GTL S43b - Design a (small) Fuzzy Logic Tool

The aim of this project is to implement a simple python program able to handle fuzzy logic rules. In short, the aim of this work is to automatize what we handy solved during the lecture.

This implementation must be developed from scratch in python and should be integrated into the skeleton code proposed. It should follow the 3 steps of a classical fuzzy logic engine, which are reminded in the Figure 1:

- The translation of the data of the case to analyze into the fuzzy sets (fuzzycation)
- The inference of expert rules by the inference engine
- The translation of the fuzzy results of this inference into float numbers (de-fuzzycation)

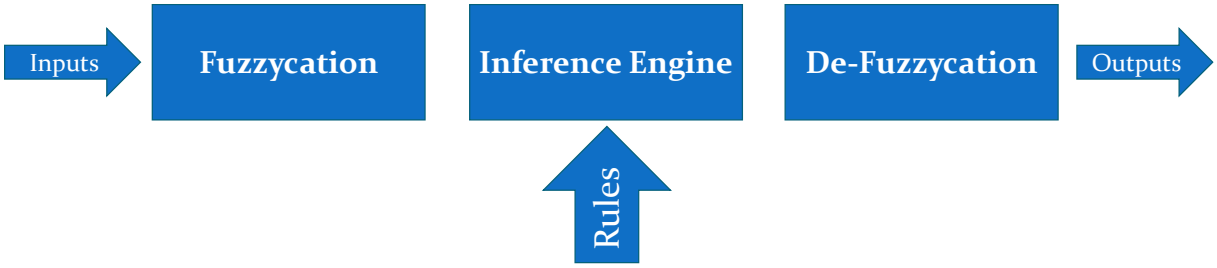

Figure 1: Classical architecture of a Fuzzy Logic engine

This work could be carried out by team of **max** 2 Students. Your program and report must be uploaded in SAVOIR in the dedicated section before **25th September (18:00)**. To complete this project, you have to use Python programming language with some classical libs (*matplotlib*, *numpy*). All the code managing the world where your robot is evolving is already coded and don't have to be modified (strongly forbidden). Your role is then to develop the fittest and smartest robot. Of course, since the goal is to design and develop a Fuzzy Logic engine, don't use lib proposing already coded fuzzy engines such as *Scikit-fuzzy*...

The goal is to build several Fuzzy Logic engines in order to support the decision making of a virtual robot which should evolve in a dangerous world. This work is partially based on the article written by *Utku Kose*, available on SAVOIR.

Robot – Characteristics and decision to take

The robot has 3 main characteristics he can directly and precisely access:

- Its **energy**, an integer from 0 to 100. This one limits the distance it can explore since moving one step consumes 10 energy units. By staying stationary, the robot recovers its energy (20% of its current energy level) without exceeding 100 units.
- Its **structure integrity**, an integer from 0 to 100. When the simulation starts this value is equal to 100, but when this one is reduced to 0, the robot explodes: the game is over.
- Its **ammo** (ammunition), an integer from 0 to 100. When the robot fire on an enemy 10 ammo units are consumed. It is possible to reload 20 ammo units by running on a cell containing an ammo supply (when consumed, it disappears).

These three characteristics can be translated into fuzzy sets, by simplifying the ones expressed by *Utku Kose* in his article. These fuzzy descriptions are plotted in Figure 2. These ones can be adapted in your

program regarding strategies you want to deploy. In that case, please justify and explain these modifications on your report.

