2nd Project (Python) report

# Michał Rejmak, s184464, Data Engineering

Every requirement apart from “abstraction for world with hexagonal fields” has been met

# Implementation of the world and its visualization

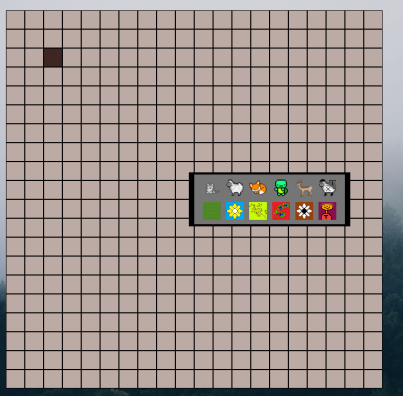
The world is made of squared board 20x20 so that it can contain maximum 400 organisms.

**Every action except from human movement is performed by user’s mouse.**

The whole screen is divided into 2 parts. On the right side, there is menu which is responsible for actions like:

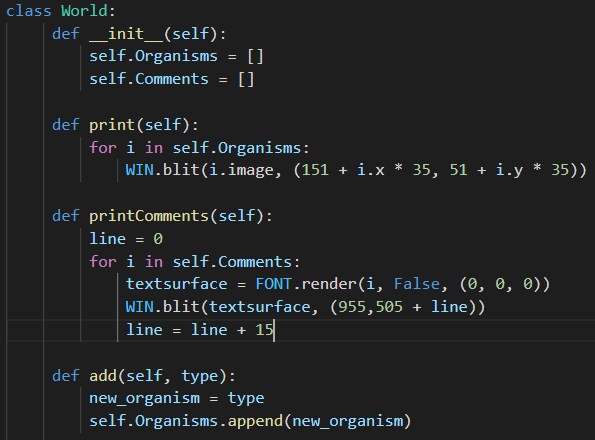
* Restarting the game (Life! The game)
* Creating a new game with some instances of every organism (New game)
* Loading recently saved game state to the board (Load game)
* Saving current game state into a text file (Save Game)
* Performing next round (Next turn)
* Usage of Human’s special ability, if it is possible (Human’s special ability also causes a “Next turn” effect.
* Displaying the comments about actions that happened in last round

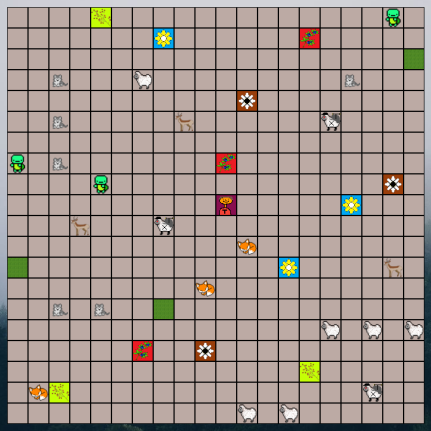
**Human’s movement performed by arrows also causes the “next turn” effect.**

****

On the left side, there is a board itself which is empty after launching the game. To start the game, it is needed to choose one of menu’s options or to add the organisms manually by clicking on a free cell and selecting a type wanted.

