50.021 -AI

Alex

Week 10: Genetic Algorithm

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

Genetic Algorithms – the game – and a simple homework - due to 31st of July

The code implements a game – big thanks to Natalie for coding an initially weird idea. The game works as follows. A group of students are trapped on a haunted island somewhere deep in the pacific ocean, and every generation sends their students in every generation to flee from the island. They will need to go on a long voyage over the ocean. Unfortunately there are lots of disasters that can end their lifes during their journey (want a scholarship on this island :P?). They take the voyage in high tech boats. But it takes skills to survive the disaster even when the boats are well-equipped.

The skills (attributes) are modelled in 7 categories and each category skill-point range from 0 to 3. *Strength, Agility, Intelligence, Charm, Vitality, Stamina, Spirit* (see `stats.py`). The total skill-points you can have is 15.

The problem is to find a combination of skills for *at least one student* to survive 50 days on the ocean. Every day (not only monday, tuesday and thursday's at 4pm), a disaster can randomly appear and the student has to fight the disaster (see `disasters.py`). Each disaster has the same 7 categories of skills.

The fight outcome depends on the skills of the student, the skills of the disaster and a portion of luck that is inherent in every fight. In a fight, each skill category of a disaster is compared with the skill level of the student (see gameplay.py, fight method). The comparison is used to derive a probability that the student wins in this particular skill category. A student survives if he wins at least 4 out of the 7 skill categories.

We will find the suitable skill combination using genetic algorithm, where the goal is to develop a skill set for the students. As the disas-

ters come to life in a random fashion, and the outcome of the fight is probabilistic with respect to the skill-point in each skill cateogry, plus that we have a discrete space of skill set values, it is hard to model it as a gradient-based optimization problem and genetic algorithms are a good choice to use here.

Follow these steps to begin familiarizing with the code,

1. Open gameplay.py and run it. The main function in gameplay.py will create a player with random skill attributes, that is each of the 7 skill categories will get either 0,1,2, or 3 skill points, and sum of the points across the 7 attributes is less than or equal to 15. Afterwards, it will show how many days this student survives before being killed by one of the 'disasters'.

2. There are two game settings, *probability_game* = true/false and *use_multinomial* = true/false. They are just different types of games. If *probability_game* is True, then, the winner of the fight between students and disaster is determined probabilistically. For example, if student's strength is 3, and if disaster's strength is 1, student has 75% chance of winning the fight. Otherwise if *probability_game* is False, then the student will win as long as it has an attribute that is higher than the disaster's. In this case, since student's strength is 3, it will win the disaster in this skill category. *use_multinomial* is the distribution used in creating the disasters. If it is True, we use a multinomial distribution, if False, we pick the disasters randomly.

Your tasks:

- There are three methods you need to implement in total. In `genetic_algorithm.py` implement `reproduce`. The method `reproduce` will make use of three already-implemented routines: `clone, mutation, crossover`, but they depend on `mutate` and `crossover` in `player.py`. Implement `mutate` and `crossover` in `player.py`.

- bonus: add a new disaster with the sum of skill points not stronger than the existing monsters, distribute the creation probability so that this new monster is included, too.