PLANISUSS WORLD

509477 - Computer programming, algorithms and data structures, Mod. 1
Barbieri Chiara *517096*, Sotgia Francesca *513067*

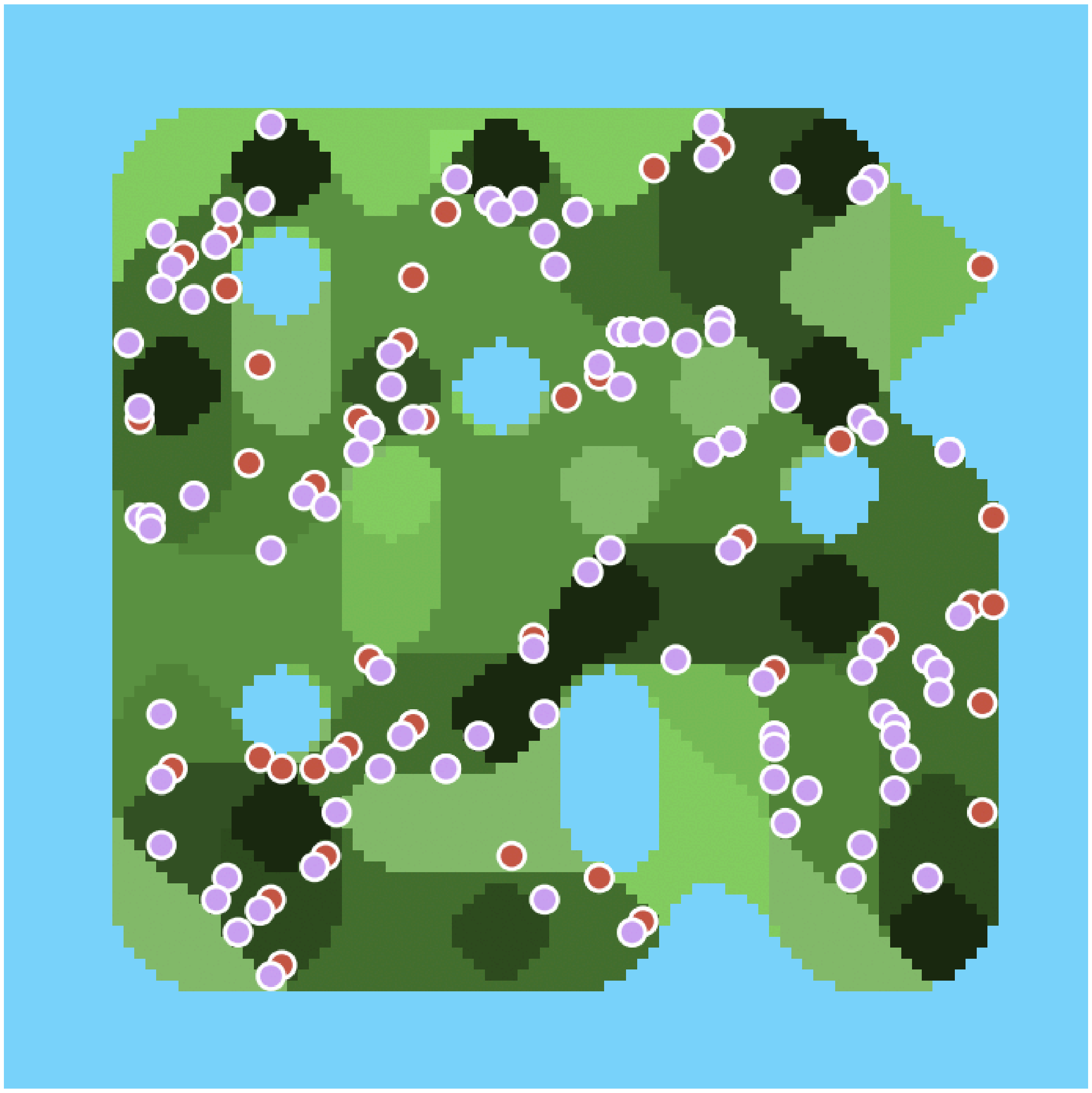


Table of contents

Goal of the project

The goal of the project is to design and implement a fictional world called "Planisuss", which is a single continent populated by three species, the Vegetob that represents the vegetation of the world, the Erbast that is the herbivorous species and the Carviz that represents the carnivorous species .