**World class:**



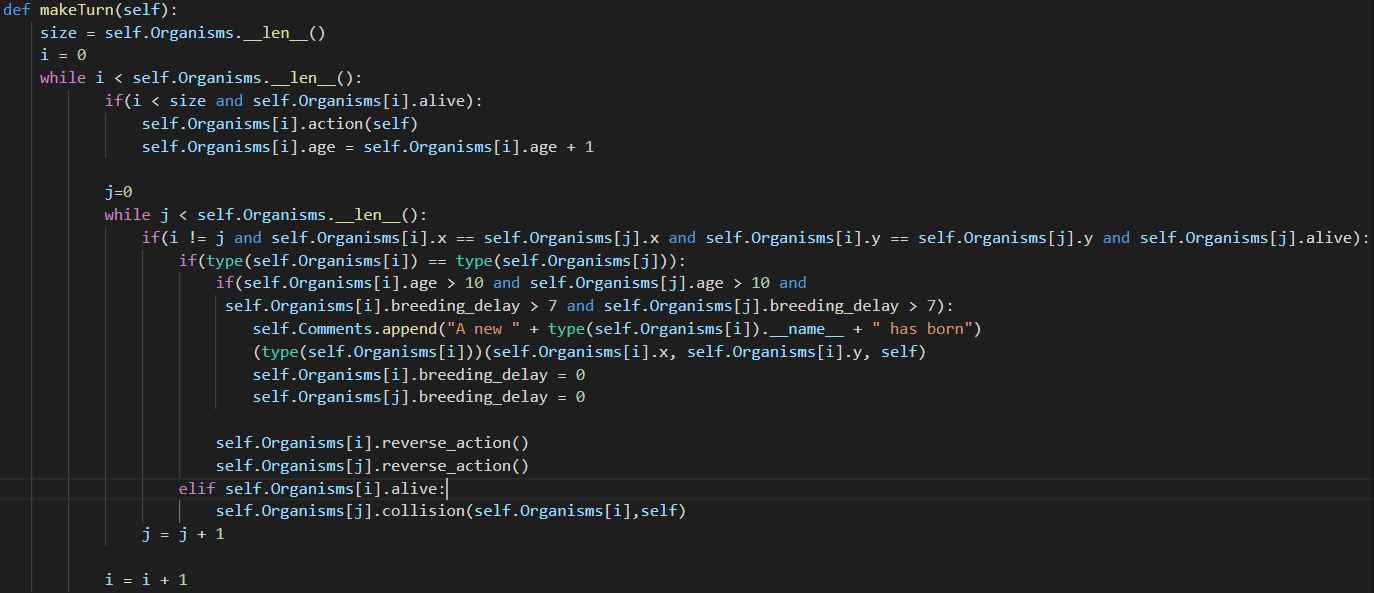


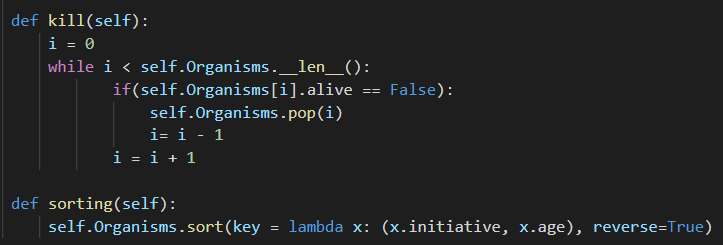
The game is performed by creating 1 object of the World class. It contains fields such as:

* List of all the Organisms that are currently alive
* List of comments of the actions and collisions that took place in the last round

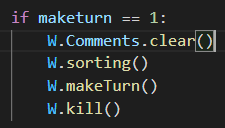
Functions:

* Constructor which creates already mentioned lists
* Print – draws the image of every living organism in the proper place on the board
* Print Comments – write all the comments in the proper section
* add – is used in the constructor of the organisms. Creating an organism will always cause adding it to World’s “Organisms” list



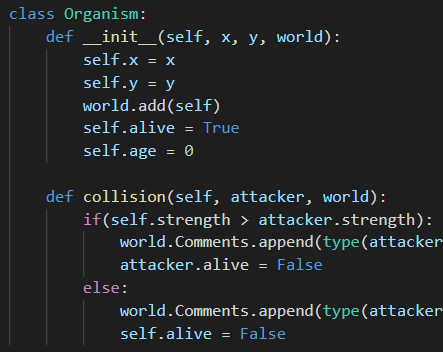


* make Turn – a function similar to the one in C++ which is responsible for performing all the actions and collisions between organisms. This time it works together with two other functions: “kill” and “sorting”.



First of all, all the comments from the previous round are being deleted. Then, all the organisms are being sorted by initiative and age. Then comes the call of the “make turn” function. There are two loops here. In the first one, particular “action” function is being called for every organism **that existed at the beginning of the round** (thankfully to the “size” variable). For the animals, it will be a change particular position field, but for the plants there is a process of spreading. Age of those organisms is increased by one. The second loop checks the collision between all the existing Organisms in a particular moment. The game is set that there will be no collision of the same two plants (they do not spread if the cell is already occupied by the same plant). Therefore when the position of the organisms of the same type collides, multiplication occurs. I had to implement some limitations (age > 10 and time from last multiplication > 7) because animals would start to multiply with their own children and the map would soon run out of the free cells. At first, new organism of a given type is created in the cell of collision. Then, both parents calls the function “reverse action” and return to their previous cells. If organisms are not of the same type, function “collision” is being called with the assumption that it is called by the organism that is being attacked. In that way, usage of their abilities will be performed. During the collision, most probably some organisms will die. In such case their “alive” attribute is being set to False. There comes the “kill” function which after the whole round of actions and collisions will delete from a list those organisms whose “alive” attribute is set to false.

