**import random**

**class GeneticAlgorithm(object):**

**def \_\_init\_\_(self, genetics):**

self.genetics = genetics

pass

**def run(self):**

population = self.genetics.initial()

**while True:**

fits\_pops = [(self.genetics.fitness(ch), ch) for ch in population]

if self.genetics.check\_stop(fits\_pops): break

population = self.next(fits\_pops)

pass

return population

**def next(self, fits):**

parents\_generator = self.genetics.parents(fits)

size = len(fits)

nexts = []

**while len(nexts) < size:**

parents = next(parents\_generator)

cross = random.random() < self.genetics.probability\_crossover()

children = self.genetics.crossover(parents) if cross else parents

**for ch in children:**

mutate = random.random() < self.genetics.probability\_mutation()

nexts.append(self.genetics.mutation(ch) if mutate else ch)

pass

pass

return nexts[0:size]

pass

**class GeneticFunctions(object):**

**def probability\_crossover(self):**

r"""returns rate of occur crossover(0.0-1.0)"""

return 1.0

**def probability\_mutation(self):**

r"""returns rate of occur mutation(0.0-1.0)"""

return 0.0

**def initial(self):**

r"""returns list of initial population

"""

return []

**def fitness(self, chromosome):**

r"""returns domain fitness value of chromosome

"""

return len(chromosome)

**def check\_stop(self, fits\_populations):**

r"""stop run if returns True

- fits\_populations: list of (fitness\_value, chromosome)

"""

return False

**def parents(self, fits\_populations):**

r"""generator of selected parents

"""

gen = iter(sorted(fits\_populations))

**while True:**

f1, ch1 = next(gen)

f2, ch2 = next(gen)

yield (ch1, ch2)

pass

return

**def crossover(self, parents):**

r"""breed children

"""

return parents

**def mutation(self, chromosome):**

r"""mutate chromosome

"""

return chromosome

pass

**if \_\_name\_\_ == "\_\_main\_\_":**

"""

example: Mapped guess prepared Text

"""

**class GuessText(GeneticFunctions):**

**def \_\_init\_\_(self, target\_text,**

limit=200, size=400,

prob\_crossover=0.9, prob\_mutation=0.2):

self.target = self.text2chromo(target\_text)

self.counter = 0

self.limit = limit

self.size = size

self.prob\_crossover = prob\_crossover

self.prob\_mutation = prob\_mutation

pass

# GeneticFunctions interface impls

**def probability\_crossover(self):**

return self.prob\_crossover

**def probability\_mutation(self):**

return self.prob\_mutation

**def initial(self):**

return [self.random\_chromo() for j in range(self.size)]

**def fitness(self, chromo):**

# larger is better, matched == 0

return -sum(abs(c - t) for c, t in zip(chromo, self.target))

**def check\_stop(self, fits\_populations):**

self.counter += 1

**if self.counter % 10 == 0:**

best\_match = list(sorted(fits\_populations))[-1][1]

fits = [f for f, ch in fits\_populations]

best = max(fits)

worst = min(fits)

ave = sum(fits) / len(fits)

print("[G %3d] score=(%4d, %4d, %4d): %r" % (self.counter, best, ave, worst, self.chromo2text(best\_match)))

pass

return self.counter >= self.limit

**def parents(self, fits\_populations):**

while True:

father = self.tournament(fits\_populations)

mother = self.tournament(fits\_populations)

yield (father, mother)

pass

pass

**def crossover(self, parents):**

father, mother = parents

index1 = random.randint(1, len(self.target) - 2)

index2 = random.randint(1, len(self.target) - 2)

if index1 > index2: index1, index2 = index2, index1

child1 = father[:index1] + mother[index1:index2] + father[index2:]

child2 = mother[:index1] + father[index1:index2] + mother[index2:]

return (child1, child2)

**def mutation(self, chromosome):**

index = random.randint(0, len(self.target) - 1)

vary = random.randint(-5, 5)

mutated = list(chromosome)

mutated[index] += vary

return mutated

# internals

**def tournament(self, fits\_populations):**

alicef, alice = self.select\_random(fits\_populations)

bobf, bob = self.select\_random(fits\_populations)

return alice if alicef > bobf else bob

**def select\_random(self, fits\_populations):**

return fits\_populations[random.randint(0, len(fits\_populations)-1)]

**def text2chromo(self, text):**

return [ord(ch) for ch in text]

**def chromo2text(self, chromo):**

return "".join(chr(max(1, min(ch, 255))) for ch in chromo)

**def random\_chromo(self):**

return [random.randint(1, 255) for i in range(len(self.target))]

pass

GeneticAlgorithm(GuessText("Hello World!")).run()

pass