The world will be a simulation of an ecosystem and we will observe how the three species interact with each other and how they will evolve with time.

Initialization of the World: the species inserted in the world

The Planisuss is structured in cells, which represent the fundamental units, and which are organized into a grid of dimension NUMCELLSxNUMCELLS.

Each cell is described by a pair of coordinates and can either be a water cell or a ground cell, and each ground cell can contain all the three species, that are Vegetob, Erbast and Carviz.

Class "Carviz"

In the Planisuss world, the Carviz represent the carnivorous species that hunt the Erbast to survive, they can also move in the world to find better conditions.

In the class *Carviz* we describe the basic characteristics of the individuals, that is their energy, lifetime, age, social attitude, position and whether they are alive or dead, as well as some of their actions, such as movement.

Initially the first individuals created all start with the same characteristics, which are:

- The energy represents the strength of the Carviz. Energy is equal to the maximum energy a carviz can have, which is set by the constant MAX_ENERGY_C,
- The lifetime represents the duration of the life of the individual. Lifetime is set as a random integer between 0 and and the constant MAX_LIFE_C,
- The age starts from 0, and with each passing day it augments,
- The social attitude is the likelihood of an individual to join a Pride or stay in the Pride. The social attitude is a value picked from a uniform distribution between 0 and 1,
- The position is defined as self.i for the y-coordinate and self.j for the x-coordinate. The position represents the position of the individual in the map.

In this class we will also keep track of how many Carviz are present in the world through the *instance_count* variable.

The Carviz can group together and create a Pride. Prides will not be specifically defined as a class, but they will be directly defined in the map called PrideMap, since in each cell in this map we have a list that can be empty (no individuals), with one item (one Carviz) or with many items (many Carviz). Therefore as a result we get that the Prides are a result of many Carviz in the same cell.

Class “Erbast”

In the Planisuss, the Erbast represent the herbivorous species that graze the Vegetobs and can move in the world to find better conditions.

In the class *Erbast*, as well as for the Carviz, we describe the basic characteristics of the individuals, that is their energy, lifetime, age, social attitude, position and whether they are alive or dead, as well as some of their actions, such as movement..

As for the Carviz, initially the first individuals are created with the same characteristics, which are:

- The energy represents the strength of the Erbast. Energy is equal to the maximum energy a Erbast can have, which is set by the constant MAX_ENERGY_E,
- The lifetime represents the duration of the life of the individual. Lifetime that is set as a random integer between 0 and and the constant MAX_LIFE_E,
- The age starts from 0, and with the passing days will be augmented,
- The social attitude is the likelihood of an individual to join a herd or stay in the herd. The social attitude is a value picked from a uniform distribution between 0 and 1,
- The position is defined as self.i for the y-coordinate and self.j for the x-coordinate. The position represents the position of the individual in the map.

In this class we will also keep track of how many Erbast are present in the world through the *instance_count* variable.

The Erbast can group together and create a Herd. Herds will not be specifically defined as a class, but they will be directly defined in the map called HerdMap, since in each cell in this map, we have a list that can be empty (no individuals), with one item (one Erbast) or with many items (many Erbast). Therefore as a result we get that the Herds are a result of many Erbast in the same cell.

To design the world we have created a class, called “*WorldInitialization*”, that is able to manipulate the individuals, under the respective methods:

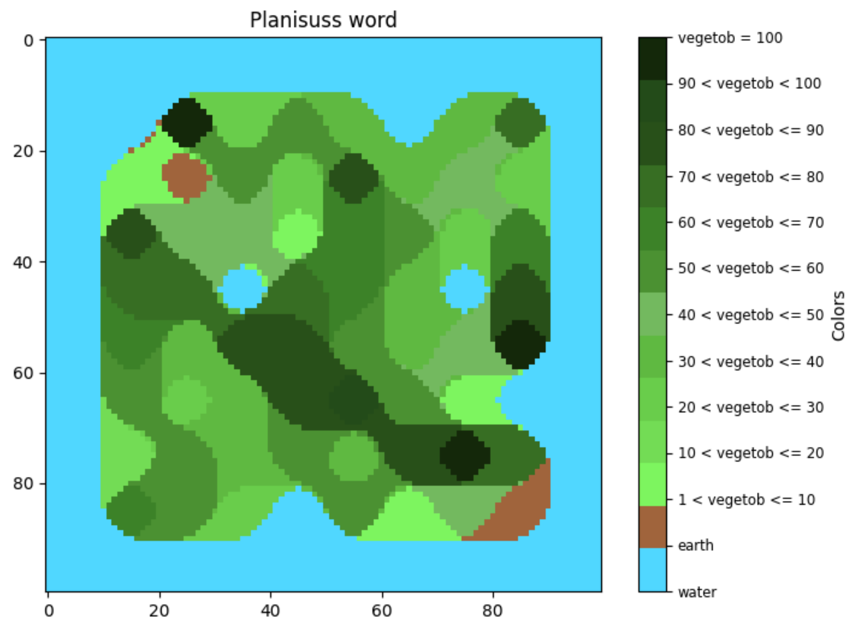
- loadTerrain for the Vegetobs,
- loadPride for the Carviz,
- loadHerd for the Erbast.

loadTerrain

To construct load terrain we have first created a 0 matrix with the numpy’s function *np.zeros((NUMCELLS, NUMCELLS))*, which given the constraint NUMCELLS =100, forms a 100x100 matrix.

Afterwards we have divided the 100x100 matrix into a 10x10 matrix, such that each “cell” of the 10x10 matrix was 10x10. We do this to have a smoother transition from water (value 0), which has to forcely be in the extremes of the matrix, to the inner part of the matrix, composed of simple terrain (value 1) or a vegetob density (value from 2 to 10). In the next explanations it will be clearer the motive behind our choice.

For the 10x10 matrix we make sure that the extremes are all 0, and on the other hand, the inner part is assigned for each "cell" a value at random between 1 and 10, but only if the random probability for each cell is greater than the "TERRAIN_PROB" (constant previously defined). Afterwards we scale the 10x10 matrix so that we have a 100x100 matrix.
After this initialization we will apply two filters, first a filter of size 10 and afterwards a filter of size 5, that will go through the whole 100x100 matrix calculating the median of the values selected and provide a smoother image of the terrain. The result will be a more realistic and natural representation of the world, since it will resemble an island, with different degrees of green, based on the density of the vegetob, brown if there are no vegetobs and light blue if there is water. However, note that for displaying the result we have to run another class, not this one.



loadPride

Before specifying how we inserted the Carviz into the map, we would like to specify our main assumption, that is that we will always nominate the Carviz as "Pride", i.e. a group of Carviz, since the group can have from 1 (an individual) to MAX_PRIDE (maximum number of individuals in a Pride, so 100) number of individuals, so it includes the possibility of a Carviz being alone.
First of all PrideMap is initialized as a list of empty lists, that means that the matrix with dimension NUMCELLSxNUMCELLS will be a matrix having inside empty lists. More precisely, we will have a list that contains NUMCELLS rows where each row contains NUMCELLS empty lists.
To insert the Carviz in our world, we have decided to select cells at random, for which we don't have water, and add an individual inside. Throughout the while loop we could have more than one individual inside the same cell, that is a list with more than one object.
The loop continues to add individuals until it reaches the maximum number of Carviz that we can have in the world initialization, that is the constant MAX_NUMBER_C.
At the end we will have a matrix that either contains empty lists or it contains a list of instances of the Carviz class.

loadHerd

The reasoning behind loadHerd is the same as in loadCarviz, so we always nominate the Erbast as "Herd", i.e. a group of Erbast, since the group can have from 1 (an individual) to MAX_HERD (maximum number of individuals in a Herd, so 1000) number of individuals, so it includes the possibility of a Erbast being alone.

First of all HerdMap is initialized as a list of empty lists, that means that the matrix with dimension NUMCELLSxNUMCELLS will be a matrix having inside empty lists. More precisely, we will have a list that contains NUMCELLS rows where each row contains NUMCELLS empty lists.