# Implementation of all required animals and plants with breeding and sawing.



**Base class: Organism**

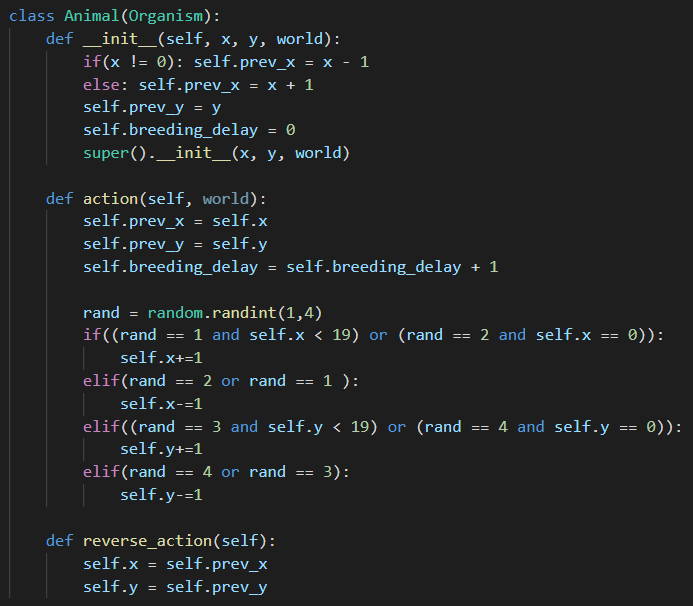
It has all the fields and functions that are common for every organism.

* X and Y are obviously the actual coordinates on the board
* Active – Boolean attribute that shows whether organism is supposed to perform an action/collision or should be deleted after the round(initially set as a True, of course).
* Age – number of rounds that an organism lives
* After creating every organism, world’s function “add” is called to add it to list of organisms.

**The only universal function, for a basic collision:**

This function only checks whether the strength of the attacker is bigger or equal than his opponent’s. In such case, it changes his “alive” status so that it will not perform any action or collision and will be deleted at the end of the round. Otherwise, the same things happens with the attacker.

Base class that inherits from Organism: Animal



It has 3 additional fields:

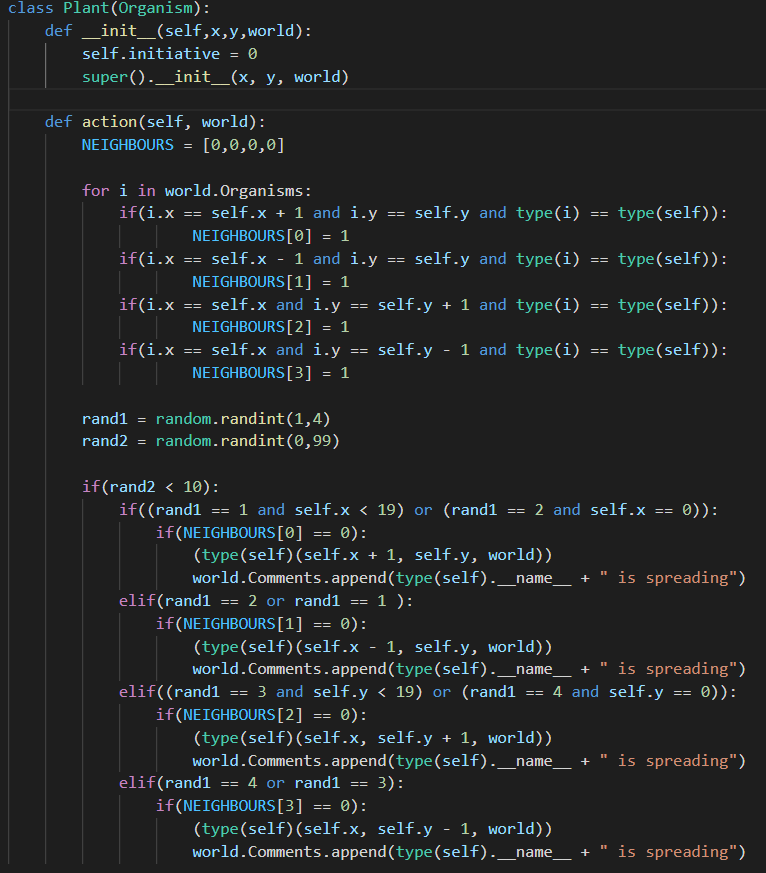
* Previous x and y – in order to perform reverse action or turtle’s reflection.
* Breeding delay – to count days since the last multiplication and limit this process

Functions:

Action – each time, it sets the current coordinates as the previous (before movement) and increases the breeding delay. Then comes the randomization of the movement. Depending on the integer from set <1,4>, organism select the direction of movement. As a result the particular coordinate will change its value by 1. In case of the situation that the animal choose the direction that would lead him out from the board, the opposite direction is selected.

