Université de
VersaillesSaintQuentinenYvelines

UFR Des Sciences

Algorithme Génétique

Cahier des spécifications

Dirigé par : KLOUL Leila

Auteurs : AIT SLIMANE Rachid, ANTHENE Nicolas, BRAHIMI Lounes, DIA Mouhamadou Moustapha, DJAMA Samy, HAMENNI Koceila, OKETOKOUN Iqbal, WALSH Mathieu

22 avril 2020

Table des matières

1 Découpage des packages :

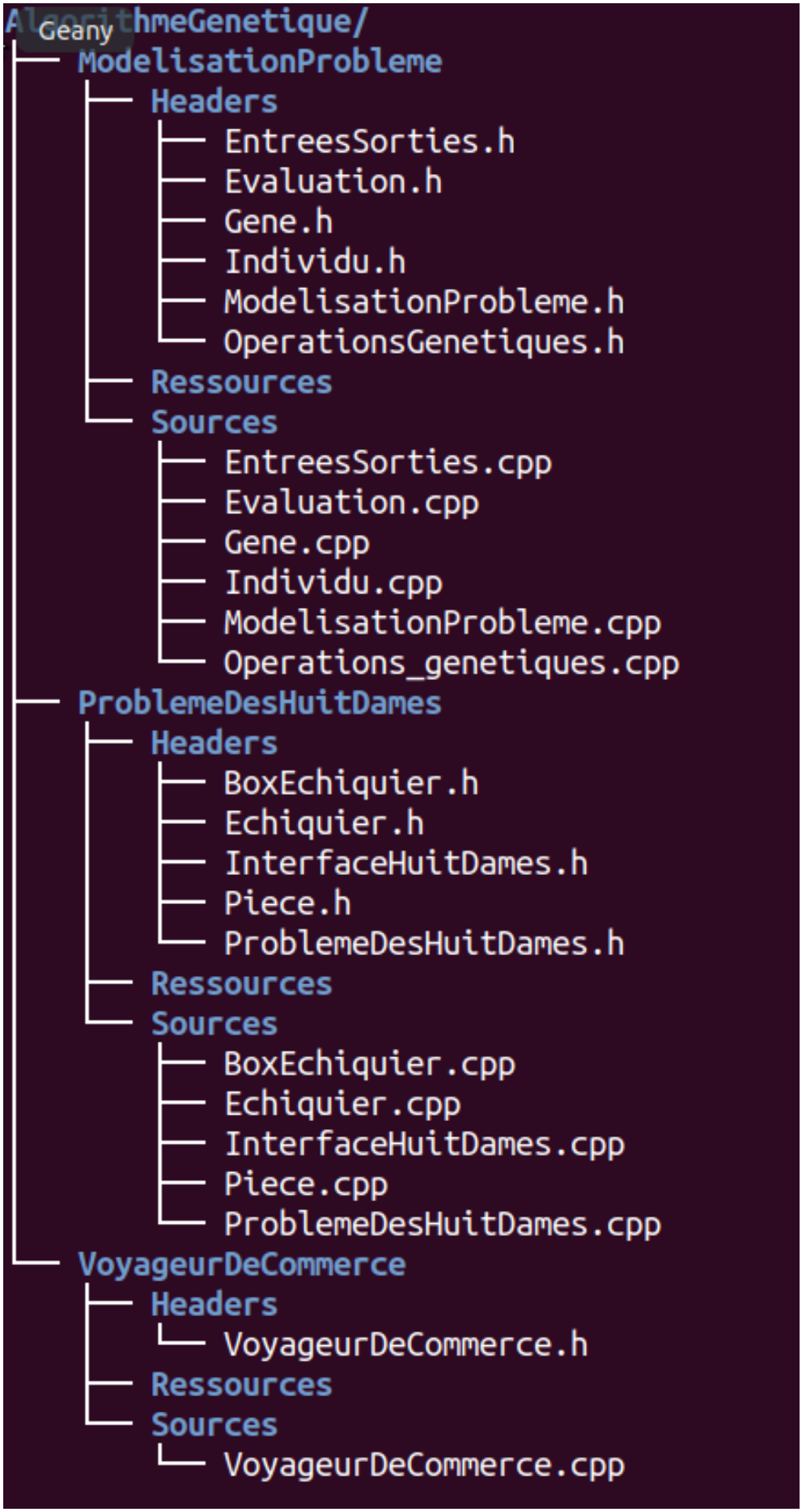

FIGURE 1 – Découpage des packages

Notre projet est découpé en 3 packages principaux, Modélisation d'un problème, Le Voyageur de commerce

et le problème de huit dames.

2 Diagramme de classes :

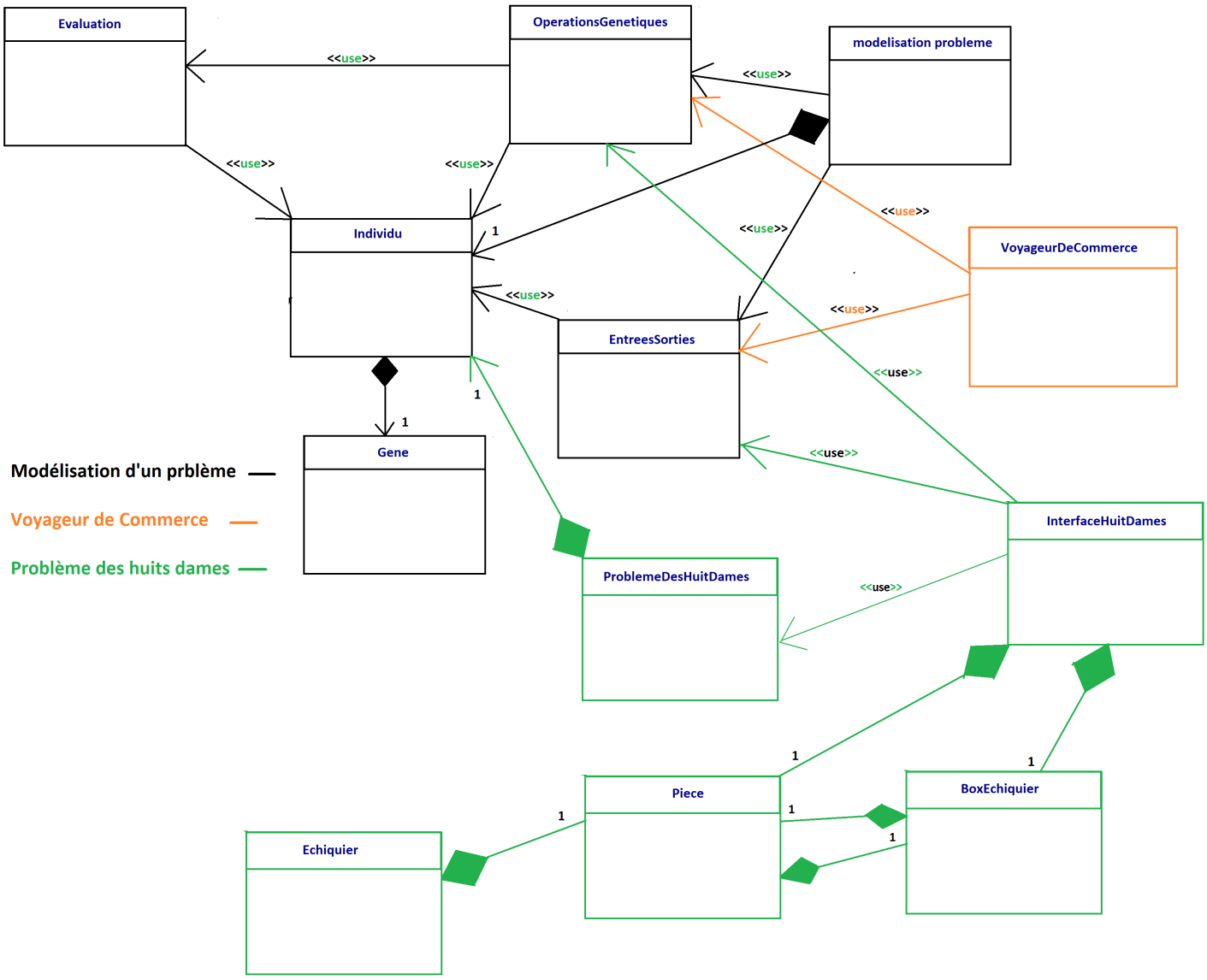

FIGURE 2 – Diagramme de classes

3 Introduction :

Un algorithme génétique est un algorithme ayant pour but de résoudre des problèmes d'optimisation avec une approche évolutionniste, il prend sa source de la théorie de l'évolution présentée par Charles Darwin au XIXe siècle. Un tel algorithme s'exécute en cinq étapes principales : l'évaluation, la sélection, l'enjambement, la mutation et l'itération du processus. Dans le cadre de notre projet de fin de cycle nous devons créer un logiciel permettant de modéliser différents types de problèmes, notre logiciel sera générique ce qui donnera la possibilité à l'utilisateur d'avoir une infinité de choix différents, notre logiciel donnera aussi la possibilité à l'utilisateur devisionner, grâce à une interface graphique, le déroulement de 2 problématiques prédéfinies par nos soins comme exemples, les 2 problématiques sont : "problème des huit dames" : ce dernier est de placer huit dames d'un jeu d'échecs sur un échiquier sans que les dames se menacent mutuellement."Voyageurs de commerce" : détermine un plus court chemin qui visite chaque sommet une et une seule fois et qui termine dans le sommet de départ. Le cahier des spécifications suivantes regroupera les fonctions, méthodes et classes que nous allons utiliser lors

de 4 l'implémentation des différents modules présents dans l'organigramme de notre application, celle-ci sera codée en C++.

