Search Based Software Engineering

Project - Evolutionary Game

2019-07-02

Deadline: ~~2019-08-01 23:59~~ 2019-08-07 23:59

Submit to: andre.karge@uni-weimar.de

Submission details: Submit your breeder.py class file
No Jupyter Notebooks!
Compress your file (.zip or .tar.gz or .rar)

Name your compressed file: <lastname>_<firstname>_<matrikelnummer>-ex<exercise-number>(.tar.gz or .rar or .zip)
example: norris_chuck_123456-project.tar.gz

Groups: submit your solved assignment **individually (no groups)** - only graded for digital engineering

Language: Python 3

Hint: The jupyter notebook of the lab class can be found at link
The game repository can be found at link

Problem Description
You have to implement your own genetic algorithm for the game EGame. EGame is an evolutionary game with different populations fighting the game of life in order to survive. You are in charge of one population and control its initialization and breeding process.

Each population consist of individuals with special traits (perception, desires and abilities). On the playground there are different items (food, poison, health potions and corpses of dead individuals) which can be consumed by the individuals. Each individual has a health value which is constantly reduced each frame. To increase health, they can eat food, health potions or corpses.

Food simply increases health by a certain amount. If they eat poison, they get poisoned and their health will be reduced faster. Health potions cure the poison and increase health by a small amount. When an individual dies, a corpse is generated which can be eaten by all individuals who are alive. Corpses act like food but also contain the poison value of the deceased individual (eg. when the dead individual had eaten 5 poison, the individual who eats its corpse will be healed by a fixed amount but also will get a fraction of that poison, too).

Next to the populations, predators will be generated from time to time who try to hunt individuals and eat their corpses.
Individuals can seek and attack other individuals (of the opponent population) but they cannot attack predators.

To be able to seek for certain things like items or other individuals, all individuals have their own perception. Perception is a radius around individuals in which they can see their environment. It is limited by a defined amount of pixels. Each item class perception and individual perception is a fraction of the individuals general perception. All perception values have to sum up to 1. That means, if an individual has a food perception of 1, it uses all of its perception to find food and does not perceive any other element. The perceptions are: food, poison, health potions, opponents, predators.

Additionally, individuals have desires for different things. These desires control how much they are attracted or repelled of certain things. Again, the sum of all desires has to be 1. The desires are: seek food, dodge poison, seek health potion, seek opponents, seek corpses and dodge predators. That means, if an individual has a desire of 1 to seek food, it will not try to avoid poison or predators or seek health potions.

Furthermore, individuals have different abilities. Like the other traits, the values for all abilities have to sum up to 1. Abilities are: increased armor, increased maximum speed, increased strength, poison resistance and toxicity. Increased armor will reduce the damage if an individual is attacked. Increased maximum speed enables individuals to move faster. Increased strength enables individuals to deal more damage to opponents. Poison resistance reduces the constant damage an individual takes by poison. Toxicity lets attackers take damage when they attack the individual.

Exercise 1. (0 points)
Clone the github repository `https://github.com/digital-bauhaus/EGame` and follow the install instructions.

Exercise 2. (15 points)
Write your own Breeder class (an example can be found at genetic_algorithm/breeder.py) with this template:

```
class Breeder:
        def __init__(self, parent):
                self.parent = parent
        def initialize_population(self, num_individuals, color):
                # your code here
                return population
        def breed(self, population):
                # your code here
                return updated_population
```

Exercise 3. (3 points)
Write a fitness function which assesses the fitness of an individual (you should figure out what makes up good and bad individuals).
Hint: You can access the dna by calling *get_dna()* on an individual-object. You should also use the individual statistic to calculate the fitness (can be found at individual.statistic - have a look at the *Statistic* class).

Exercise 4. (2 points)
Write a docstring for your Breeder class and explain, how you define good and bad individuals (fitness assessment) and how you perform initialization, selection, crossover, mutation. Explain what you did there and why.