**InitialisePopulation :**

n = nombre de solutions demandées par le client

m = nombre de case des solutions

plage = liste chaînée des valeurs possibles pour chaque case (tableau d’entier)

Pour n solutions faire

Créer un tableau de m entiers

Pour chaque case du tableau faire

TirageAleatoire de la plage donnée

Fin pour

Fin pour

**SelectionnerParents(nbParents)** :

Créer une liste listeParents

Pour nbParents

Tirer aléatoirement 2 parents parmi les archives

Si : parent1.fitness < parent2.fitness

listeParents.ajouter parent1

Sinon

listeParents.ajouter parent2

Fin si

Fin pour

**SélectionnerArchives(nbArchives)** :

Créer une liste d’individus

Appeler evaluerPopulation *\*L’appel à evaluerPopulation nous renvoie une liste d’individus triée en fonction du fitness\**

Enregistrer le résultat de l’appel dans la liste d’individus

Créer une instance d’archive

Si nbArchives > individus.size

Renvoyer une erreur, il ne peut y avoir un nombre d’archives plus grand que celui d’indivius

Fin si

Pour i de 0 à nbArchives

Ajouter dans l’archive l’individu i de la liste

Fin pour

Renvoyer l’archive

**caculFitness (ListInd)** :

Pour chaque indiv

fitness = rowFitness.Indiv(i) + densite.Indiv(i)

Fin pour

Return fitness

**calculRowFitness (ListIndiv)** :

Pour chaque indiv\_1

Int compteur = 0

Pour chaque indiv\_2

Si coût de l’indiv\_1 > coût indiv\_2 et durée indiv\_1>durée indiv\_2

compteur += force indiv\_2

Fin si

Fin pour

rowFitness indiv\_1 = compteur

Fin pour

Return ListIndiv