# *Bibliothèque ''Genetics''*

# *de CAPSIS*

## Documentation

I. Seynave, C. Pichot - mars 2004

# SOMMAIRE

## Introduction

Capsis est un simulateur de dynamique de peuplements forestiers avec élaboration d'itinéraires sylvicoles (http://membres.lycos.fr/coligny/). La version Capsis4 permet de développer des modèles de croissance de type "Arbre Indépendant des Distances" (MAID) et "Arbre Dépendant des Distances" (MADD). Grâce à sa structure modulaire, Capsis4 peut s'appliquer à de nombreuses situations forestières. En outre, les outils d'interventions qu'elle propose permettent de l'utiliser dans une démarche d'établissement de scénarios sylvicoles.

L'ajout de la bibliothèque *Genetics* dans la plateforme Capsis4 a pour objet de fournir des **outils génériques** permettant l'intégration dans différents modules :

- des données génétiques, qualitatives et quantitatives, en complément des données dendrométriques déjà intégrées par les différents modules,
- et des processus de flux de gènes (dispersion des pollens et des graines, reproduction, phénologie) en interaction avec les processus de dynamique forestière (croissance, régénération et mortalité).

La bibliothèque prend en charge la gestion des génotypes ainsi que les processus de reproduction (méiose, fécondation). Elle a été développée dans un contexte **d'arbres individualisées et spatialisées** (dans Capsis, catégorie des modules MADD). Les simulations sont envisagées à **court ou moyen terme** (faible nombre de générations) ; certains processus génétiques, comme les mutations, ne sont donc pas pris en compte. La structure des données génétiques et les processus de reproduction actuellement développés dans la bibliothèque sont adaptés aux espèces diploïdes à reproduction sexuée conforme (hérédité biparentale de l'ADN nucléaire). Enfin, **les génotypes sont discrétisés en loci** portant des allèles neutres ou avec une valeur adaptative.

# 1 Principes généraux de fonctionnement et d'utilisation

Ce paragraphe a pour objet d'introduire toutes les notions utilisées dans la bibliothèque (par exemple, qu'est ce qu'un génotype dans la bibliothèque ?), et les principes de calcul des différentes grandeurs génétiques (à l'échelle d'un arbre ou d'une population), notamment les hypothèses génétiques qui sont faites.
Il présente ensuite les grandes lignes pour l'utilisation de la bibliothèque : les données à fournir, les conventions à respecter ainsi que les grands types d'opérations réalisées par la bibliothèque (validation des données et générations de données génétiques).

## *1.1 Description générale de la bibliothèque*

### 1.1.1 Définition d'un arbre génétique

La bibliothèque *genetics* est conçue pour être connectée avec des modules de type MADD (Modèle Arbres Dépendant des Distances). L'unité élémentaire est donc l'arbre et la première fonction de la bibliothèque est de définir un arbre génétique (*GeneticTree*).

L'arbre génétique est un arbre spatialisé génotypé. Il hérite des variables d'instance et des méthodes de l'arbre spatialisé *(GMaddTree*[1]). Il possède en plus des variables d'instance génétiques qui lui confère son statut d'arbre génétique :

- un génotype et un multigénotype (information génétique discrétisée en loci)
- les identifiants de la mère et du père
- sa date de création
- ses coefficients de consanguinité individuel et global
- les valeurs génétiques d'un ou plusieurs paramètres[2]
- les valeurs environnementales d'un ou plusieurs paramètres
- les valeurs phénotypiques d'un ou plusieurs paramètres

L'arbre génétique possède des méthodes qui calculent les valeurs de ces variables d'instance, à l'exclusion de la date de création et des identifiants des parents qui sont communiqués à la bibliothèque par le module. A chaque arbre est également associé (au niveau du module) un effectif qui peut être supérieur à « 1 » lorsque l'arbre représente une population.

Pour pouvoir être utilisée dans des contextes variables et à diverses échelles, la bibliothèque est conçue pour gérer simultanément différents types d'arbres génétiques :

- l'arbre génétique individuel,
- l'arbre génétique non génotypé : toutes les valeurs des variables génétiques sont inconnues : ce type d'arbre permet la prise en compte dans le module d'espèces non étudiées sur le plan génétique.
- l'arbre génétique moyen : ce type d'arbre permet la prise en compte dans les simulations de peuplements (ou populations) génotypés et non individualisés.

[1] GMaddTree est un arbre spatialisé. Il est décrit dans capsis kernel (Madd = Model Arbre Dependant des Distances).
[2] Un paramètre peut être par exemple un paramètre de croissance ou de phénologie intervenant dans le module.

Les variables d'instance d'un arbre génétique et leur principe de calcul sont décrits ci-dessous pour chaque type d'arbres géré par la bibliothèque.

#### *1.1.1.1 L'arbre génétique individuel*

L'arbre génétique individuel a deux parents, une mère et un père, et une date de création. Ces informations permettent de reconstruire sa généalogie. Elles sont communiquées à la bibliothèque par le module puisque celui-ci prend en charge le processus de régénération.

Son génotype est discrétisé en allèles portés par différents loci sur l'ADN nucléaire, et les ADN cytoplasmiques d'origine maternelle et paternelle. C'est un génotype individuel (*IndividualGenotype* pour Capsis). La bibliothèque le calcule à partir du génotype de ses parents en simulant les processus de méiose et fécondation.

Le coefficient de consanguinité d'un arbre individuel est égal au coefficient de parenté de ses parents. C'est la probabilité que deux gènes, à un locus donné, soient identiques par descendance ; c'est-à-dire qu'ils dérivent d'un même gène ancêtre. Dans *genetics,* c'est le coefficient de consanguinité individuel. La bibliothèque le calcule en remontant la généalogie de l'arbre jusqu'à retrouver ses ancêtres appartenant au peuplement initial (peuplement donné dans le fichier d'inventaire). Un arbre individuel n'a pas de coefficient de consanguinité global (il est égal à -1 par convention). Cette grandeur n'est définie que pour les arbres moyens (voir 1.1.1.3).

Les valeurs génétiques, environnementales et phénotypiques sont les valeurs de un ou plusieurs paramètres adaptatifs. Ces paramètres peuvent être par exemple un coefficient d'une équation d'un processus d'évolution, ou plus simplement un caractère directement exprimé par l'arbre (ex : précocité…).
La valeur génétique d'un paramètre est calculée par la bibliothèque sous l'hypothèse d'additivité des valeurs des allèles (voir 1.2.1.2 page 13) : la valeur génétique d'un paramètre est la somme des valeurs des allèles portés par les loci intervenants sur ce paramètre. On parle bien ici de la valeur adaptative d'un allèle et non de son effet dans la population qui dépend de sa fréquence.
La valeur environnementale représente l'effet du milieu sur le paramètre. Elle ne dépend que de la variabilité du milieu (voir 1.2.1.2 page 13).
La valeur phénotypique est la somme des valeurs génétiques et environnementales. L'hypothèse d'indépendance entre le génotype et le milieu est donc faite au sein de la bibliothèque ; d'éventuelles interactions peuvent être prises en compte au niveau du Module.

#### *1.1.1.2 L'arbre génétique non génotypé*

Un arbre non génotypé est un arbre génétique dont toutes les variables génétiques sont inconnues. Pour Capsis, elles seront égales à -1 ou null selon leur type.

#### *1.1.1.3 L'arbre génétique moyen*

Un arbre moyen est disponible pour permettre la prise en compte dans les simulations de peuplements (ou populations) génotypés mais non individualisés. Cet arbre moyen est un arbre génétique. En tant que tel, il possède les mêmes variables d'instance qu'un arbre génétique individuel.

Le génotype d'un arbre moyen contient les effectifs des allèles dans le peuplement. Par locus, la somme des effectifs alléliques est proportionnelle à l'effectif (= nombre d'arbres) du peuplement : une fois l'effectif pour l'ADN cytoplasmique et deux fois l'effectif pour l'ADN nucléaire (diploïdie).
C'est un génotype multiple (*MultiGenotype* pour Capsis).
Contrairement au génotype d'un arbre génétique individuel, le génotype d'un arbre moyen peut varier dans le temps. Dans la bibliothèque, l'évolution du génotype d'un arbre moyen dépend uniquement des variations de l'effectif du peuplement que l'arbre moyen représente.
Ainsi, si le processus de mortalité du module simule de la mortalité dans un peuplement non individualisé, l'effectif du peuplement baisse de n1 à n2. Par locus, le nombre d'allèles à enlever est (n1-n2) (ou 2 * (n1-n2)). Cette opération est faite en tenant compte des fréquences alléliques dans le peuplement avant mortalité. L'hypothèse sous-jacente est que la mortalité n'est pas génotype-dépendante.
Le module peut également simuler de la régénération dans un peuplement non individualisé. Dans ce cas, le module ne doit pas augmenter l'effectif du peuplement mais créer un nouveau peuplement qui contient uniquement la régénération. Il doit communiquer à la bibliothèque la liste des n couples de parents (arbres individuels ou moyens). La bibliothèque calcule le génotype du nouvel arbre moyen à partir des génotypes des parents et de leur contribution (= nombre de descendants par couple).

Un arbre moyen peut avoir 1 ou plusieurs couples de parents. Lorsque l'arbre moyen a plus de 1 couple de parents, les identifiants de la mère et du père seront égaux à -1 par convention. Le tableau des couples parents qui est communiqué par le module n'est pas stocké.

L'arbre moyen a, comme l'arbre génétique individuel, une date de création communiquée à la bibliothèque par le module.

Le coefficient de consanguinité individuel d'un arbre moyen est égal à la moyenne des coefficients de parenté des n couples de parents. Le coefficient de consanguinité global d'un arbre moyen est la probabilité pour que 2 gènes tirés aléatoirement dans la population soient identiques par descendance. Ce coefficient est défini pour permettre de calculer le coefficient de consanguinité d'un nouvel individu issu d'une reproduction intra MultiGenotype (voir 4.1.5.5).
Ces coefficients sont supposés invariables dans le temps même lorsque l'effectif de la population diminue.

Les valeurs génétiques, environnementales et phénotypiques sont calculées selon les mêmes principes et sous les mêmes hypothèses que pour l'arbre génétique individuel. La valeur génétique d'un arbre est la somme des valeurs des allèles pondérées par l'effectif des allèles.

**Dans la pratique** :

- un accesseur est disponible pour chaque variable d'instance. Ce sont les mêmes quel que soit le type d'arbre (Tableau 1)
- pour savoir si un arbre est un arbre moyen, le modélisateur doit tester le type de génotype porté par l'arbre.

- pour savoir si un arbre est non génotypé, le modélisateur peut utiliser la méthode isGenotyped ().

Tout se passe donc comme s'il n'existait qu'un seul type d'arbre.

**Tableau 1 : Liste des accesseurs sur les variables d'instance d'un arbre génétique**

| variable génétique | accesseur | |
|---|---|---|
| Genotype | *getGenotype ()* | p.31 |
| Identifiant de la mère | *getMId ()* | p.30 |
| Identifiant du père | *getPId ()* | p.30 |
| Date de création | *getCreationDate ()* | p.30 |
| Coefficient de consanguinité individuel | *getConsanguinity ()* | p.30 |
| Coefficient de consanguinité global | *getGlobalConsanguinity ()* | p.30 |
| Valeur génétique pour un paramètre | *getGeneticValue (nom du paramètre)* | p.30 |
| Valeur phénotypique pour un paramètre | *getPhenoValue (nom du paramètre)* | p.30 |

## 1.1.2 Les processus simulés par la bibliothèque

La bibliothèque ne simule que deux processus : la méiose et la fécondation.

### *1.1.2.1 La méiose*

La bibliothèque simule le génotype du gamète produit par un arbre. L'arbre producteur de gamète peut être un arbre génétique individuel ou moyen. Le génotype du gamète produit est toujours un génotype individuel.
Les allèles portés à chaque locus de l'ADN nucléaire sont issus d'un processus stochastique dépendant :

- des probabilités de recombinaison entre les loci successifs (uniquement probabilités de recombinaison d'ordre 1) si l'arbre producteur est un arbre génétique individuel
- des effectifs alléliques de l'arbre producteur si l'arbre producteur est un arbre génétique moyen. Dans ce cas, le déséquilibre de liaison est supposé nul : le tirage de l'allèle porté à un locus ne dépend que des effectifs des allèles à ce locus.

Les allèles portés par les loci de l'ADN cytoplasmique (maternel et paternel) sont directement transmis au gamète.

### *1.1.2.2 La fécondation*

La bibliothèque simule la fécondation en fusionnant les génotypes individuels de deux gamètes. Le génotype calculé est un génotype individuel.

Ces processus sont simulés par deux méthodes *getGamete ()* et *fuseGametes ()*. Ces deux méthodes sont appelées par la bibliothèque pour calculer le génotype d'un arbre.

**Dans la pratique**, le modélisateur n'appellera pas ces méthodes. Pour calculer le génotype d'un nouvel arbre individuel ou celui d'une nouvelle instance d'un arbre moyen, le modélisateur appellera la méthode *getGenotype ()*. Pour calculer le génotype d'un nouvel arbre moyen, il appellera la méthode *computeNewMultiGenotype ()* (Tableau 2). Ces deux méthodes appelleront *getGamete ()* et *fuseGametes ()*.

**Tableau 2 : méthodes utilisées pour le calcul du génotype d'un arbre.**

| action | méthode | paramètre | |
|---|---|---|---|
| Calcul du génotype d'un nouvel arbre individuel<br>Calcul du génotype d'un arbre moyen après éclaircie ou mortalité (nouvelle instance) | getGenotype () | Aucun | p. 30 |
| Calcul du génotype d'un nouvel arbre moyen | computeNewMultiGenotype() | Tableau des parents du nouvel arbre moyen | p. 33 |

## *1.2 Utilisation de la bibliothèque*

### 1.2.1 Les entrées de la bibliothèque : description d'un fichier d'inventaire

Le fichier d'inventaire doit contenir la liste des arbres du peuplement initial ainsi que des informations génétiques caractéristiques de (ou des) espèce(s).

#### *1.2.1.1 Les arbres du peuplement initial*

Un arbre du peuplement initial appartenant à une espèce étudiée génétiquement doit être décrit sur le plan génétique par (Figure 1) :

- son génotype
- les identifiants de ses parents
- sa date de création
- ses coefficients de consanguinité

Ces données sont classiquement lus dans un fichier de chargement (« fichier d'inventaire ») ; certaines peuvent être renseigné dans le Module au niveau de l'initialisation. Les identifiants des parents et la date de création peuvent être inconnus (pour capsis, ils sont égaux à -1). Les valeurs génétiques, environnementales et phénotypiques ne sont pas données dans le fichier de chargement : ce sont des valeurs calculées par la bibliothèque à partir du génotype et des données espèce.

| ADN nucléaire | ADN cytoplasmique maternel | ADN cytoplasmique paternel | mId | pId | Date de création | consanguinité |
|---|---|---|---|---|---|---|
| {2;2;3;4;3;4;2;2;2;2;2;3;3;3} | {} | {135;156} | -1 | -1 | -1 | 0.2 |

Genotype

**Figure 1 : Exemple des données d'entrée génétique d'un arbre génétique individuel du peuplement initial**

Un format de données standard est proposé pour l'écriture de ces données dans le fichier d'inventaire. Si ce format est respecté, des méthodes pourront être utilisées par l'utilisateur pour lire ces données et les stocker dans le format de la bibliothèque (voir 5.1).

#### *1.2.1.2 Les données espèce*

Des informations caractéristiques de l'espèce doivent être données en entrée car elles sont lues par la bibliothèque pour le calcul des différentes données génétiques d'un arbre génétique.

**La carte génétique de l'espèce**

Elle contient les probabilités de recombinaison entre les loci successifs (probabilités de recombinaison d'ordre 1). Elle est appelée dans la méthode *getGamete ()* qui simule la méiose.

**La diversité allélique de l'espèce**

C'est l'ensemble des allèles pouvant être portés sur les différents loci des ADN nucléaire, cytoplasmiques d'origine maternelle et paternelle. Elle est appelée dans de nombreuses méthodes de la bibliothèque et notamment dans la méthode *getGamete ()* lorsque celle-ci est appelée sur un arbre génétique moyen.

**Les coefficients de parenté entre les arbres du peuplement initial et le coefficient de parenté par défaut**

Pour tous les couples d'arbres du peuplement initial pour lesquels le coefficient de parenté est connu, il doit être donné dans le fichier d'inventaire. En plus un coefficient de parenté par défaut doit être donné. Ces valeurs sont utilisées par la bibliothèque pour le calcul des coefficients de consanguinité d'un arbre généré en cours de simulation.

**Les informations sur la valeur adaptative des allèles**

Ces informations sont :

- les valeurs des allèles (effets uniquement additifs)
- l'héritabilité théorique du paramètre
- la variance environnementale totale du paramètre $\sigma^2_{total}$
- la part de la variance environnementale inter Step $x\sigma^2_{\text{int}er}$

Les valeurs des allèles sont utilisées pour le calcul des valeurs génétiques. Elles sont supposées invariables dans le temps.

Les autres informations sont utilisées pour le calcul des valeurs environnementales et phénotypiques.

Le principe de calcul de la valeur environnementale est basé sur l'hypothèse que la variance environnementale d'un paramètre se décompose en une variance environnementale intra Step[3] et une variance environnementale inter Step.

Par exemple, si le paramètre étudié est la potentialité de croissance, la variance environnementale de ce paramètre est la somme :

- de la variance inter Step qui représente la variabilité annuelle de l'effet du milieu liée, par exemple, à des variations de pluviométrie d'une année à l'autre
- et de la variance intra Step qui représente la variabilité spatiale de l'effet du milieu liée, par exemple, aux variations des conditions stationnelles supposées constantes dans le temps et qui ne seraient pas prises en compte dans le module

La valeur environnementale est alors la somme de :

[3] Un Step est une étape dans un projet Capsis. Il porte un peuplement à une date donnée.

- la valeur environnementale fixe, tirée dans une loi normale de moyenne 0 et de variance la variance environnementale intra Step ($\sigma^2_{\text{intra}} = \sigma^2_{total}(1 - x\sigma^2_{\text{inter}})$),
- et de la valeur environnementale inter Step, tirée dans une loi normale de moyenne 0 et de variance la variance environnementale inter Step ($\sigma^2_{\text{inter}} = \sigma^2_{total}(x\sigma^2_{\text{inter}})$).

L'utilisateur doit toujours fournir la part de la variance environnementale inter Step. En revanche il peut soit fournir la variance environnementale totale du paramètre, soit l'héritabilité théorique du paramètre. La bibliothèque en déduira la variance environnementale totale d'après l'équation *héritabilité = variance génétique / (variance génétique + variance environnementale)*, la variance génétique étant calculée sur l'ensemble des arbres (individuels et moyens) du peuplement initial sous l'hypothèse de panmixie.

### 1.2.2 Modularité de la bibliothèque

La bibliothèque est générique. Parmi les informations génétiques :

- certaines sont obligatoires : tous les modules connectés à la bibliothèque doivent fournir les données d'entrées nécessaires à leur calcul,
- d'autres sont facultatives ou optionnelles : un module connecté à la bibliothèque pourra totalement les ignorer et n'appeler aucune des méthodes de la bibliothèque portant sur ces informations.
-

#### *1.2.2.1 Informations obligatoires*

C'est le minimum d'informations que doit porter un arbre pour être un arbre génétique. Un arbre génétique doit obligatoirement avoir un génotype, les identifiants de sa mère et de son père, ainsi que sa date de création.

Au niveau spécifique, les espèces doivent obligatoirement avoir une carte génétique et un tableau de leur diversité allélique.

#### *1.2.2.2 Informations facultatives*

En plus de ces trois paramètres, un arbre peut avoir :

- des coefficients de consanguinité
- des valeurs génétiques, environnementales et phénotypiques pour plusieurs paramètres.

Dans ce cas un certain nombre de données supplémentaires doivent être fournies à l'initialisation du module respectivement :

- la consanguinité des arbres du peuplement initial et le coefficient de parenté entre les arbres du peuplement initial.
- la valeur adaptative des allèles de chaque paramètre, héritabilité et/ou variance environnementale totale du paramètre ainsi que la part de la variance environnementale inter Step.

### 1.2.3 Possibilité de différer le calcul de certaines données génétiques

Les informations génétiques portées par les arbres occupent souvent une place mémoire très importante. Pour permettre d'économiser du temps de calcul et de la mémoire, le calcul de la plupart des informations est différé, notamment le calcul du génotype. Par exemple, si un arbre meurt avant que l'on ait besoin de connaître son génotype, celui-ci ne sera jamais calculé ni à fortiori stocké en mémoire.

### 1.2.4 Procédure d'initialisation

A chaque fois que le modélisateur crée un arbre ou une nouvelle instance d'un arbre, les variables d'instances de cet arbre doivent être initialisées. Les règles pour l'initialisation d'une donnée génétique (génotype, consanguinité …) dépendent de son niveau d'information (obligatoire ou facultatif, et de la possibilité ou non de différer le calcul) et également de l'arbre (arbre du peuplement initial ou arbre généré en cours de simulation).

#### *1.2.4.1 Initialisation des identifiants des parents et de la date de création*

**Ces données sont initialisées à la création de tout nouvel individu**, aussi bien pour les individus du peuplement initial que pour les individus générés par simulation. C'est pourquoi, ces valeurs sont des paramètres du constructeur de *GeneticTree.* Si l'une de ces données n'est pas donnée au constructeur, une erreur sera générée.

**Pour les arbres non génotypés, ces données sont initialisées à la** valeur –1 ou à null

**Pour les arbres moyens, les identifiants des parents sont initialisés à -1 lorsque l'arbre a plus de deux parents**. La date de création est initialisée à -1 pour les arbres du peuplement initial s'il elle n'est pas connue. Pour les arbres générés en cours de simulation, cette variable doit être égale à la date du peuplement dans lequel il est généré.

Pour les arbres individuels du peuplement initial, ces données peuvent être initialisées à -1 si elles ne sont pas connues. Par contre, **les arbres individuels générés en cours de simulation ont obligatoirement les identifiants de leurs parents et leur date de création différents de -1**.

**Dans la pratique** : pour initialiser la date de création d'un arbre généré en cours de simulation, l'utilisateur appelle la méthode *GTCstand.getDate ()*.

#### *1.2.4.2 Initialisation du génotype*

Le génotype est une donnée obligatoire, en tant que telle c'est un paramètre du constructeur de *GeneticTree*. La bibliothèque ne peut donc pas construire un arbre sans connaître son génotype.

1-Arbres non génotypés

**Pour les arbres non génotypés, le génotype est initialisé à null dès la création de l'arbre.**

2-Arbres génotypés

Pour permettre de différer le calcul du génotype, deux variables, l'une de type *IndividualGenotype* et l'autre de type *MultiGenotype*, ont été définies. Ces variables sont les constantes *Genotype.EMPTY_INDIVIDUAL_GENOTYPE* et *Genotype.EMPTY_MULTI_GENOTYPE*.
Un génotype initialisé à l'une de ces 2 constantes (selon le type individuel ou moyen) ne sera calculé qu'au premier appel de la méthode *getGenotype ()*.
Ces constantes peuvent également être utilisées pour initialiser un arbre individuel ou moyen génotypé du peuplement initial si son génotype n'est pas connu. La procédure de validation appelée à l'issue du chargement lui attribuera un génotype par défaut.

Arbres individuels :

**Pour les arbres individuels du peuplement initial, le génotype peut être initialisé à sa valeur (si elle est connue) ou à *Genotype.EMPTY_INDIVIDUAL_GENOTYPE*.** A l'issue de la procédure de validation, tous les *IndividualGenotype* vides auront été remplacés par un *IndividualGenotype* par défaut.

**Pour les arbres individuels générés en cours de simulation, le génotype est toujours initialisé à *Genotype.EMPTY_INDIVIDUAL_GENOTYPE*.** Le calcul et l'initialisation du génotype seront réalisés au premier appel de la méthode *getGenotype()*.

Arbres moyens :

- A la création d'un nouvel arbre moyen :

**Pour les arbres du peuplement initial, le génotype peut être initialisé à sa valeur ou à vide**. A l'issue de la procédure de validation, tous les *MultiGenotype* vides auront été remplacés par des *MultiGenotype* par défaut.

