Algorithmes génétiques et intelligence artificielle

I – Introduction aux algorithmes génétiques:

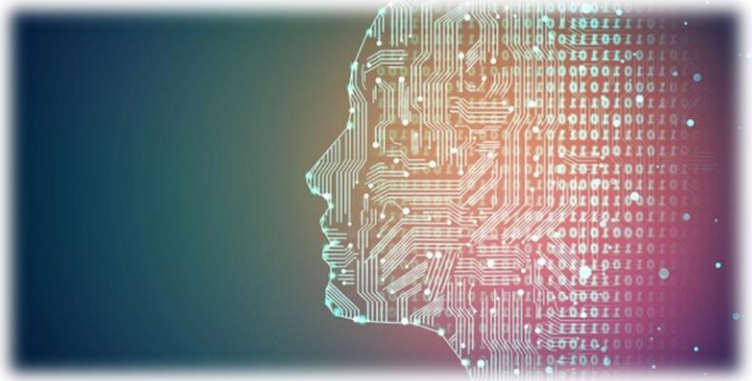
- A- Position du problème
- B- Notion et utilité d'un algorithme génétique

II- Convergence des algorithmes génétiques:

- A- La théorie des chaînes de Markov
- B- Convergence globale de l'algorithme

III- Modélisation informatique de la situation

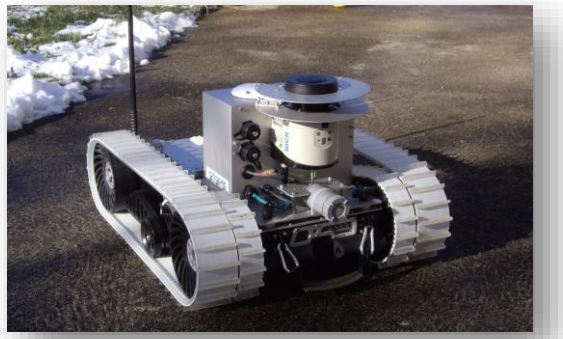
- A- Implémentation de l'algorithme
- B- Interprétation des résultats



I – Introduction aux algorithmes génétiques:

- A- Position du problème:

Robot d'exploration

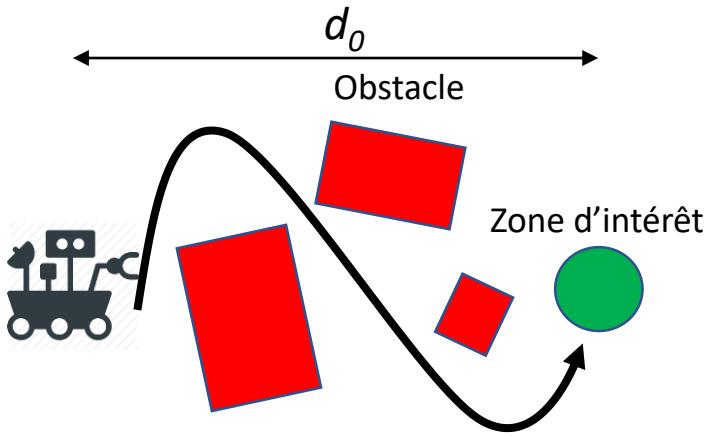


Caméléon (ECA)

Contraintes : Obstacles, énergie…

Exploration autonome de milieux inaccessibles à l'être humain

Objectif : Développer un automate/robot intelligent capable de sonder un environnement sans l'intervention de l'opérateur



Il s'agit donc de résoudre le problème d'optimisation suivant:

$$min\{d(x); x \in E\}$$

Où E désigne l'ensemble de trajectoires possibles pour le robot

d(x) est la distance séparant le robot du but après le parcours de la trajectoire x

- B- Notion et utilité d'un algorithme génétique:

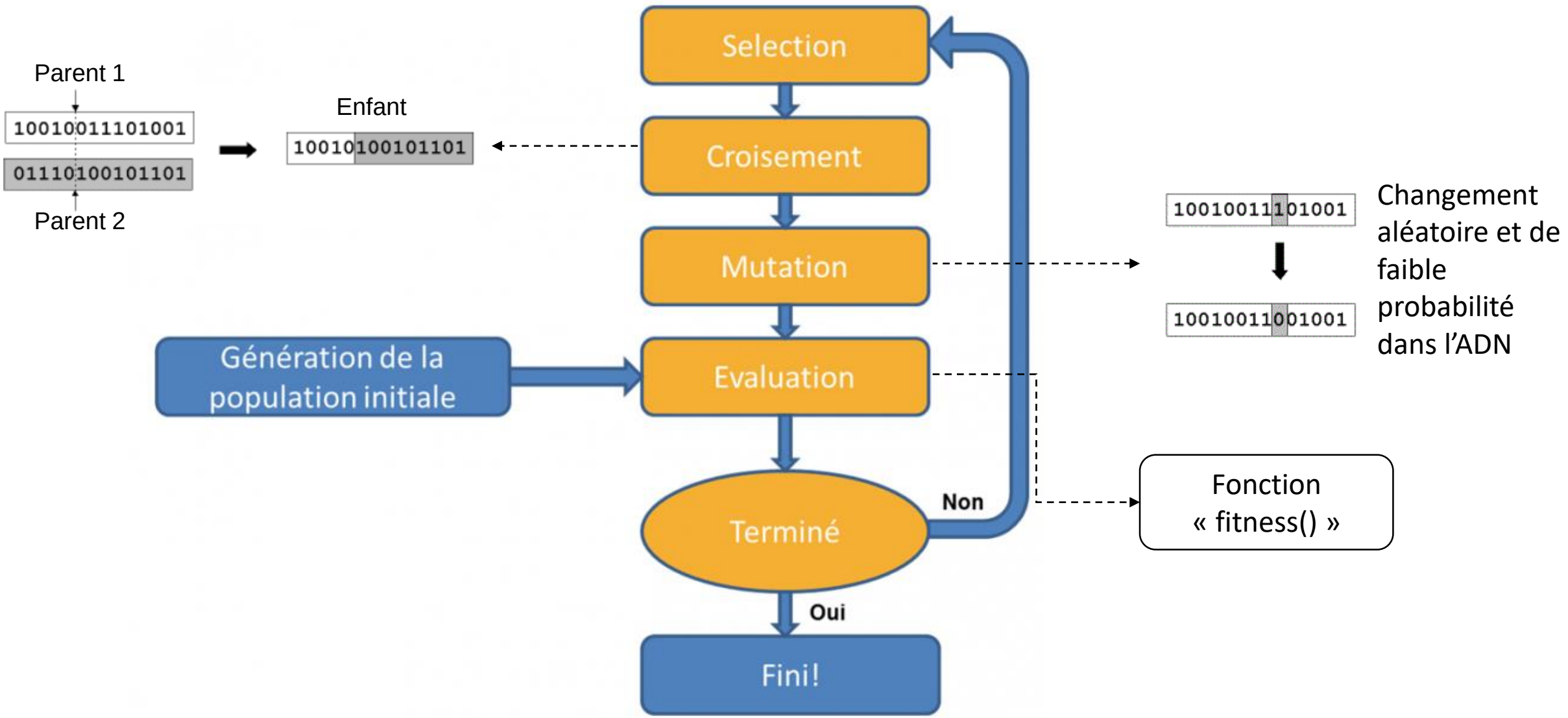
  - C'est un algorithme d'optimisation s'inspirant de l'évolution naturelle.
  - Population = plusieurs solutions éventuelles au problème.
  - La solution est entièrement déterminée par son ADN :

1	0	1	0	1	1

Exemple d'ADN - peut être une chaîne de bits, d'entiers, de vecteurs etc.

  - Les solutions évoluent en groupe de manière à atteindre l'optimum et ceci en quatre étapes majeures : L'évaluation, la sélection, le croisement et la mutation.

• Schéma de fonctionnement d'un algorithme génétique :



Source : genetic.io

- Pseudo-code d'un algorithme génétique :

```
Population ← generer_population()
evaluer(population)
tant que !conditionarret() faire
      parents ← selection(population)
      enfants ← croisement(parents)
      enfants ← mutation(enfants)
      population=enfants
fin tant que
```

II- Convergence de l'A.G:

- A- La théorie des chaînes de Markov:

- Hypothèses :

1) Ensemble d'états (populations) possibles fini qui sera confondu avec $\{0, \dots N-1\}$

2) Afin de simplifier, l'état 0 désignera tout état contenant la solution optimale.

3) On suppose de plus que le meilleur individu est toujours gardé dans la population.

