Agent-Based simulations and modeling applications in studying human social interactions

1. Introduction

Social sciences received a long-needed upgrade from a new form of computational methods: SIMULATIONS. Unlike classical methods of modelling with differential equations this can actually introduce in the environment a factor of interactions (which is, as expected, very important in this topic). One very interesting approach in this subfield of ‘Social simulation’ is Agent-based simulation and modelling. This aims to create a model for a very complex system (a human society or a subgroup of it) by implementing and highlighting a set of rules. We will properly define the problem, describe the proposed solution, present the platform created for this project, the implementation and the dynamics of the system (that should hopefully acquire some form of equilibrium or entropy.

2. Problem definition and specification

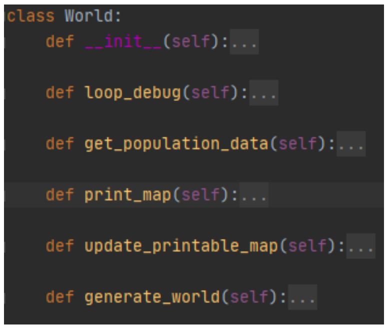
We want to illustrate a simplified concept of survival for early human societies. The idea is to create an environment where multiple agents (humans) are able to survive for multiple generations based on their perception and rule-based actions. In order to confirm this achievement and analyze any emergent properties of the system, we will also need debug information or a form of visualization.

3. Solution description

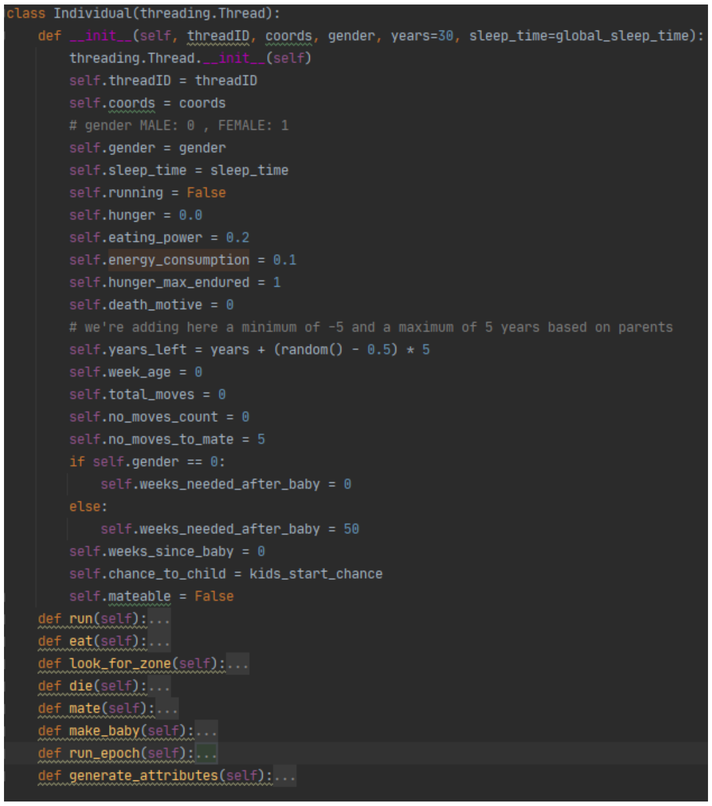
To use agent-based system there is a need for at least one type of agent. We will have two main categories of agents: Area and Individual. The area is an agent that represents one portion of the environment (as generated like a grid map), the individual represents one human.
We will need to first generate the environment that will contains resources used by the humans. Then design the way each area has its resources used and regenerated. The rules for each area make the resource it has available only to humans in that certain area, and it will regenerate its resources once human are no longer present (although, this is obviously not a precise representation of reality).
For the humans agent we will need a set of attributes based on which each of them will have a drive for eating and for reproduction which will translate into survival of the species.