**Pour tous les arbres moyens générés en cours de simulation, le génotype doit être initialisé à sa valeur (c'est-à-dire non null et non vide)**, pour éviter de stocker la liste des parents. Ce génotype est calculé par la méthode *computeNewMultiGenotype ()* avant la création de l'arbre et est donné en paramètre au constructeur de *GeneticTree*. Le modélisateur doit donc : 1- construire le tableau des parents, 2- calculer le *MultiGenotype* et 3- construire le nouvel arbre.

- A chaque nouvelle instance de l'arbre moyen :

**Le génotype est automatiquement initialisé à *Genotype.EMPTY_MULTI_GENOTYPE* par le constructeur.** Le modélisateur peut le calculer et l'initialiser en appelant la méthode *getGenotype ()*.

#### *1.2.4.3 Initialisations des données facultatives*

Toutes les données facultatives sont initialisées par le constructeur de *GeneticTree* à null ou -1 si elles sont de type numérique). Ceci aussi bien pour les arbres du peuplement initial que pour les arbres générés en cours de simulation.
Leur calcul et leur initialisation sont réalisés au premier appel de leur accesseur.

### 1.2.5 Validation des données initiales

Les données génétiques ont des structures complexes ce qui implique des règles strictes dans la structure des fichiers d'inventaire. C'est pourquoi une procédure de validation a été mise en œuvre afin de tester les données génétiques à l'issue du chargement du fichier d'inventaire. Il est conseillé au modélisateur d'appeler la méthode *validate ()* à l'issue de sa phase de chargement.

Par ailleurs, l'acquisition des données génétiques est lourde. Les fichiers d'inventaire pourront contenir des informations manquantes. La procédure de validation des données initiales permet de remplacer automatiquement ces données manquantes par des données par défaut. Les données qui peuvent être définies par défaut par *validate ()* sont les suivantes :

- le génotype à compléter

- la carte génétique à compléter
- la diversité allélique à compléter

### 1.2.6 Génération de données initiales

A l'issue du chargement et avant toute simulation, une procédure de génération de données génétiques peut être appelée. Cette procédure permet de générer des génotypes nucléaires et/ou cytoplasmiques (loci et allèles), ainsi que les valeurs adaptatives des allèles tirés selon une loi de distribution (actuellement seule la loi Normale centrée et de variance 100 est proposée).

# 2 Données génétiques gérées par la bibliothèque

Les données génétiques traitées par la bibliothèque portent à la fois sur les arbres et sur les espèces. On peut les regrouper en trois thèmes :

- génotype et généalogie
- consanguinité
- valeur adaptative des allèles

Le génotype est totalement discrétisé en loci. Par espèce et par type d'ADN, tous les loci sont identifiés par un numéro d'ordre. Ce numéro n'est pas donné mais est déduit des données génétiques : c'est le numéro de la ligne dans les tableaux des génotypes et des *AlleleDiversity*. C'est pourquoi, il est important de toujours respecter le même ordre. Par la suite on parlera du numéro ou du rang d'un locus.

## *2.1 Génotype et généalogie*

### 2.1.1 Données arbre

#### *2.1.1.1 Le génotype*

Le génotype est la seule caractéristique de l'arbre qui diffère selon le type d'arbre, individuel ou moyen. Un arbre individuel possède un *IndividualGenotype* alors qu'un arbre moyen possède un *MultiGenotype*.

***IndividualGenotype***

Un *IndividualGenotype* est composé de 3 tableaux (Figure 2).

Le premier nommé *nuclearDNA*, est à deux dimensions et contient le génotype porté par l'ADN nucléaire. Ce tableau est constitué de n lignes (n correspondant au nombre de loci) et 2 colonnes, la première pour l'ADN issu de la mère et la seconde pour l'ADN issu du père. Chaque cellule du tableau contient un entier codant l'allèle du locus considéré.

Le second et le troisième tableaux (nommés respectivement *mCytoplasmicDNA* et *pCytoplasmicDNA)* sont à une dimension et contiennent respectivement le génotype porté par l'ADN cytoplasmique transmis par la mère contenu, selon l'espèce, dans les mitochondries et/ou dans le chloroplaste, et le génotype porté par l'ADN cytoplasmique transmis par le père (mitochondrial et /ou chloroplastique selon l'espèce). Ces tableaux sont constitués de n lignes (n étant le nombre de loci) et une colonne, chaque cellule contient un entier codant l'allèle à ce locus.

Pour les arbres dont l'espèce est prise en compte sur le plan génétique, deux des trois tableaux peuvent être vides.

Tous les arbres d'une même espèce doivent avoir le même format de génotype c'est-à-dire que le nombre de loci étudiés par type d'ADN (nucléaire, cytoplasmique maternel et paternel) est identique pour tous les arbres d'une espèce et que les loci sont toujours dans le même ordre.
Par conséquent un seul et même format de génotype devra être utilisé pour les espèces susceptibles de s'hybrider. La nature intra ou inter spécifique d'un croisement sera déterminer par l'origine des parents.

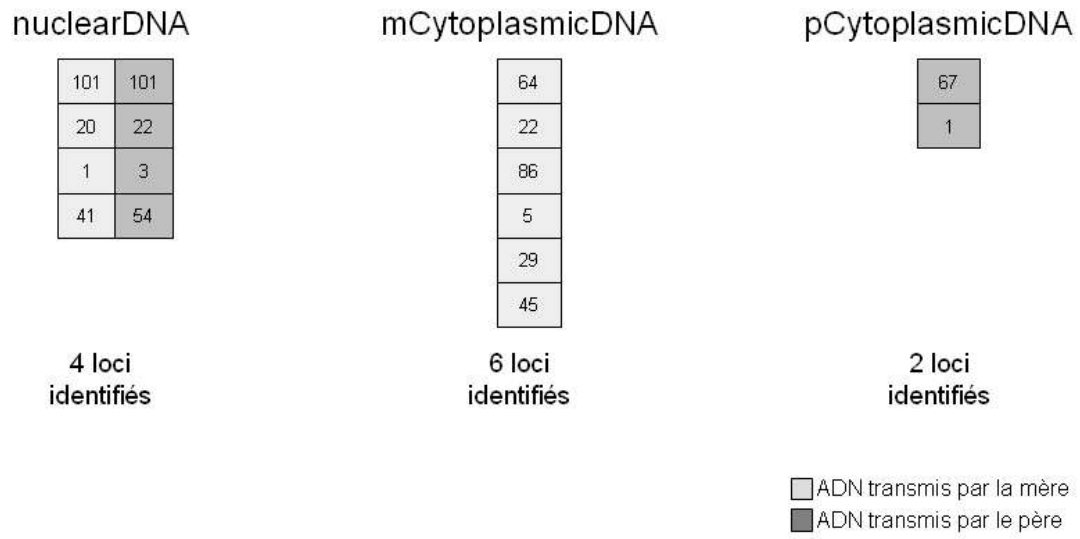


***Figure 2 : Exemple d'un IndividualGenotype***

### *MultiGenotype*

Comme l'*IndividualGenotype*, le *MultiGenotype* est composé de trois tableaux (Figure 3). Le premier code le génotype de l'ADN nucléaire, le second et le troisième codent respectivement le génotype des ADN cytoplasmique d'origine maternelle (mitochondrial et/ou chloroplastique selon l'espèce) et cytoplasmique d'origine paternelle (mitochondrial et/ou chloroplastique selon l'espèce).

Ces trois tableaux ont le même format. Ce sont des tableaux à 2 dimensions comportant autant de lignes que de loci étudiés sur l'ADN correspondant. Chaque ligne du tableau a une

longueur (nombre de colonnes) égale au nombre total d'allèles possibles pour le locus correspondant. La longueur des lignes diffère d'une ligne à l'autre, c'est-à-dire d'un locus à l'autre.

**ATTENTION: Dans la version actuelle de la bibliothèque, seules les fréquences alléliques (ou plus exactement les effectifs des allèles) pour chacun des loci sont connues au niveau d'un multigénotype. On suppose donc implicitement que les loci sont indépendants et qu'il n'existe pas de déséquilibre de liaison. Cette hypothèse vaut pour les 3 types d'ADN et se répercute au niveau de la production des gamètes et au niveau de la « mise à jour » des multigénotypes (actualisation du multigénotype lorsque l'effectif de l'arbre moyen diminue).**
**En ce qui concerne les ADN cytoplasmiques, il est conseillé (pour ne pas dire indispensable ) de coder les haplotypes sous forme d'allèles d'un seul locus. Les deux tableaux correspondant aux génotypes cytoplasmiques d'origine maternelle et paternelle auront donc généralement une seule ligne.**

Chaque cellule contient un entier égal à l'effectif de l'allèle dans la population représentée par l'arbre moyen. Le déséquilibre de liaison (association préférentielle d'allèles pour différents loci) est supposé nul. La somme des effectifs alléliques par locus est égal à 2*l'effectif de la population pour l'ADN nucléaire. Elle est égale à l'effectif de la population pour les ADN cytoplasmiques.

Ces tableaux sont respectivement nommés : *nuclearAlleleFrequency*, *mCytoplasmicAlleleFrequency* et *pCytoplasmicAlleleFrequency*.

Comme précédemment, le nombre de loci étudiés sur chaque ADN peut être différent. Chacun des trois tableaux peut être vide mais l'un au moins doit être renseigné, pour les arbres dont l'espèce est prise en compte sur le plan génétique.

Comme les *IndividualGenotype*, les *MultiGenotype* doivent être de format identique pour tous les arbres moyens d'une même espèce. De plus, si une espèce est prise en compte sur le plan génétique à la fois en tant qu'individus et en tant qu'arbres moyens, l'ordre des loci dans les *MultiGenotype* (c'est-à-dire le numéro de la ligne) doit être identique à l'ordre des loci dans les *IndividualGenotype*.

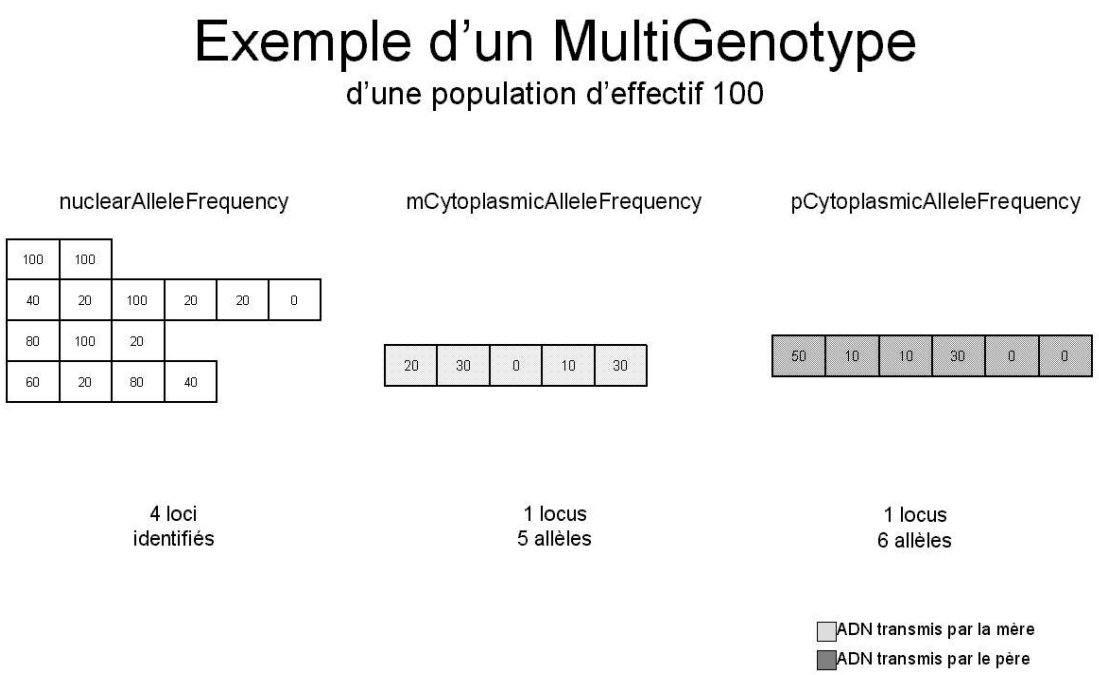


***Figure 3 : Exemple d'un MultiGenotype***

#### *2.1.1.2 Les identifiants de la mère et du père*

Ces identifiants sont toujours donnés dans le même ordre : d'abord l'identifiant de l'arbre mère (*mId*) ensuite l'identifiant de l'arbre père (*pId*). Ce sont tous les deux des entiers. Les arbres du peuplement initial, pour lesquels les parents sont inconnus, ont, par convention, les identifiants de la mère et du père égaux à -1. Un arbre moyen issu de plusieurs couples de parents a ses *mId* et *pId* égaux à -1.

#### *2.1.1.3 La date de création*

C'est l'entier définissant la date de création d'un arbre *(creationDate)*. La date de création d'un arbre n'est pas obligatoirement égale à sa date de naissance. En effet, dans les modules déjà implémentés dans Capsis, certaines relations de recrutement simulent l'apparition de nouveaux individus non pas dès la germination de la graine mais lorsqu'ils atteignent un seuil de dimension (h=30 cm, diamètre = 100 cm …).

Pour tous les arbres du peuplement initial, elle est égale, par convention, à -1 (valeur inconnue). Pour les arbres créés en cours de simulation, elle est définie au cours de l'exécution du processus de régénération comme égale à la date du peuplement obtenue (par la méthode *getDate ()*).

### 2.1.2 Données espèce

#### *2.1.2.1 GeneticMap*

La *GeneticMap* contient un tableau, appelé *recombinationProbas*, rassemblant les probabilités de recombinaison entre les loci successifs de l'ADN nucléaire. Il est constitué de n-1 lignes, n étant le nombre de loci étudiés sur l'ADN nucléaire. Chaque ligne i contient un réel, de type double, compris entre 0 et 0,5 qui représente la probabilité de recombinaison entre le $i^{ème}$ locus et le $(i+1)^{ème}$. L'ordre des probabilités de recombinaison dans *recombinationProbas* respecte l'ordre des loci tels qu'ils sont donnés dans le génotype des arbres (*IndividualGenotype* et *MultiGenotype*).

#### *2.1.2.2 AlleleDiversity*

*AlleleDiversity* contient 3 tableaux à 2 dimensions (Figure 4).

Le premier tableau contient tous les allèles possibles par locus de l'ADN nucléaire. Il est constitué de n lignes, n étant le nombre de loci étudiés. Chaque ligne du tableau comporte un nombre de colonnes variable égale au nombre total d'allèles potentiels sur le locus correspondant. Chaque cellule du tableau contient un entier, de type short, codant l'allèle. Ce tableau est nommé *nuclearAlleleDiversity*.

Les second et troisième tableaux sont construits selon le même principe. Ils correspondent respectivement à l'ADN cytoplasmique de la mère et du père. Ces tableaux sont respectivement appelés *mCytoplasmicAlleleDiversity* et *pCytoplasmicAlleleDiversity*. Lorsque les haplotypes sont des allèles d'un seul locus, ces tableaux n'ont qu'une ligne.

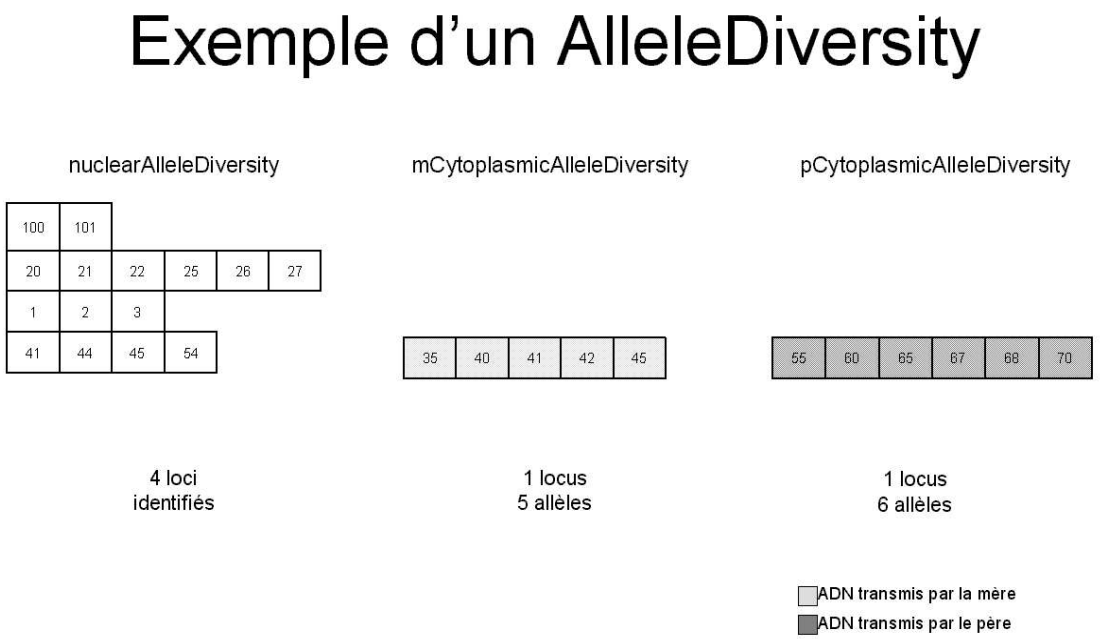


***Figure 4 : AlleleDiversity. Exemple.***

## *2.2 Consanguinité*

### 2.2.1 Données arbre

Un arbre possède deux coefficients de consanguinité : le coefficient de consanguinité individuel, appelé *consanguinity* et le coefficient de consanguinité global, appelé *globalConsanguinity*. Ce sont tous les deux des doubles.

Par convention, pour tous les arbres dont l'espèce n'est pas prise en compte sur le plan génétique, ses coefficients de consanguinité sont égaux à -1.
Pour un arbre individuel, le coefficient de consanguinité individuel est le coefficient de consanguinité de l'arbre et le coefficient de consanguinité global est égal à -1.
Pour un arbre moyen, le coefficient de consanguinité individuel est égal à la moyenne des coefficients de consanguinité des arbres qui constituent la population et le coefficient de consanguinité global est la probabilité que deux gènes tirés dans la population de gènes portés par le multigénotype soient identiques. Il est généralement inférieur au coefficient de consanguinité individuel.

### 2.2.2 Données espèce : l'apparentement

Le coefficient de parenté entre couple d'individus n'est pas intrinsèquement une donnée espèce. Il est utilisé pour calculer la consanguinité en remontant la généalogie. Cette procédure nécessite la connaissance des coefficients de parenté entre tous les individus (2 à 2) du peuplement initial, information enregistrée au niveau de l'espèce.

*Kinship* contient un tableau de doubles à 2 dimensions ainsi qu'un double.

Le tableau contient les couples d'individus du peuplement initial pour lesquels le coefficient de parenté est connu. Ce tableau contient autant de lignes que de couples. Il est composé de trois colonnes. Les deux premières colonnes contiennent les identifiants des arbres du couple (l'ordre n'a pas de sens : couple (a, b) = couple (b, a)). La troisième colonne contient le coefficient de parenté de a et b. Ce tableau est appelé *initialPhiArray*.

Le double donné dans *Kinship* est le coefficient de parenté par défaut. Lorsqu'un couple d'arbres n'est pas inclus dans le tableau *initialPhiArray*, le coefficient de parenté de ce couple est égal au coefficient de parenté par défaut. Le coefficient de parenté par défaut est appelé *defaultPhi*.

## *2.3 Valeur adaptative des allèles*

### 2.3.1 Données arbre : valeurs génétiques, environnementales et phénotypiques

Les valeurs génétiques, environnementales et phénotypiques sont stockées dans des Map[4] où : les clés correspondent aux paramètres étudiés (par exemple, un paramètre quantitatif qui agit sur la croissance ou la phénologie des arbres) et les valeurs correspondent (selon la Map) aux valeurs génétiques, environnementales ou phénotypiques des paramètres.

La Map des valeurs génétiques d'un individu contient les valeurs génétiques (ou « génotypiques » des différents paramètres quantitatifs étudiés. Par définition, ces valeurs étant invariables dans le temps, la Map des valeurs génétiques est également invariable dans le temps. Cette variable est appelée *genoValue*.

La Map des valeurs environnementales d'un individu contient les valeurs environnementales fixes (représentant les effets environnementaux constants) des différents paramètres étudiés. La Map des valeurs environnementales a été définie pour permettre la sauvegarde des valeurs environnementales fixes dès qu'elles sont calculées, ces valeurs étant utilisées pour le calcul des valeurs phénotypiques. La Map des valeurs environnementales fixes est nommée *fixedEnvironmentalValue*.

La Map des valeurs phénotypiques d'un individu contient les valeurs phénotypiques des différents paramètres à une étape (Step) donnée de l'évolution du peuplement. La valeur phénotypique d'un paramètre est égale à la somme de la valeur génétique et de la valeur environnementale totale (partie fixe + partie variable). Les valeurs phénotypiques d'un individu étant variables d'un Step à l'autre (puisqu'une partie de la valeur environnementale varie d'un Step à l'autre), pour un individu donné, une Map des valeurs phénotypiques peut être calculée à chaque Step. Cette Map est appelée *phenoValue*.

**Contrairement à toutes les variables précédentes, les valeurs de ces variables pour les arbres du peuplement initial ne sont pas données à l'initialisation du module mais**

[4] Dans le langage Java, une Map représente une liste d'objets, appelés clés, mis en correspondance avec une liste d'objets, appelés valeurs.

**peuvent être calculées à tout moment de la simulation, comme les Map des arbres créés en cours de simulation, à partir des valeurs des allèles définis à l'initialisation (voir AlleleEffect paragraphe 2.3.2).**

### 2.3.2 Données espèce : AlleleEffect

AlleleEffect est une Map. La liste des clés correspond à la liste des paramètres quantitatifs (exemples : croissance, phénologie …) étudiés. La valeur associée à une clé est un *ParameterEffect*, qui contient trois tableaux ainsi que trois réels.

Le premier tableau (appelé *nuclearAlleleEffect*) contient en ligne les loci de l'ADN nucléaire qui influencent le paramètre correspondant. Les lignes ont un nombre de colonnes variable et égal au nombre d'allèles possibles sur le locus considéré, noté n, +1.
Dans chaque ligne, la première valeur est égale à la position du locus dans le tableau des allèles possibles correspondant (*nuclearAlleleDiversity*), c'est-à-dire son numéro de ligne. Les n valeurs suivantes sont les valeurs des n allèles possibles du locus. Sur chaque ligne, la somme des n valeurs des valeurs des allèles est égale à 0.
Cette construction a été retenue car elle permet de faire le lien entre les tableaux des allèles possibles et les tableaux des valeurs (ce qui est nécessaire, par exemple, pour faire le calcul des valeurs génétiques individuelles) sans être obliger d'inclure dans le tableau des valeurs les loci qui n'influencent pas le paramètre.

Les second et troisième tableaux, respectivement *mCytoplasmicAlleleEffect* et *pCytoplasmicAlleleEffect*, sont construits sur le même principe et contiennent les loci des ADN cytoplasmiques, respectivement maternel et paternel, qui influencent le paramètre considéré.

Le premier réel, de type double, contenu dans *ParameterEffect* correspond à l'héritabilité théorique du paramètre ; cette variable est appelée *heritability*. Viennent ensuite un double correspondant à la variance environnementale totale, variable appelée *totalEnvironmentalVariance*, puis un double correspondant à la part de la variance environnementale inter Step dans la variance environnementale totale du paramètre, appelé *interEnvironmentalVariance*.