Conséquences :

- Soit X_t v.a dans $\{0, \dots, N-1\}$ donnant l'état de la population à la génération t. $(X_t)_{t\in\mathbb{N}}$ définit une chaîne de Markov. Elle est homogène car :

$$\forall (i,j) \in \{0, \dots, N-1\}^2 \; \forall t \in \mathbb{N} : \; \mathbb{P}(X_{t+1} = j | X_t = i) = \mathbb{P}(X_{t+2} = j | X_{t+1} = i)$$

- La matrice $P = (p_{i,j})_{0 \leqslant i,j \leqslant N-1}$ est la matrice de transition de la chaîne, avec :

$$\forall t \in \mathbb{N} : \; p_{i,j} = \mathbb{P}(X_{t+1} = j | X_t = i)$$

Propriétés :

- ✓ $\forall i \in \{0, \dots, N-1\} : \sum_{j=0}^{N-1} p_{i,j} = 1$ P est dite stochastique
- ✓ On a :$\forall i \in \{0, \dots, N-1\} : p_{i,0} > 0$ l'état 0 est accessible depuis n'importe quel autre état
- ✓ On a $p_{0,0} = 1$ l'état 0 est dit absorbant .

D'où $P = \begin{pmatrix} 1 & 0 \\ R & Q \end{pmatrix}$

Proposition 1:

Notons $P^k = (p_{i,j}^{(k)})_{0 \leq i,j \leq N-1}$ la k-ème puissance de P . Alors :

$$\forall (i,j) \in \{0, \dots, N-1\} : \; p_{i,j}^{(k)} = \; \mathbb{P}(X_{t+k} = j | X_t = i)$$

Corollaire :

Si μ_0 est la loi de X_0 (loi initiale), alors la loi de X_t est donnée par:

$$\mu_t = \mu_0 P^t$$

les lois étant notées par des vecteurs lignes.

B- Convergence de l'A.G :

- **Proposition 2:**

$$\forall t \in \mathbb{N} : \; P^t = \begin{pmatrix} 1 & 0 \\ [I + Q + \cdots + Q^{t-1}]R & Q^t \end{pmatrix}$$

- **Théorème 1 :**

Soit $P = \begin{pmatrix} 1 & 0 \\ R & Q \end{pmatrix}$ la matrice de transition du processus de Markov associé à l'algorithme génétique, alors on a:

(1) : $\lim_{t \longrightarrow +\infty} Q^t = 0$

(2) : $I - Q$ est inversible et $[I - Q]^{-1} = \sum_{k=0}^{+\infty} Q^k$

- Corollaire :

$$\lim_{t\to+\infty} P^t = \begin{pmatrix} 1 & 0 \\ \sum_{k=0}^{+\infty} Q^k R & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \dots & 0 \end{pmatrix}$$

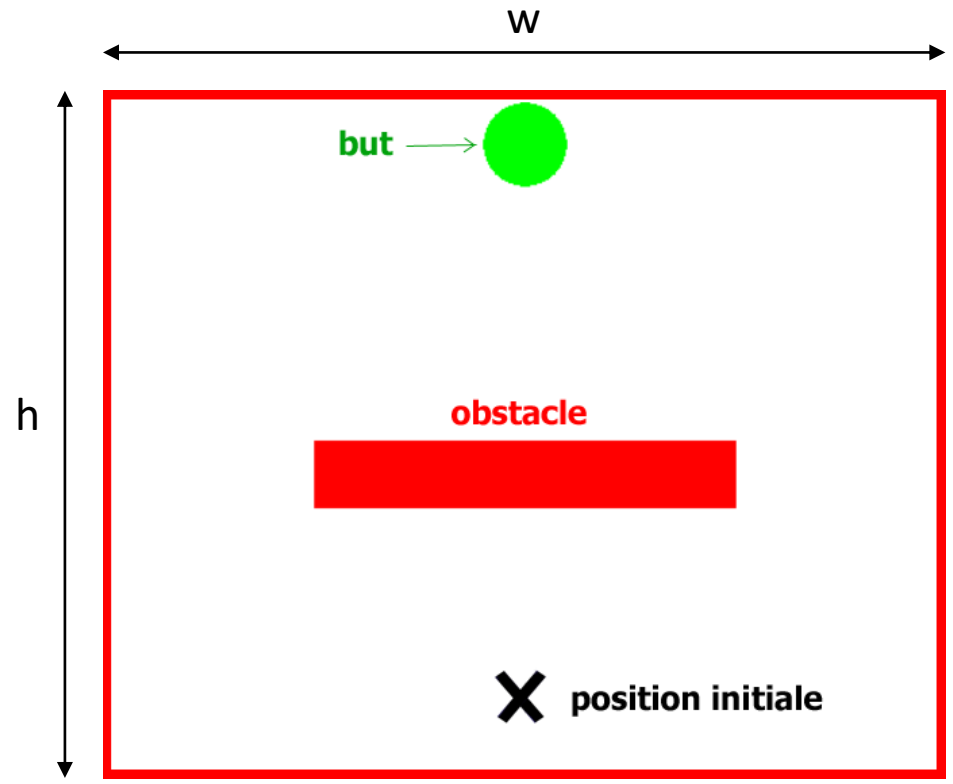
- Conclusion : (Convergence globale de l'algorithme)