4. The Multi-Agent System

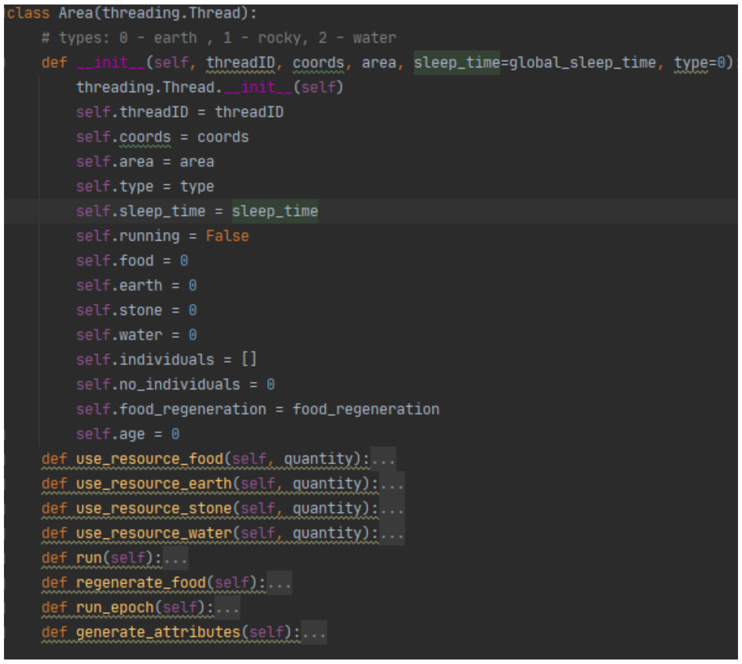
The structure of the code was modelled in 3 components and global data. The main component (WORLD) generated the other two components (Individual and Area).



```
class World:
    def __init__(self):...

    def loop_debug(self):...

    def get_population_data(self):...

    def print_map(self):...

    def update_printable_map(self):...

    def generate_world(self):...
```



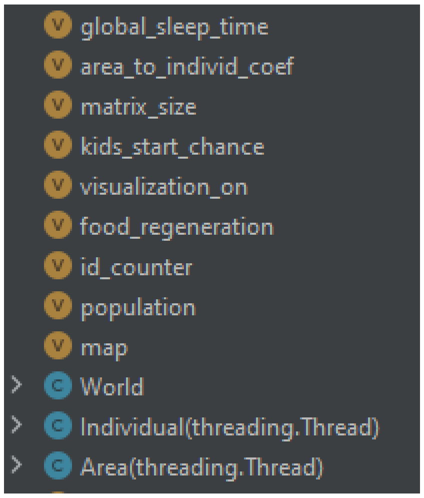
```
class Individual(threading.Thread):
    def __init__(self, threadID, coords, gender, years=30, sleep_time=global_sleep_time):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.coords = coords
        # gender MALE: 0 , FEMALE: 1
        self.gender = gender
        self.sleep_time = sleep_time
        self.running = False
        self.hunger = 0.0
        self.eating_power = 0.2
        self.energy_consumption = 0.1
        self.hunger_max_endured = 1
        self.death_motive = 0
        # we're adding here a minimum of -5 and a maximum of 5 years based on parents
        self.years_left = years + (random() - 0.5) * 5
        self.week_age = 0
        self.total_moves = 0
        self.no_moves_count = 0
        self.no_moves_to_mate = 5
        if self.gender == 0:
            self.weeks_needed_after_baby = 0
        else:
            self.weeks_needed_after_baby = 50
        self.weeks_since_baby = 0
        self.chance_to_child = kids_start_chance
        self.mateable = False
    def run(self):...
    def eat(self):...
    def look_for_zone(self):...
    def die(self):...
    def mate(self):...
    def make_baby(self):...
    def run_epoch(self):...
    def generate_attributes(self):...
```



