Simulace evoluce

Prokop Divín

July 26, 2023

Contents

1 User Part

1.1 About program

The program reads the input folder, where the environment and the animals are defined. Then it simulates the environment and the behaviour of the animals in it. When the simulation finishes, the program generates a text file, where the simulation progress is recorded (printed ASCI grid showing the simulation status after each day) and a csv file where the average values of the animals are recorded.

1.2 Rules

The animal must eat food or other animals during the day, if at the end of the day its hunger is greater than the eaten food, animal dies. If it has eaten enough food, several times more than the hunger, animal will reproduce. If it reproduces, it creates a new animal, with slightly changed characteristics (height, sense, dexterity) by the "mutation" parameter. The animal's characteristics affect its hunger, how much depends on the chosen hunger equation.

1.3 Control

Run the program from the command line and give it two parameters. The first parameter is the path to the input data file. The second is the name of the output data file. Two files are created from the input data file name. A file with the extension ".csv" where the statistics from each day of the simulation are recorded and a file with the extension "_log.txt" where the simulation status after each day is printed. There is a makefile for running on Linux and a .sln folder for running on window.

Example startup for linux:

```
\user> make
\user>./evolution tests/big.txt tests/big
```

1.4 Parameter file

The parameter file is a text file that specifies the definition of the environment. If we want to use add a comment, then anything after " #" until the end of the line will not be perceived by the application.

The file can contain definitions of three things, namely the header, the food, and the animal. More food definitions means more types of food, and similarly for animals. If there are multiple header definitions then the last one applies.

1.4.1 Header definition

This is an example of writing a header definition in a file. Where each parameter has a comment explaining the meaning.

```
*head # indicates the beginning of the header definition
days 50 # number of days of simulation, same as number of generations
```

```
map_hight 20 # map height
map_width 25 # map width
nutritions_divider 2 #- indicates how much of the animal’s food is left after
                  # subtracting it’s hunger (the nutrients left to
                  # the animal are divided by this number)

calculator 2 3 4 5 # 2- number of the function that will calculate the hunger
# 3- the number that determines the units of the "size" parameter of the animal
# 4- the number that determines the units of the parameter "sense" for the animal
# 5- the number that determines the units of the parameter "dexterity" of the animal
--------- # terminates the header definition
```

Calculator x

We can choose from four functions, where size, dexterity, sense are the properties of the animal for which the function calculates hunger.

-function 0 is inspired by the formula for kinetic energy.

$$hlad = (\frac{size}{size_r})^3 * (\frac{dexterity}{dexterity_r})^2 + \frac{sense}{sense_r}$$

-function 1 does not favour any parameter.

$$hlad = \frac{size}{size_r} + \frac{dexterity}{dexterity_r} + \frac{size}{size_r}$$

-function 2 increases the price of Sense, otherwise it is like function 0.

$$hlad = (\frac{size}{size_r})^3 * (\frac{dexterity}{dexterity_r})^2 + (\frac{sense}{sense_r})^2$$

-function 3 corresponds to the fact that the weight of an animal does not depend on 3 times its height but on 2.

$$hlad = (\frac{size}{size_r})^2 * (\frac{dexterity}{dexterity_r})^2 + \frac{sense}{sense_r}$$

Let’s assume we know:

```
calculator 1 2 3 4
```

and we are calculating hunger of animal with properties: $size = 4, sense = 6, dexterity = 8$ than this is how we calculate hunger:

$$hlad = \frac{4}{2} + \frac{6}{3} + \frac{8}{4} = 6$$

1.4.2 Food definition

This is an example of writing a food definition in a file. Where each parameter has a comment explaining the meaning.

```
*plant #must be first, indicates the beginning of the definition
```

```
name p1 # the name under which we see the animal in the simulation, it must have two chara

size 1.01 # if the animal wants to eat the plant, it must have a larger height.

count 25 # initial number of plants
nourishment 3 # is how much nutrients one piece of this plant will provide
changer 2 0.85 25 10 # 2...every two days the maximum number of plant will change,
# 0.85 ...times will be reduced
# 25 is the maximum number it can reach
# 10 ...is the minimum number it can reach
# if it crosses one of the two thresholds, the maximum number of food
                                          # does not change anymore.

----- # must end with a line with at least one "-"
```