To insert the Erbast in our world, we have decided to select cells at random, for which we don't have either water nor Prides, and add an individual inside. Throughout the while loop we could have more than one individual inside the same cell, so inside the same list.

The loop continues to add individuals until it reaches the maximum number of Erbast that we can have in the world initialization, that is the constant MAX_NUMBER_E.

At the end we will have a matrix that either contains empty lists or it contains a list of instances of the Carviz class.

Representation of the World: the Display

With the class *Display* we want to visualize the world and the individuals.

For the world representation we first define the colors we want to use for the species: for the Vegetob we chose degrees of green, for the Prides degrees of red and the Herds degrees of purple, these colors were personally chosen using the color codes.

Afterward we have used matplotlib to display the numpy matrix for the vegetob and the scatterplots for the Herds and the Prides, which were created in the WorldInitialization class. Then we put them one on top of the other (through *make_axes_locatable*) to be able to show the World and the positions of the Prides and Herds on the map.

Moreover, we have that each dot will change its color based on the amount of individuals within each position.

Day in Planisuss

In the class *DayInPlanisuss* we describe everything that happens everyday in the world, so we include functions such as the movements of the animals, the growing of the Vegetobs, the hunting and the fighting, the aging of the animals or the catastrophes.

In each passing day (an iteration of the update function) all the functions are run once, showing in real time how the Carviz (shown as Prides) and the Erbast (shown as Herds) move in the map and how they associate.

We defined the functions to update the maps of the three species (*update_terrain*, *update_scatterPride* and *update_scatterHerd*) and we started the count for the amount of Carviz, Prides, Erbast and Herds present in the map, allowing for the possibility of checking for each day the amount of animals and groups present in the display. Additionally, we calculated

the media of the Vegetobs in the world and the media of the energies of the Carviz and the Erbast, which we will use later in the visualization of the chosen plots.
Once we have defined the animation, we also have defined some constraints to terminate the animation, as well as some buttons that pause, reset and terminate as well as speed up and slow down the animation whenever the user wants to. For a more interactive user experience, we have also added a method for which the user can see directly the characteristics of a cell.

The actions in a day in Planisuss are:

- Growing of the Vegetob,
- Movement of both animals, Erbast and Carviz,
- Grazing of the Erbast,
- Struggle of the Carviz that is composed of two phases, fight and hunt,
- Aging of both animals, Erbast and Carviz,
- Spawning of both animals, Erbast and Carviz,
- Drought,
- Meteorite.

Growing of the Vegetob

Everyday the Vegetobs grow one unit in each cell, until they reach the maximum density that is 100.
When an animal, Carviz or Erbast, is in a cell surrounded by the maximum density of Vegetobs (100), the animal dies overwhelmed by the Vegetob.

Movement

The animal can decide on whether to stay in a cell or move in a given neighborhood, that is defined by the cells surrounding the cell where the animal is.

For the Carviz each individual can choose whether to follow the Pride, stay in the cell or move alone.
Based on the surrounding neighborhood, the Prides first look for any Erbast around them, if the Erbast are present they will move in that cell, however if there is no Erbast nearby, the Prides will go towards cells with higher density of Vegetobs, since they are aware that the Erbast must graze to survive, but on the other hand they will not move towards a cell with a too high density of Vegetobs to avoid being overwhelmed by them.
Therefore the Carviz move to hunt their preys, the Erbast, or to look for them and at the same time to avoid being killed.
The individuals decide to move based on their social attitude and energy, meaning that they will move only if their social attitude is greater than 0.5 or if their energy is greater than 50, or in the case they move with the whole pride both.
In the function *update_movementPride* we also add a part that calls the *fight* function (that we will explain later in the report).

Similarly, for the Erbast, each individual can choose whether to move or not, either following the herd, staying in the cell or moving without following the herd.
Their choice to move is based upon whether there are cells with a higher density of Vegetobs around them, where they can graze more to get more energy to escape from the Carviz, in fact they can also choose to move in a cell where there is the least

amount of Carviz and therefore a higher chance to survive. The Erbast also avoid the cells with a too high density of Vegetobs to avoid being overwhelmed by them.
The individuals decide to move based on their social attitude and energy, meaning that they will move only if their social attitude is greater than 0.5 or if their energy is greater than 50, or in the case they move with the whole pride both.