4 Module initialisations :

Le module initialisations permet d'interpréter un grand nombre de données entré par l'utilisateur pour constituer en détail notre population initiale et de mettre à la disposition des autres modules les outils nécessaires pour l'exécution de l'algorithme génétique sur un problème donné que l'utilisateur aura modélisé.

Pour permettre la création de l'échantillon (la population initiale) correspondant aux souhaits de l'utilisateur, ce module sera composé de deux classes principales : la classe **Individu** et la classe **Gene**.

4.1 La structure individus :

Les deux classes Individu et Gene sont utilisées pour créer un objet "individu" membre de la population, l'ensemble de tous ces membres "**la population** en question" est contenu dans un tableau dynamique du type "**vector**" nommé "individus".

Cette structure sera déclaré dans une classe extra Gene et Individu, probablement dans le main ou une classe qui lui est très proche.

Cette dernière contiendra la population initiale et participera presque dans toutes les opérations qui auront lieu, vu que la recherche des solutions au problème se fera sur l'ensemble des individus en observant la variation, les changements et l'évolution de notre population donc des éléments que contient cette structure "**individus**".

4.2 La classe Gene :

Cet objet va contenir tout les gènes dont sera constitué un individu.

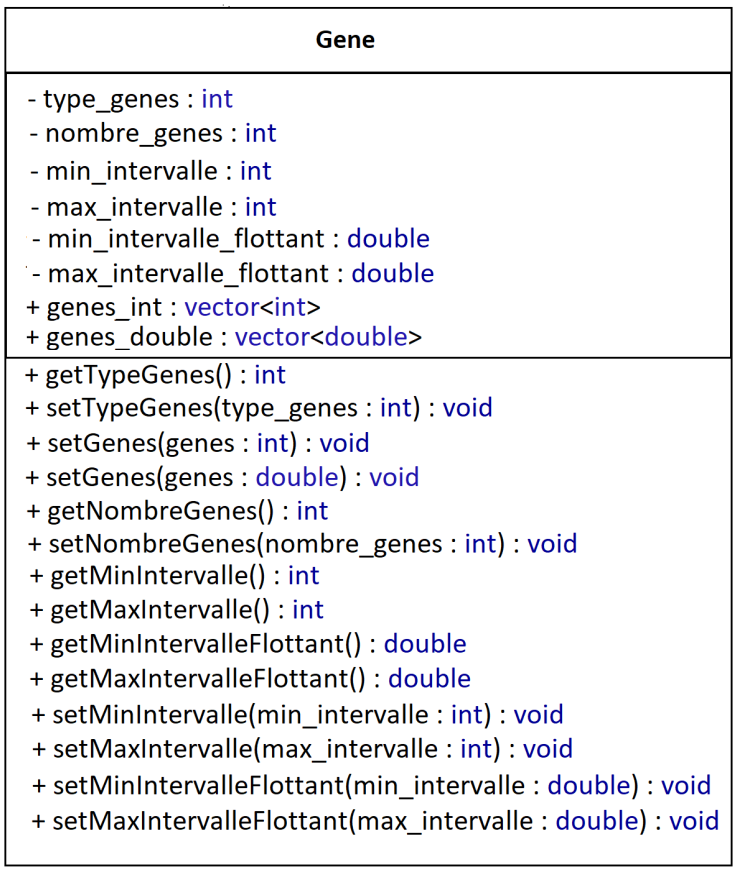

FIGURE 3 – La classe Gene

4.2.1 Les Attribus :

- **int type_genes :** C'est le type primitif choisi par l'utilisateur pour les gènes, **1** pour un entier, **2** pour un double et **3** pour un type binaire.
- **int nombre_genes :** Cette attribut contient le nombre de gènes désiré par l'utilisateur d'en sera constitué chaque individu de la population.
- **int min_intervalle** et **int max_intervalle :** Ils définissent l'intervalle de valeurs entières ou binaires que peut prendre les gènes.
- **double min_intervalle_flottant** et **double max_intervalle_flottant :** Ils définissent l'intervalle de valeurs flottantes que peut prendre les gènes.

4.2.2 Tableaux dynamiques :

Les gènes étant d'un nombre dépendant du choix de l'utilisateur, nous avons choisi de les héberger dans un tableau dynamique du type vector.

Ces derniers sont d'un type de nature primitif choisi par l'utilisateur, la variance de choix fait qu'on a définis 2 tableaux dynamiques :

. **vector<int> genes_int :** contenant des gènes du type **entier** ou du type **binaire**.

. **vector<double> genes_double :** contenant des gènes du type **flottant**.

4.2.3 Les constructeurs :

Le constructeur "**gene(int type_genes, int min_intervalle, int max_intervalle, int nombre_genes)**", est utiliser pour les gènes de type **entier** ou **binaire**, il initialise les attributs type_gene, nombre_genes, **"min_intervalle et max_intervalle de type entier"**.

Le constructeur "**gene(int type_genes, double min_intervalle, double max_intervalle, int nombre_genes)**", est utiliser pour les gènes de type **flottants**, il initialise les attributs type_gene, nombre_genes, **"min_intervalle et max_intervalle de type flottant"**.

"**gene()**" est un constructeur par défaut.

Tout les constructeurs font un "clear()" pour les 2 tableaux dynamiques pour des raisons de sécurité (avoir des tableaux vides).

4.2.4 Les Accésseurs :

- **int getTypeGene() :** Un accesseur retournant le type des gènes.
- **int getNombreGenes() :** Un accesseur retournant le nombre de gènes.
- **int getMinIntervalle() :** Retourne la valeur minimale pour un gène de type entier ou binaire.
- **int int getMaxIntervalle() :** Retourne la valeur maximale pour un gène de type entier ou binaire.
- **double getMinIntervalleFlottant() :** Retourne la valeur minimale pour un gène de type flottant.
- **double getMaxIntervalleFlottant() :** Retourne la valeur maximale pour un gène de type flottant.

4.2.5 Les Mutateurs :

- **void setTypeGenes(int type_genes) :** Modifie le type des gènes.
- **void setNombreGenes(int nombre_genes) :** Modifie le nombre de gènes.
- **void setMinIntervalle(int min_intervalle) :** Modifie la valeur minimale que peut prendre un gène de type entier ou binaire.
- **void setMaxIntervalle(int max_intervalle) :** Modifie la valeur maximale que peut prendre un gène de type entier ou binaire.
- **void setMinIntervalleFlottant(double min_intervalle) :** Modifie la valeur minimale que peut prendre un gène de type flottant.
- **void setMaxIntervalleFlottant(double max_intervalle) :** Modifie la valeur maximale que peut prendre un gène de type flottant.

4.2.6 Les mutateurs surchargés :

Le tableau dynamique correspondant au type de gènes choisis sera initialisé et modifié à travers le mutateur portant le prototype qui paramètre la méthode avec le même type de variable :

. **void setGenes(int genes) :** pour porter des modification sur le tableau dynamique **"genes_int"**.

. **void setGenes(double genes) :** pour porter des modification sur le tableau dynamique **"genes_double"**.

4.2.7 Le destructeur :

∼ **gene()** est un destructeur implémenter pour libérer la mémoire allouée par les tableaux plus proprement.

4.3 La classe Individu :

L'ensemble des objets de cette classe constitueront notre population.