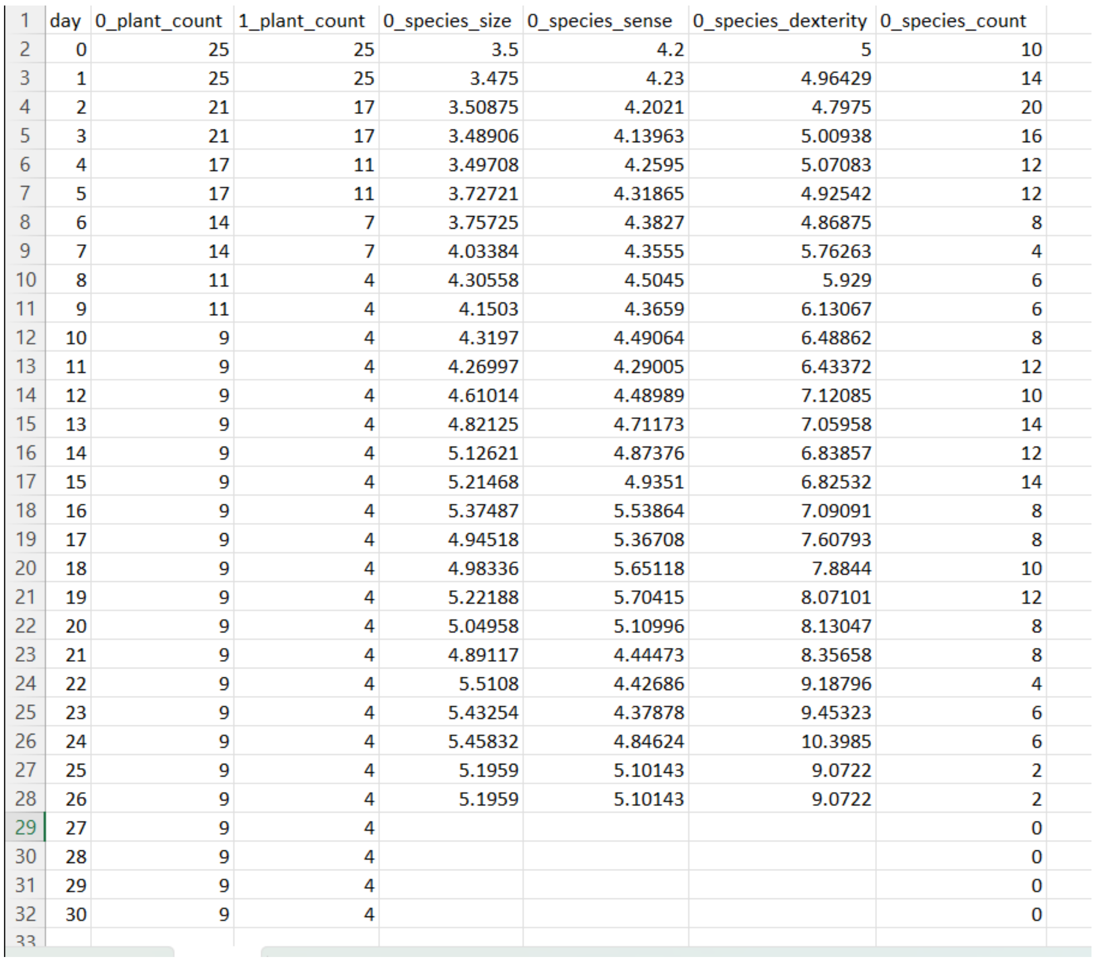
1.4.3 Animal definition

```
*species # must be first, indicates the beginning of the definition

name a1 # the name under which we will see the animal in the simulation, it must have two

size 3.5 # height of the animal
sense 4.2 # how far it can see. It is measured in the same way as dexterity
dexterity 5 # how many fields it moves in a day
reproduction 1.2 # how many times more nutrients it needs to reproduce (than its hunger)
mutation 0.1 # and is a random element, a number from 0-1.
# determines how much will new animal change

food 1.2 2 # 1.2 how many times the nutrients from the animal multiply, when eaten

count 10 # number of individuals of this species at the beginning of the simulation

stop_eat 1.5 # the maximum amount of food an animal can eat is calculated as follows:
# hunger*reproduction*stop_eat

------------ # must end with a line with at least one "-"
```