**ATTENTION : les allèles inconnus (codés –1) ont une valeur nulle ( = 0). Ce qui n'est pas du tout équivalent à un effet moyen (ou 'neutre') dans la population. Autrement dit, la valeur 0 ( correspondant à un allèle inconnu) peut avoir un très fort effet adaptatif.**

# 3 Architecture de la bibliothèque

## *3.1 La bibliothèque Genetics*

La bibliothèque Genetics est composée de 17 classes et 2 interfaces :

- 10 classes définissent les objets génétiques (par exemple un *IndividualGenotype*). Elles contiennent les variables d'instance de l'objet, dont les types sont ceux définis dans le paragraphe 2 (par exemple, pour l'objet *IndividualGenotype*, les variables d'instance sont 3 tableaux d'entiers) et les méthodes attachées à ces classes. Ces classes sont : *GeneticTree*, *Genotype*, *IndividualGenotype*, *MultiGenotype*, *EmptyIndividualGenotype*, *EmptyMultiGenotype*, *AlleleParameters*, *AlleleDiversity*, *GeneticMap* et *AlleleEffect* (Figure 5)

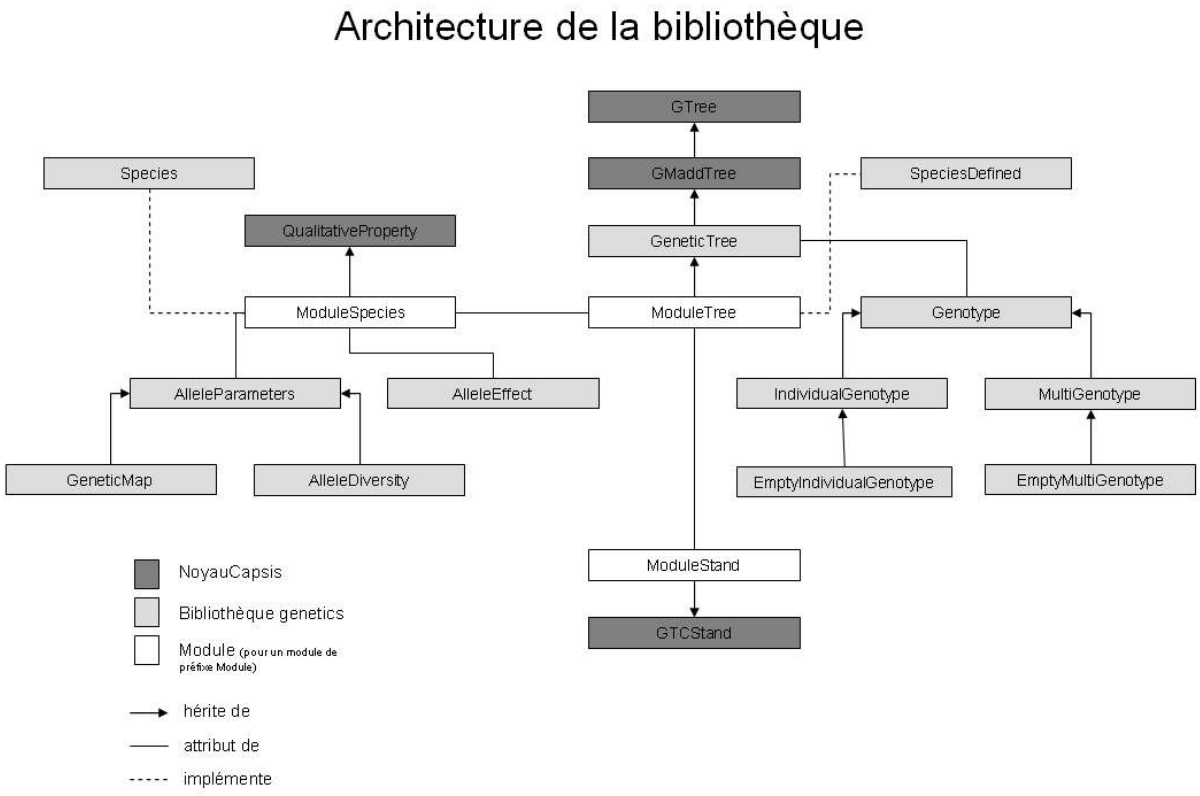


**Figure 5 : Architecture de la bibliothèque Genetics**

- 6 classes contiennent uniquement des méthodes : *GeneticsTools*, *ImportGeneticData*, *Validate*, *ValidateTools*, *CompleteInitialData*, *AreInitialDataCompatible*,

- 2 interfaces : *SpeciesDefined* et *Species* permettent à la bibliothèque de savoir si le module gère plusieurs ou une seule espèce (*SpeciesDefined*) et quelle classe, implémentant *QualitativeProperty*, définit les espèces et leurs variables d'instance,

- 1 classe définit un objet : *VertexND* et les méthodes permettant de construire cet objet. Cette classe constitue une généralisation de la classe *Vertex2D*, qui existe dans Capsis et qui a été définie pour contenir les coordonnées des points d'un polygone. *Vertex2D* est un couple de « double » de format : (a, b). *VertexND* contient une suite de « double » encadrés par des crochets et séparés par des virgules, par exemple : *[0.5, 0.2, 0.3]*. Il est utilisé pour le chargement de certaines données du fichier d'inventaire, notamment la liste des allèles possibles par locus et les effectifs alléliques des loci pour les arbres moyens (voir 5.1). Les classes font partie du package bin/capsis/lib/genetics

## *3.2 Les extracteurs de données*

Deux classes supplémentaires permettent d'extraire les données génétiques : *DEAlleleFrequencies* et *DEGenotypeFrequencies* (incluses dans le package kernel/extension/dataextractor). Elles permettent de réaliser des sorties graphiques, respectivement histogramme cumulé des fréquences alléliques par locus et histogramme cumulé des fréquences génotypiques par locus.

## *3.3 Les exportateurs de données*

Trois classes ont également été crées : *GenePopExport*, *GeneticsDGenePopExport* et *GenePopExportSettings* (package kernel/extension/ioformat). Elles permettent de réaliser des exportations dans le format de fichier du logiciel GenePop. *GenePopExport* effectue l'exportation, *GeneticsGenePopExport* gère la boîte de dialogue qui permet de définir le format d'export (quels arbres, éclatés en population selon quels critères …) et *GenePopExportSettings* permet de communiquer les informations saisies par la boîte de dialogue à *GenePopExport*.

## *3.4 Les outils de modélisation*

La classe *GeneticsGeneration* du package kernel/extension/modelTool permet de générer un certain nombre de données génétiques (*Genotype*, *AlleleEffect*).

# 4 Présentation des classes de la bibliothèque définissant des objets génétiques

Toutes ces classes définissent des objets par leurs variables (on parle de variables d'instance) et les opérations possibles sur l'objet (on parle de méthodes). Elles sont construites de la façon suivante :

- les variables d'instances de l'objet,
- un constructeur qui initialise les variables d'instance de l'objet,
- une ou plusieurs méthodes qui travaillent sur l'objet,
- les accesseurs (méthodes particulières) aux variables d'instance de l'objet (« get » pour lire les variables d'instance de l'objet et « set » pour les positionner).

## *4.1 GeneticTree*

### 4.1.1 Les variables d'instance de GeneticTree

La classe *GeneticTree* hérite de GMaddTree. Ce qui signifie qu'un arbre de type *GeneticTree* possède les mêmes variables d'instance qu'un GMaddTree : identifiant, référence à son peuplement, âge, hauteur, diamètre, coordonnées x, y et z. Il possède en plus 9 variables d'instance publiques et statiques : un *Genotype*, un *MultiGenotype*, 2 entiers (de type int) représentant l'identité de la mère et celle du père, un entier (de type int) codant la date de création, deux doubles codant les coefficients de consanguinité individuel et global et trois Map contenant les valeurs génétiques, environnementales et phénotypiques des paramètres quantitatifs étudiés.

***GeneticTree* possède donc deux variables d'instances pouvant contenir un génotype, *genotype* et *multiGenotype*. La première est de type *Genotype* et est *Immutable*. La seconde est de type *MultiGenotype* et est non *Immutable*. Cette double référence à un génotype a été mise en place pour gérer à la fois :**

- **des arbres individuels dont le génotype, de type *IndividualGenotype,* est invariable au cours du temps**
- **et des arbres moyens dont le génotype, de type *MultiGenotype,* est lui variable dans le temps.**

**Selon le type d'arbre (individuel ou moyen), le génotype est enregistré sous l'une des deux références (respectivement *genotype* et *multiGenotype*) et l'autre référence prend la valeur null. Dans le cas d'un arbre non génotypé, les deux références prennent la valeur null.**

La plupart de ces variables d'instance ont la particularité d'être invariable au cours du temps (*Immutable* : classe du noyau de Capsis qui définit les variables d'instance qui ne varient pas en cours de simulation). Pour un arbre, les valeurs de ces variables d'instance sont initialisées une seule fois et ne sont pas recopiées à chaque étape de la chaîne d'évolution. A chaque

étape, la copie de l'arbre à la date correspondante garde seulement un lien vers ces variables d'instance.

Les 2 seules variables d'instance variables au cours du temps sont le multiGenotype et la Map des valeurs phénotypiques qui, comme les autres variables non *Immutable* (comme le diamètre, par exemple), peuvent être calculées à chaque étape.

Par ailleurs, *GeneticTree* possède une variable d'instance privée (private) et statique (static) qui est la dernière étape connue appelée *lastKnownStep*. L'utilité de cette variable d'instance est expliquée aux paragraphes 4.1.5.2 et 4.1.5.3).

### 4.1.2 Les constructeurs

Cette classe possède deux constructeurs. Le premier initialise les variables d'instance d'un arbre à sa création. Le second actualise les variables d'instance non *Immutable* d'un arbre au cours de son évolution.

#### *4.1.2.1 Le premier constructeur de GeneticTree*

Il prend en paramètres les valeurs des variables d'instances *Immutable* et non *Immutable* à l'exclusion de *consanguinity*, *genoValue*, *fixedEnvironmentalValue* et *phenoValue*. Ces dernières sont initialisées par le constructeur :

- *consanguinity* et *globalConsanguinity* sont initialisées à la valeur -1,
- *genoValue*, *fixedEnvironmentalValue* et *phenoValue* sont initialisées à null,
- l'initialisation de *multiGenotype* dépend du type du génotype donné en paramètre au constructeur. S'il est de type *IndividualGenotype*, *multiGenotype* est initialisée à null et le génotype donné en paramètre est affecté à la référence *genotype*. S'il est de type *MultiGenotype*, *genotype* est initialisé à null et *multiGenotype* prend comme valeur le génotype donné en paramètre.

**Pour le modélisateur tout se passe comme si l'arbre n'avait qu'une seule référence à son génotype.**

#### *4.1.2.2 Le second constructeur de GeneticTree*

Il prend en paramètres les nouvelles valeurs des variables d'instance non *Immutable* à l'exclusion de *phenoValue* et de *multiGenotype*. Ces dernières sont initialisées par le constructeur :

- *phenoValue* est initialisée à null
- l'initialisation du *multiGenotype* dépend de la valeur du *multiGenotype* de l'instance précédente de l'arbre (= au pas précédent). Si l'instance précédente de l'arbre avait un *multiGenotype* null, *multiGenotype* est initialisé à null. Sinon, il est initialisé à Genotype.EMPTY_MULTI_GENOTYPE.

### 4.1.3 Principes d'initialisation des variables d'instance de GeneticTree

Les règles d'initialisation ne sont pas identiques pour toutes les variables d'instance de *GeneticTree*. Toutes sont données dans le tableau ci-dessous.

**Tableau 3 : principes pour l'initialisation des variables d'instance de *GeneticTree***

| | Nouvel arbre | | Nouvelle instance d'un arbre |
|---|---|---|---|
| | du peuplement initial | généré en cours de simulation | |
| *mId, pId et creationDate* | Par l'utilisateur à :<br>- -1 si arbre non génotypé ou valeur inconnue<br>- leur valeur | Par l'utilisateur<br>- -1 si arbre non génotypé<br>- valeur si arbre génotypé | |
| *genotype* | Par l'utilisateur à :<br>- null si arbre non génotypé<br>- vide ou sa valeur si arbre génotypé* | Par l'utilisateur à :<br>- null si arbre non génotypé<br>- vide si arbre individuel<br>- sa valeur si arbre moyen** | |
| *multiGenotype* | Par le constructeur | Par le constructeur | Par le constructeur |
| *consanguinity* | Par l'utilisateur à sa valeur par la méthode *setConsanguinity ()* | Par le constructeur | |
| *globalConsanguinity* | Par l'utilisateur à sa valeur par la méthode *setGlobalConsanguinity ()* | Par le constructeur | |
| *genoValue* et *fixedEnvironmentalValue* | Par le constructeur | Par le constructeur | |
| *phenoValue* | Par le constructeur | Par le constructeur | Par le constructeur |

* une valeur par défaut sera alors attribuée à l'arbre par la méthode *validate () à l'issue du chargement*
** valeur calculée, avant la création de l'arbre, par l'appel de la méthode *computeNewMultiGenotype ()*.

## 4.1.4 Les accesseurs

### *4.1.4.1 Au niveau de l'arbre*

Pour les variables d'instance *mId*, *pId* et *creationDate*, qui sont initialisés à la création de l'arbre, les accesseurs ***getMId ()***, ***getPId ()*** et ***getCreationDate ()*** sont simples : ils permettent de lire les valeurs de ces variables d'instance. Ils sont appelés sur un *GeneticTree* et renvoient un entier du type int. Il n'y a pas d'accesseur de type set sur ces variables.

Pour les variables d'instance *genotype*, *genoValue*, *fixedEnvironmentalValue* et *phenoValue*, les accesseurs sont un peu plus complexes. Ils fonctionnent tous sur le schéma suivant : l'accesseur lit la valeur de la variable d'instance. Si elle est différente de *null* ou -1, il la renvoie, sinon, il la calcule et l'initialise (ou l'actualise pour *phenoValue*) avant de la renvoyer. Le calcul de la valeur de la variable d'instance est réalisé en appelant des méthodes définies dans d'autres classes :

- **getGenotype ()** appelle *getGamete ()* et *fuseGametes ()* qui sont définies dans la classe *Genotype* (voir 4.2) pour calculer le génotype d'un arbre individuel ou *actualizeMultiGenotype ()* (voir 4.1.5.6) pour calculer le génotype d'un arbre moyen.
- **getConsanguinity ()** appelle *phi ()* (voir 4.1.5.8) pour calculer le coefficient de consanguinité uniquement lorsque l'arbre est un arbre individuel génotypé, n'appartenant pas au peuplement initial et dont l'apparentement est défini pour l'espèce (*Kinship* différent de null). Dans tous les autres cas, le coefficient de consanguinité est toujours connu.

- **getGlobalConsanguinity ()** appelle *getConsanguinity ()* lorsqu'elle est appelée sur un arbre individuel sinon elle retourne la valeur lue.
- **getGeneticValue (String parameterName)** appelle *getGeneticValue ()* de la classe *AlleleEffect* (voir 4.4).
- **getFixedEnvironmentalValue (String paramaterName)** appelle *getFixedEnvironmentalValue ()* de la classe *AlleleEffect* (voir 4.4).
- **getPhenoValue (String paramaterName)** appelle *getPhenoValue ()* de la classe *GeneticTools* (voir 5.2).

Ces accesseurs sont appelés sur un *GeneticTree* et renvoient un génotype pour *getGenotype()*, un double pour *getConsanguinity ()* et *getGlobalConsanguinity ()* et une Map pour *getGeneticValue (), getFixedEnvironmentalValue ()* et *getPhenoValue ()*.
Il n'y a pas d'accesseurs de type set sur ces variables d'instances hormis pour les variables *consanguinity* et *globalConsanguinity* car le modélisateur doit initialiser ces variables pour tous les arbres du peuplement initial (sauf pour les arbres non génotypés et s'il ne s'intéresse pas à la consanguinité).

#### *4.1.4.2 Au niveau de l'espèce*

Il existe une troisième catégorie d'accesseurs permettant de lire les données liées à l'espèce : **getGeneticMap (), getAlleleParameters (), getAlleleDiversity (), getAlleleEffect ()** et **getKinship ()**. Ils sont définis dans *GeneticTree* comme des méthodes abstraites (méthodes sans corps). C'est au modélisateur de définir le corps de ces méthodes dans la sous-classe de *GeneticTree* correspondant à l'arbre de son module (ModuleTree) (voir 10). Il y a un accesseur de type set pour toutes ces variables sauf *AlleleParameters*.

Par ailleurs des accesseurs sont définis sur *multiGenotype* (**getMultiGenotype ()** et **setMultiGenotype ()**) ainsi que sur *genotype* (**setIndividualGenotype ()**) qui ne doivent pas être utilisés par le modélisateur. *getMultiGenotype ()* est un accesseur privé. Les autres sont publics car ils sont utilisés par *Validate ()*.

Enfin une méthode abstraite *getNumber ()* est déclarée. Elle doit être définie dans *ModuleTree* pour renvoyer l'effectif d'un arbre (voir 10).

### 4.1.5 Les méthodes de GeneticTree

#### *4.1.5.1 La méthode isGenotyped ()*

Cette méthode teste si un arbre est génotypé. Si *genotype* et *multiGenotype* sont null, la méthode retourne *false* sinon, elle retourne *true*. C'est une méthode publique qui ne prend aucun paramètre. Elle a été définie pour que l'utilisateur puisse tester si un arbre a un génotype sans utiliser *getGenotype ()* qui, si l'arbre est génotypé et son génotype non calculé, déclencherait le calcul du génotype.

#### *4.1.5.2 La méthode searchTree (CTCStand stand, int treeId)*

Cette méthode a été construite pour rechercher, à une étape de la simulation, un arbre dont on connaît l'identifiant ; le peuplement attaché à l'étape considérée peut, ou non, contenir l'arbre

recherché. Dans la bibliothèque *genetics*, elle est appelée par *getGenotype ()* (à sa première requête) pour retrouver les parents de l'arbre dont on calcule le génotype.

Cette méthode, statique et publique, prend comme paramètres un peuplement (type *GTCStand*) et l'identifiant d'un arbre (type int), et renvoie un *GeneticTree*.

Si le peuplement donné en paramètre contient l'arbre recherché, l'arbre est renvoyé directement. Sinon, *searchTree ()* remonte successivement les étapes antérieures de la chaîne d'évolution jusqu'à trouver un peuplement qui contienne l'arbre recherché. Dans ce cas, la méthode fonctionne de manière récursive : elle s'appelle elle-même à chaque étape, même non visible, qu'elle rencontre en remontant un scénario jusqu'à trouver l'arbre recherché. Pour remonter les étapes successives, *searchTree ()* utilise *GTCStand.getStep ()* et *Step.getFather ()*.

Cas particulier : *getGenotype ()* peut être appelée, au cours des processus de régénération/croissance/mortalité, sur un arbre dont le génotype n'a pas encore été calculé. *getGenotype ()* appelle alors *searchTree ()* pour rechercher les parents de l'arbre. Or, dans ce cas, le peuplement donné en paramètre à la méthode *searchTree ()* est en cours de construction et donc ne connaît pas encore son *Step*. *getStep ()* et *getFather ()* ne peuvent donc pas être appelés, *searchTree ()* utilise alors *lastKnownStep* qui est une variable d'instance de l'arbre recherché.

#### *4.1.5.3 La méthode searchStand ()*

Cette méthode a été construite pour rechercher, à une étape de la simulation, un peuplement à partir d'un GTCStand et d'une date de création. Cette méthode est appelée dans *phi ()* pour retrouver, connaissant la date de création d'un arbre t, l'état (notamment son effectif) de l'arbre moyen parent au moment de la génération de t.

Cette méthode, statique et publique, prend comme paramètres un peuplement (type *GTCStand*) et une date de création (type int), et renvoie un *GTCStand*.

Si la date du peuplement donné en paramètre est égale à la date de création donnée en paramètre, la méthode renvoie le peuplement donné en paramètre. Sinon, *searchStand ()* remonte successivement les étapes antérieures de la chaîne d'évolution jusqu'à trouver le peuplement dont la date est égale à la date de création. Dans ce cas, la méthode fonctionne de manière récursive : elle s'appelle elle-même à chaque étape, même non visible, qu'elle rencontre en remontant un scénario jusqu'à trouver le peuplement recherché. Pour remonter les étapes successives, *searchStand ()* utilise *GTCStand.getStep ()* et *Step.getFather ()*.

Cas particulier : *getConsanguinity ()* et donc *phi ()* peut être appelée, au cours des processus de régénération/croissance/mortalité ; Dans ce cas la step du stand donné en paramètre à la méthode peut être null. Les méthodes *Step*. *getStep ()* et *getFather ()* ne peuvent donc pas être appelés, *searchSatnd ()* utilise alors *lastKnownStep* qui est une variable d'instance de l'arbre recherché.

#### *4.1.5.4 La méthode searchLastMultiGenotype ()*

Cette méthode a été construite pour rechercher le dernier *multiGenotype* connu d'un arbre moyen. Dans la bibliothèque *genetics*, elle est appelée par *actualizeMultiGenotype ()*.

Cette méthode privée ne prend aucun paramètre.

*searchLastMultiGenotype ()* remonte successivement les étapes antérieures de la chaîne d'évolution (à partir de l'étape sur laquelle elle est appelée) jusqu'à trouver une instance antérieure de l'arbre qui possède un *multiGenotype* non vide. La méthode renvoie le *multiGenotype* non vide qu'elle a trouvé ainsi qu'un Vector contenant les *Step* qu'elle a remontés jusqu'à trouver le *multiGenotype* (voir 4.1.5.10)

Cp 25/02/ 04

#### *4.1.5.5 La méthode computeNewMultiGenotype ()*

Cette méthode calcule le *multiGenotype*, les coefficients de consanguinité et les identifiants des parents d'un nouvel arbre moyen généré par simulation, à partir du tableau de parents envoyés par le module. Les valeurs calculées sont stockées dans un tableau d'objets à une dimension. Ce tableau est composé de 5 lignes, la première contenant le *multiGenotype*, la deuxième l'identifiant de la mère, la troisième l'identifiant du père, la quatrième le coefficient de consanguinité individuel et la dernière le coefficient de consanguinité global. Ce tableau est renvoyé par la méthode *computeNewMultiGenotype ()*.

Cette méthode publique prend comme paramètres un tableau à deux dimensions qui contient les couples de parents et un *GTCStand* qui est le peuplement auquel appartient l'arbre généré. Le tableau des parents est composé d'autant de lignes que de couples de parents. Il peut contenir 2 ou 3 colonnes. La première colonne contient l'identifiant de la mère et la seconde l'identifiant du père.

- cas du tableau à 2 colonnes : chaque couple doit apparaître autant de fois qu'il génère de descendants.
- cas du tableau à 3 colonnes. La troisième colonne contient le nombre de descendants (n) générés par le couple correspondant.

**Calcul du multigenotype**

Pour chaque couple, la méthode fonctionne ainsi :

- cas 1 : le nombre de descendants (n) est supérieur à 10000. Un *MultiGenotype* moyen est calculé à partir des *MultiGenotype* des parents. Pour les parents présentant un *IndividualGenotype*, celui-ci est converti en *MultiGenotype* en début de procédure. Le *MultiGenotype* calculé est ajouté à la « population des descendants ».
- Cas 2 : n est inférieur à 10001. La méthode calcule n *IndividualGenotype* en appelant les méthodes *getGamete ()* et *fuseGametes ()*. Les *IndividualGenotype* sont donc issus de processus stochastiques. Les *IndividualGenotype* calculés sont ajoutés à la « population des descendants ».

En appelant la méthode *computeAlleleFrequencies ()* de la classe *GeneticTree* (voir 4.1.5.7), la méthode calcule ensuite les effectifs alléliques de la « population des descendants » en sommant les effectifs alléliques des *multiGenotype* et en dénombrant chaque allèle dans l'ensemble des *IndividualGenotype*.

Précisions sur les méthodes de calcul :

Cas 1 :
Le *multiGenotype* des *n* descendants est calculé à partir des effectifs alléliques des parents.
Pour un allèle d'un locus de l'ADN nucléaire l'effectif est égal à :

$$e=E(n*(\frac{e_{mère}}{2N_{mère}}))+E(n*(\frac{e_{père}}{2N_{père}}))$$

Pour un allèle de l'ADN cytoplasmique, l'effectif est égal à :

$$e = E(n(\frac{e_{mère}}{N_{mère}}))$$ cas de ADN cytoplasmique maternel

(*E(x)* est la partie entière de *x*, $N_{mère}$ est l'effectif représenté par la mère et $e_{mère}$ est l'effectif, chez cette mère, de l'allèle considéré.

Par locus, la somme des effectifs des allèles est inférieure ou égale à 2*n (ou 1*n) du fait des arrondis. Les allèles « manquants » sont tirés aléatoirement dans la population parentale (mère et/ou père) selon le même principe que celui développé dans *getGamete ()* (voir 4.2.2.4).