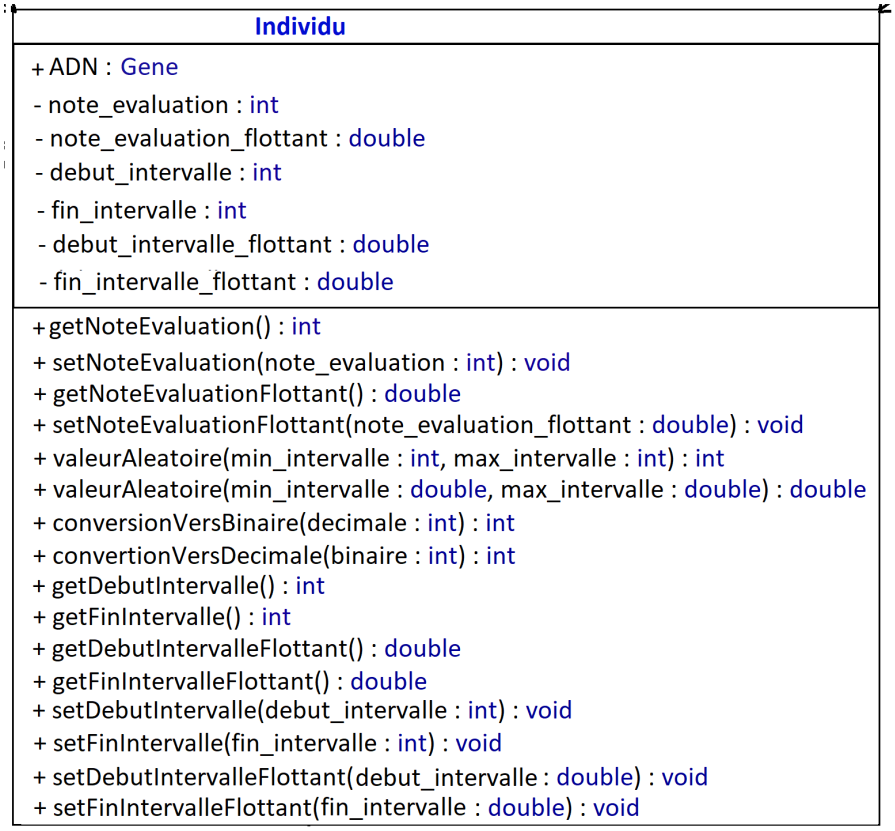

FIGURE 4 – La classe Individu

Nous commençons par cité les données qui seront transmises vers la classe individu :

. **Le type des gènes :** entiers, flottants ou bien binaire.

. **Le nombre des gènes** d'en sera composé chaque individu.

. **un intervalle :** représentant les valeurs que peut prendre un gène.

4.3.1 Attributs :

— **int note_evaluation :** C'est la note représentant la capacité d'adaptation d'un individu doté de gènes de type entier ou binaire dans l'environnement.
— **double note_evaluation_flottant :** C'est la note représentant la capacité d'adaptation d'un individu doté de gènes de type flottant dans l'environnement.
— **gene ADN :** instance de la classe Gene, qui va contenir tous les gènes de l'individu et permettra leurs modification et la récupération de leurs valeurs.

— **debut_intervalle et fin_intervalle :** outils nécessaires pour concevoire les intervalles utiliser dans la séléction par roulette dans le cas où les gènes sont de type entier ou binaires.
— **double debut_intervalle_flottant et double fin_intervalle_flottant :** outils nécessaires pour concevoire les intervalles utiliser dans la séléction par roulette dans le cas où les gènes sont de type flottant.

4.3.2 Constructeurs :

— **individu(int min_intervalle, int max_intervalle, int nombre_genes, int type_genes)** est le constructeur qui sera utiliser dans le cas où les gènes sont de type entiers ou booléens, il permet d'instancier ADN avec le type et le nombre de gènes passé en paramètres et il initialise les gènes avec des valeurs aléatoire comprises entre min_intervalle et max_intervalle en utiliser la méthode **int valeurAleatoire(int, int)**.
— **individu(double min_intervalle, double max_intervalle, int nombre_genes, int type_genes)** est le constructeur qui sera utiliser dans le cas où les gènes sont de type flottant, il permet d'instancier ADN avec le type et le nombre de gènes passé en paramètres et il initialise les gènes avec des valeurs aléatoire comprises entre min_intervalle et max_intervalle en utiliser la méthode **int valeurAleatoire(double, double)**.
— **individu()** est un constructeur par défaut qui appele le constructeur par defaut de la classe Gene.

4.3.3 Les méthodes :

— **int valeurAleatoire(int min_intervalle, int max_intervalle) :** génére une valeur entière comprise entre min_intervalle et int max_intervalle, il est utiliser dans le cas où les gènes sont de type entiers ou binaires.
— **int valeurAleatoire(double min_intervalle, double max_intervalle) :** génére une valeur flottante comprise entre min_intervalle et max_intervalle, il est utiliser dans le cas où les gènes sont de type flottants.
— **int conversionVersBinaire(int decimale) :** Convertie un nombre **décimale** en **binaire**.
— **int conversionVersBinaire(int decimale) :** Convertie un nombre **binaire** en **décimale**.

4.3.4 Les Accesseurs :

. **int getNoteEvaluation() :** retourne l'attribut note_evaluation dans le cas où les gènes sont de type entier ou binaire.
. **double getNoteEvaluationFlottant() :** retourne l'attribut note_evaluation_flottant.
. **int getDebutItervalle() :** retourne l'attribut debut_intervalle.
. **int getFinItervalle() :** retourne l'attribut fin_intervalle.
. **double getDebutIntervalleFlottant() :** retourne l'attribut double debut_intervalle_flottant.
. **double getFinIntervalleFlottant() :** retourne l'attribut double fin_intervalle_flottant.

4.3.5 Les Mutateurs :

. **void setNoteEvaluation(int note_evaluation) :** permet de modifier la valeur de l'attribut note_evaluation.
. **void setNoteEvaluationFlottant(double note_evaluation_flottant) :** permet de modifier la valeur de l'attribut note_evaluation_flottant.
. **void setDebutIntervalle(int debut_intervalle) :** permet de modifier la valeur de l'attribut int debut_intervalle.
. **void setFinIntervalle(int fin_intervalle) :** permet de modifier la valeur de l'attribut int fin_intervalle.
. **void setDebutIntervalleFlottant(double debut_intervalle) :** permet de modifier la valeur de l'attribut int debut_intervalle_flottant.
. **void setFinIntervalleFlottant(double fin_intervalle) :** permet de modifier la valeur de l'attribut int fin_intervalle_flottant.

5 Module Test et Evaluation :

Ce module permet d'analyser la syntaxe et d'interpréter l'équation sur les gènes entrée par l'utilisateur et de tester si la génération (la population) actuelle est satisfaisante pour arrêter l'éxecution de l'algorithme et de rendre les résultats, nous avons donc décidé de créer la classe "**Evaluation**".

Cette dernière utilise dans plusieurs de ses méthodes la classe "**Individu**" car un individu une fois évaluer, dans le but d'être noté, on modifie son attribut note_evaluation et pour faire cela, on a besoin de récupérer les valeurs de ses gènes.

5.1 La classe Evaluation :

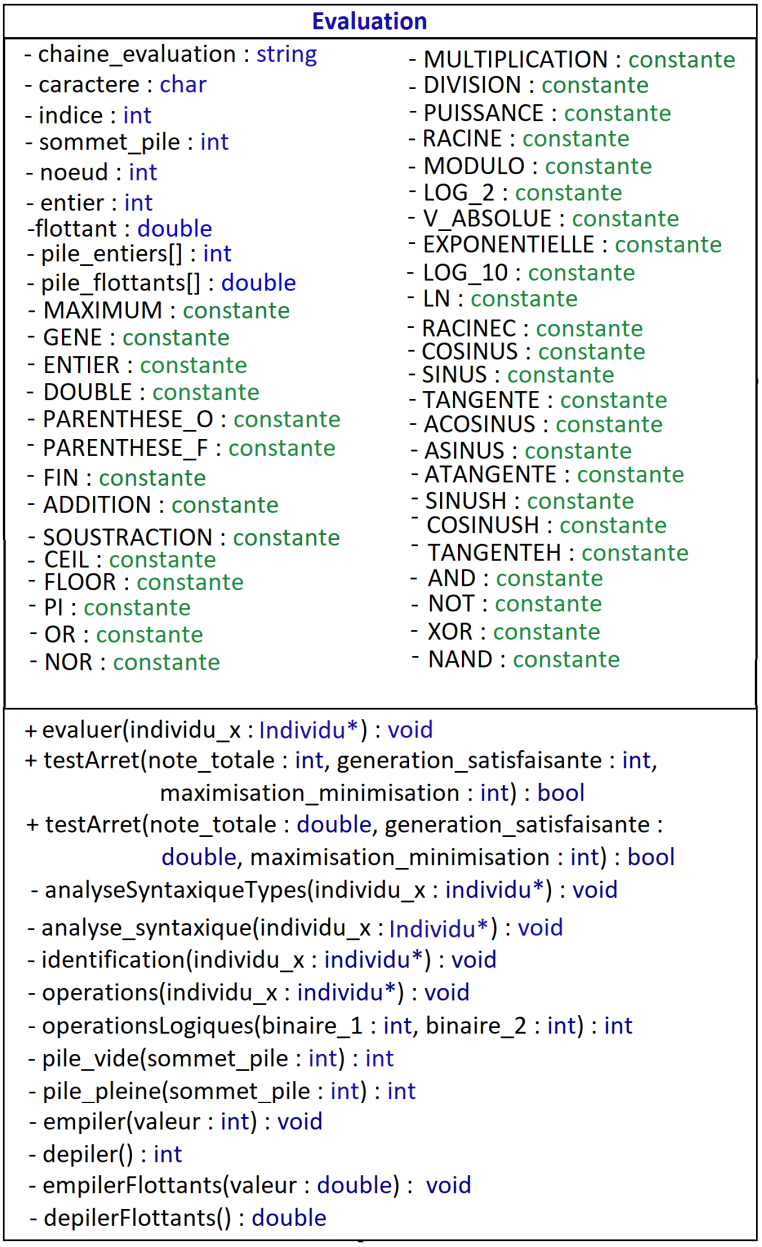

FIGURE 5 – La classe Evaluation

5.1.1 Les Attributs :

— **string chaine_evaluation** contient l'équation que l'utilisateur aura entrée.
— **char caractere** porte le caractère actuel.
— **int indice :** représente l'indice du caractère actuel sur la chaine de caractère qui contient l'équation.