Grazing

Everytime an Erbast doesn't move, it eats one unit of Vegetob of the cell it is in. The grazing entails that the energy of the Erbast increases by 1 and that the density of the Vegetobs in that cell decreases by 1.
If an entire herd decides not to move, every Erbast gaines 1 point of energy for every point of Vegetob density, however if the density of the Vegetob is lower than the number of the Erbast, we have that only the Erbasts with the least amount of energy get to graze and the social attitude of the individuals that didn't eat will be decreased by 0.2.

Struggle for Carviz

The struggle phase is composed of two moments: fight and hunt.

Fight

If two Prides are in the same cell and they decide not to join in a single Pride, they fight.
The fighting consists in one-to-one matches between the individuals with higher energy of each Pride. This means that in each match the individual of each Pride that has the higher energy (called champion) will fight against the strongest Carviz of the other Pride and the champion with higher energy will survive, while the other will die. The fighting ends when one of the two Prides has no more components.
Moreover, the individuals of the winning Pride will have their social attitude increased by 0.2.
The *fight* function will also be called in the *update_movementPride*.

Hunt

The hunting takes place when the Pride doesn't move and there is a herd or an Erbast present in the cell.
The Pride identifies the strongest Erbast (i.e. the one with the highest energy) and it will try to hunt it; the probability of success of the hunt depends on the relative value of the cumulative energy of the Pride and of the Erbast.
If the hunting is successful (i.e. the Erbast dies), the energy of the prey is divided between each individual of the Pride and in the case where there is some spare energy, this is assigned to the Carviz with lower energy; moreover the social attitude of the individuals of the Pride increases by 0.2.
When the hunting is unsuccessful (i.e. the Erbast survives), the social attitude of the Erbast increases by 0.2 and on the other hand the energy and social attitude of each Carviz in the Pride decreases by 0.2.

Aging

Everyday the animals get older, meaning that their age will increase by 1, and every month (10 days) their energy is diminished by 2 because they get older. When the energy of an individual gets to zero, it dies.

Spawning

The spawning, i.e. the birth of the offspring, can happen for two reasons: if an animal dies due to the reaching of its maximum lifetime (when their age is equal to their lifetime) or when the animal dies because its energy reaches 0.
When an offspring is created its properties are:

- the age is set to 0,
- the lifetime is equal to the one of its parent,
- the social attitude is equal to the one of its parent,
- the sum of the energies of the two offspring is equal to the one of the parent.

When a Carviz or an Erbast reaches its maximum lifetime, it generates two offspring with the properties stated above, the same happens when an Erbast's energy gets to 0, whereas the energy of a Carviz gets to 0 only one offspring is generated and its energy is a random value between 1 and MAX_ENERGY_C (we decided to do so to avoid an exponential growth of the Carviz population, since they use up their energy faster due to their higher likelihood of moving).

We decided to also implement two functions that represent catastrophes: drought and meteorite.

Drought

The drought happens every month (10 days) and it decreases the density of the Vegetobs by 5 in each cell of the world.
We decided to add this function to the Planisuss to avoid all the animals being overwhelmed by the Vegetobs too fast.

Meteorite

Every month (10 days) a meteorite hits the world and causes a reduction of all the Vegetobs' density and the death of all the animals in the impacted area.
The area of the impact is chosen at random and can vary in size, from a square of side 2 to a square of size 6.

Interactivity

Once we have defined all the functions that describe what happens everyday in the Planisuss, we focused on the interactive part of the project.
We decided to incorporate in the visualization some buttons that allow the users to interact with the Planisuss, these are: pause button, stop button, reset button, speed up button and slow down button.
Moreover, as mentioned before, we have added the ability to read the properties of a cell directly on the display. If the user has stopped or terminated the animation, he can select a cell and directly see on the display the properties of that cell; these properties are the amount of vegetob, Herds and Prides present, and if the Prides or Herds are not empty we return the characteristics of each animal. Once the user has clicked a cell and read the characteristics, he only has to press "r" on the keyboard and the plot will go back to show the previous display.