Cas 2 :
La méthode génère n *IndividualGenotype* en appelant les méthodes *getGamete ()* et *fuseGametes ()*.
Dans le sous-cas particulier où un seul des parents est un *IndividualGenotype*, le temps de calcul peut être raccourci en limitant le nombre de tirages aléatoires. En effet, si, par exemple, c'est la mère qui est un arbre individuel et le père un arbre moyen, l'ADN cytoplasmique maternel des n descendants est connu : c'est celui de la mère. La méthode le transmet directement aux descendants sans utiliser les méthodes *getGamete ()* et *fusesGametes ()*. On évite ainsi de réaliser les tirages aléatoires dans l'ADN cytoplasmique maternel du père. Pour déterminer l'ADN cytoplasmique paternel et l'ADN nucléaire des n descendants, la méthode utilise *getGamete ()* et *fusesGametes ()*.

**Calcul des coefficients de consanguinité**
Ce calcul n'est effectué que lorsque l'apparentement (*Kinship*) est défini pour l'espèce.

Pour chaque couple donné en paramètre, la méthode calcule le coefficient de parenté en appelant la méthode *phi ()*. La moyenne de ces coefficients, pondérée par la contribution de chaque couple, est le coefficient de consanguinité individuel.

La méthode construit la Map des contributions gamétiques en passant en revue chaque couple parent. Cette Map contient comme clé les identifiants des parents et le nombre total de gamètes produits par parent (en tant que mère ou père).

Le calcul du coefficient de consanguinité global d'un arbre moyen est basé sur le partitionnement des 2n gènes du *MultiGenotype* en fonction de leur origine parentale ($C_p$ gènes proviennent du parent $P_p$).
Deux gènes tirés (sans remise) dans le *MultiGenotype*,
soit :

- proviennent du même parent ($P_p$) avec une probabilité : $\frac{C_p}{2n}\frac{C_p-1}{2n-1}$

- et sont identiques (chez $P_p$) avec une probabilité : $\frac{1}{2n_p} + (1 - \frac{1}{2n_p}) f_p$

soit :

- proviennent de 2 parents ($P_p$ et $P_q$) avec une probabilité : $\frac{C_p}{2n}\frac{C_q}{2n-1} + \frac{C_q}{2n}\frac{C_p}{2n-1}$
- et sont identiques avec une probabilité : *phi* ($P_p, P_q$)

Sur l'ensemble des parents, on obtient :

$$f = \sum_{p=1}^{P} \frac{C_p}{2n}\frac{C_p - 1}{2n-1}(\frac{1}{2n_p} + (1 - \frac{1}{2n_p}) f_p) \quad + \quad \sum_{p=1}^{P}\sum_{p \neq q} \frac{C_p}{2n}\frac{C_q}{2n-1} phi(P_p, P_q)$$

Notation : $P$ = nombre de parents
$C_p$ = contribution gamétique du parent $P_p$
$n_p$ = effectif du parent $P_p$
$f_p$ = consanguinité globale de $P_p$

#### *4.1.5.6 La méthode actualizeMultiGenotype ()*

Cette méthode calcule le *multiGenotype* d'une nouvelle instance d'un arbre moyen. Dans la bibliothèque *Genetics*, elle est appelée par *getGenotype ()*.

Cette méthode privée ne prend pas de paramètre.

Elle appelle *searchLastMultiGenotype ()* pour retrouver le dernier *multiGenotype* connu de l'arbre sur lequel elle est appelée et connaître la liste des *Step* à remonter. La méthode passe en revue chaque *Step* du Vector envoyé par *searchLastMultiGenotype ()* dans l'ordre chronologique. A chaque Step i, la méthode compare l'effectif de l'instance de l'arbre au *Step* i à l'effectif de la dernière instance de l'arbre ayant un *multiGenotype* non vide (à la première étape, il s'agit du *multiGenotype* envoyé par *searchLastMultiGenotype ()*). S'ils sont égaux, la méthode passe au *Step* suivant et le multiGenotype de l'arbre au Step i reste vide. S'ils sont différents, la méthode calcule la nouvelle valeur du *MultiGenotype* et l'alloue à *multiGenotype* de l'arbre au *Step* i. La dernière instance de l'arbre ayant un *multiGenotype* non vide devient alors l'instance correspondant au *Step* i.

Calcul du nouveau *multiGenotype* : newN étant l'effectif de l'instance de l'arbre au pas i et oldN étant l'effectif de la dernière instance de l'arbre ayant un *multiGenotype* non vide.

1- Si oldN-newN est inférieur à 10000 :
La méthode réalise, par locus, (oldN-newN) tirages aléatoires d'allèle sans remise dans le dernier génotype. Pour cela la méthode construit d'abord le nouveau *MultiGenotype mg*. Il est égal au dernier *MultiGenotype* connu. Pour chaque locus i, à chaque tirage, la méthode calcule l'effectif cumulé des allèles du locus i de *mg*, tire un nombre compris entre 0 et l'effectif

cumulé total, détermine le premier effectif cumulé supérieur à ce nombre et soustrait 1 à l'effectif de l'allèle correspondant dans *mg* (Figure 6).

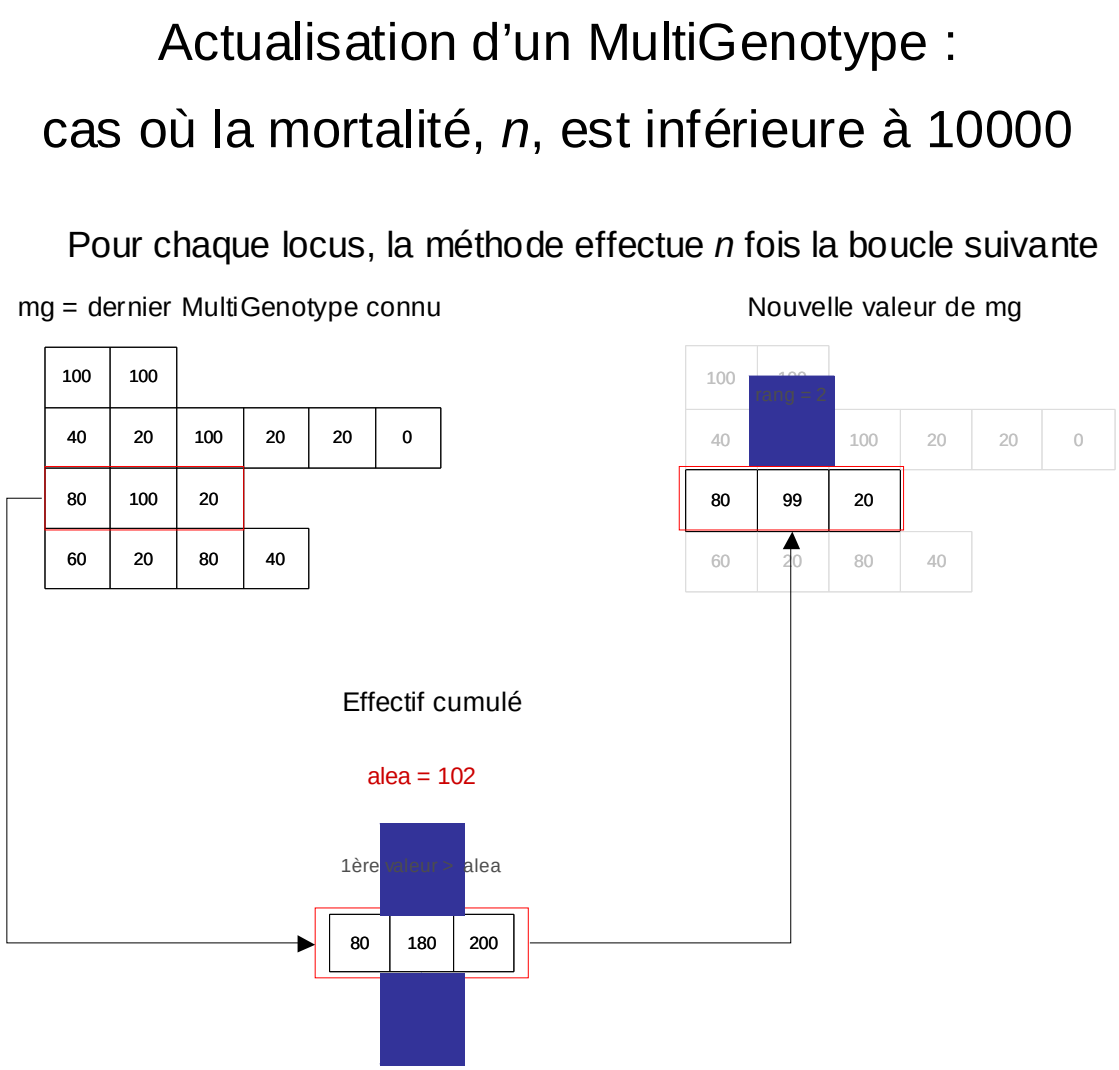


**Figure 6 : actualisation du génotype d'un arbre moyen lorsque la mortalité est inférieure à 10000.**

2- Si oldN-newN est supérieur à 10000 :
La méthode calcule le nouvel effectif d'un allèle comme suit :

$$e_{après} = e_{avant} - E(\frac{oldN - newN}{oldN} e_{avant})$$

*E()* est la partie entière
Si par locus, la somme est inférieure à 2*newN (ou 1*newN pour les ADN cytoplasmiques), du fait des arrondis, la méthode tire aléatoirement le nombre d'allèles manquants et les soustrait aux effectifs calculés selon le même principe que dans le cas oldN-newN inférieur à 10000.

#### *4.1.5.7 La méthode computeAlleleFrequencies ()*

Cette méthode calcule les effectifs alléliques d'une collection de génotypes à partir de l'*AlleleDiversity* de l'espèce. Dans la bibliothèque, cette méthode est appelée par *computeNewMultiGenotype ()*.

C'est une méthode privée et statique qui prend comme paramètre une collection de génotypes et une *AlleleDiversity*.

Cette méthode fonctionne de façon homologue à la méthode *computeAlleleFrequencies ()* de la classe *GeneticTools* (voir 5.2.3).

#### *4.1.5.8 La méthode phi ()*

Cette méthode calcule le coefficient de parenté des deux arbres dont les identifiants sont donnés en paramètres. Dans *genetics*, elle est appelée par *getConsanguinity ()* et *computeNewMultiGenotype ()* pour calculer le coefficient de consanguinité d'un arbre :

$$f(id) = phi\ (mId, pId)$$

avec *f* coefficient de consanguinité de l'arbre id
et *phi (mId, pId)* coefficient de parenté des parents de id

Cette méthode est privée et statique. Elle prend comme paramètres, en plus des identifiants des parents, un *GTCStand* et un *int* qui correspond à la date de création du descendant. Ce paramètre est utilisé lorsque l'arbre est issu d'autofécondation d'un arbre moyen.

La méthode recherche tout d'abord les instances des arbres en appelant *searchTree ()*. Pour calculer le coefficient de parenté d'un couple, la méthode remonte la généalogie des arbres donnés en paramètres jusqu'à retrouver leurs parents dans le peuplement initial où tous les arbres ont un coefficient de consanguinité et tous les couples un coefficient de parenté connus. A chaque étape, la méthode applique l'une des relations suivantes :

si $I_i \neq I_j$

$$Phi\ (I_i \text{ et } I_j) = 0.5 * Phi\ (I_i, \text{mère de } I_j) + 0.5 * Phi\ (I_i, \text{père de } I_j) \quad (1)$$

si $I_i = I_j$ :

$$Phi\ (I_i \text{ et } I_i) = 1/2n + (1 - 1/2n) * f(I_i) \quad (2)$$

avec $f(I_i)$ coefficient de consanguinité globale

La méthode fonctionne de manière récursive. A chaque étape, elle teste si l'on se trouve dans un cas où la boucle s'arrête (voir Figure 7) :

1 – Si un des deux arbres est inconnus (identifiant = -1), le coefficient de parenté est nul – Arrêt de la boucle.
2 – Si les deux arbres sont identiques, le coefficient de parenté est calculé par l'équation (2) – Arrêt de la boucle si le coefficient de consanguinité global est connu sinon la méthode *getGlobalConsanguinity ()* appelle de nouveau la méthode *phi ()*.
3 – Si les deux arbres appartiennent au peuplement initial leur coefficient de parenté est connu – Arrêt de la boucle.
4 – Si M n'appartient pas au peuplement initial, le coefficient de parenté est calculé par l'équation (1), la méthode s'appelle elle-même
5 – Si P n'appartient pas au peuplement initial, le coefficient de parenté par l'équation (1), la méthode s'appelle elle-même

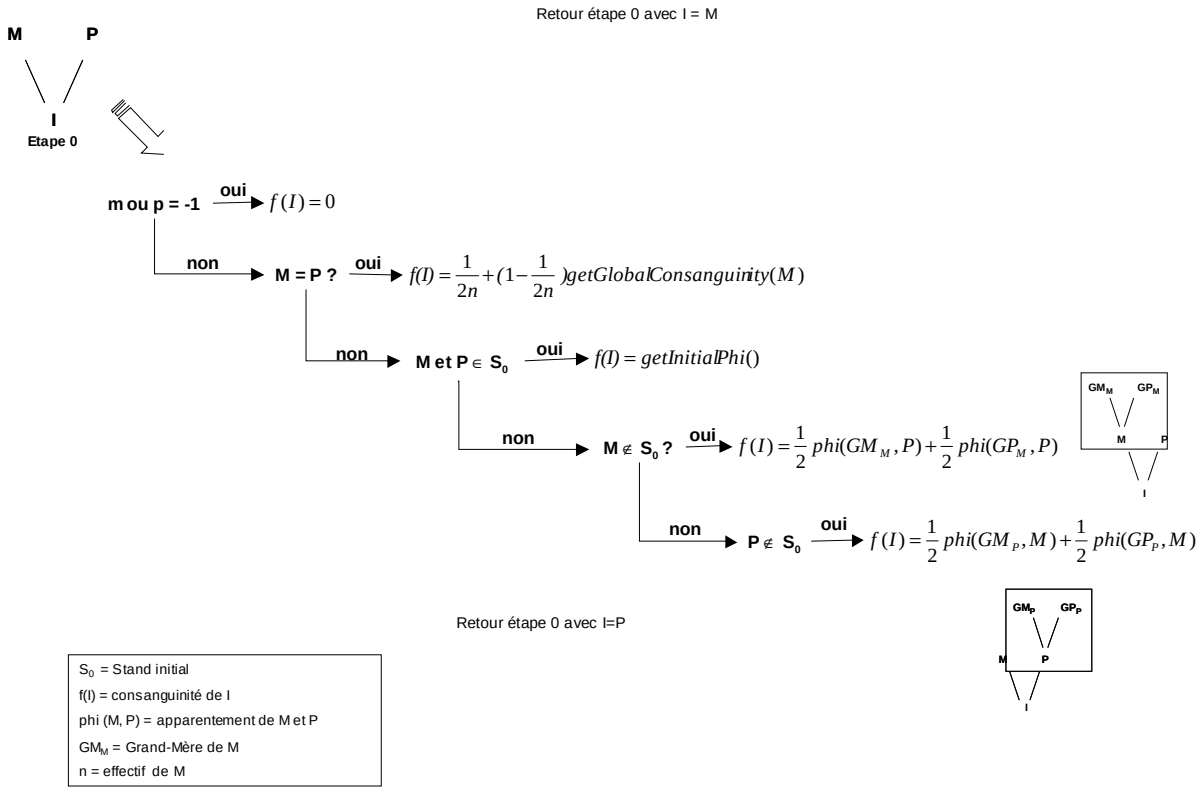


**Figure 7 : Principe de fonctionnement de la méthode phi ().**

### *4.1.5.9 La méthode getInitialPhi (GTCStand stand, int mId, int pId)*

Cette méthode retourne le coefficient de parenté de deux arbres (*mId* et *pId*) appartenant au peuplement initial. Dans *genetics*, elle est appelée par *phi ()* (voir 4.1.5.8).

Cette méthode est privée et statique. Elle prend comme paramètres un stand et deux identifiants. Elle renvoie un double.

La méthode demande l'*Apparentement* de l'espèce. Elle parcourt le tableau des coefficients de parenté *initialPhiArray*. S'il contient le couple donné en paramètre, la méthode renvoie le coefficient de parenté correspondant, sinon elle renvoie le coefficient de parenté par défaut *defaultPhi*.

### *4.1.5.10 La sous-classe MultiGenotypeTracing de GeneticTree*

Cette sous classe définit l'Objet *MultiGenotypeTracing* qui est renvoyé par la méthode *searchLastMultiGenotype ()*. Cet objet contient toutes les informations nécessaires pour actualiser le *multiGenotype* d'un arbre moyen.

*MultiGenotypeTracing* est composé d'un *MultiGenotype* et d'un Vector. Lorsque *searchLastMultiGenotype ()* est appelée, elle renvoie un *MultiGenotypeTraccing* qui contient :

- le dernier *multiGenotype* non vide d'un arbre moyen
- et un Vector de tous les *Step* compris entre le *Step* sur lequel est appelé *searchLastMultiGenotype ()* et le *Step* correspondant à l'instance de l'arbre contenant le dernier *multiGenotype* non vide.

## *4.2 Genotype*

*Genotype* est une classe abstraite qui définit les objets communs à ses deux sous-classes (*IndividualGenotype* et *MultiGenotype*). Comme toute classe abstraite, elle ne définit pas les variables d'instance de l'objet *Genotype* qui sont définies dans les sous-classes. Elle contient deux méthodes.

La méthode **getGamete (AlleleParameters ap)** crée le génotype d'un gamète produit par un arbre à partir de son génotype. C'est une méthode abstraite et elle est donc nécessairement définie dans chacune des deux sous-classes, les calculs qu'elle effectue étant différents selon le type du génotype de l'arbre producteur. La définition de la classe abstraite permet au modélisateur d'utiliser une méthode unique qui appelle, de façon transparente, la méthode adaptée au type de génotype sur lequel elle s'applique (sans tester à chaque fois le type du génotype). Cette méthode prend comme paramètre un *AlleleParameters* (voir 4.3) et renvoie un *IndividualGenotype*.

La méthode **fuseGametes (IndividualGenotype g1, IndividualGenotype g2)** simule la fusion de deux gamètes pour créer le génotype d'un nouvel arbre. Elle prend comme paramètres deux *IndividualGenotype* (qui sont en fait les génotypes, haploïdes, de deux gamètes calculés par *getGamete ()*) et renvoie un *IndividualGenotype*. Cette méthode utilise toujours des *IndividualGenotype*, elle a donc pu être définie dans la super classe.

 L'ordre des *IndividualGenotype* donnés en paramètre dans la méthode *fuseGametes ()* est important : le premier représente le génotype du gamète femelle produit par la mère, le second celui du gamète mâle produit par le père. Pour construire le génotype de l'arbre fils, la méthode fusionne les ADN nucléaires des deux gamètes et récupère l'ADN cytoplasmique maternel du gamète femelle et l'ADN cytoplasmique paternel du gamète mâle (voir la méthode *getGamete ()* paragraphes *4.2.1.3 et 4.2.2.4*).

**Les méthodes simulant les processus de méiose et fécondation (*getGamete ()* et fuseGametes ()) génèrent toujours un *IndividualGenotype*. Il n'y a pas de méthodes générant directement un *MultiGenotype*.**

Ces deux méthodes n'ont normalement pas besoin d'être appelées dans une classe du module. Elles sont appelées indirectement par les méthodes *getGenotype (),* à sa première requête sur un arbre donné, et *computeNewMultiGenotype ().*

### 4.2.1 IndividualGenotype

**Les variables d'instance de IndividualGenotype**

*IndividualGenotype* possède quatre variables d'instance :

- *nuclearDNA* : tableau de short à deux dimensions
- *mCytoplasmicDNA* : tableau de short à une dimension
- *pCytoplasmicDNA* : tableau de short à une dimension
- un *Random* (générateur de nombres aléatoires) : utilisé dans la méthode *getGamete ()* (voir 4.2.1.3)

#### *4.2.1.1 Le constructeur de IndividualGenotype*

Le constructeur prend en paramètres trois tableaux de short, un à deux dimensions (*nuclearDNA*) et deux à une dimension (*mCytoplasmicDNA* et *pCytoplasmicDNA*) et renvoie un *IndividualGenotype*.

#### *4.2.1.2 Les accesseurs*

Trois accesseurs sont définis un pour chaque tableau d'ADN :

- **getNuclearDNA ()** : est appelé sur un *IndividualGenotype*, ne prend aucun paramètre et renvoie un tableau de short à 2 dimensions (*nuclearDNA*),
- **getMCytoplasmicDNA ()** : est appelé sur un *IndividualGenotype*, ne prend aucun paramètre et renvoie un tableau de short à une dimension (*mCytoplasmicDNA*),
- **getPCytoplasmicDNA ()** : est appelé sur un *IndividualGenotype*, ne prend aucun paramètre et renvoie un tableau de short à une dimension (*pCytoplasmicDNA*).

#### *4.2.1.3 Les méthodes de IndividualGenotype*

**getGamete (AlleleParameters ap)**

La méthode *getGamete ()* construit le génotype, de type *IndividualGenotype*, d'un gamète à partir du génotype *IndividualGenotype* de l'arbre producteur.

Pour ne définir qu'une seule méthode *getGamete ()*, nous avons choisi de construire des « pseudo » gamètes qui héritent à la fois des ADN cytoplasmiques maternel et paternel de l'arbre producteur. Le « sexe » du gamète est déterminé au moment de la fusion : s'il est produit par la mère, il est donné comme premier paramètre dans la méthode *fuseGametes ()*, s'il est produit par le père, il est donné comme second paramètre.

La méthode construit un ADN nucléaire haploïde à partir de l'ADN nucléaire diploïde de l'arbre producteur en simulant la méiose. Pour déterminer la valeur de l'allèle sur le premier locus, la méthode tire un entier, aléatoirement 0 ou 1. Si cet entier est égal à 0, l'allèle sur le premier locus est celui issu de la mère (1ère colonne du tableau *nuclearDNA*), s'il est égal à 1, c'est celui issu du père (2ème colonne du tableau). La méthode passe ensuite en revue les loci suivants. A chaque étape i, la méthode détermine la valeur de l'allèle sur le $i^{ème}$ locus. Pour cela, elle tire un nombre décimal aléatoire compris entre 0 et 1 et le compare à la $(i-1)^{ème}$ valeur du tableau des probabilités de recombinaison (donné en paramètre). A chaque fois que le nombre aléatoire tiré est inférieur à la probabilité de recombinaison, l'origine de l'allèle est inversée par rapport à celle de l'allèle du locus précédent (maternelle, si l'allèle du locus précédent était hérité du père et inversement ; voir Figure 8).

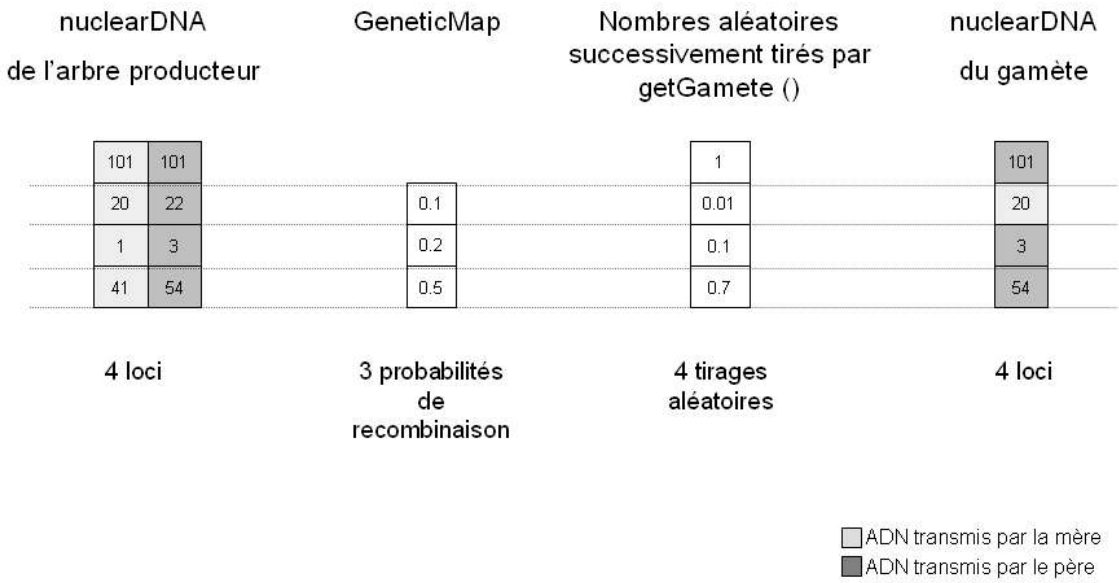


**Figure 8 : Méthode getGamete (). Construction du génotype d'un gamète produit par un arbre individuel.**

Cette méthode est appelée sur un *IndividualGenotype*. Elle prend comme paramètre un *AlleleParameters* comme la méthode abstraite du même nom. Mais pour fonctionner l'*AlleleParameters* donné en paramètre doit être de type *GeneticMap* (celle correspondant à l'espèce de l'arbre). Cette méthode renvoie un *IndividualGenotype* dont le *nuclearDNA* est un peu particulier puisqu'il ne possède qu'une seule colonne.