— **int sommet_pile :** indice du sommet de la pile.
— **int noeud :** une fois un élément de l'équation est détecté il est affecté dans l'attribut "noeud", ainsi il représente un élément (la valeur d'un gène, l'addition, la fin de l'équation...).
— **int entier :** Utiliser dans le cas ou les gènes sont de type entiers ou binaire, lorsque l'élément actuelle est un géne ou un entier relatif, il devra surment etre empiler dans la pile pour servire d'opérande pour l'opération désirer par l'utilisateur, ainsi entier contiendra la valeur de l'opérande.
— **double flottant :** Utiliser dans le cas ou les gènes sont de type flottant, lorsque l'élément actuelle est un géne ou un flottant, il devra surment etre empiler dans la pile pour servire d'opérande pour l'opération désirer par l'utilisateur, ainsi flottant contiendra la valeur de l'opérande.
— **int pile_entiers[MAXIMUM] :** structure représentant une pile, qui est utilisé pour l'analyse et l'interprétation de l'équation dans le cas où les gènes sont de type entiers ou binaire.
— **ouble pile_flottants[MAXIMUM] :** structure représentant une pile, qui est utilisé pour l'analyse et l'interprétation de l'équation dans le cas où les gènes sont de type flottant.
— **static const int GENE :** identifiant d'un noeud "gène".
— **static const int ENTIER :** identifiant d'un noeud "entier".
— **static const int DOUBLE :** identifiant d'un noeud "flottant".
— **static const int FIN :** identifiant de la fin de l'équation.
— **static const int PARENTHESE_O :** identifiant de la parenthèse ouvrante.
— **static const int PARENTHESE_F :** identifiant de la parenthèse fermante.
— **static const int ADDITION :** identifiant de l'opération d'addition.
— **static const int SOUSTRACTION :** identifiant de l'opération de soustraction.
— **static const int MULTIPLICATION :** identifiant de l'opération de multiplication.
— **static const int DIVISION :** identifiant de l'opération de division.
— **static const int PUISSANCE :** identifiant de l'opération de puissance.
— **static const int RACINE :** identifiant de l'opération racine carrée.
— **static const int MODULO :** identifiant de l'opération modulo.
— **static const int V_ABSOLUE :** identifiant de l'opération valeur absolue.
— **static const int LN :** identifiant de l'opération logarithme naturel.
— **static const int LOG_2 :** identifiant de l'opération logarithme de base 2.
— **static const int LOG_10 :** identifiant de l'opération logarithme de base 10.
— **static const int EXPONENTIELLE :**identifiant de l'opération exponentielle.
— **static const int RACINEC :** identifiant de l'opération racine cubique.
— **static const int CEIL :** identifiant de l'opération nombre entier supérieur le plus proche.
— **static const int FLOOR :** identifiant de l'opération nombre entier inférieur le plus proche.
— **static const int COSINUS :** identifiant de l'opération cosinus.
— **static const int SINUS :** identifiant de l'opération sinus.
— **static const int TANGENTE :** identifiant de l'opération tangente.
— **static const int ACOSINUS :** identifiant de l'opération arc cosinus.
— **static const int ASINUS :** identifiant de l'opération arc sinus.
— **static const int ATANGENTE :** identifiant de l'opération arc tangente.
— **static const int COSINUSH :** identifiant de l'opération cosinus hyperbolique.
— **static const int SINUSH :** identifiant de l'opération sinus hyperbolique.
— **static const int TANGENTEH :** identifiant de l'opération tangente hyperbolique.
— **static const int AND :** identifiant de l'opération du ET logique.
— **static const int OR :** identifiant de l'opération du OU logique.
— **static const int NOT :** identifiant de l'opération négation logique.
— **static const int XOR :** identifiant de l'opération ou éxclusif.
— **static const int NAND :** identifiant de l'opération NONET.
— **static const int NOR :** identifiant de l'opération NONOU.
— **static const int PI :** constante d'Archimède.
— **#define MAXIMUM :** Maximum élément que contiendra une pile.

5.2 Le constructeur :

Evaluation(std : : string chaine_evaluation) est le constructeur utilisé, il prend en paramètre l'équation donnée par l'utilisateur, avec laquelle il initialise l'attribut chaine_evaluation, l'attribut noeud, indice, entier, flottant, sommet_pile et caractere par le premier element de la chaine saisie par l'utilisateur.

5.3 Les méthodes :

- **la méthode void evaluer(individu* individu_x) :** C'est cette méthode qui est appelée pour analyser et interpréter l'équation, elle prend en paramètre l'individu qu'elle va évaluer.
  Pour analyser et interpréter l'équation, cette méthode utilise d'autres, qui sont : (analyseSyntaxique-Types, analyseSyntaxique, identification, operations, operationsLogiques, pileVide, pilePleine, empiler, depiler, empilerFlottants, depilerFlottants).
- **void analyseSyntaxiqueTypes(individu* individu_x) :** Cette méthode analyse syntaxiquement l'équation et récupére les entiers ou les flottants si ils y figurents.
- **void analyseSyntaxique(individu* individu_x) :** Cette méthode permet d'identifier les types des éléments de l'équation de les affectés dans l'attribut noeud. Elle permet d'interpréter les symboles des gènes et de récuperer leurs valeurs via individu_x et de les affectées dans l'attribut entier ou flottant. Elle permet aussi de détecter les opérations arithmétiques ou logiques et les afféctés dans noeud.
- **void identification(individu* individu_x) :** Méthode qui vérifie si le noeud actuel est un gène, un entier, un flottant ou une parenthése ouvrante, si c'est le cas, une opération est probablement à venir, donc elle empile la valeur du gène et appelle notre méthode analyse_syntaxique pour interpréter la suite de l'équation, dans le cas où c'est une parenthése ouvrante elle va continuer l'analyse syntaxique car l'équation est surment pas finit.
- **void operations(individu* individu_x) :** Vérifie si c'est une opération et fait le traitement nécessaire dans ce cas, en analysant la suite de l'équation avec analyseSyntaxiqueTypes et analyseSyntaxique, une fois les opérandes cerner dans la pile, elle exécute l'opération et l'empile.
- **int operationsLogiques(int binaire_1, int binaire_2) :** Méthode appeler par la méthode operation lorsque le noeud actuelle est une opération logique pour l'éxecuter sur les deux opérandes. Donc c'est dans cette derniere qu'est implémenter les calcules logiques (AND, OR, XOR...).
- **int pileVide(int sommet_pile) :** Elle met notre pile à vide à travers l'attribut sommet_pile.
- **int pilePleine(int sommet_pile) :** Verifie si la pile est pleine.
- **void empiler(int valeur) :** Ajouter un élément au sommet de la pile d'entiers.
- **void empilerFlottants(double valeur) :** Ajouter un élément au sommet de la pile de flottants.
- **int depiler() :** Retourner l'élément au sommet de la pile d'entiers et le supprimer de cette dernière.
- **double depilerFlottants() :** Retourner l'élément au sommet de la pile de flottant et le supprimer de cette dernière.
- **Le test d'arrêt :** Dans le cas où les gènes sont de type entiers ou binaire, le test est fait avec la méthode **bool testArret(int note_totale, int generation_satisfaisante, int maximisation_minimisation)** qui vérifie si la note totale de la génération actuelle **"note_totale"** est supérieur à une note fixé par l'utilisateur **generation_satisfaisante**, présumer représentant une population satisfaisante en prenant compte si l'éxécution a pour but une maximisation ou une minimisation.
  Dans le cas où les gènes sont de type flottants, le test est fait avec la méthode **bool testArret(double note_totale, double generation_satisfaisante, int maximisation_minimisation)** qui vérifie si la note totale de la génération actuelle **"note_totale"** est supérieur à une note fixé par l'utilisateur **generation_satisfaisante**, présumer représentant une population satisfaisante en prenant compte si l'éxécution a pour but une maximisation ou une minimisation.