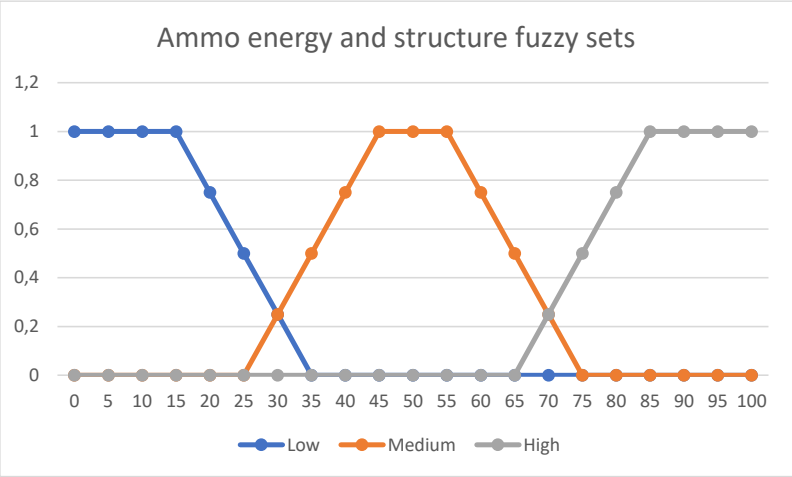

Figure 2 : Fuzzy sets definition for ammo, structure and energy parameters

In addition to these characteristics, the robot requests from its radar the **distance** to points of interest that can be: the closest enemy or the closest ammo supply. Since the quality and the precision of the radar are both poor and uncertain, the results given by this one is expressed as two fuzzy sets (belonging to domain [0; 1]), ie. the **distance** expressed with the fuzzy values: *Far*, *Medium* and *Close*

In the given python program, the manner to call the radar follows this python syntax in the code of your robot: `call_radar(target:str) -> list[]`. Where the input parameter `target` can take two values: “enemy” or “ammo”. The returned list contains tuples for the selected target (“enemy” or “ammo”) containing the fuzzy values for each output fuzzy values.

An enemy detected by the radar with a membership 0% of “far” fuzzy value is considered at range of the gun of the robot. The gun hit is certain and the enemy managed by computer doesn’t move (as opposed to robots controlled by the other students which can move).

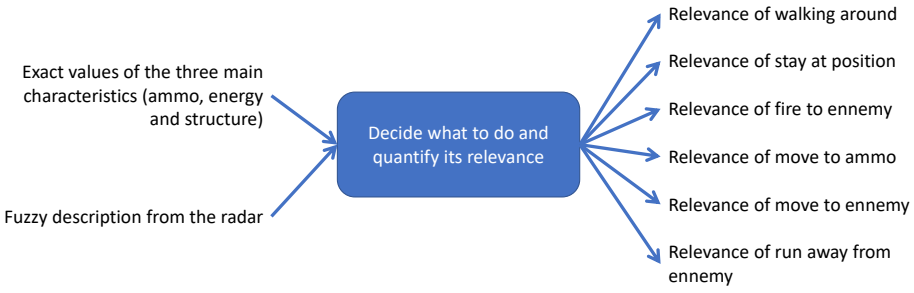

Figure 3: Inputs and outputs for the robot’s brain to design and code

In this work, the decision, and so the output is managed differently than the approach proposed in the article. Since you only have to focus on the “brain” (decision maker) of the robot, your program should calculate and return to the actuator of the robot (as a list), the relevance (expressed by a percentage from 0 to 100) of each possible decision (as illustrated in Figure 3). The actuator of the robot will select the one having the highest relevance and apply this decision accordingly. If the return of the brain is composed only by no relevance, the robot will stay at its position during this turn. And if there are more than one decision having the highest relevance, the system will take one randomly.

Regarding the actions performed by the robot's actuator:

- If it has to **move**: all moves consume 20 of energy and 2 steps will be performed (with Manhattan displacements).
- In particular, if it has to **get away**, it will go in the opposite direction to the one detected by the radar. If the robot hits the limit of the world, no effect is applied on its structure but the move is lost (and the energy as well).
- In the particular case of **walking away**, the robot will try to move to the center of the map.
- If it has to **stay**, the robot earns 20% of its current energy level (in the limit of 100 units maximum).

Work to do

The work expected follows several steps:

- **Define your strategy, design** and write (in natural language) the **rules** accordingly. You have to use only fuzzy logic approach to make your decision from the input parameters (the exact and fuzzy ones) to the evaluation of the relevance of each action the robot can do. Select as well the operators you want to use for defuzzycation and aggregation.
- **Code** in Python all steps required to apply your rules into one or several fuzzy logic engines. Don't hesitate to use `numpy` or `matplotlib`.
- To check the relevance of your strategy and its implementation, use the `try` method, to identify the behavior of your robot on different situations you defined and tune, if necessary, the rules or the selected operators.
- **Validate** your fuzzy logic approach by plotting the graph showing the effect of the different inputs on the final decision
- When the game engine is complete, you will be able to test the final behavior of your robot in the world composed by only computer managed robots, then with robots designed by your classmates.

Tips: To ease the development of your tool, you should consider this problem as a discrete one: consequently, I suggest you to split the membership functions and each intermediate result (aggregation, max, min…) into a **set of points**: the more points you have, the more precise will be your system and its results. Keep enough points to have precise enough results.