**computeDefaultIndividualGenotype (GeneticTree tree)**
Si des arbres individuels d'une espèce suivie sur le plan génétique (c-à-d que, dans le peuplement initial, des arbres de cette espèce ont un *IndividualGenotype* non null), *computeDefaultIndividualGenotype ()* leur attribue un *IndividualGenotype.*
Ce génotype :

- a le même format que celui des arbres génotypés de la même espèce (2 à 2, les 3 tableaux des allèles par locus ont la même taille)
- et tous les allèles sont inconnus c-à-d égaux à -1 par convention.

#### *4.2.1.4 La sous-classe de IndividualGenotype : EmptyIndividualGenotype*

La sous-classe de *IndividualGenotype* a été construite pour pouvoir attribuer à un arbre un génotype de type *IndividualGenotype* égal à une constante. Cette constante est affectée à un arbre lorsque l'utilisateur souhaite reporter le calcul de son génotype.

La sous-classe *EmptyIndividualGenotype* définit le constructeur de *EmptyIndividualGenotype ()*. C'est dans la classe *Genotype* qu'est construite la constante EMPTY_INDIVIDUAL_GENOTYPE par appel de ce constructeur.

## 4.2.2 MultiGenotype

### *4.2.2.1 Les variables d'instance de MultiGenotype*

*MultiGenotype* possède quatre variables d'instance :

- 3 tableaux de double à deux dimensions : *nuclearAlleleFrequency*, *mCytoplasmicAlleleFrequency* et *pCytoplasmicAlleleFrequency*
- un *Random* : utilisé dans la méthode *getGamete ()* (voir ci-dessous)

### *4.2.2.2 Le constructeur de MultiGenotype*

Le constructeur prend en paramètres trois tableaux de double à deux dimensions. Il renvoie un *MultiGenotype*.

### *4.2.2.3 Les accesseurs*

Trois accesseurs sont définis un pour chaque tableau des effectifs alléliques :

- **getNuclearAlleleFrequency ()** : est appelé sur un *MultiGenotype*, ne prend aucun paramètre et renvoie un tableau de short à 2 dimensions (*nuclearAlleleFrequency*),
- **getMCytoplasmicAlleleFrequency ()** : est appelé sur un *MultiGenotype*, ne prend aucun paramètre et renvoie un tableau de short à une dimension (*mCytoplasmicAlleleFrequency*),
- **getPCytoplasmicAlleleFrequency ()** : est appelé sur un *MultiGenotype*, ne prend aucun paramètre et renvoie un tableau de short à une dimension (*pCytoplasmicAlleleFrequency*).

### *4.2.2.4 Les méthodes de MultiGenotype*

**getGamete (AlleleParameters ap)**

La méthode *getGamete ()* construit le génotype, de type *IndividualGenotype*, d'un gamète à partir du génotype *MultiGenotype* de l'arbre producteur. Chacun des 3 tableaux du génotype du gamète est construit de la même façon (Figure 9).

La méthode passe successivement en revue chaque ligne (1 ligne correspondant à un locus) du tableau d'effectifs alléliques de l'arbre producteur. Pour chaque ligne i, elle construit un tableau de double à une dimension, de taille égale au nombre de cellules de la ligne i, contenant les effectifs alléliques cumulées. Elle tire ensuite un nombre entier aléatoire, alea, entre 0 et 2n (n étant l'effectif de la population) et repère dans le tableau des effectifs cumulées le rang j de la première valeur supérieure à alea. Elle lit ensuite dans le tableau *AlleleDiversity* l'allèle de la ligne i et de la colonne j et l'affecte à la ligne i du tableau d'ADN du gamète.

Cette méthode est appelée sur un *MultiGenotype*. Elle prend comme paramètre un *AlleleParameters* comme la méthode abstraite du même nom. Mais pour fonctionner l'*AlleleParameters* donné en paramètre doit être de type *AlleleDiversity* (celle correspondant à l'espèce de l'arbre). Cette méthode renvoie un *IndividualGenotype* dont le *nuclearDNA* est un peu particulier puisqu'il ne possède qu'une seule colonne.

# Construction du génotype d'un gamète : méthode getGamete ()

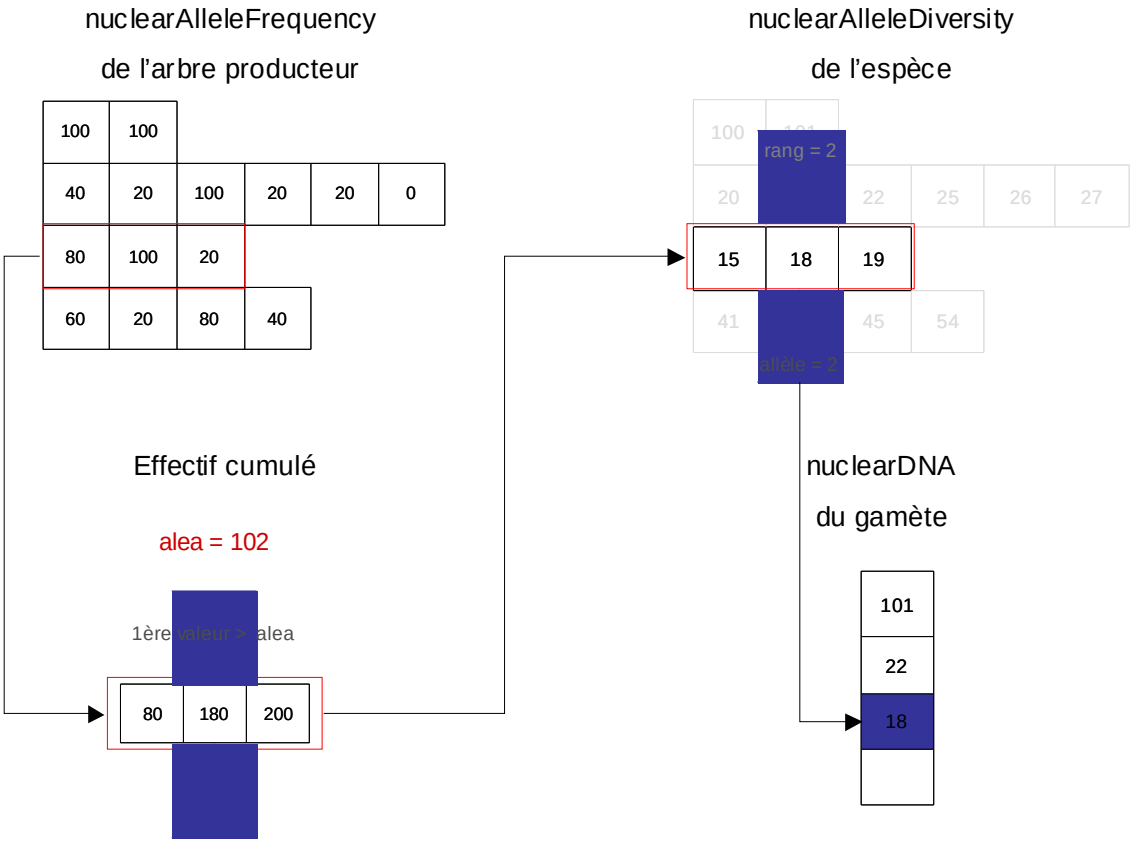


**Figure 9 : Méthode getGamete (). Construction du génotype d'un gamète produit par un arbre moyen.**

**computeDefaultMultiGenotype (Collection treesWithGenotype, Collection treesWithoutGenotype)**

Lorsqu'un ou plusieurs arbres moyens du peuplement initial, d'une même espèce S, ont un *MultiGenotype* vide (c-à-d égal à *Genotype.EMPTY_MULTI_GENOTYPE*), cette méthode permet de calculer un *MultiGenotype* par défaut et de leur attribuer. Le calcul du MultiGenotype est effectué en respectant les fréquences alléliques de la population des arbres individuels et moyens génotypés du peuplement initial.

*computeDefaultMultiGenotype ()* prend comme paramètres deux *Collection* :

- la première contient la liste des arbres, d'espèce S, appartenant au peuplement initial et ayant un génotype
- la seconde contient les arbres moyens, d'espèce S, qui n'ont pas de *MultiGenotype*.

Pour calculer les fréquences alléliques dans la population des arbres avec un génotype non vide, cette méthode appelle *computeAlleleFrequencies ()* (voir 5.2.3).
Le calcul du *MultiGenotype* de l'arbre moyen dépend de son effectif.

Si l'effectif est inférieur à 10000 :
Le *MultiGenotype* est issu d'un processus entièrement stochastique.
Pour chaque locus i, la méthode calcule le tableau des fréquences alléliques cumulées à partir des fréquences alléliques dans la population initiale (arbres individuels et moyens inclus). Elle tire ensuite un nombre réel aléatoire, alea, entre 0 et 1, et repère dans le tableau des fréquences cumulées le rang j de la première valeur supérieure à alea. Elle ajoute 1 à la valeur dans la cellule (i,j) du tableau des effectifs alléliques de l'arbre moyen. Pour le calcul des effectifs alléliques sur l'ADN nucléaire, la méthode réalise 2*n tirages aléatoires par locus (n étant l'effectif de l'arbre moyen). Pour le calcul des effectifs alléliques sur les ADN cytoplasmiques, la méthode réalise n tirages aléatoires. Les haplotypes étant habituellement codés comme des allèles d'un seul locus, une seule série de n tirages est nécessaire pour générer chacun des ADN cytoplasmiques.

Si l'effectif est supérieur à 10000 :
La méthode calcule le *MultiGenotype* en appliquant la formule suivante :

$$e_i = E(2 * n * p_i)$$

$e_i$ est l'effectif de l'allèle i
$p_i$ est sa fréquence dans la population initiale
$n$ est l'effectif de l'arbre moyen

*E()* est la partie entière.
Si par locus, la somme des effectifs alléliques est inférieure à *2n* (ou *n* pour les ADN cytoplasmiques), la méthode réalise $2n - \sum_{locus} e_i$ (ou $n - \sum_{locus} e_i$ pour les ADN cytoplasmiques) tirages aléatoires selon le même principe que ci-dessus.

#### *4.2.2.5 La sous-classe de MultiGenotype : EmptyMultiGenotype*

La sous-classe de *MultiGenotype* a été construite pour pouvoir attribuer à un arbre un génotype de type *MultiGenotype* égal à une constante. Cette constante est affectée à un arbre lorsque l'utilisateur souhaite reporter le calcul de son génotype.

La sous-classe *EmptyMultiGenotype* définit le constructeur de *EmptyMultiGenotype*. C'est dans la classe Genotype qu'est construite la constante EMPTY_MULTI_GENOTYPE par appel de ce constructeur.

## *4.3 AlleleParameters*

*AlleleParameters* est une classe abstraite. Elle a été définie pour faire hériter *GeneticMap* et *AlleleDiversity* de *AlleleParameters*.

Cette démarche permet :
- de construire un accesseur unique qui renvoie soit *GeneticMap*, soit *AlleleDiversity* selon le type de génotype de l'arbre sur lequel il est appelé (*getAlleleParameters ()*).
- de construire la méthode *getGamete ()* de la classe abstraite *Genotype* qui prend comme paramètre un *AlleleParameters* dont le type est testé dans les sous-classes de *Genotype*.

Ainsi lorsqu'on effectue sur un génotype g (de type *IndividualGenotype* ou *MultiGenotype*) d'un arbre t : *g.getGamete (t.getAlleleParameters ())* :

- si g est un *IndividualGenotype*, la ligne de commande lit la *GeneticMap* de l'espèce et effectue la méthode *getGamete ()* de la sous-classe *IndividualGenotype*,
- si g est un *MultiGenotype*, la ligne de commande lit l'*AlleleDiversity* de l'espèce et effectue la méthode *getGamete ()* de la sous-classe *MultiGenotype*.

Cette classe ne contient ni variables d'instance, ni constructeurs ni méthodes.

### 4.3.1 GeneticMap

*GeneticMap* hérite de *AlleleParameters*.

#### *4.3.1.1 Les variables d'instance de GeneticMap*

*GeneticMap* contient un tableau de float à une dimension rassemblant les probabilités de recombinaison (lors de la méiose) entre loci successifs : *recombinationProbas*,

#### *4.3.1.2 Le constructeur de GeneticMap*

Le constructeur de *GeneticMap* initialise la variable d'instance de *GeneticMap* (*recombinationProbas*) dès l'initialisation du module (au chargement du fichier ou durant la validation). Il prend deux paramètres, l'un des deux pouvant être *null*.

#### *4.3.1.3 Les accesseurs*

*GeneticMap* définit l'accesseur **getRecombinationProbas ()**.

#### *4.3.1.4 La méthode computeDefautlRecombinationProbas (GeneticTree tree)*

Si *GeneticMap* n'est pas définie, *computeDefautlRecombinationProbas ()* calcule une *GeneticMap* par défaut qui contient :

- un tableau *recombinationProbas*, dont la taille est égale au nombre de loci sur l'ADN nucléaire moins 1 et où toutes les probabilités de recombinaison sont égales à 0.5,

Dans ce cas, lors de la méiose, tous les loci seront donc indépendants.

### 4.3.2 AlleleDiversity

*AlleleDiversity* hérite de *AlleleParameters*.

#### *4.3.2.1 Les variables d'instance de AlleleDiversity*

*AlleleDiversity* contient trois tableaux de short à deux dimensions contenant la liste des allèles possibles sur chacun des loci des ADN nucléaire, cytoplasmiques maternel et paternel.

#### *4.3.2.2 Le constructeur de AlleleDiversity*

Toutes les variables d'instance de *AlleleDiversity* sont initialisées par le constructeur, qui prend donc trois paramètres, dès l'initialisation du module (au chargement du fichier ou durant la validation).

#### *4.3.2.3 Les accesseurs*

*AlleleDiversity* définit un accesseur pour chaque variable d'instance : **getNuclearAlleleDiversity ()**, **getMCytoplasmicAlleleDiversity ()** et **getPCytoplasmicAlleleDiversity ()**.

#### *4.3.2.4 Les méthodes de AlleleDiversity*

**computeNuclearAlleleDiversity (Collection trees)**
Cette méthode calcule *nuclearAlleleDiversity* à partir d'une liste d'arbres individuels. Elle prend comme paramètre une Collection contenant une liste d'arbres individuels de la même espèce et renvoie un tableau de short à deux dimensions.

La méthode passe en revue chaque locus des arbres de la Collection donnée en paramètre et sauvegarde dans le tableau de résultat chaque nouvel allèle rencontré.

**computeCytoplasmicAlleleDiversity (Collection trees, String parent)**
Cette méthode calcule un tableau contenant la liste des allèles possibles par locus sur l'ADN cytoplasmique. Elle prend comme paramètres une Collection, contenant des arbres individuels de la même espèce et une String égale à « maternal » ou « paternal ».

La principe de calcul est le même que celui de la méthode ci-dessus.

**computeAlleleDiversity (Collection trees)**
Cette méthode calcule l'*AlleleDiversity* correspondant à une population d'arbres individuels.
Elle prend comme paramètres une Collection contenant des arbres individuels de la même espèce.

Pour le calcul de chacun des trois tableaux, elle fait appel aux deux méthodes décrites ci-dessus.

## *4.4 AlleleEffect*

### 4.4.1 Les variables d'instance de AlleleEffect

*AlleleEffect* possède une seule variable d'instance : une Map appelée *effect*. Les objets contenus dans cette Map sont définis dans une sous-classe de *AlleleEffect* : *ParameterEffect*.

### 4.4.2 ParameterEffect, sous-classe de AlleleEffect

Cette sous-classe est construite sur le même schéma que les classes précédentes.

#### *4.4.2.1 Les variables d'instance de ParameterEffect*

*ParameterEffect* possède 6 variables d'instance :

- trois tableaux de short à deux dimensions contenant les valeurs des allèles de chacun des trois ADN nucléaire, cytoplasmiques maternel et paternels
- l'héritabilité du paramètre (double)

- la variance environnementale totale du paramètre (double)
- la part de la variance environnementale inter Step (double)

Chacun des 3 tableaux possède un nombre de lignes égal au nombre de loci impliqués (habituellement 1 pour les ADN cytoplasmiques). Le nombre de colonnes est égal au nombre d'allèles + 1. La première colonne donne le numéro d'ordre du locus, les suivantes contiennent les effets des allèles dans le même ordre que celui des tableaux décrivant la diversité allélique (*AlleleDiversity*).

#### *4.4.2.2 Le constructeur de ParameterEffect*

Les variables d'instances de *ParameterEffect* sont initialisées à l'initialisation du module (au cours du chargement du fichier ou par simulation).
Les règles pour l'initialisation :

- les tableaux des valeurs des allèles doivent être donnés à l'initialisation du module par chargement du fichier de données ou par simulation (voir 9.2.2.7). L'un au moins des trois tableaux doit être non *null*.
- l'héritabilité et/ou la variance environnementale totale doivent être données à l'initialisation. Si l'une des deux valeurs n'est pas connue, elle doit être, par convention, donnée égale à -1 (valeur inconnue). La bibliothèque *Genetics* calculera une valeur par défaut (par la procédure de validation ou par la procédure de génération de données génétiques).
- la part de la variance environnementale inter *Step* doit être donnée à l'initialisation. Elle permet de séparer les effets environnementaux invariables au cours de la simulation (tirés une seule fois à l'étape racine) et les effets environnementaux variables (re - tirés à chaque étape).

#### *4.4.2.3 Les accesseurs de ParameterEffect*

Un accesseur est défini pour chacune des variables d'instance : **getNuclearAlleleEffect ()**, **getMCytoplasmicAlleleEffect ()**, **getPCytoplasmicAlleleEffect ()**, **getHeritability (), getTotalEnvironmentalVariance ()** et **getInterEnvironmentalVariance ().** Ces accesseurs s'appellent sur un *ParameterEffect*, ne prennent aucun paramètre et renvoient un tableau de short à deux dimensions pour les trois premiers, un double pour les trois derniers.

La sous-classe *ParameterEffect* ne contient pas d'autres méthodes.

### 4.4.3 Le constructeur de AlleleEffect

Le constructeur de *AlleleEffect* renvoie une nouvelle Map vide. Il ne prend aucun paramètre. Ce constructeur ne doit être appelé qu'une seule fois par espèce pour construire la table des valeurs pour l'espèce.

### 4.4.4 Les accesseurs de AlleleEffect

*AlleleEffect* contient deux accesseurs.

**getParameterEffect (String parameterName)** permet de lire le *ParameterEffect* d'un paramètre. Cet accesseur s'appelle sur la Map *AlleleEffect,* prend en paramètre une String (le nom du paramètre) et renvoie un *ParameterEffect*.

**getParameterName ()** permet de lire la liste des noms de paramètre contenus dans *AlleleEffect*. Cet accesseur ne prend aucun paramètre et renvoie un Set[5] contenant la liste des noms des paramètres.

### 4.4.5 Les méthodes de AlleleEffect

*AlleleEffect* possède sept méthodes.

**addParameterEffect ()**
Cette méthode permet de construire un nouvel objet de type *ParameterEffect* et de l'ajouter dans la Map *effect*. C'est une méthode de type void (c-à-d qu'elle ne renvoie rien) qui prend comme paramètres :

- une String (le nom du paramètre),
- 3 tableaux de short à deux dimensions (les valeurs des allèles sur chaque ADN)
- trois double (successivement l'héritabilité du paramètre, la variance environnementale totale et inter Step)

qui permettent de définir le nouveau *ParameterEffect* (à l'aide du constructeur de *ParameterEffect*). Le couple, nom du paramètre et ParameterEffect associé, est ensuite ajouté à la Map.

**removeParameterEffect ()**
Cette méthode permet de supprimer un ParameterEffect dans la Map *effect*. C'est une méthode de type void (c-à-d qu'elle ne renvoie rien) qui prend comme paramètre une String (le nom du paramètre : *parameterName*). Le *parameterEffect* supprimé est celui associé à la clé *parameterName*.

**isEmpty ()**
Cette méthode permet de tester si la Map des valeurs est vide. Cette méthode renvoie *true* si la Map ne contient aucune clé et renvoie *false* dans le cas contraire.

**getHeritability (Collection trees, String parameterName)**
Cette méthode est appelée par la méthode *ValidateInitialData ()* de la classe *Validate* si l'héritabilité, à l'issue de l'initialisation du module, est égale à -1 pour calculer une valeur d'héritabilité par défaut.
La valeur d'héritabilité par défaut est calculée à partir de la variance environnementale totale (qui doit être différente de -1) de la façon suivante :

$$h^2 = \frac{\sigma_G^2}{\sigma_G^2 + \sigma_E^2}$$

avec $\sigma_G^2$ variance génétique et $\sigma_E^2$ variance environnementale totale.

[5] Dans le langage Java, Set est une liste d'éléments qui ne contient jamais deux éléments égaux.

$\sigma_E^2$ est celle donnée à l'initialisation du module

$\sigma_G^2$ est calculée sur l'ensemble de la population initiale, en incluant les arbres moyens et individuels, en utilisant la méthode computeGeneticVariance () de la classe GeneticTools.

Cette méthode prend comme paramètre une Collection qui contient une liste d'arbres (moyens et/ou individuels) et une String correspondant au nom du paramètre. Lorsque cette méthode est appelée dans la méthode *ValidateInitialData ()*, la Collection donnée en paramètre est la Collection de tous les arbres, de l'espèce considérée, présents dans le peuplement initial.

**getTotalEnvironmentalVariance (Collection trees, String parameterName)**

Cette méthode est appelée par la méthode *ValidateInitialData ()* de la classe *Validate* si la variance environnementale totale, à l'issue de l'initialisation du module, est égale à -1 pour calculer une variance environnementale totale par défaut.

La valeur de la variance environnementale par défaut est calculée à partir de l'héritabilité du paramètre (qui doit être différente de -1) de la façon suivante :

$$\sigma_E^2 = \sigma_G^2 \frac{1-h^2}{h^2}$$

avec $\sigma_G^2$ variance génétique et $h^2$ héritabilité du paramètre.

$h^2$ est celle donnée à l'initialisation du module

$\sigma_G^2$ est calculée sur l'ensemble de la population initiale, en incluant les arbres moyens et individuels, en utilisant la méthode computeGeneticVariance () de la classe GeneticTools.

Cette méthode prend comme paramètre une Collection qui contient une liste d'arbres (moyens et /ou individuels) et une String correspondant au nom du paramètre. Lorsque cette méthode est appelée dans la méthode *ValidateInitialData ()*, la Collection donnée en paramètre est la Collection de tous les arbres, de l'espèce considérée, présents dans le peuplement initial.

**getGeneticValue (String parameterName, GeneticTree tree)**

Cette méthode calcule, pour un arbre, la valeur génétique pour un paramètre sous l'hypothèse de l'additivité des valeurs des allèles. Elle est appelée par la méthode *getGeneticValue ()* de la classe *GeneticTree*, qu'il est conseillée d'utiliser car elle s'appelle directement sur un *GeneticTree*. D'autre part lorsque la valeur a déjà été calculée, elle est conservée au niveau de l'arbre ; l'appel sur un *GeneticTree* ne fait que la renvoyer.

La méthode *getGeneticValue ()* de la classe *AlleleEffect* lit les données dont elle a besoin : le génotype de l'arbre, l'*AlleleDiversity* correspondant à l'espèce de l'arbre et le *ParameterEffect* du paramètre pour lequel on calcule la valeur génétique. Elle teste ensuite le type de génotype de l'arbre. S'il est de type *IndividualGenotype*, la méthode somme les valeurs des allèles. S'il est de type *MultiGenotype*, la méthode somme les valeurs des allèles pondérés par leur fréquence dans la population représentée par l'arbre.