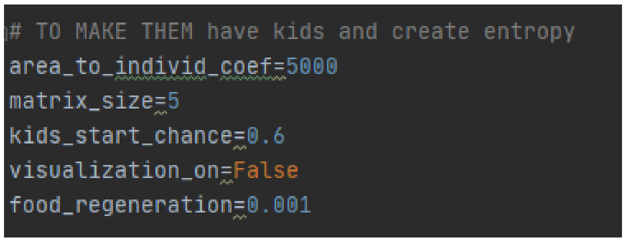
```
class Area(threading.Thread):
    # types: 0 - earth , 1 - rocky, 2 - water
    def __init__(self, threadID, coords, area, sleep_time=global_sleep_time, type=0)
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.coords = coords
        self.area = area
        self.type = type
        self.sleep_time = sleep_time
        self.running = False
        self.food = 0
        self.earth = 0
        self.stone = 0
        self.water = 0
        self.individuals = []
        self.no_individuals = 0
        self.food_regeneration = food_regeneration
        self.age = 0
    def use_resource_food(self, quantity):...
    def use_resource_earth(self, quantity):...
    def use_resource_stone(self, quantity):...
    def use_resource_water(self, quantity):...
    def run(self):...
    def regenerate_food(self):...
    def run_epoch(self):...
    def generate_attributes(self):...
```

As can be seen in the image on the right the two agents have been implemented as Threads that have access globally to the other individuals (population) and to areas (map) based on their location. This represents the agent's PERCEPTION and the main kind of communication.
We also defined a list of other parameters that represent the most important tweaking arguments in order to create one outcome or another. We have as couple of incremental examples that will five main parameters in order to achieve the following scenarios:

- Making the agents die of old age
- Making the agents die of hunger
- Making them have unlimited children
- Making the environment regeneration and reproduction obtain entropy



```
V global_sleep_time
V area_to_individ_coef
V matrix_size
V kids_start_chance
V visualization_on
V food_regeneration
V id_counter
V population
V map
> C World
> C Individual(threading.Thread)
> C Area(threading.Thread)
```

