GENETIC ALGO

A CASE OF MACHINE LEARNING

Jean-Michel Torres
(torresjm@fr.ibm.com)

[nov 2017- nov 2024]

AGENDA

- Very simplified genetics
- Demo with TSP (traveller sales person)
- Hands-on : banana !
- Sources, refs

GENETICS CRASH COURSE:

Selection : mechanism increasing the probability and/or size of the offspring for the most successful individuals (the best adapted).

Heredity : the process by which children receive their parents' properties.

Mutation : a random modification of traits in a new individual.

EVOLUTION WITHOUT GENES

« According to Darwin ». Alain, French Philosopher, Sept 1908

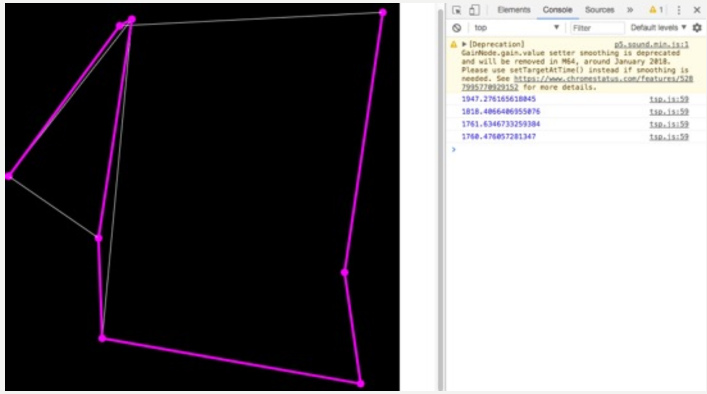
Michel Raymond, Agora des savoirs, 2013

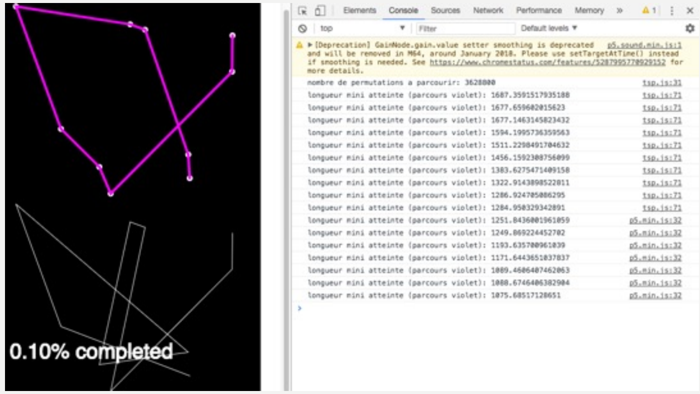
Travelling Sales Person (TSP)

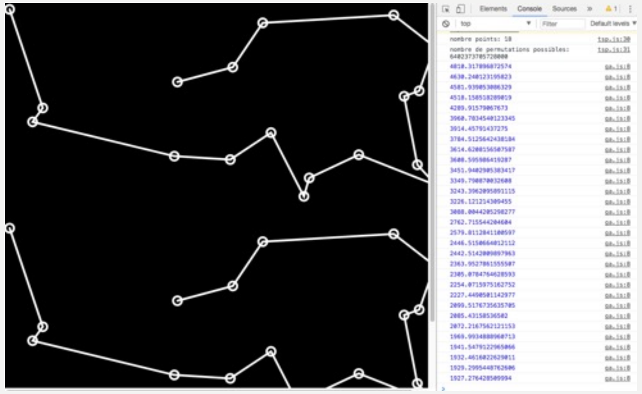
Find the shortest path joining N points :

Demo 3 scripts solving with differnet algos :

1. Try paths at random, measure their length, keep the shortest so far in memory…
   - Very long, no warranty for finding the shortest
2. Try all possible paths (lexicograpic)
   - Very very long
3. Genetic algo :
   - reasonably fast and qualitative, but no warranty for finding the shortest.