Reverse action – performed when animals are supposed to return to the previous cells. Then simply, previous coordinates becomes the current ones.

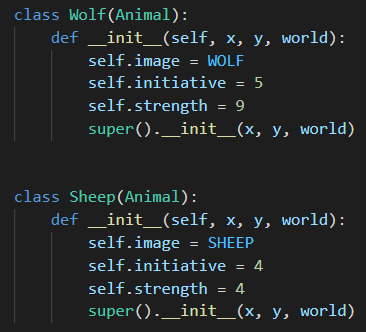
Base class that inherits from Organism: Plant



It has no special fields that any animal would not have. However, it sets the initiative which is equal to 0 for all the Plants.

It has only one function: action

At the beginning, there is a loop which checks whether in the neighbouring cells, there are plants of the same class. If so, particular element of the NEIGHBOURS array is set to 1. Then, there is another random variable which will state if the plant will try to spread (10%). If so, it proceeds similarly to process of selecting the dimension of spread as in the animal’s function. The differences are that the organism will not move but will create a new organism in selected place and the process will not happen at all if selected cell is occupied by the plant of the same type.

Derived from animal: Wolf, Sheep

Those classes are the most basic animal-derived classes.

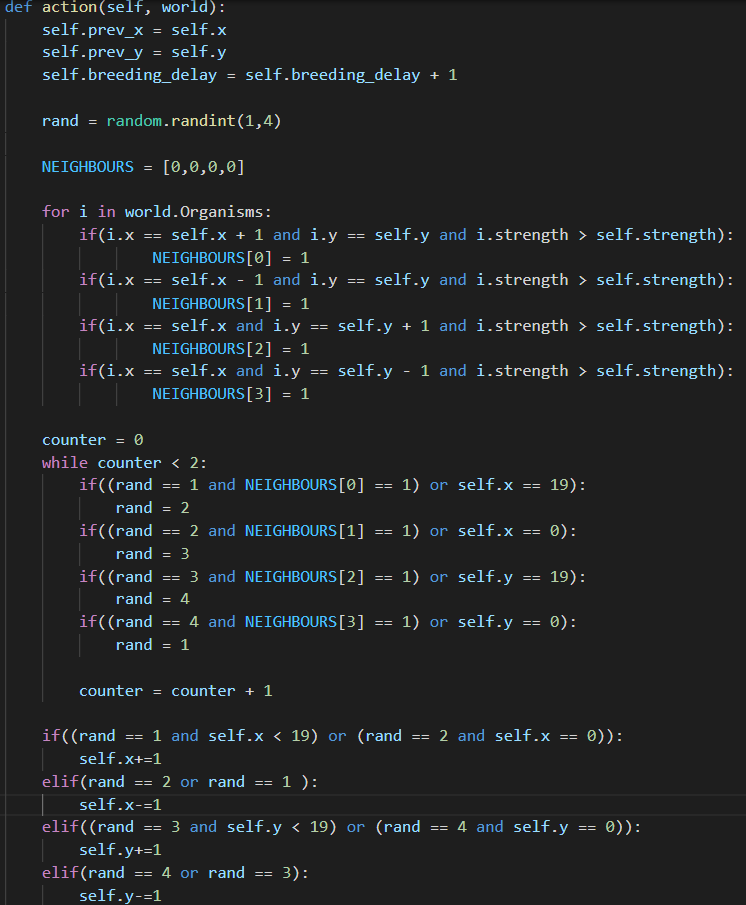
In the constructor, particular values of initiative and strength are set.

New field “image” occurs. There are images defined in “constants.py” file. It is graphical representation of objects of those classes at the game board.

Instances of these classes, use the basic action and collision function implemented in their base classes.

Derived from animal: Fox

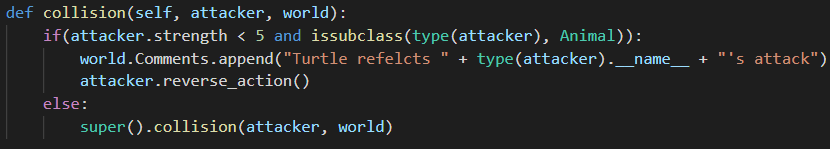
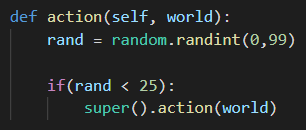
The only thing that differentiate Fox from Wolf and Sheep is its complex action function



The beginning is the same as in the Plants’ action function. We are looking for the neighbours, but this time, not of the same type, but those with the higher strength.

