COMP 204 – Assignment #4

Due date: Monday, November 18, 23:59

- Submit *one* Python program on MyCourses, which should contain all your functions. The file should be called <u>ecosystem_simulator.py</u>
- You may submit your program more than once, in which case the latest submission will be graded.
- Write your name and student ID at the top of the program
- For each question, complete the relevant functions and methods. **Do not change the names of the class, functions, methods, or arguments.** You can create additional functions/methods if needed.
- For each question, your function will be automatically tested on a variety of test cases, which will account for 75% of the mark.
- For each question, 25% of the mark will be assigned by the TA based on (i) Re-use of functions as appropriate; (ii) appropriate naming of variables; (iii) commenting of your program; (iv) simplicity of your program (simpler = better).
- **Important notes:**
    - This assignment focuses on function re-use and object-oriented programming. You will be writing several functions and using some classes. Some of those functions should call other functions in order to re-use the code already written. Make sure to re-use your functions as much as needed!
    - The question statements include hints as to how many lines of code are needed to answer each question. <u>These are merely suggestions, and you will not be penalized for using more/fewer lines</u>. If you find yourself coding much more lines than what it suggested, it may be a hint that you are overcomplicating the problem.
- **<u>Super important</u>:** For your submission, include the original non-random my_random_choice( ) function.

<u>Background</u>

In this assignment, you will write a Python program that simulates the evolution of a simple ecosystem. In doing so, you will also reinforce your understanding of object-oriented programming.

Let us start by describing the simulation we are aiming to program. Our goal is simulate an African ecosystem. The ecosystem consists of an L x L grid, where each cell represents a 1 km by 1 km square. Each cell is either occupied by a *single* animal, or is empty. We will consider only two species: zebras and lions. The Figure below illustrates a possible state of the ecosystem (with L = 10).

	0	1	2	3	4	5	6	7	8	9
0										
1			Z						Z	
2		Z								
3						L				
4					Z				Z	
5			Z						Z	
6										
7					L					
8										
9			Z		Z					

The ecosystem evolves over time. We will assume that we model its evolution in time steps of one month. This means that the ecosystem will be in one state at month 0, then in another state at month 1, then another in month 2, etc. The parameters of the simulation (described below) are not very realistic; they've been chosen to make the problem interesting and not too hard.

Parameters of the simulation:

1. The size of the grid: L=10
2. Duration of the simulation: 20 months
3. Maximum age of lion: If a lion makes it to 18 months, it dies of old age
4. Maximum age of zebra: If a zebra makes it to 7 months, it dies of old age
5. Death by starvation of lions: If a lion has not eaten in 6 months, it dies of starvation
6. Death by starvation of zebras: Zebras never die of starvation, because we assume there's always grass for them to eat.
7. Reproduction:
    a. Lions reproduce at age exactly 7 and 14 months only
    b. Zebras reproduce at age exactly 3 and 6 months only

State of the system: An *L* x *L* grid, where each cell is either empty, or contains an animal, with a certain species (lion or zebra), age, and time since last meal.

Update of the state from one month to the next:
Each month, the following events take place, *<u>in that order</u>*:

1) **Aging**: Make all animals grow older by one month
2) **Hunger:** For all animals, increase by one month the time since the last meal
3) **Death of old animals:** Remove ALL animals that have exceeded their life span from the simulation (see parameters above). Animals that die are simply removed from the simulation.
4) **Death of malnourished animals**: Remove ALL animals which have starved (see parameters above).
5) **Moves:** Each animal that is still alive tries to move randomly to one of the (up to) 8 cells that surround it horizontally, vertically, or diagonally. If the animal is near the edge/corner or the grid, it is only allowed to move to a valid cell (within the grid). When an animal tries to move to a cell that is already occupied by another animal, three things can happen:
    - If the animal that is moving is a lion and it moves to a cell that contains a zebra, it eats the zebra and replaces it in that cell. The zebra is then removed from the simulation.
    - If the animal that is moving is a zebra and it moves to a cell that contains a lion, it is eaten by the lion and is removed from the simulation.
    - If a lion tries to move to a cell occupied by another lion, or if a zebra tries to move to a cell occupied by another zebra, the move is forbidden so the animal remains where it was. The animal does not try to find another cell to move to.
6) **Reproduction:** If an animal reaches the time of reproduction (see above), a new animal of the same species is born and is placed in one of the empty neighboring cells from the parent (randomly chosen). If there is no empty neighboring cell, the baby dies (i.e. it never makes it to the simulation). Note: We are assuming here that each individual animal is capable of having a baby by itself (like bacteria); there is no need for a second animal of the same species to be in the vicinity.

Important notes:

- The 6 steps above will need to be executed in successive stages where each stage has an effect on all animals (e.g. have a piece of code to age all animals by one month, then a piece of code to remove all animals that have starved, etc…). DO NOT make one animal age, check if they died of hunger/old age, move, etc… and move on to the next animal; this will give you the incorrect answers.

