AML-Assignment#4

(Artificial Life (PyLifeSimbot))

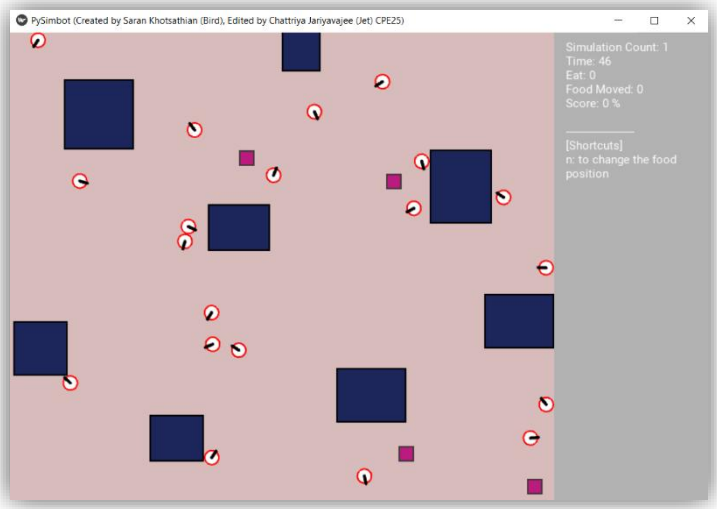
Introduction:

Artificial Life (often abbreviated ALife or A-Life) is a field of study wherein researchers examine systems related to natural life, its processes, and its evolution, through the use of simulations with computer models, robotics, and biochemistry. … from https://en.wikipedia.org/wiki/Artificial_life

Now, after you have learned something about using GA which is used for creating fuzzy rules for you. This assignment is about to extend the algorithm a little bit. Since GA emulates the process of evolution which could be considered as a natural process of many living things. Instead of looking the process by fixed time of many generations, this time, all living things will be live their lives until they die. One who lives longer would have more chances to reproduce or spread some of its DNA into the population. Once there is someone to die, the nature is called upon to create a new individual from the genetic operations by mating a couple from the remaining population. When the time goes on, the improved ones survive and live their lives, finally become the dominated ones.

To create an Artificial Life, these are the steps ..

- At start, a population of robot individuals is generated. (In this case, population is set to be 20). As usual, their chromosomes are randomly formed. To stay alive, all robots must have their energy (they were born with 300). The energy is not only decreased every time, but also consumed more when the robot is not in its well status. For example, if the robot hits something, the energy is spent faster. Other usual statuses are lazy, headache, and etc.
- In the environment, all robots are put into the test. The energy is the key. To live longer, the robots have to preserve their energy and (more importantly) to find food. When food is eaten by a robot, that robot has more additional energy to life.
- Each robot is able to see others. Therefore, other robots are considered as moving obstacles. There are also static obstacles, which are not moving at all.
- Food are limited, but not empty. If one food was taken, another food is formed in another place randomly located.
- When one robot dies, the process to generate the new robot is taking place for replacing the dead. A couple of robots are randomly picked from the remaining ones in the population. Then, the crossover is called to reproduce a new robot from its parents. In addition, mutation is also applied rarely. To introduce new genes to the population, generate new robots are rarely presented.
- The simulation time is counting continue until reaching the maximum tick (100000).
- Learning curve is plotted from amount of dead robots in different time intervals.
- At the end, the robot with highest energy is selected to be the final answer.



```python
if __name__ == '__main__':

    app = PySimbotApp(robot_cls=StupidRobot,
                      num_robots=20,
                      num_objectives=4,
                      theme='default',
                      simulation_forever=False,
                      max_tick=100000,
                      interval=1/1000.0,
                      food_move_after_eat=True,
                      robot_see_each_other=True,
                      # map="no_wall",
                      customfn_before_simulation=before_simulation,
                      customfn_after_simulation=after_simulation)
    app.run()
```

```python
            if int(answerMove) == 0 and int(answerTurn) == 0:
                self.lazy_count += 1

            if int(answerMove) < 0 or abs(answerTurn) > 30:
                self.headache_count += 1

            self.turn(answerTurn)
            self.move(answerMove)

            # every step the robot lost its energy
            self.energy -= 1

            # if the robot hit, it also lost energy
            if self.just_hit :
                # self.energy -= Y
                pass

            # if the robot eat food, it gets some energy back
            if self.just_eat :
                # self.energy += X
                pass

            if self.energy < 0 :
                # die and find a new robot
                print (self._sm.iteration)

                # record that there is a dead one

                # generate new robot by usubg genetic operations
                temp = StupidRobot()
                temp = self.generate_new_robot()
                self.RULES = deepcopy(temp.RULES)

                # give this new born some energy
                self.energy += 300
                pass
```

- The population of 20 robots with 4 available food locations and 8 stable obstacles (20 robots considered as moving obstacles).

```python
    def generate_new_robot(self):
        simbot = self._sm
        num_robots = len(simbot.robots)

        def select() -> StupidRobot:
            index = random.randrange(num_robots)
            return simbot.robots[index]

        select1 = select()   # design the way for selection by yourself
        select2 = select()   # design the way for selection by yourself

        while select1 == select2:
            select2 = select()

        temp = StupidRobot()

        # Doing crossover
        #     using next_gen_robots for temporary keep the offsprings, later they will be copy
        #     to the robots

        # Doing mutation
        #     generally scan for all next_gen_robots we have created, and with very low
        #     propability, change one byte to a new random value.

        # Making a new random robot

        return temp
```