1.5 File with statistics

There is a csv file with a record for each day. One record is one line. For each plant, we find here the number of individuals at the beginning of the day. For each animal species, we find here, the number of individuals at the end of the day and the average value of the parameters "Size", "Sense", "Dexterity".

here is an example:



day	0_plant_count	1_plant_count	0_species_size	0_species_sense	0_species_dexterity	0_species_count
0	25	25	3.5	4.2	5	10
1	25	25	3.475	4.23	4.96429	14
2	21	17	3.50875	4.2021	4.7975	20
3	21	17	3.48906	4.13963	5.00938	16
4	17	11	3.49708	4.2595	5.07083	12
5	17	11	3.72721	4.31865	4.92542	12
6	14	7	3.75725	4.3827	4.86875	8
7	14	7	4.03384	4.3555	5.76263	4
8	11	4	4.30558	4.5045	5.929	6
9	11	4	4.1503	4.3659	6.13067	6
10	9	4	4.3197	4.49064	6.48862	8
11	9	4	4.26997	4.29005	6.43372	12
12	9	4	4.61014	4.48989	7.12085	10
13	9	4	4.82125	4.71173	7.05958	14
14	9	4	5.12621	4.87376	6.83857	12
15	9	4	5.21468	4.9351	6.82532	14
16	9	4	5.37487	5.53864	7.09091	8
17	9	4	4.94518	5.36708	7.60793	8
18	9	4	4.98336	5.65118	7.8844	10
19	9	4	5.22188	5.70415	8.07101	12
20	9	4	5.04958	5.10996	8.13047	8
21	9	4	4.89117	4.44473	8.35658	8
22	9	4	5.5108	4.42686	9.18796	4
23	9	4	5.43254	4.37878	9.45323	6
24	9	4	5.45832	4.84624	10.3985	6
25	9	4	5.1959	5.10143	9.0722	2
26	9	4	5.1959	5.10143	9.0722	2
27	9	4				0
28	9	4				0
29	9	4				0
30	9	4				0

2 Programing part

2.1 Reproduction

If an animal eats enough food than its ”*hunger*”∗”*reproduction*”, where ”*hunger*” is calculated using the function chosen in the ”calculator” parameter and ”reproduction” is the number chosen using the same parameter, it will reproduce. Reproduction is done by making copies of a given individual and changing its ”size”, ”sense”, ”dexterity” parameters by randomly subtracting or adding their percentage, determined by the ”mutation” parameter. The new individual is placed at a random location on the map. For this reason, all three traits are represented by a decimal number. If they weren’t they couldn’t change gradually at a small mutation rate and so would remain the same. The property ”size”, is rounded when used.

2.2 Movement

The movement of each animal is divided into three parts where the **MakeStap** function is called in three rounds on each instance of an animal. The function returns the coordinates to a field the animal should move. On each of these rounds, the animal can eat some food or approach it and travel about a third of the maximum distance. If the **dexterity** is not divisible by three, then it moves the extra amount on its last turn.

2.3 Classes

2.3.1 Simulation

This class controls the running of the simulation and is responsible for loading, running and saving the results. It only manages these tasks, other classes are responsible for their execution.

2.3.2 ImputReader

It has the task of reading the input file and passing the simulation parameters obtained from it. The Chacker class and its descendants take care of checking the parameters and parsing them. There is one descendant of the Chacker class for each section of the input file. For each section, the number, type of parameters and the type of their values are checked.

2.3.3 Entities

The ImputEntities.h, simulationEntities.h and Map.h files contain class headers for representing things on the map and auxiliary structures. Some of them will be mentioned here. In order for food and animals to be stored in the same data structure they must have a common ancestor. This ancestor is the **Item** class, which has a virtual function **type** that returns the type of the object.

The **Food** class is a descendant of the **Item** class, it represents a single instance of a food on the map and it has a reference to an instance of the **PlantSpecies** class, which is only one for each type of a food. The variable **count** in the PlantSpecies class, is the number that the food will be refilled at the end of the day. The **count** variable can change over time, the parameters for this are stored in the **changer** structure.

The **Animal** class is also a descendant of the **item** class and represents a single instance of the animal on the map. It has a reference to an instance of the **AnimalSpecies** class, which holds information about the species of the animal.