- For each of the 6 steps above, you should proceed from left-to-right and top-to-bottom in the grid. So the first cell to be considered is the cell (0,0), then the cell (0,1), then (0,2)… If you do it in a different order, you will obtain different results from those we expect.

Assignment

Download *ecosystem_simulator.py* from MyCourses.
The file contains a class called "Animal" and several functions that you will need to complete, and the *my_test( )* function, which can be used to run your simulation. The grid of animals is represented as a two-dimensional list, each empty cells containing the empty object None, and other cells contain an object of the class Animal. The program also contains the *initialize_grid(size)* function (which has already been written ny us) that you will use to initialize your grid and start your simulation.

Download *expected_output.txt* from MyCourses.
This file contains examples of correct output for each question. This output is produced by the function *my_test( )*, provided in the code.

Important notes:

- To make it possible for our TAs to import your code within their grading system:
  - All your code should either be in a function, a class, or a method, or it should be within the *if* block at the end:
    ```
    if __name__=="__main__":
            # PUT CODE HERE IF YOU WANT; IT WILL NOT BE GRADED
    ```
  - Make sure the file you submit is called ecosystem_simulator.py (not something like ecosystem_simulator(1).py or anything else).

Question 1. (10 points) [8-20 lines of code] Write the function *list_neighbors(current_row, current_col, grid),* which takes as input the row index and column index of a position in the grid and returns a list of all neighboring positions in the grid. This list should include 8 tuples in cases where current_row and current_col are not close to an edge of the grid. Otherwise, it would contain 5 tuples (if next to an edge) or 3 tuples (if close to a corner). This list's tuples can be in any order.

Question 2. (10 points) [8-20 lines of code] Write the function
random_neighbor(current_row, current_col,animal_grid, only_empty=False)
that returns a randomly selected neighbour of the current position (*current_row*, *current_col)*.
If keyword argument *only_empty* is True, then only empty neighboring cells should be considered, i.e. your function should only return a neighbouring cell that is not already occupied by an animal. If all neighboring cells contain animals, this function should return None.
If keyword argument *only_empty* is False, then your function should pick randomly among the neighbors, irrespective of whether an animal occupies the cell or not.

Your function should make use of the *list_neighbors( )* function and the *my_random_choice( )* function.

Question 3. (5 points) [2-5 lines of code] Write the Animal class method *can_eat(self, other),* that returns True if self can eat other, and False otherwise. Remember that only lions can eat zebras.

Question 4. (5 points) [2-5 lines of code] Write the Animal class method *time_passes(self)*, which increments by one the age and time_since_last_meal of the object it is called on.

Question 5. (5 points) [4-6 lines of code] Write the Animal class method *dies_of_old_age(self)*, which returns True if the animal is in age of dying, and false otherwise. The function does not actually kill the animal.

Question 6. (5 points) [4-6 lines of code] Write the Animal class method *dies_of_hunger(self),* which returns True if the animal is supposed to die of malnourishment, and False otherwise. The function does not actually kill the animal.

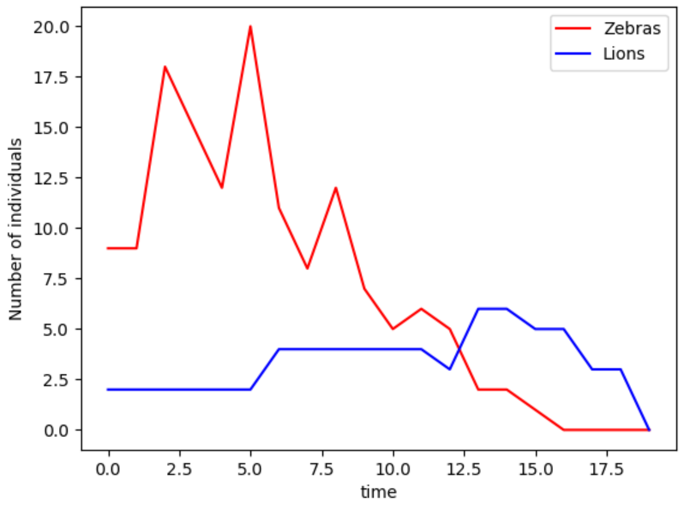
Question 7. (5 points) [4-6 lines of code] Write the Animal class method *will_reproduce(self),* which returns True if the animal is supposed to reproduce this month (depending on their age), and False otherwise.

Question 8. (40 points) [50-100 lines of code] Write the function *one_step(animal_grid)* that performs one step (i.e. one month) of the simulation, starting from a given animal_grid. The function should modify the animal_grid to reflect changes in the animals' positions. Cells containing an animal that died should be restored to None. It should also return the list of all events (described by Strings) that took place during that time step. Example of events would be:
Lion dies of old age at position 7 2
Lion dies of hunger at position 3 5
Zebra moves from 1 4 to empty 0 3
Zebra moves from 0 2 to 0 1 and is eaten by Lion
Lion moves from 8 3 to 8 4 and eats Zebra
Birth of a Zebra at1 4

In order to get results identical to the expected output, you will need to carry out each step of the simulation in the order shown below (refer to the Background section for more details):
For each step, you should proceed row by row (top to bottom), and for each row proceed from left to right (so the first cell to be considered will be that in row=0 and column=0, and the last cell to be considered will be in row=*L*-1, column=*L*-1).