Then, it is loop that iterate twice, to make sure that selected future cell is not occupied by the stronger organism. In this loop, there is a correction of the randomized value, repeated as long as proper cell will be found (unless there is no such cell).

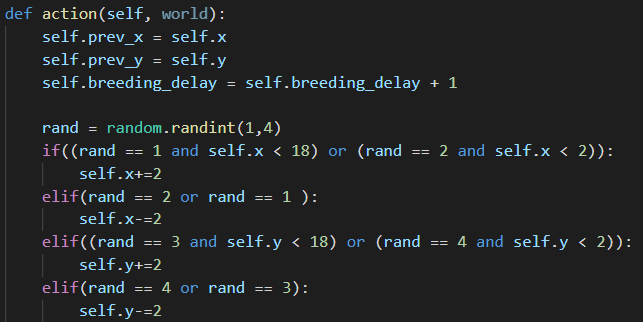
The last part of this function is exactly the same as in the basic animal’s action function.

Derived from animal: Turtle

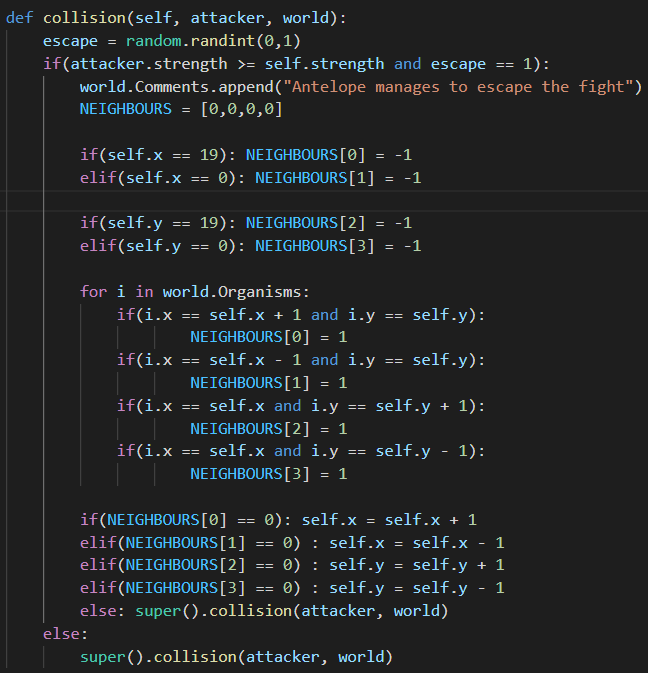
It has implemented its own action and collision function. However, both of them call a base function after all.

Action: simple randomization a number between 0 and 99 inclusive. Then, to achieve the 25% chance of movement, only if the randomized variable is lesser than 25, base action function is called.

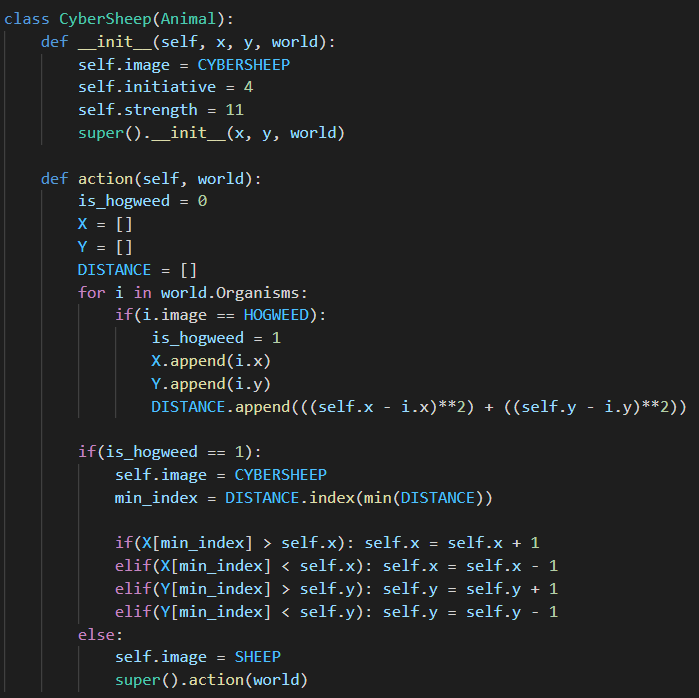
Collision: if the attacker is derived from animal and its strength is lesser than 5, turtle forces the usage of reverse action by the attacker. No collision occurs in fact. Otherwise, base collision function is being called.

Derived from animal: Antelope

Antelope’s action function is generally the same as the basic animal’s one. It has only difference that the x or y variables changed not by 1 but by 2 and some more restriction not to let it go out of the board are implemented.