WIKIPEDIA : GENETIC ALGORITHM

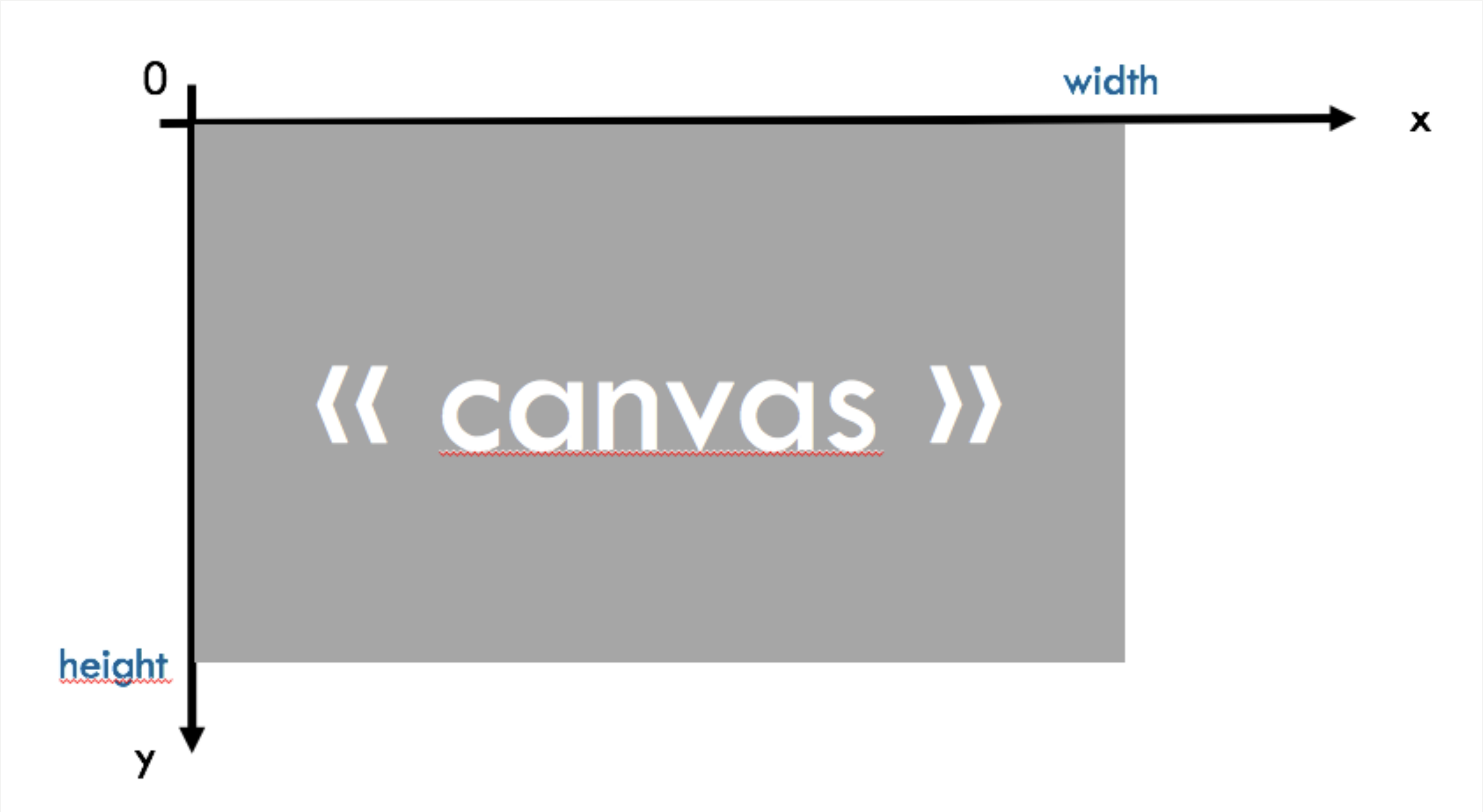
The 2006 NASA ST5 spacecraft antenna. This complicated shape was found by an evolutionary computer design program to create the best radiation pattern. It is known as an evolved antenna.

More deta

BANANA !!!