Please note that in the terminal the day at which you stopped and its characteristics will be printed again, so that the user can precisely see where he had left in the animation.

Pause button

The pause button, created by the function *create_pause_button*, allows the user to pause the simulation of the Planisuss, meaning that the simulation will freeze in the exact state it was in when the user decided to pause it.
If the user wants the simulation to continue, they can press again on the button (that is now showing the text "Restart").
We defined how the pause button works with the function *toggle_pause*.

Stop button

The stop button, created by the function *create_stop_button*, allows the user to terminate the simulation, without being able to restart it after.
We defined how the pause button works with the function *toggle_stop*.

Reset button

The reset button, created by the function *create_reset_button*, allows the user to reset the simulation, meaning that it will start again from Day 0 and won't take into consideration what happened in the previous simulation.
We defined how the pause button works with the function *toggle_reset*. To allow this to work correctly we defined the *reset_satate* in which we defined how the world is initialized when the user presses the reset button: the day is set again to 0, the number of the animals are set to 0 as well, we reset the world maps to their initial state and empty the maps to then reinitialize the PrideMap and HerdMap with old instances.

Speed up button

The speed up button, created by the function *create_speedUp_button*, allows the user to increase the speed of the simulation. To do so we need to decrease the time interval between one iteration and another (in our case between each day) and we did so in the function *increase_speed*. We defined how the pause button works with the function *toggle_speedUp*.

Slow down button

The slow down button, created by the function *create_slowDown_button*, allows the user to decrease the speed of the simulation. To do so we need to increase the time interval between one iteration and another (in our case between each day) and we did so in the function *decrease_speed*. We defined how the pause button works with the function *toggle_slowDown*.

Passing of time

As we said before, all the events on Planisuss happen in discrete time units, called days. Each day corresponds to one iteration and we keep track of time by *self.days*.
Since the simulation can, and most probably will, last several days we also define months, which correspond to 10 days, years (10 months), decades(10 years) and centuries (10 decades) and we do so in the function *increment_time*.

Update the simulation

We want to be able to see in the visualization how the world evolves each day, meaning that we want to visualize how the animals move in the world, the growth of the Vegetobs but also the spawning. So we defined several update functions that allow us to do so.
Firstly we defined the *update_statistics* function that returns us the current day, the number of Carviz, Erbast, Prides and Herds present in the simulation in that moment as well as the media of the Vegetobs and how many days, months, years, decades and centuries have passed since the start of the simulation.
Then we defined the *update_world_state* function that update the state of the world, this means that it keeps track of everything that happens in each day: how the animals move, the grazing, the hunts, the aging of the individuals, the spawning, the growth of the Vegetobs and the catastrophes (drought and meteorite).
Afterwards we update the display of the world with the new data through the *update_display* function.
Finally the whole graph is updated through the *update* function, in which we take into consideration the passing of time as well as the previously defined update functions and it also ends the simulation if certain conditions are met, those are if the simulation reaches the maximum number of days (NUMDAYS = 100000) or if the number of either the Carviz or the Erbast is less than 0. The *update* function allows us to visualize the passing of time and everything that happens in each day in the simulation.
All the above can be seen on the terminal, where the actions that happen in a day are printed.

Animation

To create the animation we have used *FuncAnimation* that creates a new frame of the world and its actions everyday. The *update* method tells us what happens in the day, and the animation returns an updated display.