$$\lim_{t\to+\infty} \mathbb{P}(X_t = 0) = 1$$

III-Modélisation informatique:

A- Implémentation de l'algorithme:

- Langage informatique : Python
- Bibliothèques utilisées : pygame, numpy, random.

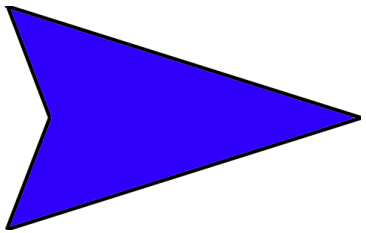


➢ Terrain à explorer = surface 2D

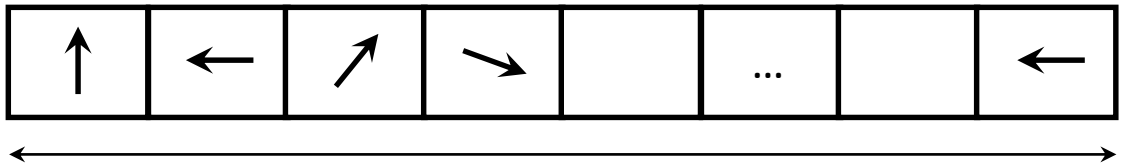
➢ Obstacle(s) = rectangle inaccessible

➢ But/zone d'intérêt = disque de rayon r

Objet : Robot



- Repéré par un vecteur pos(x,y)
- Possède un vecteur vitesse (vel), et accélération (acc).
- ADN : Liste de forces s'appliquant successivement.



len(ADN)=durée de vie d'un individu

- La mise à jour des individus entre deux instants :

```
if self.succes==False and self.collision==False:
    self.appliquerforce(self.DNA[count])
    self.vel+=self.acc
    self.pos+=self.vel
    self.acc*=0
```