Collision in this case, again uses the same algorithm of checking neighbours. This time we are looking for a free cell. If an antelope is weaker than the attacker, it is randomized 50/50 whether she will escape or not. If so, antelope moves into the first free cell.

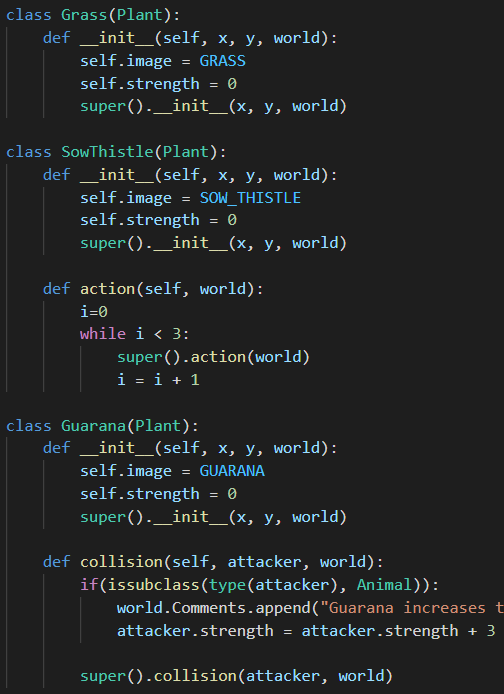
Derived from animal: Cyber Sheep

The whole point of cyber sheep is that it moves towards the Sosnowsky’s Hogweed, because it is the only one who can kill it without dying.

Therefore in its action function it checks whether there is any hogweed alive. It gets all the coordinates of them and calculate squared distance between cyber sheep and each hogweed. Then smallest distance is being chosen and under the same index, there are coordinates of the closest hogweed . Then, the whole movement is about moving towards the Hogweed. First in x axis, then y axis. Organisms do not move across, so the order of those moves does not matter.

If there is no Hogweed in the game, cyber sheep changed its image into to basic sheep however it keeps its strength and will not multiplicate with normal sheeps.

Derived from Plant: Grass, Sow Thistle, Guarana



There are three quite basic plant-derived classes.

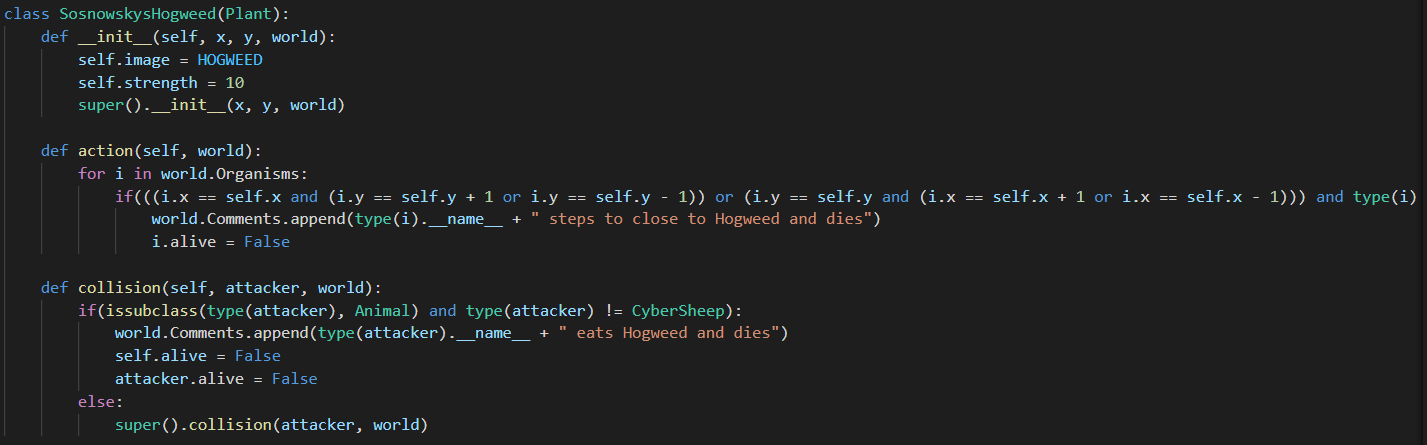
Grass has no special functions implemented.

Sow Thistle calls the basic action function 3 times, as it performs 3 attempts to spread each turn.

When Guarana is attacked by the organism animal-derived. It firstly increases the strength of it and then, basic collision function is called. It is the reason why, in saving the game state, all the animals saves also their strength.