6 Module opérations génétiques :

Ce module regroupe toutes les principales opérations sur une population, à savoir la sélection (avec ses trois types), le croisement et la mutation, ainsi nous avons décidé de créer la classe OperationsGenetiques.

Cette dernière manipule dans toutes ses méthodes des objets de la classe **Individu** ou indirectement via l'ensemble **individus**, elle utilise aussi dans certaines de ses méthodes telles que le croisement la classe **Evaluation**.

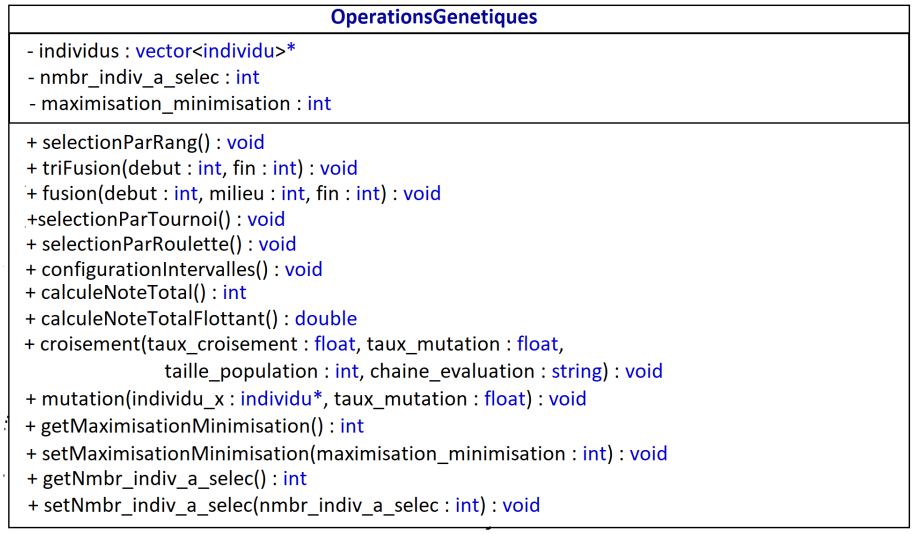

FIGURE 6 – La classe operations genetiques

6.0.1 Constructeurs :

Notre classe a pour constructeur **operationsGenetiques(std : :vector<individu>* individus, int maximisation_minimisation, int nmbr_indiv_a_selec)**, il initialise ses attributs, ainsi il indique si l'utilisateur souhaite une maximisation ou bien une minimisation, le nombre d'individus qui seront séléctionnés et fait pointer notre variable individus sur la population sur laquelle va s'opérer les opérations génétique.

6.1 Les Attributs :

- **<individu>* individus :** pointeur sur l'ensemble individus qui héberge notre population qui sera modifier dans plusieurs méthodes tel que les séléction, le croisement ou la mutation.
- **int nmbr_indiv_a_selec** Le nombre d'individus à séléctionné.
- **int maximisation_minimisation :** indique si la configuration maximise ou minimise un critère donné par l'utilisateur.

6.2 La séléction par rang :

. **void selectionParRang() :** c'est cette méthode qui est appelée pour faire la sélection par rang, elle tri notre population à l'aide de la méthode **triFusion** et sélectionne le nombre d'individus souhaité et supprime les autres.

. **void triFusion(int debut, int fin) :** méthode qui fait un tri de type fusion sur notre population.

. **void Fusion(int debut, int milieu, int fin) :** méthode utilisé par la fonction **triFusion** pour faire le tri fusion et avoir ainsi une complexité en nlog(n).

6.3 La séléction par tournoi :

On utilise la méthode **"void selectionParTournoi()"** pour faire ce traitement, elle crée deux objets du type Individu aux quels elle affecte deux éléments de notre population individus, elle les comparent, et insère

le plus adapté au problème dans un tableau dynamique temporaire qui sera à la fin affecté à notre structure individus.

6.4 La séléction par roulette :

. **void selectionParRoulette() :** C'est cette méthode qui est appelée pour faire la sélection par roulette, elle utilise la méthode **configurationIntervalles** pour créer un intervalle pour chaque individu proportionnel à sa note d'évaluation et la note total de la population, elle sélectionne le nombre d'individus souhaité en supprimant les autres.

. **void configurationIntervalles() :** elle génère un intervalle indiquant l'importance de la note d'évaluation d'un individu par rapport a la note total de la population pour chaque membre de notre population, l'ensemble des intervalles se complètent et constitue la note total.

. **int calculeNoteTotal() :** Cette méthode utiliser dans **configurationIntervalles** calcule la note total de la population dans le cas où les gènes sont de type entier ou binaire.

. **double calculeNoteTotalFlottant() :** Cette méthode utiliser dans **configurationIntervalles** calcule la note total de la population dans le cas où les gènes sont de type flottant.

6.5 Le croisement :

L'opération du croisement se fait sur les individus sélectionnés en appelant la méthode **"void croisement(float taux_croisement, float taux_mutation, int taille_population, std : :string chaine_evaluation)"**, elle calcule et choisit avec le taux de croisement les individus qui participeront au croisement, elle boucle le nombre d'individus manquant pour avoir le nombre d'individus initiaux, à chaque tour de boucle elle prend 2 individus parmi ceux sélectionnés et crée grace à eux deux nouveau individus dont les gènes sont un mélange de ceux des 2 parents, elle évalue une note d'adaptation pour ces deux nouveaux individus et l'affecte dans leurs attributs, elle fait un calcul aléatoire qui suivant le taux de mutation peut amener l'un des individus ou les deux à muter, en appelant la méthode **mutation**.

6.6 La Mutation :

Elle ce fait via la méthode **"void mutation(individu* individu_x, float taux_mutation)"**, cette dérnière modifie un gène ou plusieurs d'un individu avec une valeur aléatoire comprise entre l'intervalle fixé par l'utilisateur.

6.7 Les Accésseurs :

— **int getMaximisationMinimisation() :** retourne l'attribut maximisation_minimisation.
— **int getNmbr_indiv_a_selec() :** retourne l'attribut nmbr_indiv_a_selec.

6.8 Les Mutateurs :

— **void setMaximisationMinimisation(int maximisation_minimisation) :** modifie l'attribut maximisation_minimisation.
— **void setNmbr_indiv_a_selec(int nmbr_indiv_a_selec) :** modifie l'attribut nmbr_indiv_a_selec.

7 Le module entrees sorties :

Le module Entrées et sorties sert soit à constituer soit à charger une configuration, il offre aussi à l'utilisateur la possibilité de sauvegarder une configuration donnée.

Ce dernier nous permet aussi de générer des statistiques montrant l'évolution des individus de génération en génération.

FIGURE 7 – La classe entrées et sorties

Nous avons décidé de creer une classe EntreesSorties qui implémente toutes ces fonctionnalités.

7.0.1 Les Attributs :

— **std : :string chaine_evaluation :** contiens l'équation à évaluer saisie par l'utilisateur.
— **taille_population :** représente le nombre d'individus dans la population.
— **int nombre_iterations :** nombre d'itérations maximale "nombre maximale de générations qui seront produites".

— **int nmbr_indiv_a_selec :** nombre d'individus qui seront séléctionnés.
— **int choix_selection :** type de séléction choisie par l'utilisateur.
— **int nombre_genes :** nombre de gènes d'en sera constituer chaque individu.
— **float taux_croisement :** pourcentage à multiple utilité dans le croisement, utiliser par exemple pour déterminer la taille stable dans les individus séléctionnés.
— **float taux_mutation :** le pourcentage utilisé pour calculer la probabilité qu'un individu mute.
— **int generation_satisfaisante :** la note d'évaluation moyenne minimum de la population présumer satisfaisante dans le cas où les gènes sont de type entier ou binaire.
— **double generation_satisfaisante_flottant :** la note d'évaluation moyenne minimum de la population présumer satisfaisante dans le cas où les gènes sont de type flottant.
— **int type_genes :** soit entier, double ou bien binaire.
— **int min_intervalle :** valeur minimale que peux prendre un gène dans le cas où ils sont de type entier ou bien binaire.
— **int max_intervalle :** valeur maximale que peux prendre un gène dans le cas où ils sont de type entier ou bien binaire.
— **double min_intervalle_flottant :** valeur minimale que peux prendre un gène dans le cas où ils sont de type flottant.
— **double max_intervalle_flottant :** valeur maximale que peux prendre un gène dans le cas où ils sont de type flottant.
— **std : :vector<int> note_moyenne :** tableau dynamique contenant la note d'évaluation moyenne de chaque génération généré, dans le cas où les gènes sont de type entier ou bien binaire.
— **std : :vector<double> note_moyenne_flottant :** tableau dynamique contenant la note d'évaluation moyenne de chaque génération généré, dans le cas où les gènes sont de type flottant.
— **std : :vector<int> meilleur_individu :** tableau dynamique contenant la note d'évaluation du meilleur individu de chaque génération généré, dans le cas où les gènes sont de type entier ou bien binaire.
— **std : :vector<double> meilleur_individu_flottant :** tableau dynamique contenant la note d'évaluation du meilleur individu de chaque génération généré, dans le cas où les gènes sont de type flottant.
— **int maximisation_minimisation :** indique si l'utilisateur souhaite une maximisation ou bien une minimisation du critère choisie.