1) Reset the *has_moved* attribute to False for all animals in the grid
2) Execute the *time_passes*() function on all animals in the grid
3) Remove animals that died (as determined by the *die_of_old_age* and *die_of_hunger* methods)
4) Move each animal alive that has not moved already (*has_moved* == False) to a random neighboring cell (generated by the *random_neighbor* function). Set the *has_moved* attribute to True (this will prevent an animal from moving more than once per turn). If the move results in one animal eating another, perform the necessary adjustments to the grid. Remember that when an animal eats another, its *time_since_last_meal* attribute is reset to 0.
5) Generate births for the animals that have reached the right age.

Question 9. (15 points) [5-10 lines of code] Modify the *run_whole_simulation()* function so that it generates exactly the figure shown here. The image should not be shown to screen, but should instead be saved in a file whose name is provided as argument to the method.



Important notes: The simulation involves animals making random moves to neighboring cells, and babies being placed at randomly chosen neighboring positions from their parent.
In the Python's random module, there is a function that we will use for that <u>after your program is finished and debugged</u>:
random.choice(my_choices), where my_choices is a list of objects, from which one is randomly selected.

However this randomness means that different runs of the same program will probably produce different results. This makes debugging difficult, and makes it hard for me to give you what the expected output. So, in order to alleviate this difficulty, you will use for development and testing a *non-random* selection function, which always selects the top-most, left-most element in the list. This function is already written for you as *my_random_choice( ).*

Super important: For your submission, use the original non-random my_random_choice() function.

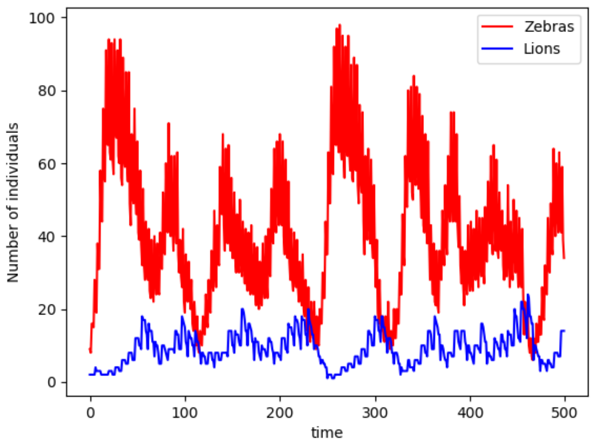
For fun, not graded, nothing to submit.
When you are happy with your program and are ready to switch to actual random moves, comment out the line "return min(choices, key=getKey)" of the *my_random_choice()* function, and uncomment the line located below it.

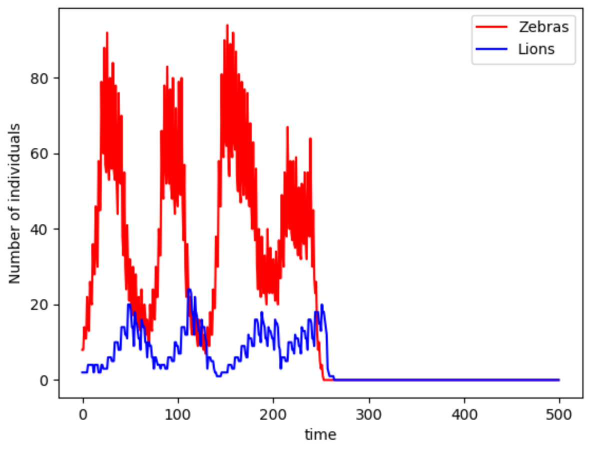
Now, you will get a different result every time you run your simulation. Change the duration of the simulation to 500 steps, and look at the image file that gets generated. You'll see different kinds of things happening. Most of the times, the population of zebras will oscillate widely (approximately from 5 to 90), over time periods of 50-100 months. This will be accompanied by similar oscillation of the lion population (from 1 to 20). Notice how the peaks of zebra population is followed by an increase in lion population, which causes a rapid drop in zebra population, which eventually causes a drop in lion population. And the cycle starts over! Until, by bad luck, either

- All the lions die, which can happen more or less quickly, and then the zebras living happy
- All the zebras die or are eaten, and then the lions die of hunger shortly thereafter

At the end of this document, I show some of the results I've obtained. You will not get exactly the same thing, but you should find similar kinds of results.

Super important, again: For your submission, use the original non-random *my_random_choice()* function. If you've switched to the random version for this section, make sure to switch back before submitting!


Zebras
Lions
Number of individuals
time
0
20
40
60
80
100
0
100
200
300
400
500


Zebras
Lions
Number of individuals
time
0
20
40
60
80
0
100
200
300
400
500