Derived from Plant: Belladonna, Sosnowsky’s Hogweed

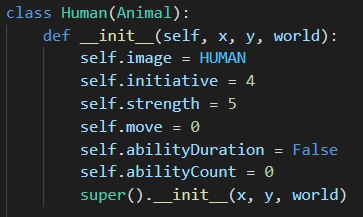
Its collision is simple. If it collides with the animal. Both of them dies (alive = False). Otherwise basic collision is called.



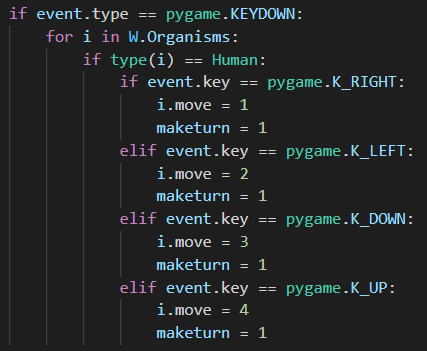
Sosnowsky’s Hogweed does not spread. Instead it checks whether there is any organism around it. If so, it is being killed.

The collision is exactly the same as in the Belladonna but both action and collision do not work when the type of the organism is cyber sheep which can kill hogweed without dying.

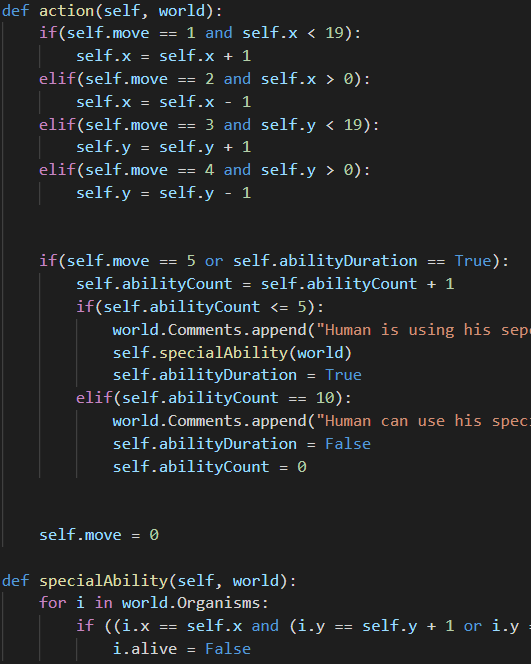
# Human and his special ability implementation

Human has additional fields such as move, ability duration, ability count.

Move is in integer which causes a move of the human if it has a value between 1 and 4

ability duration is a Boolean variable which is set as true to be able to perform special ability for 5 rounds after activating it.

Ability count is integer which after activation will increase by 1 each turn up to 10. First 5 times, special ability will be activated, rest 5 is for 5 rounds break between another usage.

When the action is called, Human moves dependably on “move” variable. There is no random factor. The logic is the same as with the animals but when human reaches the end of board and tries to proceed in that direction, it just do not move at all.

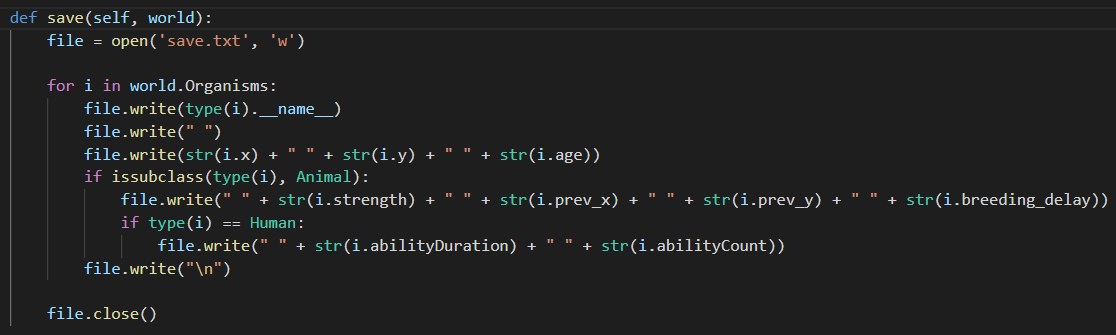
Then is the check of the special ability activation. If the “move” has been set to 5 by the menu button, ability duration is set to “True” and in next 5 rounds, special ability will be called. Then, for 5 rounds nothing happens, even though ability duration is still true, until the ability count will reach 10 and reset so that ability will be available again.

Special ability just looks if there is any organisms around a human. If so, it sets its “alive” attribute to False, so that it will be deleted.