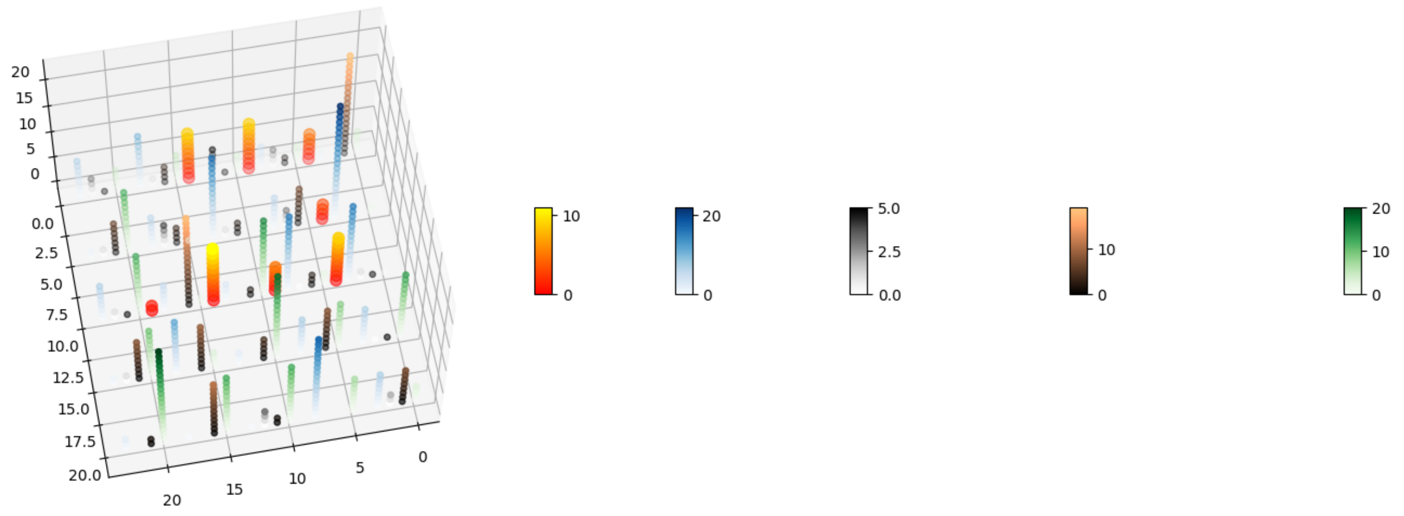


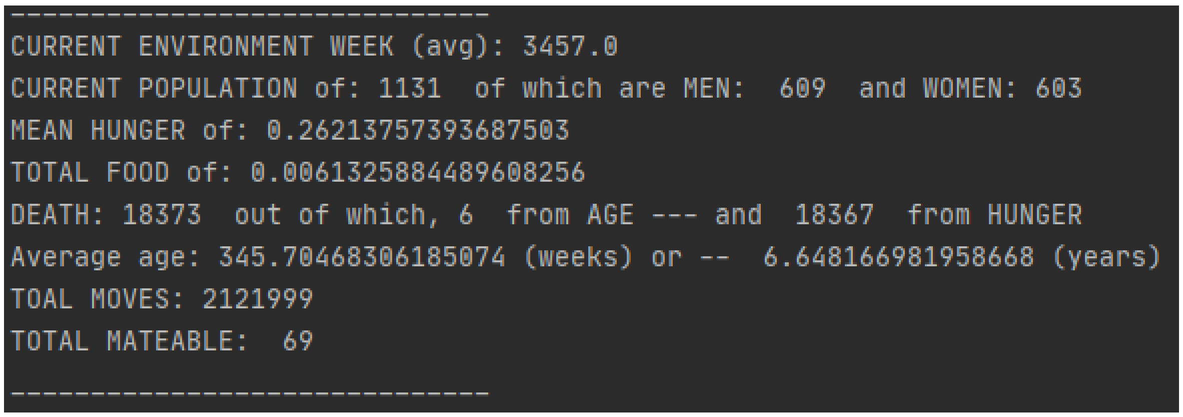
```
# TO MAKE THEM have kids and create entropy
area_to_individ_coef=5000
matrix_size=5
kids_start_chance=0.6
visualization_on=False
food_regeneration=0.001
```

The main dynamics of the systems are based on the drives on the humans and their ACTION capabilities. The system is based on a set of rules that create the GOAL for the humans.

- Each iteration (week) the human agent will grow hungrier depending on how much energy he consumes. If he gets hungry, he tries to eat from his area (the precepted environment), or goes searching in a nearby area hoping to find food, otherwise it will die of hunger. If it doesn't die from hunger, disease might get him and we modelled this using an approximate hereditary age (or life expectancy) with a slight variation on each generation of new children.
- Each iteration the agent will also update its mating status based on whether he is stable enough (he hasn't moved searching for food in a while) and enough time has passed since the last baby was born (in case the individual is a woman). If a random chance happens with a nearby individual who is also open to mating, there could be a new child born that inherits some of the changes in his/her parents attributes. The older they get the smaller the chance to have a baby though.
- Each iteration the area agents will have their resources used and if no humans are around the area, then the resources will regenerate.

To visualize the procedure, we have two methods: graphical visualization and debug information. The graphical visualization will print the grid map (using 3d plotting through matplotlib) on which we'll have each resource (the green one represents food – it's the most important) and the population in each area (the red-yellow). The bigger the column the bigger the quantity of resources or population. The second visualization is based on simple statistics in the debug menu updated each second.




```
------------------------------
CURRENT ENVIRONMENT WEEK (avg): 3457.0
CURRENT POPULATION of: 1131  of which are MEN:  609  and WOMEN: 603
MEAN HUNGER of: 0.2621375739368750З
TOTAL FOOD of: 0.0061325884489608256
DEATH: 18373  out of which, 6  from AGE --- and  18367  from HUNGER
Average age: 345.70468306185074 (weeks) or --  6.648166981958668 (years)
TOAL MOVES: 2121999
TOTAL MATEABLE:  69

------------------------------
```