Report on the project “Planissus”

by Davide Bulfone

Short Introduction

The aim of this project was to create a simulation of a “world” using the python library matplotlib, a useful library that allows the user to represent data on a graph.
Planissus is formed by ground and water tiles, the water tiles are inhabitable.
This “world”, which we called “Planissus”, has 3 different living creatures:

- vegetob: they represent the vegetation of Planissus
- erbast: they represent the herbivores that inhabit Planissus
- carniz: they represent the carnivores that inhibit Planissus

Goal and problematics

The ideal goal of this work should be having a simulation that doesn’t weigh too much on the computer calculation capabilities, providing not only a simple set of rules but also the capability to adjust the simulation according to the user's necessity.
The main problem that I encountered so far is the RAM overutilization by the program, in fact I’m pretty sure I could simplify some of the rule sets.

Development and Methodology

The first attempt I made in writing the program was a complete failure; I started collecting the data I was generating in some lists. At first it worked for the ground tiles and the vegetob “population”, but then I realized that using lists for the erbast and carniz data would be uselessly difficult so I switched to dictionaries to simplify my job. By using dictionaries I could easily store multiple data in each array tile of the map, and access them way more easily.
My program structure is as follows:

- **Settings of the simulations**(world dimension, growth rate for vegetob, …)
- **Zero day input**(generates a set of data from which the simulation will start)
- **Defining of useful functions**(I specified function that I’ll use in the simulation)
- **Sets of rules for each living being**(I created a set of rules for all the animals and plants, to define their simulated behavior in the world)
- **Data representation**(By using matplotlib I represented the data in a graph)

With this design I could easily modify each part of the code and I could make it more readable by dividing it in different sections.
No living being can live, stay or expand in water tiles.
Vegetob are the simplest creatures of Planissus, for each day they grow by one on their original tile and once they reach the maximum inhabitable space of their tile they start expanding in the nearest lowest tile(see `get_low_tile`).

Erbast are the natural enemy of vegetob; they have a total day energy(see `"e_daily_energy"`) which represent the total actions they can do for each day on Planissus. They have 3 different actions they can take: moving, grazing or, eventually, die and reproduce.
Carniz are the natural enemy of erbast; they also have a total day energy but their actions are a bit different; they can: hunt, move or, eventually, die and reproduce.
All the specifications for these actions are written inside the ruleset.
The rulesets I wrote are a series of if() conditions that apply a change in the data set of each element of the world.

Tools and Resources

To learn how to use matplotlib I read the official documentation for it on his site.
For the implementation of the function that I used inside my rulesets I used chatGPT.
I also found some helpful discussions on Github and other programming forums in which they resolved similar problems that I encountered .

Testing and Results

To test the rules I implemented, I printed the data for each variable on different days, then I compared the results and looked for what I was expecting from my rules.
The resulting output of the program is a graph divided in tiles, with each color representing a state of the tile; to make it more understandable, I defined a hierarchy of the data. The color blue represents water tiles, green represents tiles in which only vegetobs are present, red represents the presence of erbast (if there are erbast we assume that there are vegetob) and the color purple to represent carniz.

Possible Improvements

With more work on the project I'm pretty sure I could add the presence of herds and prides, and also re-work the starting data to get a more uniform simulation.
During the writing of the program I overlooked the social attributes of the animals (erbast, carniz). I simply defined a random integer, either zero or one, to define if an individual was "social" or not. This data could be more sophisticated by adding a wider range of values and a more complex "behavior" by analyzing the situation of each individual.
Another implementation could be adding a "god mode" from which the user could stop, reset and restart a simulation; also there could be a tab of data showing the data for each selected tile.