2.4 API

The main data structure is a polymorphic vector of vectors containing unique pointers, named **map**. This object represents a two-dimensional map of the environment in which the simulation will take place. The position of each item is determined by its coordinates. The individual items point to an object representing an animal or food. Two items cannot be in the same position if there is nothing at the position there is a null pointer.

The **map** object is in the Map class.

2.4.1 Map

The class inherits from **MapLoader**, where the functions for loading parameters and initial initialization of the Map are located. This class is the core of the program. It holds the main data structure **map**, and stores all the information readed from the input file. All the information about the food types is stored in a vector **plants**, and all information about animal species is in the vector **species**, where instances of the class **AnimalSpecies** are located. If an instance of the class **Animal** wants to retrieve information about its species, then it has a reference to the appropriate instance in it. This information is separated in the program because many instances of **AnimalSpecies** can correspond to a single instance of **Animal**. The information in **AnimalSpecies** is common to all animals of a given species.

Similarly with the **Food** class, where each instance of it also has a pointer to an instance of the **PlantSpecies** class. This reference is the only data item of the **Food** class. It would be possible to directly reference from all places on the map, where food should be directly to an instance of the class with species information. This solution would save one reference and the **Food** class would not have to exist. The current solution is chosen because it offers a better possibility of future extensions to the plant features on the map.

Unlike food, the features of each individual of the same species can change, so the **Animal** class is necessary here.

2.4.2 Function Day

Simulates one day of simulation. It starts by calling the **moveAnimal** function, which moves all the animals

After it moved all the animals, it calculates hunger for each animal and decides whether it will survive, die or reproduce. At the very end, it adjusts the maximum number for all meals and then adds each type of meal to that number

2.4.3 moveAnimal, and Movement of the animals

This function traverses the coordinates of all animals three times in a row and calls the **MakeStap** function on them and moves them to the coordinates obtained from this function. If an animal eats another animal, it moves to that animal's location. This means that these coordinates will be included twice in the new pass, but this problem is solved by storing the coordinates in the unordered set.

The **MakeStap** function is part of the **Animal** class and returns the coordinates that the animal can move to. All conditions on whether this move is possible are checked in this function. We did it this way because we wanted to leave as much freedom as possible when creating a new type of animal with a different way of moving or viewing the environment. This means that the **Animal** class is passed a constant reference to the main map vector with the environment map.

In **MakeStap**, using the **SearchAround** function, it searches the environment but only up to the size of the **sense** variable to find the largest and closest food source it can eat. If the food is within range it will return its coordinates. If it is not in range it calls the function **GoCloser** which returns the coordinates of the empty field that is closer. If there is no food, it will use the function

RandomEmptyBox to return the coordinates of empty place where it can move as far away as possible.

2.4.4 Statistic

This class holds the data. There is only one instance of it in the program and that is in the Simulation class, which gives it the command to save the data.

2.5 Tests

All tests are in the tests folder. Randomness plays a big role in the program. The placement of new animals, food and changes in breeding parameters are decided randomly. Therefore, the Random Number Generator is always initialized with the number 42.

2.5.1 Incorrect inputs

These tests are in the errorinputs folder, they are meant to test that the program does not crash on a bad input and responds appropriately.

2.5.2 Correct inputs

On all these inputs, the program should exit after some time and produce a text file with the recorded simulation run and a csv file with the results. For larger map sizes, I recommend opening the text file in an editor that does not automatically block out excessively long lines.

These tests also test the functionality of comments and other things that should not affect the runtime. Such as different order of sections.

Test **race.txt and race2.txt**

For these tests, there are two kinds of animals and two kinds of food. One can be eaten by any animal, and one animal has to grow at least a little bit, the other kind is big anought from the start. The number of food gradually decreases over the course of the simulation.

You should see a trend in the results where the taller animal gets smaller and the shorter one grows a bit and then stays the same size. in the first test, the second animal survives. While in the second test it does not, because there is a requirement for a minimum height for the second plant species as well.

Test **test4.txt, test1.txt, test2.txt, test3.txt**

These tests are identical except for the type of calculator. So we can see the effect of the hunger calculation on the environment.

Test **big.txt**

In this test there is a 100x100 map, 3 types of plants and 5 types of animals. It may take some time to run.