# Algorithme Génétique

## Introduction

Pour le développement d’une librairie d’algorithme génétique, nous nous sommes basés sur un exemple d’implémentation disponible sur Github :

Nous avons décidé de développer notre librairie en mettant en pratique différents mécanismes architecturaux et bonnes pratiques vues en cours, et en implémentant l’exemple précédant avec notre librairie développée.

## Exemple d’algorithme génétique

UML :

Les classes :

* **Ga.java** : c’est le main de l’application où est exécuté l’algorithme.
* **Algorithm.java** : contient les fonctions relatives à l’algorithme génétique (mutation, croisement
* **Individual.java** : représente un individu
* **Population.java** : représente une population d’individus
* **Skill.java** : permet d’évaluer les individus

Fonctionnement :

Dans cet exemple, la population est composée de 50 individus. Les individus sont représentés par un tableau de 64 gènes (tableau de 64 octets) ayant pour valeur 0 ou 1.

L’évaluation des individus est faite par rapport à un objectif (appelé *solution*) : "1111000000000000000000000000001111100000000000000000000000001111"

Une variable *skill* comptabilise les gènes similaires d’un individu par rapport à cet objectif. Elle est calculée par la fonction d’évaluation.

La fonction de mutation est aléatoire.

La fonction de croisement mélange aléatoirement les gènes des parents pour les donner à un individu enfant.

La Sélection des parents se fait aléatoirement ; 5 individus sont tirés aléatoirement dans la population. Parmi ces 5 individus, celui ayant la meilleur compétence (*skill*) est sélectionné pour être un parent.

L’algorithme s’arrête lorsqu’un individu présente les mêmes gènes que la solution.

Pour plus de précision, voir le code de l’exemple.

## Librairie développée

UML

Les classes

Interface IGeneticAlgorithm

Design Pattern Façade

Création d’individus

Design Pattern Factory

Point de vue utilisateur

Déclaration des fonctions d’évaluation, de croisement et de mutation via des lambdas

Implémentation des individus et de leur création via les interfaces Individual et IndividualCreator

Utilisation des setters

Initialisation

Run

Resultats

Gestion des exceptions

Classe GeneticAlgorithmException

Gestions des modes de fonctionnement

Via des énumérations

(pas complet)

Découpage des fonctions : SRP

Une fonction = Une responsabilité

Nommage, Commentaires, JavaDocs, gitlab, Logger

Fonctionnement :

voir exemple

A Améliorer :

* Tests unitaires : Junit, Mockito
* Parallélisation
* Gestion des modes
* Contraintes sur les entrées utilisateurs