

0 représentant un espace vide.

Différentes fonctions sont présentes, elles sont ici résumées. Voici celles qui placent les bâtiments :

- PlacerBatiments : si des bâtiments sont à placer, alors pour tout bâtiment à placer on contrôle s'il peut être placé (fonction **PlacerBâtiment**). Si c'est le cas, on met à jour l'aire du terrain utilisée et on la retourne (=le résultat du placement, plus l'aire est grande, mieux c'est)
- **PlacerBâtiment** : on regarde pour chaque case du terrain x,y si elle est occupée. Si elle est vide alors on regarde si notre bâtiment à placer est en conflit avec d'autres bâtiments déjà placés. (Pour tout bâtiment déjà placé ->fonction **enConflitAvec**). S'il n'est pas en conflit, alors on indique au bâtiment sa position (exemple, b1 pos(3,4) de largeur 8 et hauteur 7, utile pour contrôler les conflits) puis on appelle **AjouterLeBâtiment** qui l'ajoute à la liste des bâtiments placés dans le terrain.
- **enConflitAvec** : Contrôle si un bâtiment n'est pas en conflit avec un autre sur le terrain. Cette fonction utilise également **incluDansLeTerrain** pour contrôler que le bâtiment ne dépasse pas du terrain.

Les autres fonctions sont : trier(type de tri), générer(nombre de générations aléatoires), afficher, initialisation etc.

FileHandler

Permet de créer les bâtiments et le terrain à partir d'un fichier.

Main

Contrôle s'il y a des arguments. Si non, il lance à partir du fichier. Si oui, il génère des bâtiments de dimensions aléatoires et compare les heuristique d'espace, d'encombrement et de générations aléatoires.

Interprétation des résultats

Les heuristique d'espace et d'encombrement donne dans l'ensemble des résultat similaires. Le résultat du nombre de générations aléatoire dépend énormément du nombre de générations faites. Le résultat est globalement en dessous des deux heuristiques précédentes. J'ai dû monter à 10 000 générations (50 bâtiments) pour obtenir des résultats meilleurs que l'espace et l'encombrement. Cependant cela prend beaucoup plus de temps et l'on pourrait se demander s'il ne vaudrait pas mieux calculer toutes les permutations.

3000 ordres aléatoire : aire 2393 > aléatoire 2376

```
Résultat tri aire: 2393
ordre des batiments triés 36 4 5 23 18 14 9 27 39 44 12 17 38 22 31 26 16 7 8 13 25 37
Résultat tri encombrement: 2393
ordre des batiments triés 36 4 5 23 18 14 9 27 39 44 12 17 38 22 31 26 16 7 8 13 37 25
Début de l'aléatoire:
Génération de 3000 ordres aléatoires
meilleur solution (numéro de batiment trié dans l'ordre) : 19 36 23 41 48 6 11 9 31 12 29 14 5 43 17 44 34 0 38 1 13 24 49 45 28 21 37 25 47
Résultat par génération aléatoire : 2376
Affichage:
```

10 000 ordres aléatoire : aire 2375 < aléatoire 2410

```
Résultat tri aire: 2375
ordre des batiments triés 18 27 31 29 21 48 5 38 26 41 9 39 33 4 19 22 46 28 42 17 32 13 3
Résultat tri encombrement: 2375
ordre des batiments triés 18 27 31 29 21 48 5 38 26 41 9 39 33 4 19 22 46 28 42 32 13 17 3
Début de l'aléatoire:
Génération de 10000 ordres aléatoires
meilleur solution (numéro de batiment trié dans l'ordre) : 27 5 23 16 19 1 28 15 31 37 38 29 46 8 42 4 41 17 33 43 38 24 7 49 32 2 13 47
Résultat par génération aléatoire : 2410
Affichage:
```

On pourrait également penser que le meilleur résultat aléatoire donne une liste triée proche de celle des tris d'espaces et d'encombrement. Mais cela ne semble pas être le cas :

Aire = 4 3 ; aléatoire = 1 4 0

Aire = 2 7 3 5 ; aléatoire = 4 0 1 3 5 6

<p>Résultat tri aire: 20 ordre des batiments triés 4 3</p> <p>Résultat tri encombrement: 20 ordre des batiments triés 4 3</p> <p>Début de l'aléatoire: Génération de 10000 ordres aléatoires meilleur solution (numéro de batiment trié dans l'ordre) : 1 4 0 Résultat par génération aléatoire : 22 Affichage: 1 1 4 4 4 1 1 4 4 4 0 0 4 4 4 0 0 4 4 4 0 0 0 0 0</p> <p>Program ended with exit code: 0</p>	<p>Résultat tri aire: 54 ordre des batiments triés 2 7 3 5</p> <p>Résultat tri encombrement: 54 ordre des batiments triés 2 7 3 5</p> <p>Début de l'aléatoire: Génération de 10000 ordres aléatoires meilleur solution (numéro de batiment trié dans l'ordre) : 4 0 1 3 5 6 Résultat par génération aléatoire : 58 Affichage: 4 4 4 4 0 0 0 0 4 4 4 4 0 0 0 0 1 1 5 5 0 0 0 0 1 1 5 5 0 0 0 0 1 1 3 3 3 3 0 0 1 1 3 3 3 3 0 0 1 1 6 6 0 0 0 0 0 0 6 6 0 0 0 0</p> <p>Program ended with exit code: 0</p>
--	--

Génération aléatoire vs toutes les permutations

Les générations aléatoires sont bien lorsque toutes les permutations seraient trop longues à calculer. On pourrait calculer une équation permettant de savoir, suivant le nombre de bâtiments, quand est-ce que toutes les permutations sont plus rentables à faire qu'un certain nombre d'aléatoire. Par exemple, pour 5 bâtiments, il est plus rentable de faire toutes les permutations que de calculer 10 000 fois des permutations aléatoire. On répèterait trop de fois la même permutation pour rien.

Donc si si toutes les permutations sont trop longues à calculer alors on en prend un certain nombre qui sont aléatoires et on retourne la meilleure.

Les heuristiques d'espace et d'encombrement restent les plus rentables en terme de cout/résultat.

Améliorations :

D'un point de vue technique, la mise en place de threads aurait été intéressante. En effet, les calculs sont lents lorsque nous voulons faire beaucoup de permutations ou de génération aléatoires.

Nous aurions pu tester d'autres heuristiques tel qu'alterner les gros bâtiments avec des plus fins dans l'ordre à placer.