Planisuss Properties

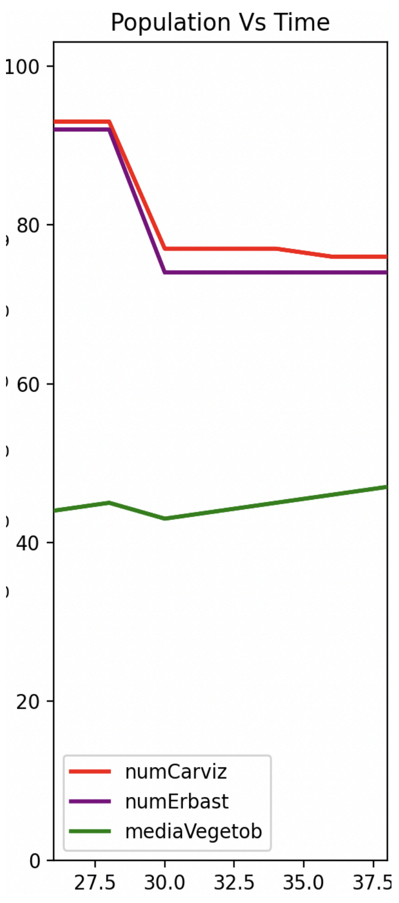
With the map we visualize how the animals are distributed in the world and how they behave, but we also want to represent plots that allow us to better understand how some properties evolve with respect to time to have a better understanding of the progression of the species.
To do so we defined the class *PlanisusProperties*, in which we create two plots:

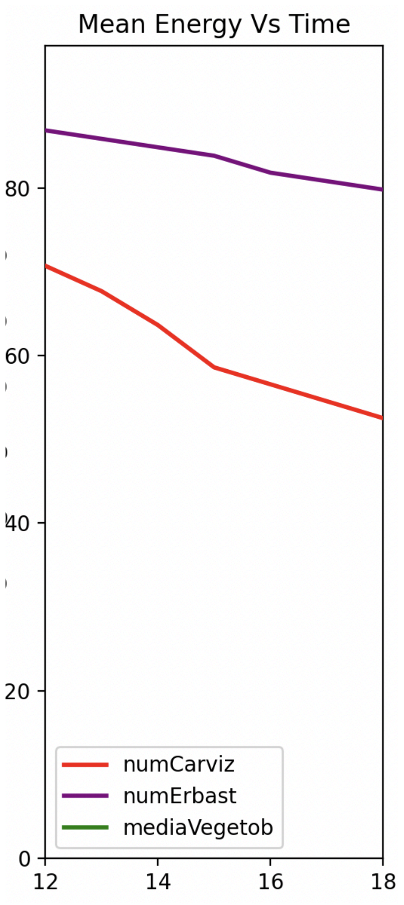
- Population Vs Time,
- Mean Energy Vs Time.

We start by calling the class DayInPlanisuss, so that we could have daily data and initializing the first plot.
In this class as well, to make the user have a more interactive experience, we have added the possibility to have the plot change on a click, in fact if you click one plot the other one will appear. To this we have used the method *mpl_connect.*

Moreover, another similarity with the class DayInPlanisuss, is that here as well to have a graph that changes overtime, we have used *FuncAnimation,* the choice to use the same method was driven by the fact that likewise we can have the same velocity in changing the frame for both the world graph and the plots.

Because of the limited space in the display, the axes change their values adaptively to the data present at the moment. Moreover, we have chosen that time is always represented on the x-axis and the dependent variable (either the energies or the amount of individuals in each population) always on the y-axis.





Calling of the classes

To run the code we had to call the classes on the __*main*__, used to specify without uncertainties the order with which the functions have to be called. Moreover, we would like to specify the fact that creating a variable to specify each class was necessary to be able to always have in consideration the same display, since at each run we have different combinations of initializations in *WorldInitialization*. Likewise we create one initialization that is called for everything.

Possible implementations

One implementation we could have done is the logs and replay at any given moment of the simulation. To do this we should have created a dictionary where for each day (logged as key) we associate a stored version of the actions of that day, so how the display changes (logged as value). Therefore, we would get a dictionary where the values are the values of each cell (noted as the values in the positions *[i][j]* for each map of the display, which are *terrainMap*, *PrideMap* and *HerdMap*).

This would have allowed us the ability to pick any given day and return its value, even as a sequence if we wished (using yet again *FuncAnimation*, by defining the update function as a loop through the dictionary that takes certain keys chosen by a constraint and returns their values in order displaying them).

Obviously, in the described dictionary it would be possible to log any kind of information based on the use decided by the person implementing it.

We have not decided to implement it because we wanted to focus on other aspects of the project, especially how to manipulate *FuncAnimation* for a sequence of frames that depend on more than one plot.