7.0.2 Constructeurs :

Le constructeur **EntreesSorties(std : :string nom_fichier)** est utiliser pour charger une configuration déja sauvegarder dans un fichier.

Le constructeur EntreesSorties(std : :string chaine_evaluation, int taille_population, int nombre_iterations, int nmbr_indiv_a_selec, int choix_selection , int nombre_genes, float taux_croisement, float taux_mutation, int generation_satisfaisante, int type_genes, int min_intervalle, int max_intervalle,
int maximisation_minimisation) est utiliser lorsque l'utilisateur saisie les données de la configuration initiale et spécialment lorsque le type des gènes est soit entier ou bien binaire.

Le constructeur EntreesSorties(std : :string chaine_evaluation, int taille_population, int nombre_iterations, int nmbr_indiv_a_selec, int choix_selection , int nombre_genes, float taux_croisement, float taux_mutation, double generation_satisfaisante, int type_genes, double min_intervalle, double max_intervalle, int maximisation_minimisation) est utiliser lorsque l'utilisateur saisie les données de la configuration initiale et spécialment lorsque le type des gènes est flottant.

7.1 Les Méthodes :

— **void sauvegarde(std : :string nom_fichier, std : :vector<individu>* individus)** utiliser pour sauvegarder la configuration actuelle.
— **void genererLatex(std : :vector<individu>* individus, std : :string nom_fichier)** utiliser pour générer le fichier latex illustrant les statistique de l'évolution des systèmes.

7.2 Les Accésseurs :

— **std : :string getChaineEvaluation() :** retourne l'attribut chaine_evaluation.
— **int getTaillePopulation() :** retourne l'attribut taille_population.
— **int getNombreiterations() :** retourne l'attribut nombre_iterations.
— **int getNmbr_indiv_a_selec() :** retourne nmbr_indiv_a_selec.
— **int getChoixSelection() :** retourne l'attribut choix_selection.
— **int getNombreGenes() :** retourne l'attribut nombre_genes.
— **float getTauxCroisement() :** retourne l'attribut taux_croisement.
— **float getTauxMutation() :** retourne l'attribut taux_mutation.
— **int getGenerationSatisfaisante() :** retourne l'attribut generation_satisfaisante.
— **double getGenerationSatisfaisanteFlottant() :** retourne l'attribut generation_satisfaisante_flottant.
— **int getTypeGenes() :** retourne l'attribut type_genes.
— **int getMinIntervalle() :** retourne l'attribut min_intervalle.
— **int getMaxIntervalle() :** retourne l'attribut max_intervalle.
— **double getMinIntervalleFlottant() :** retourne l'attribut min_intervalle_flottant.
— **double getMaxIntervalleFlottant() :** retourne l'attribut max_intervalle_flottant.
— **std : :vector<int> getNoteMoyenne() :** retourne l'attribut note_moyenne.
— **std : :vector<double> getNoteMoyenneFlottant() :** retourne l'attribut note_moyenne_flottant.
— **std : :vector<int> getMeilleurIndividu() :** retourne l'attribut meilleur_individu.
— **std : :vector<double> getMeilleurIndividuFlottant() :** retourne l'attribut meilleur_individu_flottant.
— **int getMaximisationMinimisation() :** retourne l'attribut maximisation_minimisation.

7.3 Les Mutateurs :

— **void setChaineEvaluation(std : :string chaine_evaluation) :** modifie l'attribut chaine_evaluation.
— **void setTaillePopulation(int taille_population) :** modifie l'attribut taille_population.
— **void setNombreiterations(int nombre_iterations) :** modifie l'attribut nombre_iterations.
— **void setNmbr_indiv_a_selec(int nmbr_indiv_a_selec) :** modifie l'attribut nmbr_indiv_a_selec.
— **void setChoixSelection(int choix_selection) :** modifie l'attribut choix_selection.
— **void setNombreGenes(int nombre_genes) ; :** modifie l'attribut nombre_genes.
— **void setTauxCroisement(float taux_croisement) :** modifie l'attribut taux_croisement.
— **void setTauxMutation(float taux_mutation) :** modifie l'attribut taux_mutation.
— **void setGenerationSatisfaisante(int generation_satisfaisante) :** modifie l'attribut generation_satisfaisante.
— **void setGenerationSatisfaisanteFlottant(double generation_satisfaisante) :** modifie l'attribut generation_satisfaisante_flottant.
— **void setTypeGenes(int type_genes) :** modifie l'attribut type_genes.
— **void setMinIntervalle(int min_intervalle) :** modifie l'attribut min_intervalle.
— **void setMaxIntervalle(int max_intervalle) :** modifie l'attribut max_intervalle.
— **void setMinIntervalleFlottant(double min_intervalle) :** modifie l'attribut min_intervalle_flottant.
— **void setMaxIntervalleFlottant(double max_intervalle) :** modifie l'attribut max_intervalle_flottant.
— **void setNoteMoyenne(std : :vector<int> note_moyenne) :** modifie l'attribut note_moyenne.
— **void setNoteMoyenneFlottant(std : :vector<double> note_moyenne) :** modifie l'attribut note_moyenne_flottant.
— **void setMeilleurIndividu(std : :vector<int> meilleur_individu) :** modifie l'attribut meilleur_individu.
— **void setMeilleurIndividuFlottant(std : :vector<double> meilleur_individu) :** modifie l'attribut meilleur_individu_flottant.
— **void setMaximisationMinimisation(int maximisation_minimisation) :** modifie l'attribut maximisation_minimisation.

FIGURE 8 – La classe Modelisation Probleme

8 Modelisation Probleme :

Ce module permettra à l'utilisateur de saisir toutes les données nécessaires pour configurer notre algorithme génétique, de l'exécuter et d'observer les résultats statistiques, il permettra aussi de sauvegarder et de charger

une configuration.

Plusieurs outils sont mis en place à ce niveau pour que l'utilisateur contrôle la simulation, ainsi notre interface servira de tableau de bord pour notre application.

8.0.1 les constructeurs :

ModelisationProbleme() c'est le constructeur utiliser pour instantier notre fenêtre modélisation du problème et de configurer son paramétrage en utilisant les attributs.

8.0.2 les attribut :

- L'utilisateur saisira le **nombre de gènes** grace à un **QSpinBox**.
- La **taille de la population** aussi sera récuperer grace à un **QSpinBox**.
- Le **nombre d'itérations** aussi subira le meme traitement, ainsi que le **nombre d'individus et le nombre d'individus à séléctionnés "QSpinBox"**.
- le Le **taux de croisement et le taux de mutation** sont de type **QDoubleSpinBox**.
- configurer, sauvegarder, charger, demarrer_simulation, pause, reprendre, reconfigurer, acceuil, aide, quitter et consulter seront des boutons de type **QPushButton**
- layout_configurer, layout_sauvegarder, layout_charger, layout_demarrer_simulation, layout_pause, layout_reprendre, layout_reconfigurer, layout_acceuil, layout_aide, layout_quitter et layout_consulter sont de type **QHBoxLayout**.
- QVBoxLayout *layout_nombre_genes, layout_taille_population, layout_nombre_iterations, layout_nombre_individu_selectionnes, layout_type_genes, layout_type_genes_principale, layout_choix_selection, layout_maximisation_minimisation, layout_maximisation_minimisation_principale, layout_chaine_evaluation, layout_min_intervalle, layout_max_intervalle, layout_generation_satisfaisante, layout_taux_croisement, layout_taux_mutation, layout_nom_fichier_sauvegarde, layout_nom_fichier_latex, layout_nom_fichier_chargement est de type **QVBoxLayout** ;
- Layout_conteneur est le layout principale qui contiendra tout les autres, il est de type **QGridLayout**.
- label_nombre_genes, label_taille_population, label_nombre_iterations, label_nombre_individu_selectionnes, label_choix_selection,label_chaine_evaluation, label_min_intervalle, label_max_intervalle, label_generation_satisfaisante , label_taux_croisement, label_taux_mutation, label_nom_fichier_sauvegarde, label_nom_fichier_latex t label_nom_fichier_chargement sont des étiquettes de type **QLabel**.
- L'attribut type_genes est de type **QGroupBox** il contient 3 **QRadioButton** : "entier", "flottant" et "binaire".
- L'attribut maximisation_minimisation est de type **QGroupBox** il contient 2 **QRadioButton** : "maximisation" et "minimsation".
- L'attribut choix_selection est de thype **QComboBox**.
- chaine_evaluation, min_intervalle, max_intervalle, generation_satisfaisante, nom_fichier_sauvegarde, nom_fichier_latex et nom_fichier_chargement sont des champus de texte de type **QLineEdit**.