Cette méthode prend comme paramètre une String (le nom du paramètre) et un *GeneticTree*. Elle renvoie un réel de type double.

**getFixedEnvironmentalValue (String parameterName)**

Cette méthode calcule, pour un arbre, la valeur environnementale fixe d'un paramètre. Elle est appelée par la méthode *getFixedEnvironmentalValue ()* de la classe *GeneticTree*, qu'il est

conseillée d'utiliser car elle s'appelle directement sur un *GeneticTree*. D'autre part lorsque la valeur a déjà été calculée, elle est conservée au niveau de l'arbre ; l'appel sur un *GeneticTree* ne fait que la renvoyer.

La méthode *getFixedEnvironmentalValue ()* de la classe *AlleleEffect* lit les données dont elle a besoin : les variances environnementales totale et inter *Step* pour l'espèce et le paramètre considéré. Elle calcule ensuite la variance environnementale « intra *Step* » :

$$\sigma^2_{intra} = (1 - p(\sigma^2_{inter})) * \sigma^2_{total}$$

avec $\sigma^2_{intra}$ variance environnementale intra *Step*

$p(\sigma^2_{inter})$ part de la variance environnementale inter *Step* dans la variance environnementale totale

et $\sigma^2_{total}$ variance environnementale totale

La valeur environnementale fixe est le résultat d'un tirage aléatoire dans une loi normale de moyenne 0 et de variance la variance environnementale « intra *Step* ».

Cette méthode prend comme paramètre une String (le nom du paramètre) et un GeneticTree. Elle renvoie un réel de type « double ».

**getPhenoValue (String parameterName, GeneticTree tree)**
Cette méthode calcule, pour un arbre, la valeur phénotypique d'un paramètre. Elle est appelée par la méthode *getPhenoValue ()* de la classe *GeneticTree*, qu'il est conseillée d'utiliser car elle s'appelle directement sur un *GeneticTree*.

La valeur phénotypique est calculée de la façon suivante :

$$P_i = G + E_{fixe} + E_{inter\ \ i}$$

avec : G valeur génotypique
$E_{fixe}$ valeur environnementale fixe
$E_{inter\ i}$ valeur inter environnementale de l'étape i

*getPhenoValue ()* utilise deux méthodes *getGeneticValue (), getFixedEnvironmentalValue ()* pour tirer la variance environnementale inter *Step* dans une loi normale de moyenne 0 et de variance la variance inter environnementale qui est égale à :

$$\sigma^2_{inter} = p(\sigma^2_{inter})\sigma^2_{totale}$$

avec $\sigma^2_{inter}$ : la variance environnementale inter Step.

et $p(\sigma^2_{inter})$ : la part de la variance environnementale inter Step

Elle prend comme paramètre une String (le nom du paramètre) et un *GeneticTree* (l'arbre pour lequel est calculé la valeur phénotypique du paramètre). Elle renvoie un double.

# 5 Présentation des classes outils de la bibliothèque

Les deux classes suivantes ne contiennent que des méthodes. Par la suite, si des méthodes génériques sont ajoutées, elles pourront être définies dans ces classes. Toutes les méthodes définies dans ces classes sont de type static. Elles s'appellent donc de la façon suivante : *nomDeLaClasse.nomDeLaMéthode (paramètres)*.

## *5.1 ImportGeneticData*

Les données génétiques chargées à l'initialisation sont relativement complexes (souvent des tableaux à deux dimensions). Pour faciliter le chargement de ces données, des méthodes générales ont donc été définies. Mais leur utilisation suppose de respecter, dans le fichier d'inventaire, des formats standard de données.

### 5.1.1 Les formats standard de données dans le fichier de chargement

Deux types de formats de données :
- une Collection d'éléments qui peuvent être des entiers ou des réels. C'est le format conseillé pour les données stockées dans des tableaux à une dimension, comme par exemple les génotypes et les probabilités de recombinaison.
Une Collection est encadrée par des accolades et contient une suite de nombres séparés par des virgules.

*Ex : {1, 6, 7, 3, 15}*

- une Collection de *VertexND*, de dimension variable. C'est le format conseillé pour les données stockées dans des tableaux à deux dimensions tels que le génotype nucléaire d'un arbre moyen, les valeurs des allèles ou encore les allèles possibles par locus.
Une Collection de V*ertexND* est encadrée par des accolades et contient une suite de *VertexND* séparés par des virgules. Chaque *VertexND* est délimité par des crochets et les éléments qu'il contient sont séparés par une virgule suivie d'un espace.

*Ex : {[0.5, 0.2, 0.3], [0.5, 0.5], [0.1, 0.1, 0.1 , 0.7], [1]}*

### 5.1.2 Les différentes classes de lecture de ces formats et de chargement

Cinq méthodes sont définies.

**importDiploideDNA (Collection c)**
A partir d'une Collection d'entiers donnée en paramètre, cette méthode construit et renvoie un tableau de short à deux dimensions (Figure 10).

Dans la Collection, les deux allèles d'un locus doivent être donnés successivement comme dans l'exemple ci-dessous :

*Exemple de deux loci a et b : {a1, a2, b1, b2}*
*a1 et b1 étant issus de la mère*
*a2 et b2 étant issus du père*

Le tableau construit contient, dans sa première colonne, les éléments de rang impair (maternels) et, dans sa deuxième colonne, les éléments de rang pair (paternels).

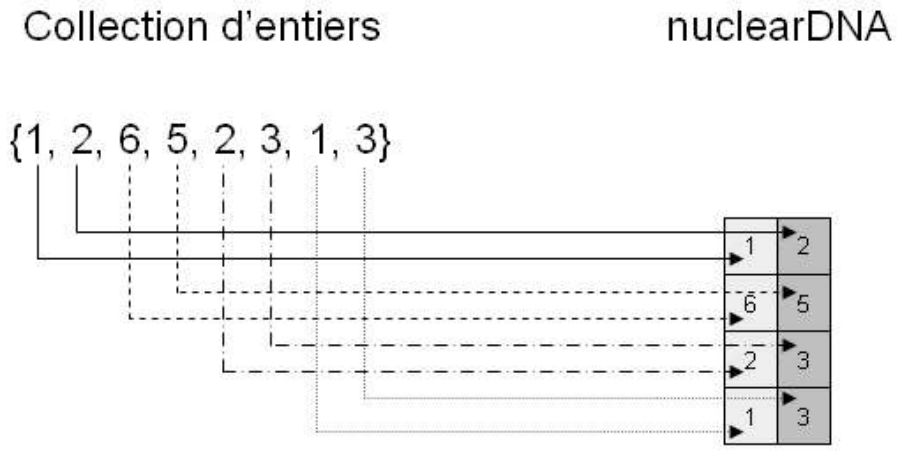


**Figure 10 : Construction d'un tableau nuclearDNA à partir d'une Collection d'entiers.**

**importHaploideDNA ()**
A partir d'une Collection d'entiers donnée en paramètre, cette méthode construit et renvoie un tableau de short à une dimension. Les éléments du tableau sont rangés dans l'ordre dans la Collection donnée en paramètre.

**importMultiGenotype ()**
A partir d'une Collection de *VertexND*, donné en paramètre, cette méthode construit et renvoie un tableau de int à deux dimensions.
Dans la Collection, chaque *VertexND* représente les effectifs de tous les allèles possibles pour un locus.
Le tableau construit comprend autant de ligne qu'il y a de *VertexND* dans la Collection. Chaque ligne contenant les éléments d'un *VertexND*. L'ordre des *VertexND* (celui des loci) et l'ordre des éléments à l'intérieur de chaque *VertexND* (celui des allèles, voir importMultiAllele) sont rigoureusement respectés.

**importMultiFrequence ()**
A partir d'une Collection de *VertexND*, donné en paramètre, cette méthode construit et renvoie un tableau de double à deux dimensions.
Dans la Collection, chaque *VertexND* représente les fréquences des allèles possibles sur un locus.
Le tableau construit comprend autant de ligne qu'il y a de *VertexND* dans la Collection. Chaque ligne contenant les éléments d'un *VertexND*. L'ordre des *VertexND* et l'ordre des éléments à l'intérieur de chaque *VertexND* sont rigoureusement respectés.

**importMultiAllele ()**

A partir d'une Collection de *VertexND*, donné en paramètre, cette méthode construit et renvoie un tableau de short à deux dimensions.
Dans la Collection, chaque *VertexND* représente la liste des allèles possibles sur un locus.
Le tableau construit comprend autant de ligne qu'il y a de *VertexND* dans la Collection. Chaque ligne contenant les éléments d'un *VertexND*. L'ordre des *VertexND* et l'ordre des éléments à l'intérieur de chaque *VertexND* sont rigoureusement respectés.

**importRecombinationProbas ()**
A partir d'une Collection de réels, donnée en paramètre, cette méthode construit et renvoie un tableau de short à une dimension. Les éléments du tableau sont rangés dans le même ordre que dans la Collection donnée en paramètre (séquence des loci).

## *5.2 GeneticTools*

Sont rassemblées dans cette classe des méthodes qui utilisent plusieurs objets de la bibliothèque. Cette classe contient 11 méthodes.

### 5.2.1 computeNuclearAlleleFrequencies ()

Cette méthode calcule, pour différents loci de l'ADN nucléaire, les fréquences alléliques dans une population d'arbres.

Cette méthode prend en paramètres une Collection, donnant la liste des arbres de la population, un tableau d'entiers donnant le rang des loci (= numéro de ligne dans le tableau des allèles possibles par locus) pour lesquels les fréquences alléliques sont à calculer et un boolean selon que le calcul des fréquences doit inclure ou exclure les allèles -1. Les arbres de la Collection peuvent être individuels (c-à-d avec un *IndividualGenotype*) ou moyens (c-à-d avec un *MultiGenotype*). Tous les arbres doivent être de la même espèce.

Elle renvoie un tableau de double à deux dimensions contenant les fréquences alléliques par locus. Ce tableau comporte autant de lignes qu'il y a d'éléments dans le tableau d'entiers donné en paramètre (soit une ligne par locus). Chaque ligne contient autant de cellules que d'allèles possibles sur le locus.

Pour calculer les fréquences alléliques, la méthode calcule d'abord les effectifs alléliques dans la sous population des arbres individuels. Puis elle somme les effectifs alléliques des arbres moyens et de la sous population des arbres individuels. Dans le calcul des fréquences, la sous population des arbres individuels est assimilée à un arbre moyen d'effectif le nombre d'arbres individuels qui le composent.
Pour faire ces calculs, la méthode passe d'abord en revue chaque arbre de la Collection et chaque locus donnés en paramètre pour calculer :

- le nombre d'arbre ayant un génotype *IndividualGenotype*,
- le nombre d'occurrence de chaque allèle possible dans la population des arbres avec *IndividualGenotype* stocké dans un tableau contenant en lignes les loci et en colonnes l'occurrence de chaque allèle en respectant l'ordre du tableau des allèles possibles,
- l'effectif total d'allèles différents de -1 dans la population des arbres avec *IndividualGenotype*

- la somme des effectifs des arbres moyens
- la somme des effectifs alléliques des arbres avec *MultiGenotype* stockée dans un tableau, qui comme précédemment, contient en lignes les loci et en colonnes les effectifs de chaque allèle en respectant l'ordre du tableau des allèles possibles,
- l'effectif total d'allèles différents de -1 sur l'ensemble des arbres moyens

Si le boolean *false* est donné en paramètre : les fréquences sont calculées en divisant le nombre d'occurrence de chaque allèle par l'effectif total d'allèles autres que -1. La fréquences de l'allèle -1 est nulle. Les fréquences alléliques sont calculées en excluant les allèles inconnus.
Si le boolean *true* est donné en paramètre : les fréquences sont calculées en divisant le nombre d'occurrence de chaque allèle par la somme des effectifs des arbres individuels et moyens. Les fréquences alléliques sont calculées en incluant les allèles inconnus.

**Note : Avant de calculer les fréquences alléliques, la méthode teste si tous les arbres de la Collection donnée en paramètre sont de la même espèce. Si ce n'est pas le cas, la méthode renvoie *null* et affiche un message dans la Log avertissant que le calcul n'a pu être réalisé. Ce test est également effectué par toutes les méthodes décrites ci-dessous.**

### 5.2.2 computeCytoplasmicAlleleFrequencies ()

Cette méthode calcule, pour différents loci d'un ADN cytoplasmique, les fréquences alléliques dans une population d'arbres (individuels et/ou moyens).

Cette méthode prend en paramètres une Collection, contenant la liste des arbres de la population, un tableau donnant le numéro des loci pour lesquels les fréquences alléliques sont à calculer, une String précisant l'origine de l'ADN (« maternal » ou « paternal ») et un boolean pour détreminer si les allèles -1 sont ou non exclus du calcul des fréquences. Comme précédemment tous les arbres de la Collection doivent être de la même espèce.

Elle renvoie un tableau de double à deux dimensions contenant les fréquences alléliques pour chaque locus. Ce tableau comporte autant de lignes qu'il y a d'éléments dans le tableau d'entiers donné en paramètre (soit une ligne par locus). Chaque ligne contient autant de cellules que d'allèles possibles pour le locus.

Le calcul des fréquences se fait selon le même principe que pour la méthode ci-dessus.

### 5.2.3 computeAlleleFrequencies ()

Cette méthode calcule, pour un ensemble de loci donné, les fréquences alléliques dans une population d'arbres (individuels et moyens). L'ensemble des loci peut contenir des loci des ADN nucléaires et cytoplasmiques. Pour distinguer l'ADN porteur de chaque locus, le numéro de chaque locus doit être précédé de « n_ » s'il est sur l'ADN nucléaire, « m_ » s'il est sur l'ADN cytoplasmique maternel ou « p_ » s'il est sur l'ADN cytoplasmique paternel.

Cette méthode prend en paramètres une Collection, contenant la liste des arbres (qui doivent tous être de la même espèce) et un Set contenant les loci.

Elle renvoie une Map dont les clés sont les numéros des loci précédés de n_, m_ ou p_ et dont les valeurs sont des tableaux à deux dimensions contenant les fréquences alléliques. La première ligne de ce tableau contient le nom des allèles (dans l'ordre où ils sont donnés dans *AlleleDiversity*) et la seconde leur fréquence.

Pour calculer les fréquences alléliques, cette méthode utilise les deux méthodes précédentes. Pour cela, à partir de l'ensemble des loci donnés en paramètre, elle construit, pour chaque ADN (nucléaire, cytoplasmiques maternel et paternel), 1 tableau d'entiers contenant les numéros des loci sur l'ADN.

### 5.2.4 computeGenotypeFrequencies ()

Cette méthode calcule, pour un ensemble de loci donné, les fréquences génotypiques dans une population d'arbres.

 Tous les arbres de la Collection doivent être des arbres individuels (avec un *IndividualGenotype*) et tous les loci doivent être portés par l'ADN nucléaire ; dans la liste des loci, les numéros de loci doivent donc tous être précédés de « n_ ».

Cette méthode prend en paramètres une Collection, contenant la liste des arbres, et un Set, contenant les numéros des loci précédés de « n_ ». Elle renvoie une Map dont les clés sont les numéros des loci précédés de « n_ » et dont les valeurs sont des tableaux à deux dimensions (3 lignes et autant de colonnes que de génotype soit n(n+1)/2 colonnes avec n le nombre d'allèles possibles) contenant les génotypes et leur fréquence (voir exemple ci-dessous) :

*Exemple d'un locus avec quatre allèles possibles 1, 2, 3 et 4 :*

*Le tableau des fréquences génotypiques ($p_i$) sera construit de la façon suivante :*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 4 |
| 1 | 2 | 3 | 4 | 2 | 3 | 4 | 3 | 4 | 4 |
| $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ | $p_7$ | $p_8$ | $p_9$ | $p_{10}$ |

Le principe de calcul est similaire à celui du calcul des fréquences alléliques. La méthode passe en revue chaque arbre de la Collection et chaque locus et comptabilise le nombre d'occurrence de chaque génotype possible hormis les génotypes qui contiennent l'allèle -1 et le nombre d'occurrence de génotypes ne contenant pas l'allèle -1. Elle divise ensuite le nombre d'occurrence par le nombre d'occurrence de génotypes ne contenant pas l'allèle -1.

**Les génotypes contenant l'allèles -1 sont exclus du calcul des fréquences génotypiques.**

**Nota :** Si un ou plusieurs arbres de la Collection possèdent un *MultiGenotype*, la méthode *computeGenotypeFrequencies ()* se réalise néanmoins entièrement, mais en ignorant les arbres avec *MultiGenotype* dans le calcul des fréquences génotypiques. Dans ce cas, un message s'inscrit dans la Log pour signaler à l'utilisateur que la Collection donnée en paramètre contient des arbres avec un *MultiGenotype* et pour préciser la liste des identifiants de ces arbres.

### 5.2.5 computePanmicticHeterozygoteFrequencies ()

Connaissant les fréquences alléliques de différents loci de l'ADN nucléaire, cette méthode calcule la fréquence théorique d'hétérozygotes, sous l'hypothèse de panmixie, dans une population d'arbres individuels.

Cette méthode prend comme paramètres une Collection, contenant la liste des arbres, et un Set, contenant les numéros de loci précédés de « n_ ».

Elle renvoie un tableau de double à une dimension contenant, pour chaque locus, la fréquence théorique d'hétérozygotes, en respectant l'ordre des loci du Set donné en paramètre à la méthode.

La fréquence théorique d'hétérozygotes pour un locus est égale à :

$$H_T = 1 - \sum_i q_i^2$$

i allant de 1 à n (nombre d'allèles)
$q_i$ = fréquence de l'allèle i

Pour calculer les fréquences alléliques, la méthode utilise la méthode *computeNuclearAlleleFrequencies ()* avec comme valeur du boolean *false*.

### 5.2.6 computeF ()

Cette méthode calcule le déficit relatif en hétérozygotes F (par rapport à la panmixie), dans une population d'arbres individuels et pour différents loci.
Elle prend comme paramètres une Collection (contenant le liste des arbres) et un Set (contenant les numéros des loci précédés de « n_ »). Elle renvoie un tableau de double à une dimension contenant pour chaque locus, le déficit en hétérozygotes, en respectant l'ordre des loci dans le Set donné en paramètre à la méthode.

Le déficit en hétérozygotes, pour un locus, est calculé selon l'équation suivante :

$$F = 1 - \frac{h_{obs}}{H_T}$$

$h_{obs}$ étant la fréquence d'homozygotes observée dans la population et
$H_T$ la fréquence théorique d'hétérozygotes (calculée par la méthode computePanmicticHeterozygoteFrequencies ).

Pour calculer F, cette méthode utilise les méthodes *computeGenotypeFrequencies ()* et *computePanmicticHeterozygoteFrequencies ()*, toutes deux prenant en paramètre la Collection et le Set donnés en paramètre à la méthode *computeF ()*.

**Nota** : Les deux méthodes utilisées par *computeF ()* ne gérant pas de la même façon les arbres avec *MultiGenotype* (*computePanmicticHeterozygoteFrequencies ()* les inclut alors que *computeGenotypeFrequencies ()* les ignore), si la Collection donnée en paramètre contient un

arbre avec un *MultiGenotype*, la méthode *computeF ()* renvoie null et affiche un message dans la Log prévenant que le calcul de F n'a pas pu être réalisé.

### 5.2.7 computePanmicticGeneticMean ()

Cette méthode calcule, pour un paramètre, la moyenne génétique dans une population d'arbres sous l'hypothèse de panmixie.
*NB1 : Dans le cadre d'un modèle purement additif, l'hypothèse de panmixie est sans effet sur la moyenne de la population*

Elle prend comme paramètres une Collection contenant des arbres individuels et/ou moyens et une String (le nom du paramètre).

Elle renvoie un double.

Sous l'hypothèse d'additivité des allèles, la moyenne génétique est calculée de la façon suivante :

$$\overline{G} = \sum_{loci\ de\ l'ADN\ nucléaire} 2\sum_{i} p_i\alpha_i + \sum_{loci\ des\ ADN\ cyto.mat} \sum_{i} p_i\alpha_i + \sum_{loci\ des\ ADN\ cyto.pat} \sum_{i} p_i\alpha_i$$

i allant de 1 à n (nombre d'allèles du locus)
j allant de 1 à n (nombre d'allèles du locus)
$p_i$ fréquence de l'allèle i du locus
$\alpha_i$ valeur de l'allèle i

Pour calculer la moyenne génétique, la méthode utilise les méthodes *computeNuclearAlleleFrequencies ()* et *computeCytoplasmicAlleleFrequencies ()* avec comme valeur du boolean *false*.

### 5.2.8 computePanmicticGeneticVariance ()

Cette méthode calcule, pour un paramètre, la variance génétique dans une population d'arbres sous l'hypothèse de panmixie et d'indépendance des loci.

Elle prend comme paramètres une Collection contenant des arbres individuels et/ou moyens et une String (le nom du paramètre).

Elle renvoie un double.

La variance génétique est calculée de la façon suivante :

$$\sigma^2{}_G = \sum_{loci\ de\ l'ADN\ nucléaire} 2 * \sum_{i} p_i(\alpha_i - \overline{\alpha})^2 + \sum_{loci\ des\ ADN\ cyto.mat} \sum_{i} p_i(\alpha_i - \overline{\alpha})^2 + \sum_{loci\ des\ ADN\ cyto.pat} \sum_{i} p_i(\alpha_i - \overline{\alpha})^2$$

$$\overline{\alpha} = \sum_{allele} p_i\alpha_i$$

i et j allant de 1 à n (nombre d'allèles sur le locus)
$p_i$ fréquence de l'allèle i

$\alpha_i$ valeur de l'allèle i

Pour calculer la variance génétique, la méthode utilise les méthodes *computeNuclearAlleleFrequencies ()* et *computeCytoplasmicAlleleFrequencies ()* avec comme valeur du boolean *false*.

### 5.2.9 computeObservedGeneticMean ()

Cette méthode calcule, pour un paramètre, la moyenne génétique dans une population d'arbres. Contrairement à la méthode *computePanmicticGeneticMean*, l'effet des allèles inconnus (codés -1) est ici pris en compte.

*NB : Dans le cadre d'un modèle purement additif, la moyenne de la population pourrait être estimée à partir des fréquences des allèles.*

Elle prend comme paramètres une Collection contenant des arbres individuels et une String (le nom du paramètre). Comme pour la méthode *computeObservedGeneticVarianc*e, les arbres moyens ne sont pas autorisés car ils ne permettent pas (dans la version actuelle de la bibliothèque) de connaître la fréquence des génotypes.

Elle renvoie un double.

La moyenne génétique est calculée à partir des valeurs génétiques des individus :

$$\overline{G} = \frac{1}{n} \sum_{individus} G$$

### 5.2.10computeObservedGeneticVariance ()

Cette méthode calcule, pour un paramètre, la variance génétique dans une population d'arbres.

Elle prend comme paramètres une Collection contenant des arbres individuels uniquement et une String (le nom du paramètre).

Elle renvoie un double.

La variance génétique est calculée à partir des valeurs génétiques des individus :

$$\sigma^2_G = \frac{1}{n} \sum_{individus} (G - \overline{G})^2$$

Pour calculer la variance génétique, la méthode utilise les méthodes *computeObservedGeneticMean ()* et *getGeneticValue ()*.