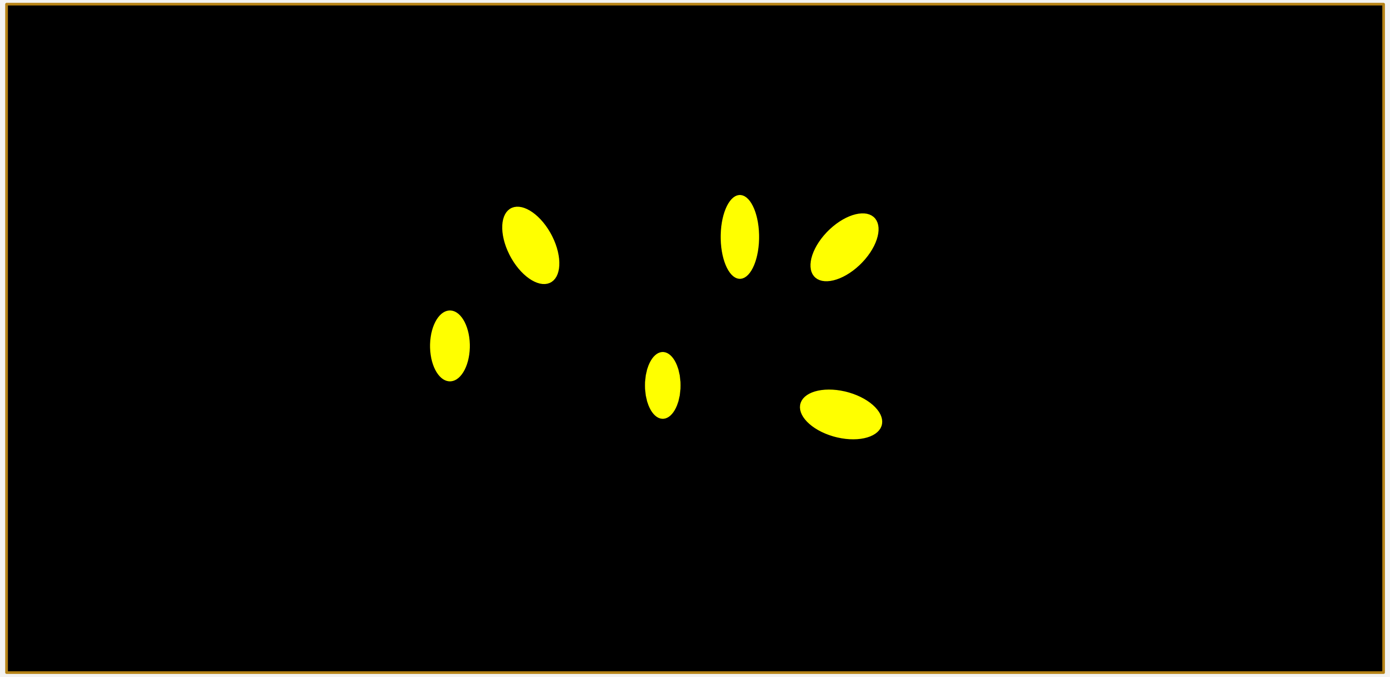
BEFORE WE START

Playground



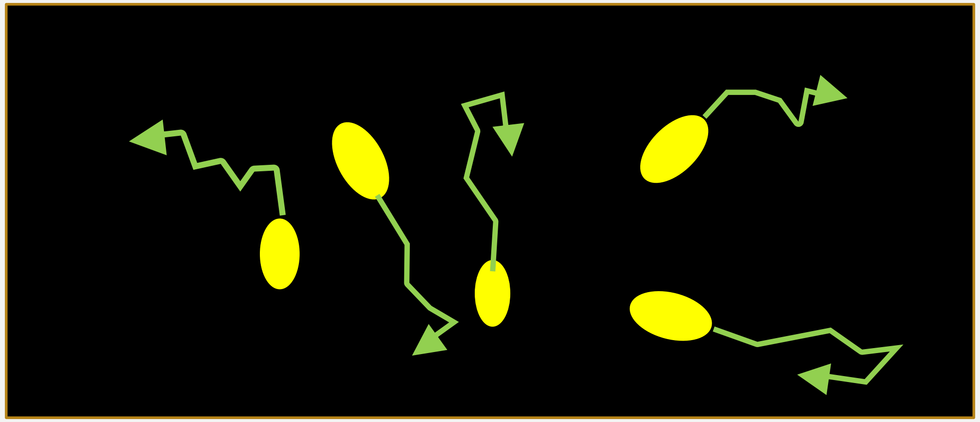
BEFORE WE START

Population = { Minions }



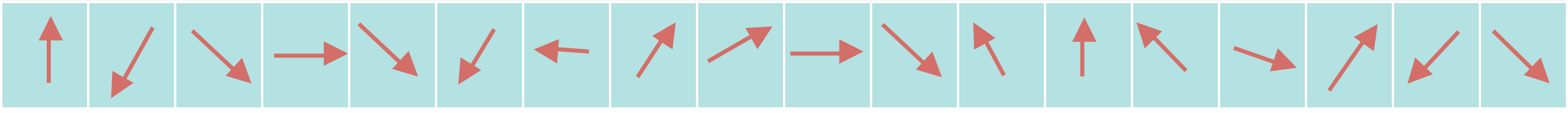
BEFORE WE START

Minions behaviors are defined by their moves during their lifetime



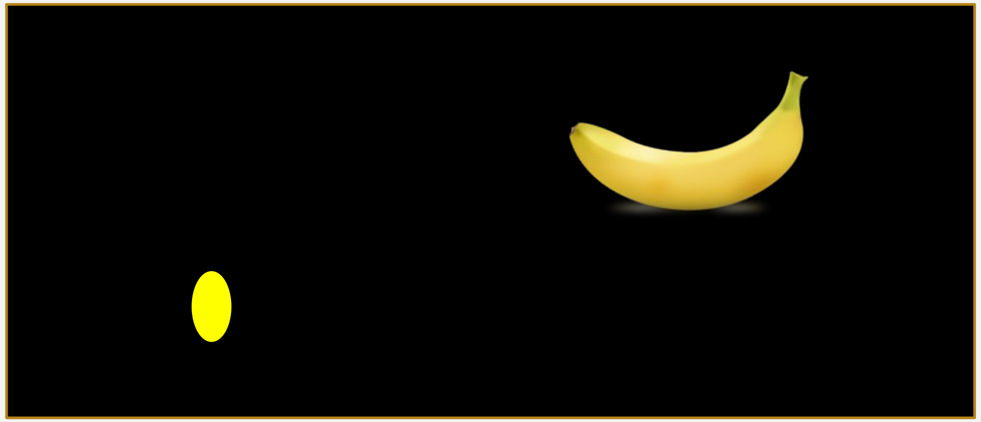
This will constitute their DNA :

a chronological array of moves from birth to death



BEFORE WE START

«Fitness» = shortest distance to the food at end of life.

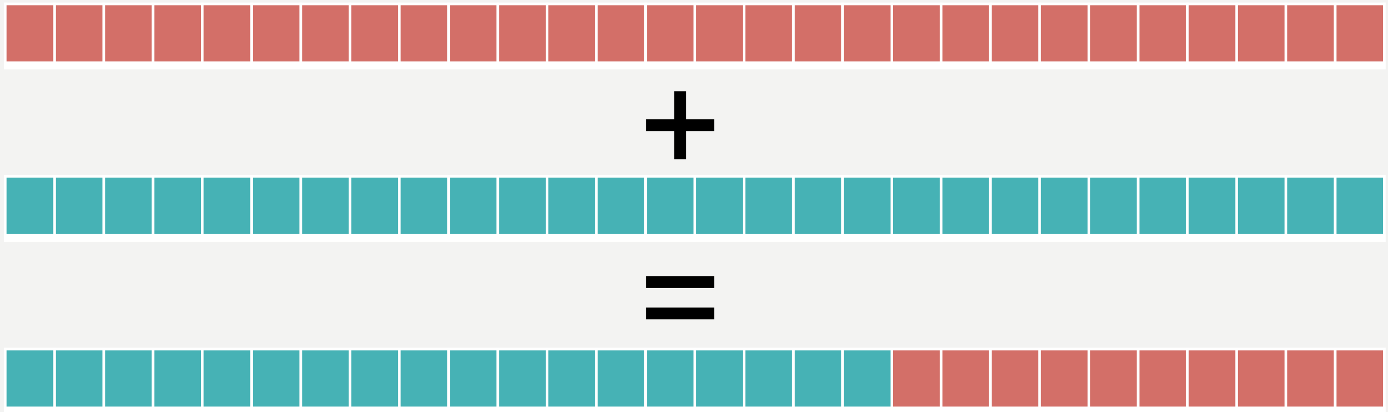


Minions with best fitness will get more chances to have offsprings: i.e. : there will be more copies of their genes in the pool that gives birth to next generation

BEFORE WE START

Crossover: method for generating a new Minion using genes from the pool by choosing and mixing :

- Select 2 genes at random
- Cut and merge at random point



Mutation: random modification of some values in the gene

STEP BY STEP ALGORITHM BUILDING

A. One minion moves up,
B. Two minions move towards random directions,
C. Many minions move in random directions,
D. Minions have DNA and move as directed by their (random) genes.
E. Compute Fitness (and fill DNA Pool), but not used yet for next generation.
F. Use DNA to create next generation.
G. Add target hit detection and make a large reward for that.
H. Now with mutations
I. Making life harder with an obstacle