ℹ « âge » de la population < durée de vie

ℹ La génération 0 est générée <u>aléatoirement</u>

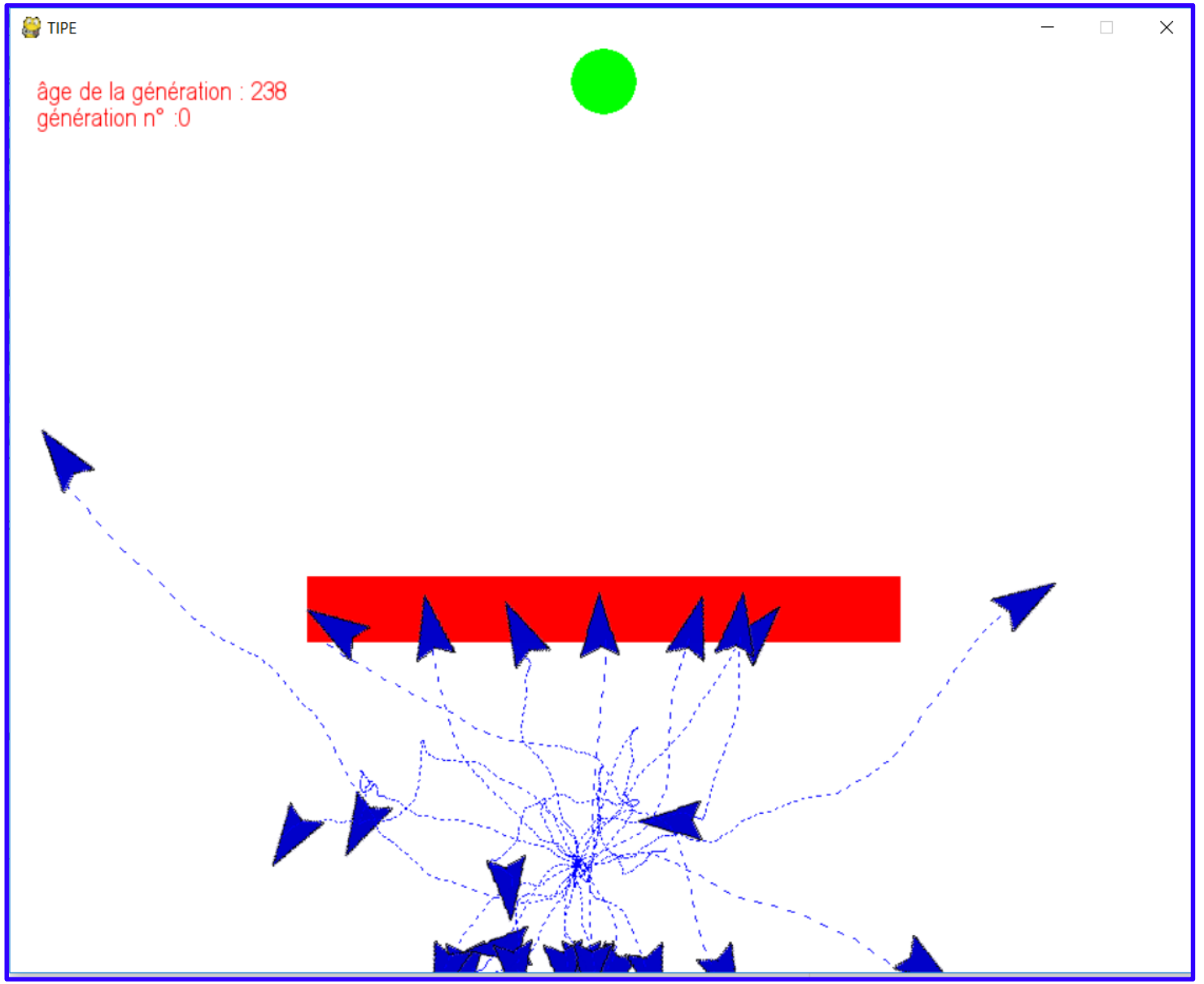


Figure 1 - *Capture d'écran affichant la population à un instant de l'exécution*

- Evaluation des individus:

Soit x un élément de la population, on choisit:

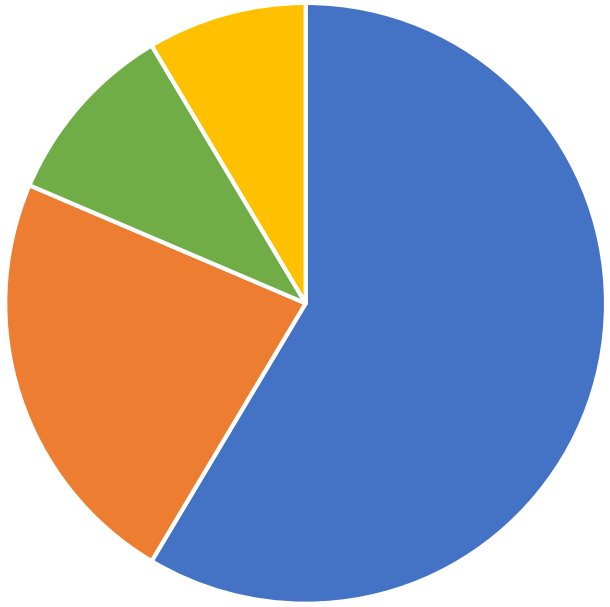
$$fitness(x) = \frac{{}^{1}/_{d(x)}}{\max\{{}^{1}/_{d(y)}\ ; y \in population\}} \in]0,1]$$

Autre choix possible : fonction affine, mais celle-ci est moins efficace :

✓ Une fonction en 1/d(x) offre une meilleure sélectivité :

- Sélection des parents : (Méthode de la roulette)