# Implementation of saving and loading state of the world to file and from file.

In order to handle saving and loading process, the class Game state has been created. It has no fields but only two methods: load and save.

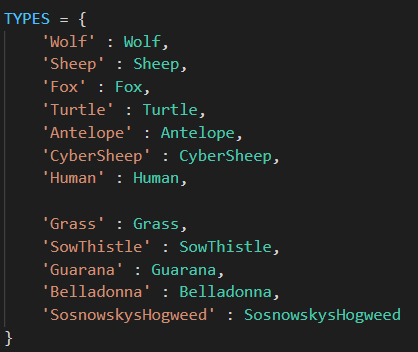
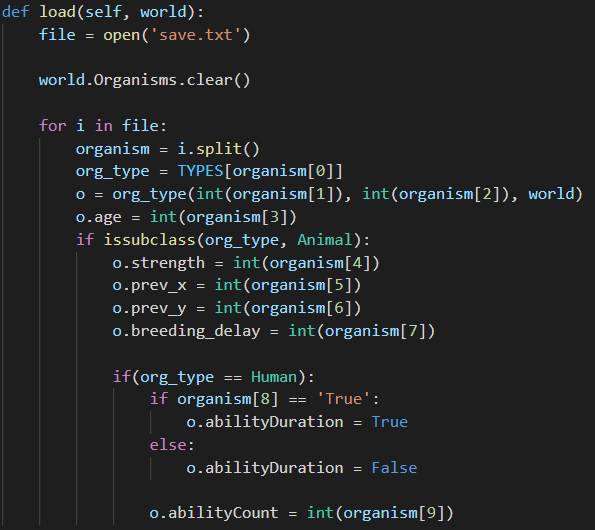
SAVE:



All the variables important to correctly load the game state are stored in the text file. Therefore, there is a loop through all the organisms.

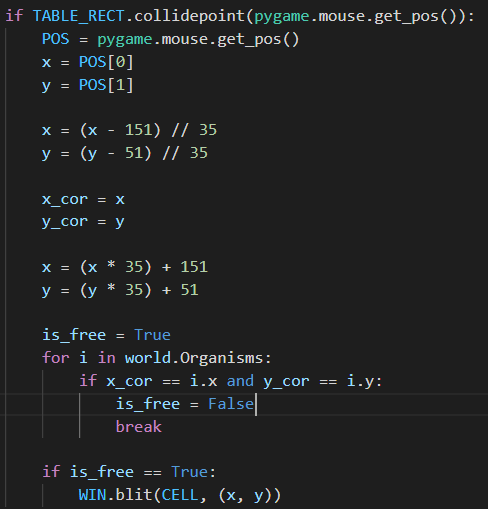
Plants saves only the name of its type, coordinates and age.

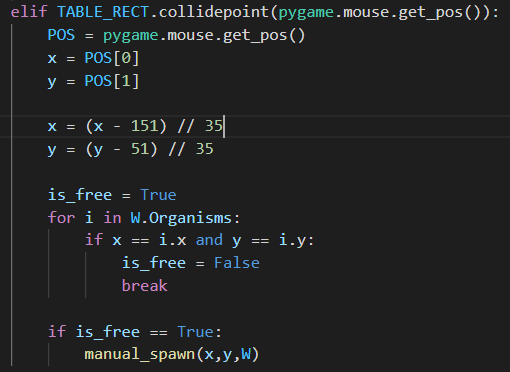
However, animals stores also strength (because of the guarana), previous coordinates and breeding delay. If Animal is a Human, his unique feilds are also stored. The whole process happens when user presses the proper button in the menu section.

LOAD:

First step is to delete current game state so to clean the organisms list. Then, all the lines are split into arrays of strings. Based on the array of types, the proper organism is added (proper constructor is called) and depending on the type, all necessary fields are being set.

# Implementation of adding a new organism to the world by clicking on a free map cell.





First step is to extract the coordinates of the free cell by the position of the mouse. Therefore, X component is decreased by the upper left x position of the board and this value is divided (integer division) by the length of the one cell (35). One function checks if its free and call the “manual spawn” function while other draws a darker square on the free cell to let the user know that he can add an organism there.

After clicking with Left mouse button on the free cell, “manual spawn” function is called. In the middle of the screen appears the window with the picture of every organism. Clicking on any will cause creation of the particular class instance in a proper cell.

ORG\_NUMBERS is defined in the “constants.py” file and allows not to write huge amount of if statements but directly create a proper object.

12 pictures that appears while choosing the right type of organism, are in the same area that rectangles that correspond to the particular integer that will lead to the proper constructor’s call.