A -« INFRASTRUCTURE » (PYGAME ENGINE)

```python
while running:
    win.fill(background_color)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_q:
                running = False

    pygame.draw.circle(win,yellow,(minion.pos.x, minion.pos.y), msize)

    minion.move()

    time.sleep(0.01)

    pygame.display.update()

pygame.quit()
```

A -« INFRASTRUCTURE » (UTILITY)

```python
# define a class Vec for managing vectors (adding, scaling, normalizing,... )
class Vec():
    ### vectors will be couples (x,y) (positions, velocity, acceleration)
    def __init__(self,v):
        self.x = v[0]
        self.y = v[1]

    def print(self):
        print(f" my coordinates are {self.x} and {self.y}")

    def vadd(self,other): # adding vectors
        self.x = self.x + other.x
        self.y = self.y + other.y

    def vtimes(self,s): # scaling vector
        self.x = self.x * s
        self.y = self.y * s

    def vnorm(self,ampl=1): # scaling to a length
        n = sqrt(self.x * self.x  + self.y * self.y) / ampl
        self.x = self.x / n
        self.y = self.y / n
```

A - DEFINE AND INSTANCIATE MINION

```python
# most used objects will be minions, so let's build the class :
class Minion(): # velocity (0,-1) by default, means the minion goes up
    def __init__(self,pos=(WIDTH/2,HEIGHT-50),vel=(0,-1),acc=(0,0)):
        self.pos = Vec(pos)
        self.vel = Vec(vel)
        self.acc = Vec(acc)

    def move(self):
        self.vel.vadd(self.acc)
        self.pos.vadd(self.vel)
        self.acc.vtimes(0)

# instantiate one minion
minion = Minion()
```

B- LET THE MINIONS CHOOSE THEIR DIRECTION

```
minion = Minion((WIDTH/2,HEIGHT-50), (random()-0.5,random()-0.5))
minion2 = Minion((WIDTH/2,HEIGHT-50), (random()-0.5,random()-0.5))
```

```
pygame.draw.circle(win,yellow,(minion2.pos.x, minion2.pos.y), msize)
pygame.draw.circle(win,yellow,(minion.pos.x, minion.pos.y), msize)


minion.move()
minion2.move()
```

C- BUILD A POPULATION OF MINIONS

```
class Population():
    def __init__(self, pop_size):
        self.pop_size = pop_size
        self.minions = []
        for i in range(self.pop_size):
            self.minions.append(Minion((WIDTH/2,HEIGHT-50), (random()-0.5,random()-0.5), (0,0)))

    def move(self):
        for m in self.minions:
            m.move()
```

```
pop = Population(40)
```

```
for i in range(pop.pop_size):
    pygame.draw.circle(win,yellow, (pop.minions[i].pos.x, pop.minions[i].pos.y), msize)

pop.move()
```

D- NOW USE GENES (NO HEREDITY YET)

```
lifespan = 400
```

```
def move(self,life_tick):
    for m in self.minions:
        m.move(life_tick)
```

← change in pop.move()

D- NOW USE GENES (NO HEREDITY YET)

```
class Minion():
    def __init__(self,pos=(WIDTH/2,HEIGHT-50),vel=(random()-0.5,random()-0.5),acc=(0,0)):
        self.pos = Vec(pos)
        self.vel = Vec(vel)
        self.vel.vnorm(1)
        self.acc = Vec(acc)
        self.dna = DNA()

    def move(self,life_tick):
        self.acc.vadd(self.dna.genes[life_tick])
        self.vel.vadd(self.acc)
        self.pos.vadd(self.vel)
        self.acc.vtimes(0)
```

← added at step 2 when vel was not more « go up »

← a minion creates it's genes when initialized

← here we use the gene value at index « life_tick » (iterating the array of genes)

```
life_count +=1
if life_count == lifespan:
    pop = Population(10)
    life_count = 0

for i in range(pop.pop_size):
    pygame.draw.circle(win,yellow, (pop.minions[i].pos.x, pop.minions[i].pos.y), msize)

pop.move(life_count)
```

← when lifespan is achieved, restart a new population

E- COMPUTE FITNESS (AND FILL DNA POOL):

```
def compute_fitness(self):
    d = self.pos.vdist(reward)
    self.fitness = maprange(d,0, WIDTH,1,0)
```