Probabilités de sélection



Individu	fitness
1	1
2	0,2
3	0,18
4	0,3

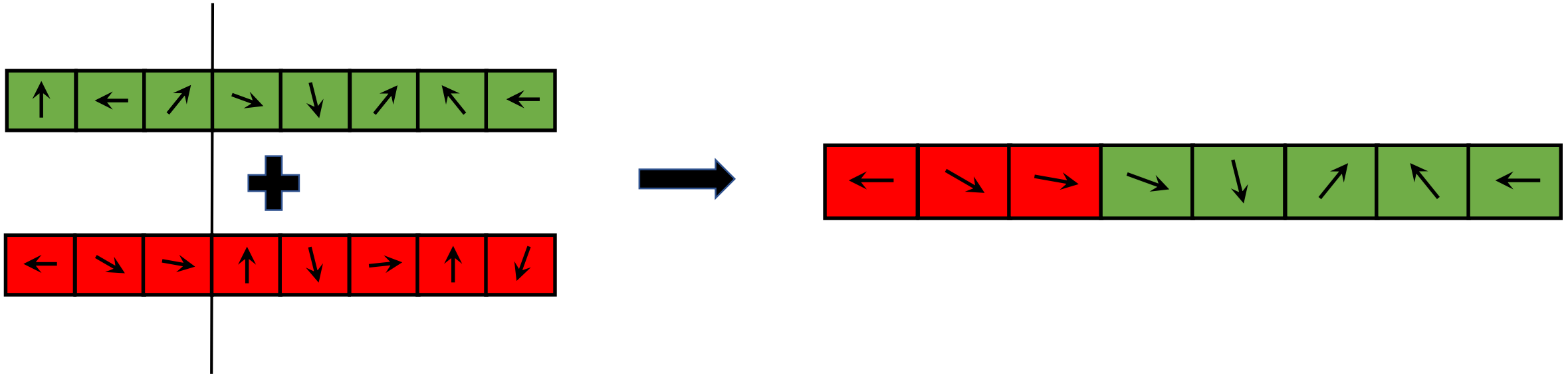
Avantage :

✓ Privilégie les meilleurs individus tout en préservant la diversité

En python : création d'une liste matingpool d'où le tirage au sort des parents est effectué :

```
def selection(self):
    self.evaluer()
    self.matingpool.clear()
    for i in range(self.taillepopulation):
        j=int(self.robots[i].fitness*100)
        for k in range(j):
            self.matingpool.append(self.robots[i])
```

- Croisement :

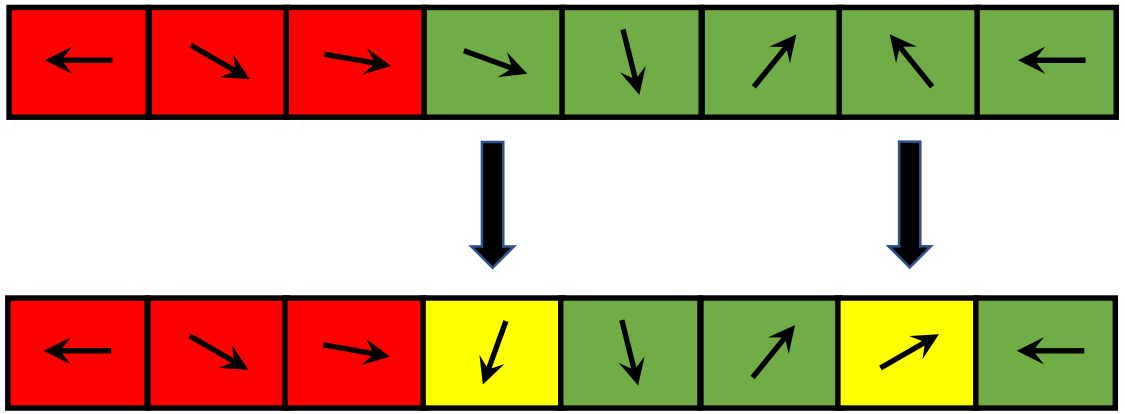


point de coupe m

En Python : fonction croisement

```
def croisement(parentA,parentB):
    A=parentA.DNA
    B=parentB.DNA
    enfant=robot()
    n=len(A)
    m=random.randrange(n)
    for i in range(n):
        if i<=m:
            enfant.DNA[i]=A[i]
        else:
            enfant.DNA[i]=B[i]
    return enfant
```

