))$$

Analogy:

Enor(E)	i=1..n
f(e)	predators[i].Attack(preys[j])
s	predators
H,+,0	$Colony^*, \ominus, predators[i]$

Enor(E)	i=1..n
f(e)	predators[i].Attack(preys[j])
s	preys
H,+,0	$Colony^*, \ominus, preys[j]$

The whole program would look like this:



3. Testing

- Test if prey and predator children classes return same object(singleton pattern)
- Check if prey and predator children classes return proper IsPrey method value(true- for preys, false- for predators)
- Check predator reproduction (offspring year, non offspring year)
- Check prey reproduction (offspring year, non offspring year), check what happens when prey population reaches maximum allowed.
- Predator attacks prey colony where no animal is alive
- Predator attacks predator colony
- Predator attacks alive prey colony, prey colony decreases.