**Tableau 4 : rappel des types des paramètres et des objets renvoyés par les méthodes de *GeneticTools*.**

| Paramètres | | | | Objet renvoyé |
|---|---|---|---|---|
| computeNuclearAlleleFrequencies | | | | |
| **Collection** Arbres individuels et /ou moyens | **int[]** : tableau des numéros de loci de l'ADN nucléaire *ex : [1, 5]* | **Boolean** : calcul des fréquences avec ou sans allèles -1 | | **double[][]** |
| computeCytoplasmicAlleleFrequencies | | | | |
| **Collection** Arbres individuels et /ou moyens | **int[]** : tableau des numéros de loci, d'un ADN cytoplasmique *ex : [2, 3, 9]* | **String** : « maternal » ou « paternal » selon l'origine de l'ADN cytoplasmique | **Boolean** : calcul des fréquences avec ou sans allèles -1 | **double[][]** |
| computeAlleleFrequencies | | | | |
| **Collection** Arbres individuels et /ou moyens | **Set** : ensemble des numéros de loci précédés d'un caractère codant l'ADN porteur du loci *ex : (n_1, n_5, m_2, m_3, m_9)* | | | **Map** |
| computeGenotypeFrequencies | | | | |
| **Collection** Arbres individuels | **Set** : ensemble des numéros de loci précédés d'un caractère codant l'ADN porteur du loci *ex : (n_1, n_5, m_2, m_3, m_9)* | | | **Map** |
| computePanmicticHeterozygoteFrequencies | | | | |
| **Collection** Arbres individuels et /ou moyens | **Set** : ensemble des numéros de loci de l'ADN nucléaire précédés de n_ *ex : (n_1, n_5)* | | | **double[]** |
| computeF | | | | |
| **Collection** Arbres individuels | **Set** : ensemble des numéros de loci de l'ADN nucléaire précédés de n_ *ex : (n_1, n_5)* | | | **double[]** |
| computePanmicticGeneticMean | | | | |
| **Collection** Arbres individuels et /ou moyens | **String** : nom du paramètre | | | **double** |
| computePanmicticGeneticVariance | | | | |
| **Collection** Arbres individuels et /ou moyens | **String** : nom du paramètre | | | **double** |
| computeObservedGeneticMean | | | | |
| **Collection** Arbres individuels | **String** : nom du paramètre | | | **double** |
| computeObservedGeneticVariance | | | | |
| **Collection** Arbres individuels | **String** : nom du paramètre | | | **double** |

# 6 Présentation des classes de validation des données initiales

Ces classes ont été définies pour permettre de vérifier, à l'issue du chargement des données, que les données initiales sont valides. Les données sont valides lorsque :

- les données sont compatibles entre elles. Exemple tous les arbres individuels d'une même espèce ont un génotype de même format.
- les données nécessaires aux processus élémentaires (exemple : méiose) sont définies. Des méthodes permettent de calculer certaines de ces données par défaut. Exemple : la carte génétique ou la diversité allélique.

Quatre classes ont été définies :

- *Validate* : dans cette classe est définie la méthode validateInitialData () qui réalise toutes les procédures de validation. L'utilisateur doit appeler cette méthode à l'issue du chargement des données.
- *ValidateTools* : cette classe contient des méthodes outils qui sont appelées par les autres classe : méthode pour construire les Map des espèces et méthodes pour compléter des tableaux à deux dimensions.
- *AreGeneticDataCompatible* : dans cette classe sont définies les méthodes qui testent la compatibilité des données
- *CompleteInitialData* : dans cette classe sont définies les méthodes qui calculent par défaut des données manquantes.

A chaque étape de la validation :
- si des données non compatibles sont repérées, le chargement du fichier est interrompu et un message d'erreur s'affiche précisant le type d'erreur détecté
- si les données ont été modifiées ou complétées, le chargement s'effectue normalement et un message d'information précisant les modifications apportées est affiché.

## *6.1 Validate*

Cette classe est composée d'une méthode ValidateInitialData (). Celle-ci construit tout d'abord différentes Map selon le type d'arbres (arbres non génotypé, arbres individuels avec un génotype vide, arbres individuel génotypés, arbres moyens avec génotype vide, arbres moyens génotypés). Différents tests sont effectués à partir de ces Map.
Elle appelle ensuite les méthodes des classes AreGeneticDataCompatible et CompleteInitialData.

### 6.1.1 Construction des Map par espèce

Cinq Map par espèce sont constituées :

- la Map contenant les arbres non génotypés : *speciesWithoutGenotype*
- la Map contenant les arbres individuels dont le génotype est vide : *speciesIndividualWithEmptyGenotype*
- la Map contenant les arbres individuels génotypés : *speciesIndividualWithGenotype*

- la Map contenant les arbres moyens dont le génotype est vide : *speciesPopulationWithEmptyGenotype*
- la Map contenant les arbres moyens génotypés : *speciesPopulationWithGenotype*

La construction de ces Map dépend du type de données : peuplement initial contenant une seule ou plusieurs espèces. La méthode teste donc si les arbres sont des instances de *SpeciesDefined* (vrai si plusieurs espèces, faux sinon). S'il y a plusieurs espèces, les clés des Map sont les espèces. Sinon chaque Map ne contient qu'une seule clé respectivement *WithoutGenotype*, *IndividualTreeWithGenotype*, *IndividualTreeWithEmptyGenotype*, *PopulationWithGenotype*, *PopulationWithEmptyGenotype*.
Pour la construction de ces différentes Map, *ValidateInitialData ()* appelle la méthode *builtMapSpecies ()* de la classe *ValidateTools*.

### 6.1.2 Test de la compatibilité des données

Pour tester la compatibilité des données, la méthode *testIfInitialDataNotValide ()* de la classe *AreGeneticDataCompatible* est appelée.
Si la méthode *testIfInitialDataNotValide ()* détecte une erreur, la procédure de validation est arrêtée et un message précisant l'erreur détectée est envoyé.

### 6.1.3 Calcul par défaut des données manquantes

La méthode *validateInitialData ()* réalise plusieurs opérations successives. A chaque étape, elle teste si des compléments sont à effectuer. Si oui, elle construit les variables à donner en paramètres aux différentes méthodes de calcul de données par défaut. Ces méthodes appelées par *validateInitialData ()* sont définies dans *CompleteInitialData* ou dans d'autres classes.
*validateInitialData ()* complète également, au fur et à mesure que des données par défaut sont calculées et des données complétées, le message d'avertissement qui est affiché à l'issue de la validation.
Les opérations successives sont :

- Si, pour une espèce génotypée, *AlleleDiversity* n'est pas donnée dans le fichier d'inventaire, la méthode calcule une *AlleleDiversity* par défaut en appelant la méthode *computeAlleleDiversity ()* de la classe *AlleleDiversity* et l'affecte à l'espèce.
- Pour chaque espèce, s'il y a des arbres individuels avec un génotype vide, la méthode calcule un *IndividualGenotype* par défaut (tous les allèles égaux à -1), et leur attribue. Elle enregistre dans un Set, appelé *completedIndividualGenotype*, les espèces pour lesquelles des arbres individuels avaient un génotype vide.
- Pour chaque espèce du Set *completedIndividualGenotype*, la méthode teste si *AlleleDiversity*, les *MultiGenotype* des arbres moyens et *AlleleEffect* sont complets (c-à-d si ils contiennent, respectivement, les allèles -1, les fréquences des allèles -1 et les valeurs des allèles -1 pour tous les loci). Elle les complète si besoin.
- Pour chaque espèce, s'il y a des arbres moyens avec un génotype vide, la méthode calcule un *MultiGenotype* par défaut, en appelant la méthode *computeDefaultMultiGenotype ()* de la classe *MultiGenotype*, et leur affecte.
- Si, pour une espèce génotypé, *GeneticMap* n'est pas donnée dans le fichier d'inventaire, la méthode calcule une *GeneticMap* par défaut, en appelant la méthode

*computeDefaultRecombinationProbas ()* de la classe *GeneticMap*, et l'attribue à l'espèce.

- Si, pour une espèce génotypée, *AlleleEffect* est incomplet (héritabilité ou variance environnementale totale égale à -1), la méthode calcule la valeur par défaut en appelant, *getHeritability ()* ou *getTotalEnvironmentalVariance ()* de la classe *AlleleEffect.*

## *6.2 ValidateTools*

### 6.2.1 builtMapSpecies ()

Cette méthode teste si la Map donnée en paramètre contient déjà la clé donnée en paramètre.
Si oui, elle ajoute dans le Set associé à cette clé le *GeneticTree* donné en paramètre.
Si non, elle crée une nouvelle clé et son Set associé dans lequel elle ajoute le *GeneticTree* donné en paramètre.
Deux méthodes du même nom ont été définies qui diffère par le type de clé de la Map :

- la première prend en paramètres : une Map, une String (le nom de la clé) et un *GeneticTree*.
- la seconde prend en paramètres : une Map, une *QualitativeProperty* (la clé) et un *GeneticTree*.

Ce qui permet d'utiliser la même méthode pour construire les Map par espèce qu'il y est une ou plusieurs espèces.

### 6.2.2 addValueInLigneOfArray ()

Cette méthode utilitaire permet d'ajouter dans une ligne d'un tableau à deux dimensions une cellule et de lui affecter une valeur. Deux méthodes du même nom ont été définies pour que la méthode puisse être réalisée aussi bien sur un tableau de short que sur un tableau de double.
Elles prennent comme paramètres

- un tableau de short (le tableau à modifier), un short (la valeur à ajouter) et un int (le numéro de la ligne à compléter)
- un tableau de double (le tableau à modifier), un double (la valeur à ajouter) et un int (le numéro de la ligne à compléter

Elle ne renvoie rien (méthode void).

## *6.3 AreGeneticDataCompatible*

Cette classe comporte une seule méthode *testIfInitialDataNotVadide ()* qui effectue successivement tous les différents tests de compatibilité des données. Successivement, la méthode :

- teste si, au sein d'une même espèce, les *IndividualGenotype* des arbres individuels génotypés ont le même format (même nombre de lignes).
- teste si, pour chaque espèce, lorsque des arbres moyens ont un génotype, *AlleleDiversity* de l'espèce est donnée.

- teste si, au sein d'une même espèce, tous les génotypes des arbres moyens génotypés ont le même format (même nombre de lignes et, pour une ligne donnée, même nombre de colonnes.
- teste si, au sein d'une même espèce, les génotypes des arbres individuels et moyens génotypés sont compatibles (même nombre de lignes)
- teste si, pour chaque espèce, l'*AlleleDiversity* contient, pour chaque locus, tous les allèles qui apparaissent dans les génotypes des arbres individuels génotypés. Pour cela la méthode construit l'*AlleleDiversity* par défaut en appelant la méthode *computeAlleleDiversity ()* de la classe *AlleleDiversity*. Elle compare ensuite cette *AlleleDiversity* à celle donnée dans le fichier d'inventaire : pour chaque tableau et chaque ligne la méthode vérifie que l'*AlleleDiversity* du fichier d'inventaire contient tous les allèles de l'*AlleleDiversity* calculée par défaut.
- teste si, pour une espèce donnée, le génotype d'un arbre moyen et l'*AlleleDiversity* de l'espèce ont des formats identiques : même nombre de lignes et, pour chaque ligne, même nombre de colonnes.
- teste si, pour chaque espèce, le tableau des valeurs des allèles (de *AlleleEffect*) et *AlleleDiversity* sont compatibles. Chaque ligne des tableaux de *AlleleEffect* doit avoir 1 colonne de plus que la ligne correspondante dans *AlleleDiversity*.
- teste si, pour chaque espèce, *AlleleEffect* est complet c'est-à-dire si les tableaux des valeurs des allèles ne sont pas vides, si la part de la valeur environnementale inter Step est différente de -1 et si héritabilité et variance environnementale totale ne sont pas toutes deux égales à -1.

La méthode teste une à une ces conditions de compatibilité. Si l'une n'est pas remplie, l'initialisation du module est arrêtée et un message d'erreur est envoyé.

## *6.4 CompleteInitialData*

Cette classe comporte trois méthodes statiques qui sont appelées par *validateInitialData ()* de la classe *Validate*.

### 6.4.1 completeAlleleDiversity ()

Cette méthode n'est appelée que si des *IndividualGenotype* par défaut ont été attribués à des arbres individuels du peuplement initial. Dans ce cas, *completeAlleleDiversity ()* vérifie que, pour chaque locus, l'allèle inconnue (-1) est donnée. Si ce n'est pas le cas, l'allèle -1 est ajouté par la méthode *addValueInLigneOfArray ()* et le message d'avertissement est complété.
NB : Par défaut, les fréquences des allèles sont calculées (par la méthode *computeAlleleFrequencies())* sans tenir compte de l'allèle inconnue (-1).
La méthode *completeAlleleDiversity()* prend en paramètres un *GeneticTree* et une String (le message d'avertissement). Elle ne renvoie rien.

### 6.4.2 completeMultiGenotype ()

Cette méthode n'est appelée que si des *IndividualGenotype* par défaut ont été attribués à des arbres individuels du peuplement initial. Dans ce cas, *completeMultiGenotype ()* vérifie que les *MultiGenotype* donnés à l'initialisation contiennent, pour chaque locus, les effectifs

d’allèles inconnus (fréquence de la valeur -1) en comparant la taille des tableaux des effectifs alléliques à la taille des tableaux des allèles possibles. Si la fréquence des allèles inconnus n’est pas incluse, elle est ajoutée par la méthode *addValueInLigneOfArray ()* et le message d’avertissement est complété.
NB : Par défaut, les effectifs des allèles inconnues sont égaux à 0.
Cette méthode prend en paramètres un *GeneticTree*, 3 tableaux de short (les tableaux de *AlleleDiversity*) et une String (le message d’avertissement). Elle ne renvoie rien.

### 6.4.3 completeAlleleEffect ()

Cette méthode n’est appelée que si *AlleleEffect* existe (non null et non vide) et si des *IndividualGenotype* par défaut ont été attribués à des arbres individuels du peuplement initial. Dans ce cas, *completeAlleleEffect ()* vérifie (indirectement) que les tableaux des valeurs des allèles donnés à l’initialisation contiennent, pour chaque locus, les valeurs des allèles inconnus (valeurs des allèles -1), en comparant la taille des tableaux des valeurs à la taille des tableaux des allèles possibles. Si les valeurs des allèles inconnus ne sont pas incluses, elles sont ajoutées par la méthode *addValueInLigneOfArray ()* et le message d’avertissement est complété.
NB : Par défaut, les valeurs des allèles inconnus sont égales à 0.
Cette méthode prend en paramètres un *GeneticTree*, 3 tableaux de short (les tableaux de *AlleleDiversity*) et une String (le message d’avertissement). Elle ne renvoie rien.

# 7 Présentation des classes d'extracteurs de données

## 7.1 DEAlleleFrequencies

*DEAlleleFrequencies* est un extracteur de données. Cette classe permet de représenter sur une ou plusieurs étapes l'histogramme cumulé des fréquences alléliques par locus.
Pour visualiser cette sortie, l'utilisateur doit d'abord définir, dans le gestionnaire, un groupe d'arbres de même espèce. Il doit ensuite configurer la sortie en la demandant sur le groupe qu'il a défini précédemment et en sélectionnant les loci pour lesquels il souhaite représenter les fréquences alléliques. Il a également le choix de plusieurs labels : aucun, uniquement les fréquences, uniquement le nom des allèles ou simultanément les fréquences et les noms des allèles (nom_fréquence).
*DEAlleleFrequencies* lit le groupe d'arbres et les loci sélectionnés. Elle utilise la méthode *computeAllelefrequencies ()* pour calculer les fréquences alléliques et construit les vecteurs de données et les vecteurs de labels de l'histogramme.

## 7.2 DEGenotypeFrequencies

*DEGenotypeFrequencies* est un extracteur de données. Cette classe permet de représenter sur une ou plusieurs étapes l'histogramme cumulé des fréquences génotypiques par locus.
Pour visualiser cette sortie, l'utilisateur doit d'abord définir, dans le gestionnaire, un groupe d'arbres de même espèce. Il doit ensuite configurer la sortie en la demandant sur le groupe qu'il a défini précédemment et en sélectionnant les loci pour lesquels il souhaite représenter les fréquences génotypiques. Il a également le choix de plusieurs labels : aucun, uniquement les fréquences, uniquement le génotype ou simultanément les fréquences et les génotypes.
*DEGenotypeFrequencies* lit le groupe d'arbres et les loci sélectionnés. Elle utilise la méthode *computeGenotypeFrequencies ()* pour calculer les fréquences génotypiques et construit les vecteurs de données et les vecteurs de labels de l'histogramme.

**NB : pour une meilleure lecture des histogrammes, les fréquences alléliques et génotypiques sont portées en %.**

# 8 Présentation des classes d'exportation de données

Un format d'exportation de données génétiques est défini. Il correspond au format d'entrée du logiciel GenePop. Ce format donne le génotype, pour les différents loci, d'individus regroupés en sous populations. L'exportation est réalisée par la classe *GenePopExport*. Une fenêtre de dialogue (gérée par la classe *GeneticsDGenePopExport)* permet de définir les loci à exporter ainsi que les clés de fractionnement en sous populations. L'information est transmise à *GenePopExport* par le biais de *GenePopExportSettings*.
Il n'est pas possible d'exporter le génotype des arbres moyens au format GenePop.

## *8.1 Description du format GenePop*

Le format GenePop donne successivement :

- une information générale sur les données
- les noms des loci
- le nom et le génotype des individus regroupés en sous populations

#### *8.1.1.1 Information générale sur les données*

La première ligne est une ligne d'information sur les données. La classe d'exportation y enregistre la date et l'heure d'enregistrement du fichier de sortie.

#### *8.1.1.2 Nom des loci*

Les lignes suivantes contiennent les noms des loci. Dans Capsis, les noms des loci sont composés d'une lettre et d'un nombre. La lettre donne l'ADN qui porte le locus : n pour ADN nucléaire, m pour ADN cytoplasmique d'origine maternelle et p pour ADN cytoplasmique d'origine paternelle. Le chiffre donne le numéro du locus (son numéro de ligne dans *AlleleDiversity*).

#### *8.1.1.3 Nom et génotype des individus*

**Nom**
La procédure d'exportation au format GenePop attribue à chaque individu le nom de la sous population à laquelle il appartient. Le nom d'une sous population est construit à partir des clés de fractionnement en sous populations.

**Génotype**
Dans GenePop, le génotype est codé par 2n chiffres (n = 2, 3 ou 4). Les allèles manquants sont codés 00 (ou 000 ou 0000) et, dans le cas de génotype haploïde, le premier allèle est codé comme un allèle manquant.
La procédure d'export au format GenePop de Capsis, code toujours un allèle par 3 chiffres (n=3). Chaque allèle peut donc être codé de 001 à 999. Les données manquantes sont codées « 0-1 », car 000 peut correspondre à un allèle et, dans le cas de génotype haploïde, le premier allèle est codé « --- ».

Les sous populations sont séparées par « pop ».

## *8.2 GeneticsDGenePopExport*

*GeneticsDGenePopExport* génère la boîte de dialogue permettant de formater le format de sortie, lit les informations saisies par l'utilisateur, construit la Map des sous populations et la Map des loci à partir de ces informations et stocke, dans un objet *GenePopExportSettings*, les informations nécessaires à *GenePopExport* pour construire le fichier de sortie.

## *8.3 GenePopExportSettings*

Cette classe définit l'objet *GenePopExportSettings*. Cet objet est construit par *GeneticsDGenePopExport* et lu par *GenePopExport*. Il est composé de deux Map :

- la Map des sous populations : les clés sont les noms des sous populations. A chaque clé est associé un Set qui contient les arbres de la sous population.
- la Map des loci : cette Map contient trois clés respectivement les chaînes de caractère « NuclearDNA », « mCytoplasmicDNA » et « pCytoplasmicDNA ». A chaque clé est associé un entier qui vaut 1 si le génotype de l'ADN correspondant est à exporter, 0 sinon.

### 8.3.1 Description de la boîte de dialogue (méthode *createUI ()*)

La première rubrique demande le nom du groupe à exporter. Un groupe doit être composé d'arbre d'une même espèce.
La deuxième rubrique demande le premier critère de ventilation des individus du groupe en sous population. Ce premier critère est un critère dendrométrique à choisir entre Age, Diamètre et Hauteur. Un seul critère peut être choisi. Ce critère étant une variable continu, l'utilisateur doit également spécifier la taille des classes à constituer (pour des classes de diamètre de 5 en 5cm par exemple, saisir 5).
La troisième rubrique demande le second critère de ventilation. Ce critère est un critère génétique. L'utilisateur doit choisir entre Identifiant de la mère, Identifiant du père et Date de création. Un seul critère génétique peut être choisi. Le critère génétique étant considéré comme une variable discrète, une sous population est construite pour chaque valeur de ce critère.
La dernière rubrique demande les loci à exporter. L'utilisateur doit cocher les ADN (nucléaire, maternel cytoplasmique et / ou paternel cytoplasmique) qui portent les loci à exporter.

### 8.3.2 Construction des sous populations (méthode *okAction ()*)

A partir des données saisies par l'utilisateur, *GeneticsDGenePopExport* construit les sous populations.
Le nom de la sous population est construit de la façon suivante :

- le nom du groupe précédé de « Group »
- le nom du critère dendrométrique et sa classe
- le nom du critère génétique et sa valeur

*Par exemple : GroupSapin_Age30:40_Identifiant de la mère=21*

La méthode commence par construire la Map des sous Populations. A ce stade la Map contient une clé « allTreeWithGenotype » associé à un Set qui contient tous les individus génotypés.

Si un groupe est sélectionné, la méthode construit une Map temporaire. Elle essaie ensuite de construire le groupe en appelant la méthode *group.apply ()* (méthode de Capsis.util). Si la procédure s'exécute sans erreur, la méthode ajoute un élément à la Map temporaire avec comme clé le nom du groupe précédé de « Group » (*Exemple clé = « GroupSapin »*) et comme Set la liste des arbres du groupe. La Map temporaire remplace alors la Map des sous populations. A ce stade la Map des sous populations contient un unique élément.
La méthode vérifie ensuite que tous les arbres du groupe sont bien de la même espèce. Si ce n'est pas le cas, la procédure d'exportation est arrêtée et un message d'erreur est affiché.

Lorsqu'un critère dendrométrique est donné, la méthode construit une Map temporaire. Elle calcule le nombre de classes (en fonction des valeurs minimales et maximales observées dans le groupe sélectionné) et la borne inférieure de la première classe. La méthode passe ensuite en revue chaque arbre du groupe, lit sa valeur et en déduit sa classe dendrométrique. Si la clé correspondant à cette classe n'existe pas dans la Map temporaire, la méthode ajoute un élément à la Map avec comme clé le nom de la sous population (*Exemple : clé = « GroupSapin_Age10 :20 »*) et comme valeur un Set qui contient l'arbre. Sinon, elle ajoute directement l'arbre dans le Set correspondant à sa classe. Lorsque tous les arbres ont été parcourus, la Map temporaire remplace la Map des sous populations.

Lorsqu'un critère génétique est sélectionné, la méthode procède de façon analogue. Elle construit une Map temporaire qu'elle complète en parcourant les arbres de chaque sous population de la Map des sous populations. Lorsque tous les arbres ont été parcourus, la Map temporaire remplace la Map des sous populations.

### 8.3.3 Construction de la Map des loci

A partir des données saisies par l'utilisateur, la classe construit la Map des loci à exporter. Pour chaque clé, respectivement les chaînes de caractère « NuclearDNA », « mCytoplasmicDNA » et « pCytoplasmicDNA », la classe associe la valeur 0 ou 1.

Finalement la classe *GeneticsDGenePopExport* construit un objet de type *GenePopExportSettings*.

## *8.4 GenePopExport*

Le fichier de sortie est construit par la méthode *createRecordSet ()* qui prend comme paramètre la Map des sous populations et la Map des loci.

A partir du génotype d'un arbre t du groupe, la méthode écrit les lignes donnant les noms de loci. Pour cela elle passe en revue chaque clé de la Map des loci. Si la valeur correspondante est égale à 1 (= si l'utilisateur demande à exporter les loci portés par l'ADN correspondant au nom de la clé), la méthode lit le tableau des génotypes correspondant (*nuclearAlleleDNA*,

*mCytoplasmicDNA* ou *pCytoplasmicDNA*) de l'arbre t. Pour chaque ligne de ce tableau, la méthode construit le nom du locus : lettre codante (n, m ou p) + numéro de la ligne.

La méthode passe ensuite en revue chaque sous population de la Map des sous populations. A chaque étape elle inscrit « pop » dans le fichier de sortie.
Pour chaque sous population, la méthode passe en revue chaque arbre et inscrit le nom de la sous population suivi du génotype de l'arbre. Le génotype est lu dans l'objet *Genotype* et est retranscrit au format adapté pour *GenePopExport* (Figure 11).