- Mutation:



➢ Chaque gène a une probabilité p_{mut} d'être remplacé par un vecteur force aléatoire

B- Observation et interprétation des résultats :

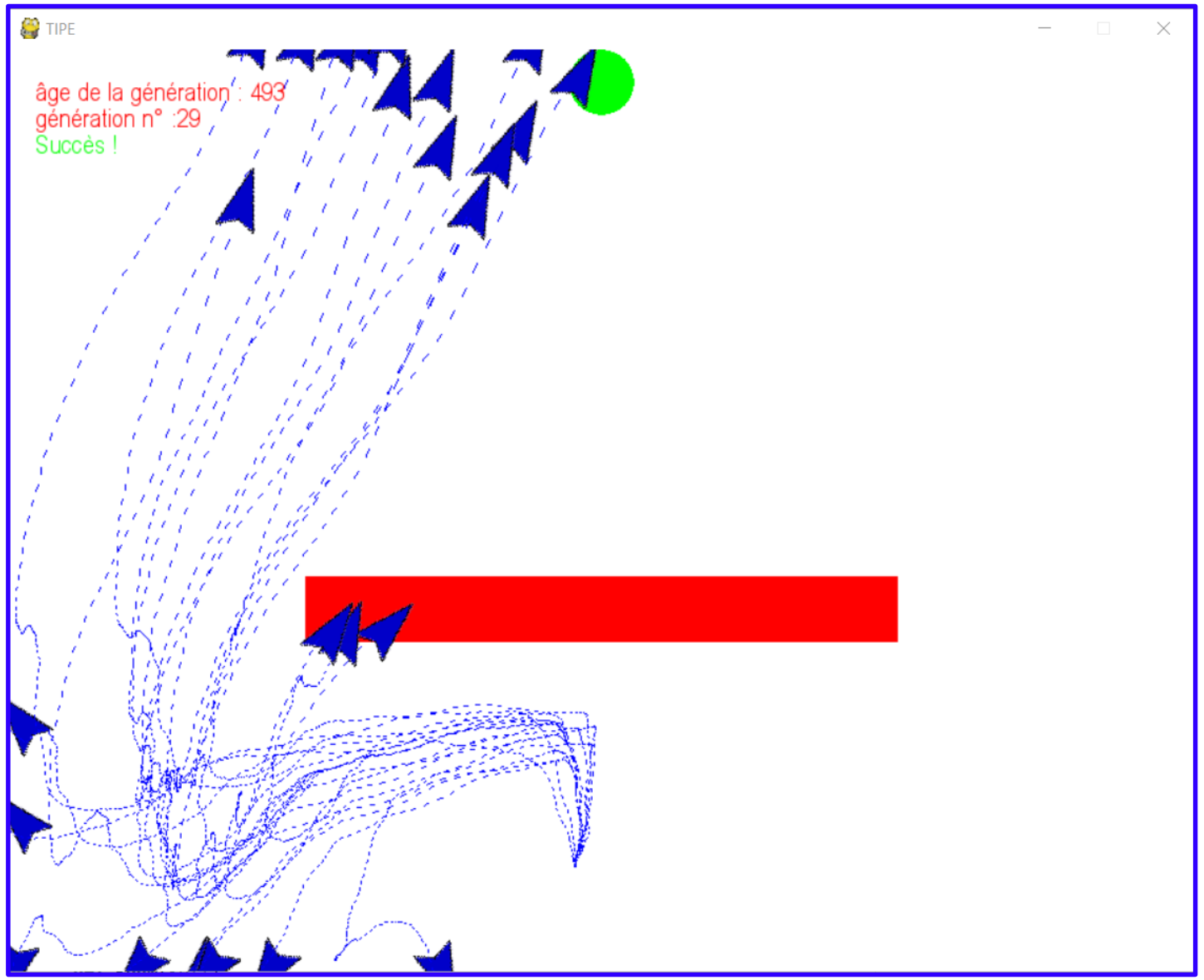


Figure 2- *État de l'algorithme après plusieurs itérations*

- Influence de la fonction fitness :