← added in Minion()

```
def evaluate(self):
    maxfit = 0
    for i in range(self.pop_size):
        self.minions[i].compute_fitness()
        if self.minions[i].fitness > maxfit:
            maxfit = self.minions[i].fitness

    self.dnapool = []
    for i in range(self.pop_size):
        qty = int(self.minions[i].fitness * 100)
        for j in range(qty):
            self.dnapool.append(self.minions[i])
```

← added in Population()

F- BUILD NEXT POP WITH DNA

← added in Population()

```
def select(self ) -> list:
    # this is where we build a next population on minions by choosing at random
    # 2 parents from the pull, making the crossover (select a cut point and build
    # a mixed dna to create a new minion in the new population
    newminions = []
    for i in range(self.pop_size):
        parentA = choice(self.dnapool)
        parentB = choice(self.dnapool)
        child_dna = parentA.dna.crossover(parentB.dna)
        newminions.append(Minion(child_dna,(WIDTH/2,HEIGHT-50), (0,0), (0,0)))
    return newminions
```

← added / changes in DNA()

```
class DNA():
    def __init__(self, dna) -> None:
        if dna == None:
            self.genes = []
            for i in range(lifespan):
                #print(i)
                self.genes.append(Vec((random()-0.5,random()-0.5)))
                self.genes[i].vnorm(0.3)
        else:
            self.genes = dna

    def crossover(self, partner):
        newdna = []
        midpoint = randint(0,lifespan)
```

G- ADD TARGET DETECTION & LARGE REWARD

```
def move(self,life_tick):
    # while target hat not been hit :
    if not self.eat_banana:
        self.acc.vadd(self.dna.genes[life_tick])
        self.vel.vadd(self.acc)
        self.vel.vnorm(1)
        self.pos.vadd(self.vel)
        self.acc.vtimes(0)

    # check if target his hit
    if self.pos.vdist(reward) < 20:
        self.eat_banana = True
```

← change .move() in Minion()

```
def compute_fitness(self):
    d = self.pos.vdist(reward)
    self.fitness = maprange(d,0, WIDTH,1,0)
    if self.eat_banana :
        self.fitness = self.fitness * 25
```

← change .compute_fitness() in Minion()

G- TARGET

```python
# this is the reward / objective
pygame.draw.circle(win,yellow, (reward.x, reward.y), 20)
pygame.draw.circle(win,background_color,(reward.x,reward.y-10),20)
pygame.draw.circle(win,green,(reward.x+20,reward.y),4)
```

H- ADD MUTATION

```
def mutation(self):
    for i in range(lifespan):
        if random() < 0.005 :
            self.genes[i] = Vec((random()-0.5,random()-0.5))
```

← add in DNA()

```
def select(self ) -> list:
    # this is where we build a next population on minions by choosing at random
    # 2 parents from the pull, making the crossover (select a cut point and build
    # a mixed dna to create a new minion in the new population
    newminions = []
    for i in range(self.pop_size):
        parentA = choice(self.dnapool)
        parentB = choice(self.dnapool)
        child_dna = parentA.dna.crossover(parentB.dna)
        child_dna.mutation();
        newminions.append(Minion(child_dna,(WIDTH/2,HEIGHT-50), (0,0), (0,0)))
    return newminions
```

← call mutation() from select in Population

I- ADD OBSTACLE

```python
obstacle = pygame.Rect(300,400,200,20)
```

```python
pygame.draw.rect(win,red,obstacle)
```

```python
# check if target is hit
if self.pos.vdist(reward) < 20:
    self.eat_banana = True
```

```python
# check if obstacle has been hit :
if obstacle.collidepoint(self.pos.x,self.pos.y):
    self.touched_obstacle = True
```

```python
def move(self,life_tick):
    # while red dot hat not been hit :
    if not self.eat_banana and not self.touched_obstacle:
        self.acc.vadd(self.dna.genes[life_tick])
        self.vel.vadd(self.acc)
        self.vel.vnorm(1)
        self.pos.vadd(self.vel)
        self.acc.vtimes(0)
```

THANK YOU FOR YOU ATTENTION !

SOURCES ET RESSOURCES

- Cro-Magnon toi-même, Michel Raymond éditions Points Sciences (et : http://www.dailymotion.com/video/x10s7n4
- The Nature of Code, The Coding Train, Daniel Schiffman