8.0.3 les méthodes :

Ces méthodes sont utilisées dans les signals/slots dans le cas où un bouton ou un événement surgit.

- **void connectConfiguration() :** Permet de configurer l'algorithme avec les données saisies par l'utilisateur et de lancer l'éxécution.
- **void connectReConfiguration() :** permet de changer la configuration de l'algorithme.
- **void connectSauvegarde() :** permet de sauvearder la configuration actuelle.
- **void connectChargement() :** permet de charger une configuration déja sauvegarder.
- **void connectLancement() :** permet de lancer l'éxécution.
- **void connectPause() :** permet mettre en attente l'éxécution.
- **void connectReprendre() :** reprendre l'éxécution de l'algorithme si elle a été mise en attente.
- **void connectAcceuil() :** retour à la page d'acceuil.

— **void connectAide() :** fournit de l'aide pour l'utilisation de l'application.
— **void connectQuitter() :** quitter l'application.
— **void connectConsulter() :** consulter les statistiques générées.

9 Voyageur de Commerce

Notre premier exemple pour tester la généricité de l'algorithme est la résolution du problème du voyageur de commerce avec notre algorithme génétique. Ce module est constitué d'une seule classe VoyageurDeCommerce, qui est à la fois l'interface et l'algorithme traitant le problème. Ce module est constitué d'une seule classe VoyageurDeCommerce, qui est à la fois l'interface et l'algorithme traitant le problème.

FIGURE 9 – La classe Voyageur

9.1 les méthodes :

- void connectConfiguration() : Permet de configurer l'algorithme avec les données saisies par l'utilisateur et de lancer l'exécution.
- void connectReConfiguration() : permet de changer la configuration de l'algorithme.
- void connectSauvegarde() : permet de sauvearder la configuration actuelle.
- void connectChargement() : permet de charger une configuration déja sauvegarder
- void connectLancement() : permet de lancer l'éxécution
- void connectAcceuil() : retour à la page d'acceuil.
- void connectAide() : fournit de l'aide pour l'utilisation de l'application.
- void connectQuitter() : quitter l'application.

9.2 les constructeurs :

- VoyageurDeCommerce() ;

9.3 les attribut :

private :

- int** graphe : c'est la structure qui nous permettra de modélisé le graphe souhaiter par l'utilisateur elle contiendra le poids des arêtes respectives de chaque sommet
- // vector<individu> individus : c'est la structure (l'enregistrement) qui hébergera notre populations
- L'utilisateur saisira le **nombre_sommets** grace à un **QSpinBox**.
- La **taille de la population** aussi sera récupérer grace à un **QSpinBox**.
- Le **nombre d'itérations** aussi subira le meme traitement, ainsi que le **nombre d'individus et le nombre d'individus à séléctionnés** "**QSpinBox**".
- le Le **taux de croisement et le taux de mutation** sont de type **QDoubleSpinBox**.
- label_generation_satisfaisante,label_nombre_sommets,label_taille_population, label_nombre_iterations,label_individu_selectionnes,label_taux_croisement,label_taux_mutation, layout_nom_fichier_sauvegarde, layout_maximisation_minimisation, layout_maximisation_minimisation_principale,layout_sauvegarder,layout_configurer, label_nom_fichier_sauvegarde, label_sauvegarder, label_charger,label_demarrer_simulation,label_reconfigurer,label_acceuil, label_aide,label_quitter,label_arc1,label_arc2,label_arc3,label_arc4,label_arc5,label_arc6,label_arc7, label_arc8,label_arc9,label_arc10 sont des étiquettes de type **QLabel**.
- layout_generation_satisfaisante,layout_quitter,layout_aide,layout_acceuil,layout_charger, layout_demarrer_simulation,layout_arc1,layout_arc2,layout_arc3,layout_arc4,layout_arc5, layout_arc6,layout_arc7,layout_arc8,layout_arc9,layout_arc10,layout_reconfigurer sont de type **QHBoxLayout**.
- layout_nombre_sommets,layout_taille_population,layout_nombre_iterations, layout_individu_selectionnes,layout_taux_croisement,layout_taux_mutation, layout_nom_fichier_chargement sont de type **QVBoxLayout**.
- Layout_conteneur est le layout principale qui contiendra tout les autres, il est de type **QGridLayout**.
- quitter,aide,acceuil,configurer,sauvegarder,charger ,demarrer_simulation,reconfigurer seront des boutons de type **QPushButton**
- nom_fichier_sauvegarde,nom_fichier_chargement ;r_chargement,generation_satisfaisante,arc1,arc2, arc3,arc4,arc5,arc6,arc7,arc8,arc9,arc10 sont des champus de texte de type **QLineEdit**.
- L'attribut maximisation_minimisation est de type **QGroupBox** il contient 2 **QRadioButton** : "maximisation" et "minimsation".

10 Problème des Huit Dames :

Nous implémenterons le problème des huit dames, qui consiste à trouver une combinaison de positions pour 8 reines d'un jeu d'échecs (dotée de la liberté de ce déplacer verticalement, horizontalement et sur les diagonales.

Cette partie sera constituée d'une sous partie algorithmique représentant la classe ProblemeDesHuitDames, et d'une deuxième partie graphique, que nous allons implémentée dans les classes BoxEchiquier, Echiquier, InterfaceHuitDames et Piece.

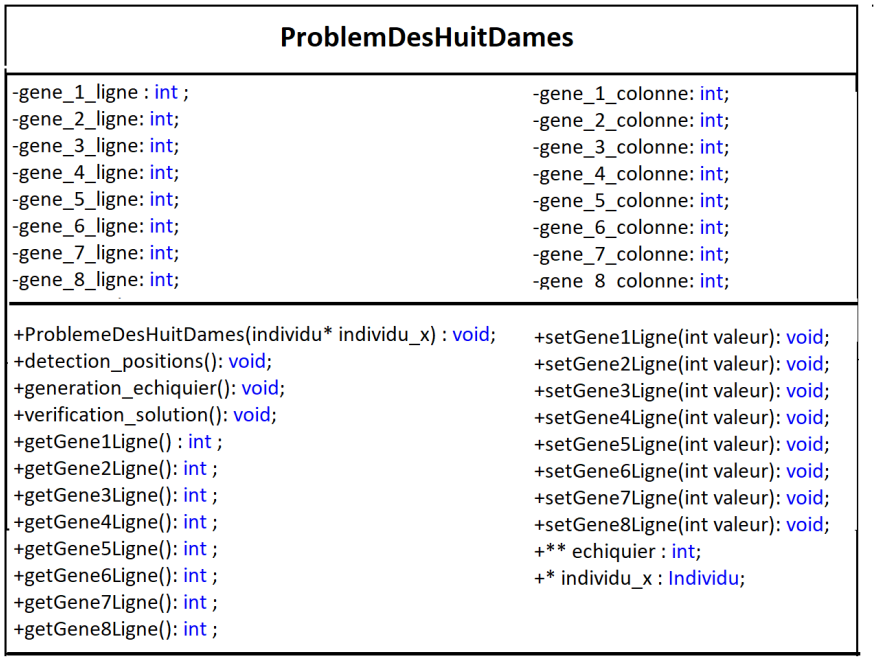

FIGURE 10 – La classe Problem Des Huit Dames

10.1 La classe Echiquier :

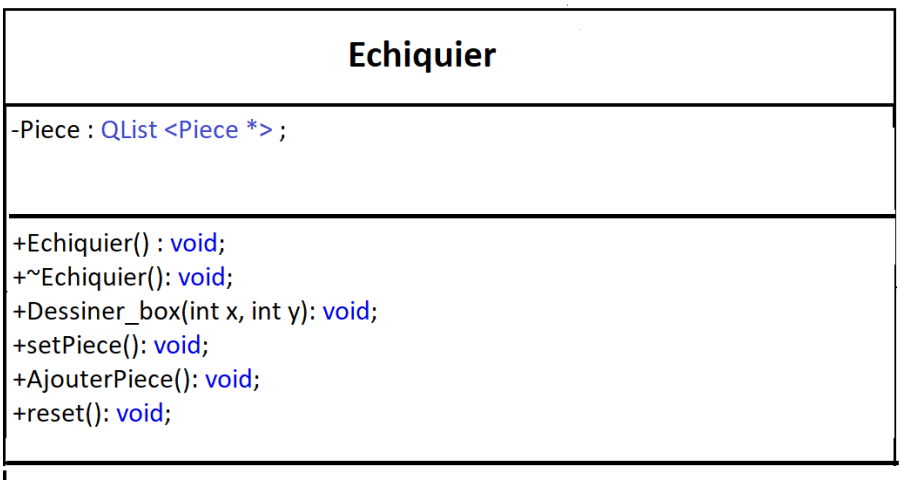

FIGURE 11 – La classe Echequier

10.1.1 les constructeurs :

Echiquier() est le constructeur par defaut.