Taille de la population : 16

Probabilité de mutation : 1%

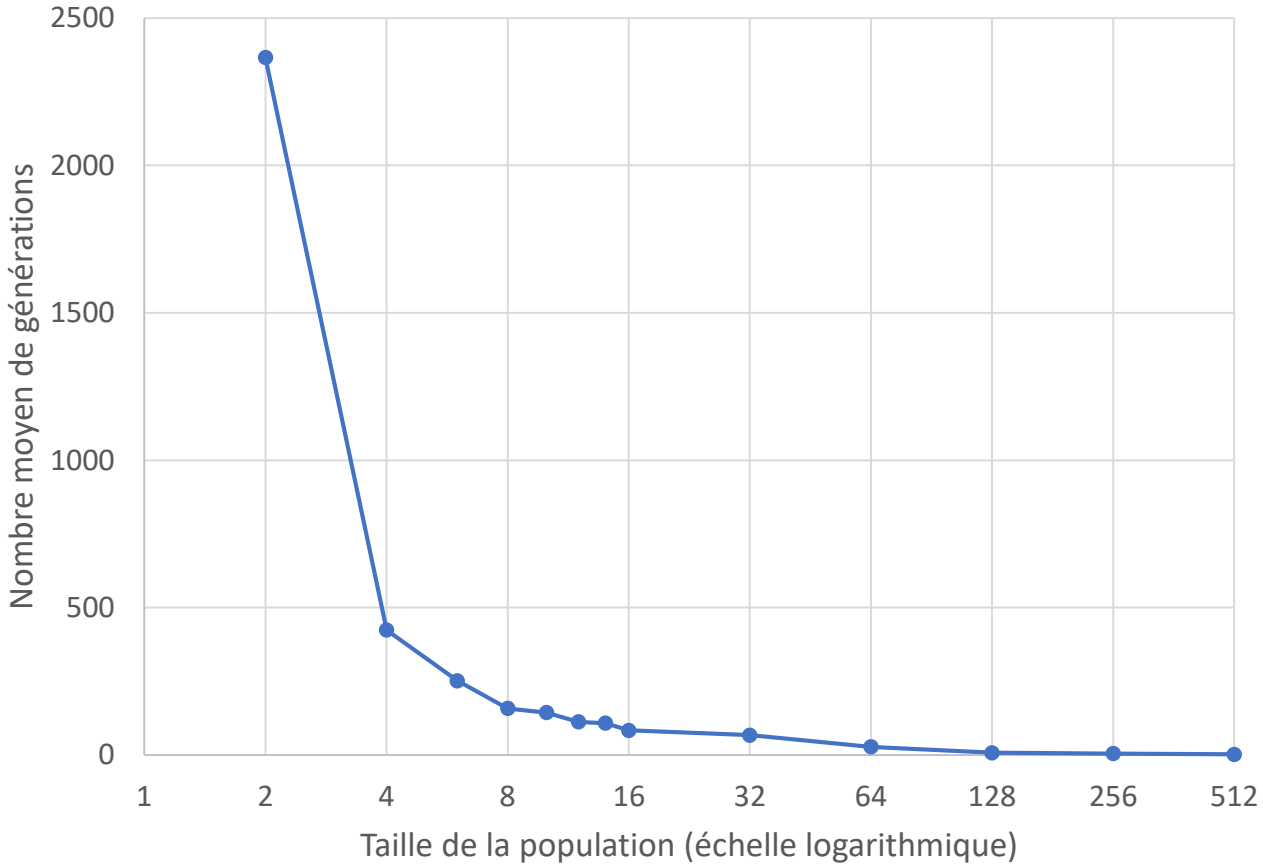
Type de fonction choisi	Nombre moyen de générations avant convergence
Inverse	83,8
Affine	109,5

✓ On observe une convergence 23.5% plus rapide pour une dépendance en 1/d(x)

- Influence de la taille de la population :

À p_{mut}=1% :

Taille de la population	2	4	6	8	10	12	14	16	32	64
Nombre moyen de générations	2366.1	423,7	251,6	157,9	144,1	112,5	108,3	83,8	67,2	27,7



- Population plus grande → Moins d'itérations

- Mais aussi :
  Population plus grande → coût élevé en opérations

- Influence de la mutation :

On fixe la taille de la population à 16.

P_{mut}	0,1%	0,2%	0,5%	1%	2%	5%	10%	20%	30%	50%	100%
Nombre moyen de générations	934,7	600,9	154,1	83,8	72,5	47,0	46,7	67,15	270,9	275,9	736,0