**NB : Si certains des allèles exportés sont inconnus, codés 0-1, ou portés par un gène haploïde, code de l'allèle précédé de ---, l'utilisateur doit remplacer ces chaînes de caractères par « 000 » avant d'utiliser GenePop. Par ailleurs, Capsis insère une ligne vide après la ligne de commentaire et en fin de fichier. L'utilisateur doit les supprimer avant d'utiliser GenePop.**

```
# Capsis 4.1.1 generated file - Fri Dec 13 16:54:04 CET 2002

n1
n2
n3
n4
n5
n6
n7
n8
n9
n10
pop
GroupSapin_Age30:40 , 002002 002002 003004 002002 002002 002003 003003 003003 002002 001001
GroupSapin_Age30:40 , 002002 002004 004004 002002 003003 002002 003003 003003 001002 001002
GroupSapin_Age30:40 , 002003 002002 004004 002002 002002 002002 003003 003003 001002 001001
GroupSapin_Age30:40 , 002002 002002 004004 002002 002002 002003 003003 002003 002002 001002
pop
GroupSapin_Age40:50 , 002002 002002 003004 002002 002002 002003 003003 003003 002002 001001
GroupSapin_Age40:50 , 002002 002002 003004 002002 002002 002003 003003 003003 002002 001001
GroupSapin_Age40:50 , 002002 002002 004004 002002 002002 002002 003003 003003 002002 001002
GroupSapin_Age40:50 , 002002 002002 004004 002002 002002 002003 003003 003003 002002 001002
GroupSapin_Age40:50 , 002002 002004 003003 002002 002002 002002 003003 002002 001002 001002
GroupSapin_Age40:50 , 002002 002002 004004 002002 002002 002002 003003 003003 002002 001001
GroupSapin_Age40:50 , 002002 002002 004004 002002 002002 002003 003003 003003 002002 001001
GroupSapin_Age40:50 , 002002 002002 003003 002002 002002 002002 003003 003003 001002 001001
```

**Figure 11 : exemple d'un fichier exporté par Capsis au format GenePop.**

# 9 Présentation de la classe « outils de modélisation » : *GeneticsGeneration*

Une classe outil a été définie pour permettre de générer des données génétiques : des loci et des valeurs adaptatives d'allèles pour différents paramètres. Cette classe peut être appelée sur l'étape racine tant qu'aucune étape fille n'a été générée.

## *9.1 Principes généraux*

Les données génétiques sont générées par espèce. L'utilisateur doit donc avant tout choisir une espèce parmi une liste proposée dans un menu déroulant.

### 9.1.1 Génération de génotype

L'utilisateur saisit pour chaque locus qu'il souhaite ajouter au génotype :

- une lettre codant l'ADN porteur du gène (n pour ADN nucléaire, m pour ADN cytoplasmique d'origine maternelle ou p pour ADN cytoplasmique d'origine paternelle)
- suivie du nombre d'allèles possibles sur ce nouveau locus

Plusieurs loci peuvent être donnés à la suite. Ils doivent être séparés par des espaces.
*Exemple : n3 n7 p5*

A partir de ces informations, la méthode complète ou crée le génotype des arbres individuels et moyens sous l'hypothèse d'équiprobabilité des allèles. Pour un arbre individuel, l'allèle est donc tiré aléatoirement (chaque allèle ayant la même probabilité 1/N d'être tiré) et pour un arbre moyen tous les allèles ont le même effectif égal à la partie entière de l'effectif de l'arbre moyen divisé par le nombre d'allèles possibles, noté N. Si du fait des arrondis, la somme des effectifs alléliques par locus est différente de 2N ou N (selon l'ADN), la méthode ajuste les effectifs alléliques par un processus stochastique. La méthode complète également *GeneticMap*, en supposant que le nouveau locus n'est pas lié au précédent (probabilité de recombinaison avec le locus précédent égale à 0.5) et *AlleleDiversity*, en codant les allèles de 1 à N.

NB : Il est prévu de modifier prochainement cette procédure de génération de loci afin de paramétrer : a) les fréquences des allèles, b) les probabilités de recombinaison entre loci.

### 9.1.2 Génération de valeur adaptative des allèles codant pour un paramètre

L'utilisateur saisit :

- le nom du paramètre,
- les loci, qui influencent ce paramètre, codés par une lettre (n, m ou p selon l'ADN porteur) et un numéro (le numéro de la ligne dans *AlleleDiversity*) *Exemple n1 n2 p1*
- la part de la variance environnementale et l'héritabilité ou la variance environnementale totale.

A partir ce des données, la classe ajoute le paramètre dans la liste des clés de *AlleleEffect* et calcule le *ParameterEffect* associé au paramètre :

- les valeurs des allèles sont tirées aléatoirement dans une loi normale de moyenne 0 et de variance 100
- si la variance environnementale totale est égale à -1, elle est calculée à partir l'héritabilité.

Si les données ne sont pas complètes, la classe affiche un message d'erreur, de même si héritabilité et variance environnementale totale sont toutes deux égales à -1.

## *9.2 Les différentes méthodes de la classe*

### 9.2.1 Conception de la classe

*createUI ()* génère la boîte de dialogue qui comporte plusieurs cadres.
Le cadre contexte permet de sélectionner l'espèce pour laquelle l'utilisateur souhaite générer des données génétiques. La liste des espèces possibles est donnée dans un menu déroulant construit par la méthode *createUI ()*. Elle est calculée à l'appel de la méthode *searchSpecies ()* au début de la méthode *createUI ()*.
Les deux cadres suivants (Génération de nouveaux loci et Génération de nouvelles valeurs) sont composés de plages de saisies et d'un bouton Action (Ajouter Loci et Ajouter). Lorsque l'utilisateur clique sur ces boutons actions, les Map contenant les descriptifs des opérations à réaliser sont mis à jour. La Map *newLociActions* est mise à jour par la méthode *addLociAction ()* déclenchée par le clic sur le bouton « Ajouter Loci » et la Map *newEffectActions* par la méthode *addEffectAction ()* déclenchée par le clic sur le bouton « Ajouter ». Ces méthodes mettent également à jour le cadre témoin.
Le cadre témoin est composé d'une plage affichant les opérations demandées et de 4 boutons action. « Aide » appelle l'aide en ligne, « Annuler » sort de la procédure de génération de données. Lorsque l'utilisateur sélectionne une opération puis clique sur « Supprimer », la méthode *removeAction ()* est appelée pour mettre à jour le cadre témoin et les deux Map décrivant les opérations à effectuer. « Ok » appelle la méthode *okAction ()* qui déclenche la réalisation des opérations demandées.

**Description de la Map *newLociAction*** : les clés sont les codes des espèces pour lesquelles des loci sont à ajouter. A chaque clé est associée une chaîne de caractères décrivant les loci à ajouter (lettre+nombre).

**Description de la Map *newEffectAction*** : les clés sont un code espèce + un tiret + un nom de paramètre (*Exemple : 0-phenology*). A chaque clé est associée une chaîne de caractères donnant la liste des loci influençant le paramètre, chaque locus étant séparé par un tiret (*Exemple : n1-n4-p2*).

| Nom de la méthode | Opération effectuée |
|---|---|
| *createUI ()* | Génère la fenêtre de dialogue et lit les données saisies |

| | |
|---|---|
| *searchSpecies ()* | Construit la liste des espèces gérées par le module à partir de la classe *ModuleSpecies* |
| *addLociAction ()* | Complète la liste des loci à générer |
| *addEffectAction ()* | Complète la liste des ParameterEffect à générer |
| *removeAction ()* | Supprime l'action sélectionnée dans la liste des opérations à sélectionner |
| *updateDisplay ()* | Met à jour la liste des opérations à effectuer |
| *okAction ()* | Exécute les différentes opérations en appelant les méthodes ci-dessous |
| *completeIndividualGenotype ()* | Complète les IndividualGenotype |
| *computeAlleleFrequencyComplement ()* | Complète les MultiGenotype |
| *completeGeneticMap ()* | Complète la GeneticMap |
| *completeAlleleDiversity ()* | Complete AlleleDiversity |
| *computeParameterEffect ()* | Calcule ParameterEffect |

## 9.2.2 Description des principales méthodes

### *9.2.2.1 searchSpecies (GTree t)*

Cette méthode calcule la liste des espèces gérées par le module et renvoie le nombre d'espèces de cette liste.
Si l'arbre t n'est pas une instance de *SpeciesDefined*, la méthode renvoie 0.
Si l'arbre est une instance de *SpeciesDefined*, la méthode passe en revue tous les accesseurs de t jusqu'à trouver celui qui renvoie une *QualitativeProperty* instance de *Species*. En utilisant la méthode *getValues ()* (méthode abstraite dans *QualitativeProperty*), la méthode *searchSpecies ()* récupère la Map des espèces (nom de l'espèce et son code).

Cette méthode est appelée par la méthode *createUI ()*.

### *9.2.2.2 okAction ()*

**Si la Map *newLociAction* n'est pas vide**
Pour chaque clé de la Map, la méthode construit :

- la Collection des arbres de l'espèce correspondante. Sur le premier arbre de l'espèce, elle lit la *GeneticMap* de l'espèce, ainsi que son *AlleleDiversity*.
- 3 vecteurs qui contiennent, pour chaque ADN, le nombre d'allèles de chaque locus à ajouter à l'ADN.

Elle appelle ensuite successivement les méthodes *computeAlleleFrequencyComplement (), completeGeneticMap ()* et *completeAlleleDiversity ()*.
Puis elle passe en revue chaque arbre de la collection. Si c'est un arbre multiple, elle complète son *MultiGenotype* en utilisant le résultat de la méthode *computeAlleleFrequencyComplement ()*. Si c'est un arbre individuel, elle appelle la méthode *completeIndividualGenotype ()*.

**Si la Map newEffectAction n'est pas vide**
Pour chaque clé de la Map, la méthode retrouve le code espèce et le nom du paramètre à partir du nom de la clé. Elle lit l'*AlleleDiversity* et l'*AlleleEffect* de l'espèce.

Si *AlleleDiversity* est null (l'espèce n'est pas génotypée), la méthode arrête la procédure et affiche un message d'erreur (les données demandées ne peuvent être générées).

Si l'espèce est génotypée et si *AlleleEffect* est null, la méthode construit une *AlleleEffect* pour l'espèce. Sinon, la méthode teste si *AlleleEffect* contient déjà le paramètre, si oui, elle le supprime. Puis, la méthode construit trois vecteurs contenant pour chaque ADN, le numéro des loci qui influencent le paramètre.
La méthode réalise ensuite plusieurs tests :

- si aucun numéro de loci donné par l'utilisateur n'est supérieur au nombre de loci existant
- si toutes les données ont été renseignées (héritabilité, variance environnementale totale et part de la variance environnementale inter Step
- si la part de la variance environnementale inter Step est différente de -1
- si héritabilité et variance environnementale totale ne sont pas toutes deux égales à -1

Si l'un de ces tests est positif, la méthode affiche un message d'erreur demandant de compléter la donnée manquante.
Si aucun test n'est positif, la méthode construit alors le *ParameterEffect* du paramètre :

- les valeurs des allèles sont générées par la méthode *computeParameterEffect ()*
- si la variance environnementale totale est égale à -1, la méthode appelle *getTotalEnvironmentalVariance ()* pour calculer une variance environnementale totale par défaut.

#### *9.2.2.3 completeMultiGenotype()*

Cette méthode complète le *MultiGenotype* d'un arbre donné en paramètre à partir de l'*AlleleDiversity* et des 3 vecteurs des loci à ajouter par brin d'ADN. Elle renvoie un *MultiGenotype*.

**Procédure de calcul du *MutiGenotype* :**
Chaque tableau du MultiGenotype est composé de :

- n1 lignes : n1 étant égal au nombre de lignes du tableau correspondant dans *AlleleDiversity*
- + n2 lignes : n2 étant le nombre de loci à générer sur l'ADN correspondant.

Les n1 premières lignes sont laissées vides. La méthode ne remplit que les n2 dernières lignes sous l'hypothèse d'équiprobabilité des allèles. Pour chaque nouveau locus i, les effectifs alléliques sont égaux à $E(N/nb_i)$ (nbi étant le nombre d'allèles possible sur le locus i et N l'effectif de l'arbre moyen) ou $E(2N/nb_i)$ pour l'ADN nucléaire.
Du fait des arrondis, la somme des effectifs alléliques par locus peut être inférieure à N ou 2N (pour l'ADN nucléaires). Les allèles manquants sont tirés aléatoirement. La méthode tire un entier, alea, compris entre 0 et $nb_i$-1 et ajoute 1 à l'effectif de l'allèle tiré (alea+1).
La méthode fusionne ensuite le *MultiGenotype* ainsi calculé avec le *MultiGenotype* de l'arbre moyen pour construire son nouveau *MultiGenotype*.

#### *9.2.2.4 completeGeneticMap ()*

Cette méthode complète une *GeneticMap*. Elle prend comme paramètres une *GeneticMap* (celle à compléter) et le vecteur des loci à ajouter sur l'ADN nucléaire. Elle renvoie une *GeneticMap*.

**Procédure de calcul :**
La méthode construit un nouveau tableau de probabilités de recombinaison de longueur n1+n2-test.

n1 : longueur du tableau des probabilités de recombinaison de la *GeneticMap* donnée en paramètre

n2 : longueur du vecteur donné en paramètre

test : test étant égal à 1 si l'espèce n'était pas génotypée ou à 0 sinon.

Elle remplit les n2-test dernières lignes en supposant que toutes les probabilités de recombinaison sont égales à 0.5. Elle fusionne ensuite le tableau des probabilités de recombinaison de la *GeneticMap* donnée en paramètre et le tableau calculé.
Elle construit ensuite une nouvelle *GeneticMap*, qui contient ce tableau, et la renvoie.

Cette méthode est appelée dans *okAction ()* qui affecte à un arbre de l'espèce (et donc l'espèce) la nouvelle *GeneticMap*.

#### *9.2.2.5 completeAlleleDiversity ()*

Cette méthode complète une *AlleleDiversity* qui lui est donnée en paramètre à partir de trois vecteurs contenant les loci à ajouter sur chaque ADN, qui lui sont donnés en paramètres. Elle renvoie une *AlleleDiversity*.

**Procédure de calcul :**
Chaque tableau (pour chaque ADN) de l'*AlleleDiversity* calculée est composé :

- n1 lignes : n1 étant égal au nombre de lignes du tableau correspondant dans *AlleleDiversity* donné en paramètre
- + n2 lignes : n2 étant le nombre de loci à générer sur l'ADN correspondant.

Les n1 premières lignes sont laissées vides. La méthode ne remplit que les n2 dernières lignes. Pour chaque nouveau locus i, les allèles sont codés de 1 à nbi (nbi étant le nombre total d'allèles sur le locus i).
La méthode fusionne ensuite chaque tableau avec le tableau correspondant dans l'AlleleDiversity donnée en paramètre pour construire une nouvelle *AlleleDiversity* qu'elle renvoie.

#### *9.2.2.6 completeIndividualGenotype ()*

Cette méthode complète l'*IndividualGenotype* d'un arbre donné en paramètre à partir des trois vecteurs contenant les loci à ajouter sur chaque ADN qui sont également donnés en paramètre. Elle renvoie un *IndividualGenotype*.

**Procédure de calcul :**
Chaque tableau de l'*IndividualGenotype* calculé est composé :

- n1 lignes : n1 étant égal au nombre de lignes du tableau correspondant dans l'*IndividualGenotype* donné en paramètre
- + n2 lignes : n2 étant le nombre de loci à générer sur l'ADN correspondant.

Les n1 premières lignes sont laissées vides. La méthode ne remplit que les n2 dernières lignes. Pour chaque nouveau locus i, l'allèle (ou les 2 allèles dans le cas de l'ADN nucléaire) est tiré aléatoirement. Pour tirer aléatoirement un allèle, la méthode tire aléatoirement un nombre *alea* entre 0 et 1 et calcule la partie entière q de alea*nbi (nbi étant égal au nombre d'allèles

possible sur le locus i). L'allèle est égal à la q ième valeur de la ligne du tableau d'*AlleleDiversity* correspondant au locus i soit q+1.
La méthode fusionne ensuite chaque tableau avec le tableau correspondant de l'*IndividualGenotype* donné en paramètre pour construire un nouvel *IndividualGenotype* qu'elle renvoie.

#### *9.2.2.7 computeParameterEffect ()*

Cette méthode calcule les valeurs des allèles influençant un paramètre et construit le *parameterEffect* du paramètre à partir de l'*AlleleDiversity* de l'espèce, de son *AlleleEffect*, du nom du paramètre, des trois vecteurs contenant les numéro des loci influençant le paramètre, de l'héritabilité, de la variance environnementale totale et de la part de la variance environnementale inter Step.

**Procédure de calcul des valeurs des allèles :**
Pour chaque locus i, la méthode récupère le nombre d'allèles possibles à partir de *AlleleDiversity*, noté n. Elle tire aléatoirement dans une loi normale de moyenne 0 et de variance 1, n-1 valeurs. Ces valeurs (multipliées par 10) sont les valeurs de n-1 premiers allèles. La valeur du dernier allèle est égale à -1 * (somme des valeurs des n-1 premiers allèles).
La méthode construit ensuite *parameterEffect* à partir des tableaux des valeurs des allèles qu'elle a calculés et des valeurs d'héritabilité, de variance environnementale totale et de part de variance environnementale inter Step données en paramètre. Le *parameterEffect* ainsi construit est ajouté à la Map *AlleleEffect*.

# 10 Spécifications requises pour le module client

Un certains nombres d'instructions doivent être obligatoirement insérées dans certaines classes du module client pour que le couplage entre la bibliothèque *Genetics* – et le module puisse être réalisé. L'objet de ce paragraphe est de préciser les lignes de codes à retranscrire.

On peut distinguer deux types d'instructions :

- des méthodes abstraites déclarées dans *GeneticTree* et qu'il faut définir dans *ModuleTree* (qui est la sous-classe de *GeneticTree*).
- un lien entre les arbres et leur espèce afin de récupérer les données génétiques peopre à l'espèce. Ce lien est établi différemment selon que le module considère une seule ou plusieurs espèces.

**Note** : *ModuleTree* est un nom générique. « module » est remplacé par un préfixe différent dans chaque module (ex : pour le module Ventoux : préfixe Vtx). Cette remarque s'applique aux autres noms de classe des modules Capsis.

## *10.1 Cas d'un module considérant plusieurs espèces*

Dans le cas d'un module considérant plusieurs espèces, il est courant de coder les espèces par une sous classe de *QualitativeProperty*, appelée *ModuleSpecies*. C'est dans cette classe que seront définis les variables génétiques spécifiques (*GeneticMap*, *AlleleDiversity* et *AlleleEffect*) et leurs accesseurs.
Par ailleurs, la classe *ModuleTree* doit implémenter l'interface *SpeciesDefined* (classe définie dans capsis/util). Cette classe constitue une interface entre chaque arbre et une *QualitativeProperty* et fournit l'accesseur à cet *QualtitativeProperty*. Par ce biais, les différents tests mis en place, dans la bibliothèque Capsis, sur les espèces pourront être réalisés (classe *Validate*).
Enfin, la classe *ModuleSpecies* doit implémenter l'interface *Species* (classe définie dans capsis/util). *Genetics* peut par ce biais reconnaître la classe héritant d'une QualitativeProperty qui définit les objets portés par l'espèce.

### 10.1.1 ModuleSpecies

Ce module doit implémenter l'interface *Species*.
En italique, les lignes de codes à retranscrire intégralement.

**Définition des variables d'instance de l'espèce**

*private GeneticMap geneticMap;*
*private AlleleDiversity alleleDiversity;*
*private Kinship kinship;*
*private AlleleEffect alleleEffect;*

**Accesseurs aux variables d'instance de l'espèce**

*public GeneticMap getGeneticMap () {return geneticMap;}*

*public void setGeneticMap (GeneticMap gene) {geneticMap=gene;}*
*public AlleleDiversity getAlleleDiversity () {return alleleDiversity;}*
*public void setAlleleDiversity (AlleleDiversity gene) {alleleDiversity=gene;}*
*public Kinship getKinship () {return kinship;}*
*public void setKinship (Kinship a) {kinship=a;}*
*public AlleleEffect getAlleleEffect () {return alleleEffect;}*
*public void setAlleleEffect (AlleleEffect effect) {alleleEffect=effect;}*

### 10.1.2ModuleTree

En italique, les lignes de code à retranscrire intégralement.

**Accesseurs à l'espèce**

```
public ModuleSpecies getModuleSpecies () {
        return ((Immutable) immutable).species;
}
public void setSpecies (ModuleSpecies species) {
        ((Immutable) immutable).species = species;
}
public QualitativeProperty getSpecies () {return ((Immutable) immutable).species;}
```

*getSpecies ()* étant une méthode abstraite déclarée dans *SpeciesDefined*, elle est définie ici dans *ModuleTree*

**Accesseur à l'effectif des arbres**

```
public int getNumber () {
        à définir par le modélisateur
}
```

**Accesseurs aux données génétiques spécifiques**

*getAlleleEffect (), getAlleleParameters ()* et *getAlleleDiversity ()* sont des méthodes abstraites de la classe *GeneticTree*. Elles doivent donc être définies dans *ModuleTree*, sous-classe de *GeneticTree*. Ces méthodes renvoient respectivement un *AlleleEffect*, un *AlleleParameters* et un *AlleleDiversity*.

Dans la classe *ModuleTree*, ces méthodes doivent être définies de la façon suivante :
- *getAlleleEffect ()* et *getAlleleDiversity ()* sont définies de façon similaire (voir exemple ci-dessous).

```
public Kinship getKinship () {
        return getModuleSpecies ().getKinship ();
}
public AlleleEffect getAlleleEffect () {
        return getModuleSpecies ().getAlleleEffect ();
}
public AlleleDiversity getAlleleDiversity () {
        return getModuleSpecies ().getAlleleDiversity ();
```

```
}
```

- *getAlleleParameters ()* est construite différemment des deux accesseurs précédents. *getAlleleParameters ()* doit tester le type du génotype de l'arbre sur lequel elle est appelée et selon le type du génotype, *IndividualGenotype* ou *MultiGenotype*, renvoyer respectivement un *AlleleParameters* de type *GeneticMap* ou de type *AlleleDiversity*. Voir exemple ci-dessous :

```
public AlleleParameters getAlleleParameters () {
        Genotype g = getGenotype ();
        if (g instanceof IndividualGenotype) {
                return getModuleSpecies().getGeneticMap();
        } else {
                return getModuleSpecies().getAlleleDiversity();
        }
}
```

## 10.2 Cas d'un module considérant une seule espèce

Dans le cas d'un module considérant une seule espèce, la classe *ModuleSpecies* n'existe pas. La définition des variables de l'espèce et des accesseurs à ces variables peut se faire dans différentes classes. Il est proposé de le faire dans la classe *ModuleModel*.

### 10.2.1 ModuleModel

Les spécifications à reprendre dans cette classe sont les mêmes que celles données au point 10.1.1.

### 10.2.2 ModuleTree

Dans *ModuleTree*, les spécifications requises sont différentes de celles données au 10.1.2 dans la mesure où les arbres n'ont pas d'espèce :

- *ModuleTree* ne doit pas implémenter l'interface *SpeciesDefined*
- *ModuleTree* ne définit pas d'accesseurs à l'espèce

**Accesseur au modèle**

Pour accéder facilement aux données génétiques spécifiques (définies dans *ModuleModel*), il est conseillé de définir un accesseur au *ModuleModel*.

```
public ModuleModel getModel () {
        return (ModuleModel) getStand ().getStep ().getScenario ().getModel () ;
}
```

**Accesseur à l'effectif des arbres**

```
public int getNumber () {
        à définir par le modélisateur
}
```

**Accesseurs aux données génétiques spécifiques**

*getKinship ()*, *getAlleleEffect ()*, *getAlleleParameters ()* et *getAlleleDiversity ()* doivent être définies dans *ModuleTree*, sous-classe de *GeneticTree*, de la façon suivante :

- *getAlleleEffect ()*, *getAlleleDiversity ()* et *getKinship ()* sont définies de façon similaire (voir ci-dessous).

```
public AlleleDiversity getAlleleDiversity () {
        return getModel ().getAlleleDiversity ();
}
public Kinship getKinship () {
        return getModel ().getKinship ();
}
public AlleleEffect getAlleleEffect () {
        return getModel ().getAlleleEffect ();
}
```

- *getAlleleParameters ()* est définie ainsi :

```
public AlleleParameters getAlleleParameters () {
        Genotype g = getGenotype ();
        if (g instanceof IndividualGenotype) {
                return getModel().getGeneticMap();
        } else {
                return getModel().getAlleleDiversity();
        }
}
```