10.1.2 Attribut :

— **QList <Piece *> Piece** représente la liste des 8 reins de notre échiquier.

10.1.3 les méthodes :

- **void Dessiner_box(int x, int y) :** déssine toutes les cases de notre échiquier.
- **void setPiece() :** Crée les piéces et les initialises.
- **void AjouterPiece() :** insére les pièces sur l'échiquier.
- **void reset() :** supprime toutes les pièces de l'échiquier ?

10.2 La classe BoxEchiquier :

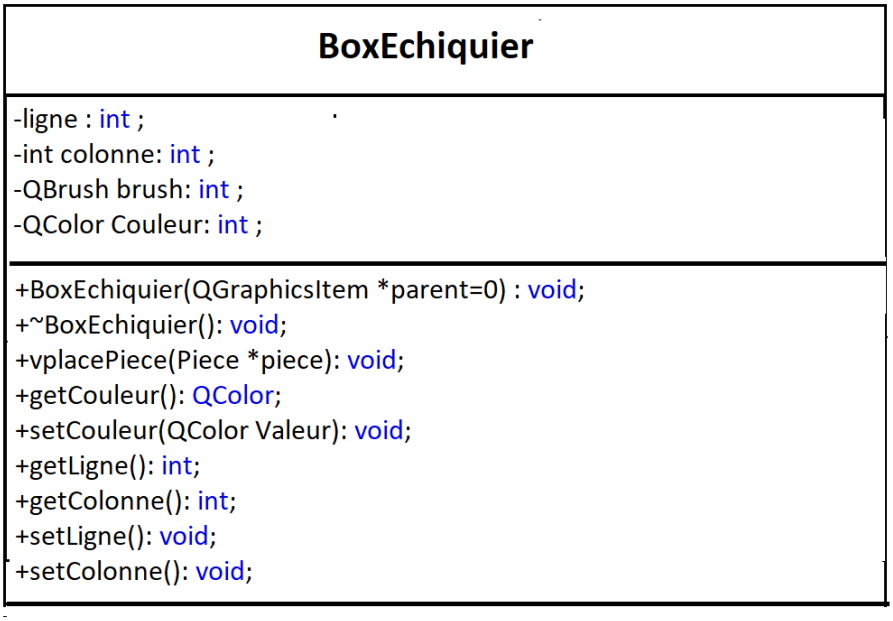

FIGURE 12 – La classe Box Echiquier

10.2.1 les constructeurs :

BoxEchiquier(QGraphicsItem *parent=0) est le constructeur utiliser, il permet de déssiner une case de l'échiquier.

10.2.2 les attribut :

- Piece * PieceActuelle : pièces présentes sur l'échquier.
- int ligne : nombre de lignes dans l'échiquier.
- int colonne : nombre de colonnes dans l'échiquier.
- QBrush brush : outil utiliser pour le déssin.
- QColor Couleur : couleur des cases de l'échiquier.

10.2.3 Les méthodes :

- **void placePiece(Piece *piece)** : place la piéce séléctionné sur l'échiquier.
- **QColor getCouleur() :** accésseur de l'attribut Couleur.
- **void setCouleur(QColor Valeur)** : mutateur de l'attribut Couleur.
- **int getLigne() :** accesseur de l'atribut ligne.
- **int getColonne() :** accésseur de l'atribut colonne.
- **void setLigne(int valeur) :** mutateur de l'attribut ligne.
- **void setColonne(int valeur) :** mutateur de l'attribut colonne.

FIGURE 13 – La classe InterfaceHuitDames

10.3 La classe InterfaceHuitDames :

10.3.1 les constructeurs :

InterfaceHuitDames(QWidget *parent = 0) est le constructeur de notre classe, il permet d'initialiser et de configurer le paramétrage nécessaire pour enfin dessiner l'intégralité de notre échiquier.

10.3.2 Les attribut :

- **QGraphicsScene *Scene_du_jeu :** Conteneur principale.
- **Echiquier *Echiquier :** objet Echiquier.
- **QList <QGraphicsItem *> listG :** elle regroupe les différents items de notre interface.
- **QVBoxLayout *layout_taille_population :**
- label_nombre_iterations, label_acceuil, label_taille_population, label_aide, label_reconfigurer, label_lancement, label_quitter sont des étiquettes de type **QLabel**.
- layout_quitter, layout_aide, layout_acceuil, layout_reconfigurer et layout_lancement sont de type **QHBoxLayout**.

— reconfigurer, lancement, acceuil, aide et quitter sont des boutons de type **QPushButton**.
— taille_population et nombre_iterations sont de type **QSpinBox**
— layout_taille_population, layout_nombre_genes, layout_nombre_iterations, layout_maximisation_minimisation et layout_maximisation_minimisation_principale sont de type
— **vector<individu> individus :** est la structure hébérgeant notre population.
— **maximisation_minimisation :** est de type QGroupBox et contient 2 QRadioButton "maimisation" et "minimisation".

10.3.3 les méthodes :

— **void DessinerEchiquier() :** déssine l'échiquier.
— **void AjouterSurScene(QGraphicsItem *item) :** ajoute un item dans le conteneur.
— **void AffichageEchiquier() :** affiche notre échiquier.
— **void RetirerPiece() :** supprime une dame de l'échiquier.
— **BoxEchiquier *matrice[8][8] :** tableau multidimentieonnel contenant les positions des dames sur l'échiquier.
— **QList <Piece *> pieceActuelle :** les pièces présentes sur l'échiquier.
— **void connectLancement() :** lance la simulation.
— **void connectAcceuil() :** retourne à l'acceuil.
— **void connectReConfiguration()** reconfigure le paramètrage.
— **void connectAide() :** aide pour l'utilisation de l'application.
— **void connectQuitter() :** Pour quitter l'interface.

10.4 La classe Piece :

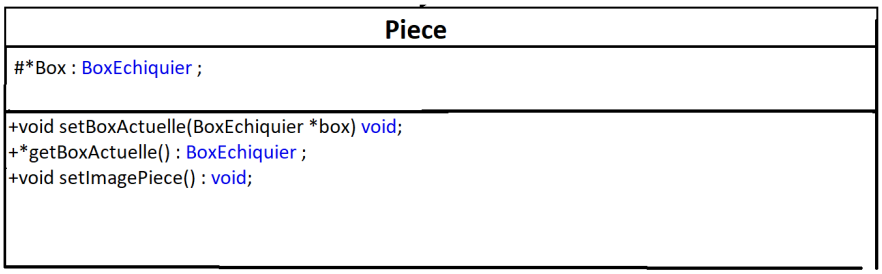

FIGURE 14 – La classe Piece

10.4.1 les méthodes :

— void setBoxActuelle(BoxEchiquier *box) : modifier la case ou se trouve la Piece.
— BoxEchiquier *getBoxActuelle() : retourne la case ou se trouve la Piece.
— void setImagePiece() : initialise l'image de la Piece dans notre cas c'est la Reine.

10.4.2 les constructeurs :

— Piece(QGraphicsItem *parent = 0) : initialise et dessine la Piece.

10.4.3 les attribut :

protected :
— BoxEchiquier *Box : la case ou se trouve la Piece.

11 Choix du langage :

Nous utiliserons le langage C++ pour développer notre application,notre choix s'explique dans les différentes utilisations de l'aspect fondamentale de ce langage. Premièrement, l'utilisation dans la notion d'objet, cela peut être observé dans les différentes classes de notre programme tel que l'interface, les individus, les gènes etc ... , pour chaque classe nous utilisons des méthodes qui décrivent le comportement de celle-si.Les variable utilisées dans nos classes sont dans la majorité des cas privés nous utilisons le principe d'encapsulation pour pouvoir modifier ses dernières, la surcharge de méthode est aussi utilisé dans le test d'arrêt par exemple. Pour ce qui est de l'héritage il est aussi grandement utilisé dans notre application principalement pour l'interface.On a aussi utilisé l'aspect procédural on faisant passer par exemple notre population de classe en classe grace aux pointeurs. Les différents points déclaré ci-dessus explique et consolide notre choix d'utiliser le langage C++

12 Conclusion :

Nous avons décrit dans ce cahier des spécifications le contenu précis de notre application, celle-ci est divisée en plusieurs modules et pour chaque module nous avons décrit les types et priorités des attributs ainsi que les méthodes. En conséquent nous avons maintenant un schéma clair et précis de l'application que nous allons développer, ce qui nous permettra de réaliser un travail propre