Mal à la Tetris

1 le background

Tout le monde connaît le jeu Tetris où il s'agit d'empiler un maximum de formes avant d'être bloqué. Les humains sont plutôt inégaux face à ce genre de problème d'optimisation spatiale, surtout dans les versions 3D du jeu. Mais sans aller dans ces extrémités, on peut s'apercevoir que l'humain est à la fois plus mauvais et meilleur qu'une machine à ce jeu. Tout dépend de si on vise l'efficacité optimale ou bien proche de l'optimal. En effet, le cerveau humain est relativement mauvais pour trouver la meilleure solution et encore plus quand il s'agit de toutes les trouver. En revanche, il sait mettre en place des stratégies très élaborées (des heuristiques) qui lui permettent, avec de l'entraînement, d'approcher régulièrement l'optimum. Ce n'est pas pour rien que les machines ont mis très longtemps à battre les meilleurs joueurs d'échecs !

On constate cela avec un problème en apparence moins complexe : le remplissage d'un rectangle avec un jeu de formes prédéterminées. Le rectangle est quadrillé de façon régulière et chaque forme prend un certain nombre de cases. Pas besoin de prendre un rectangle très grand. Pas besoin non plus d'ajouter la possibilité de tourner les pièces. Si on donne un ensemble de pièces dont la surface réunie est plus grande que le rectangle, il y a forcément des pièces qui seront inutilisées. Si la surface est égale, il y aura peut être des pièces non utilisées. Mais dans les deux cas, l'humain peut assez rapidement trouver une solution quasi optimale, voire optimale si le rectangle est petit. En revanche, plus le rectangle est grand, moins il a de chance de trouver l'optimum et encore moins s'il doit déterminer combien de solutions de placement sont optimales. Une machine peut résoudre facilement ce problème. Mais elle montre aussi rapidement ses limites car le nombre de possibilités à explorer augmente de façon exponentielle.

2 l'énoncé

L'objectif est de réaliser une fonction qui vérifie **toutes les possibilités** de placement de X formes sur un quadrillage de taille limitée et qui indique la surface maximale remplie et le nombre de placements qui donnent cette surface. Pour cela, on considère les formes suivantes :

Les noms de ces formes correspond à la classe dans laquelle elles sont définies (cf. canevas de code). Chaque forme est associée à un numéro et va occuper un nombre de carreaux différent dans le rectangle :

— Rect : n°1, 2 carreaux,

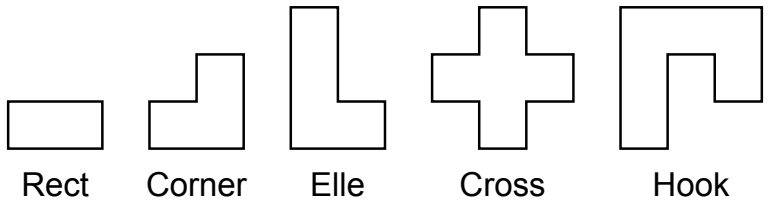

FIGURE 1 – Les 5 formes considérées

— Corner : n°2, 3 carreaux,
— Elle : n°3, 4 carreaux,
— Cross : n°4, 5 carreaux,
— Hook : n° 5, 6 carreaux.

ATTENTION : contrairement à Tetris, ces formes **ne peuvent pas** être tournées.

Votre programme doit lire sur l'entrée standard :

1. une ligne contenant un couple *largeur hauteur* correspondant aux dimensions du rectangle,
2. une ligne commençant par un entier N, suivi de N nombres allant de 1 à 5.

La deuxième ligne indique le nombre de formes à placer dans le rectangle, ainsi que leur numéro (pas forcément dans l'ordre). Par exemple : `4 1 5 2 1` indique qu'il faut essayer de placer 4 formes : 2 Rect (n° 1), 1 Corner (n°2), 1 Hook (n°5).

Votre programme doit simplement afficher sur la sortie standard le remplissage maximum trouvé ainsi que le nombre de placements qui aboutissent à ce maximum. La table 1 donne un exemple d'entrée et de résultat attendu.

entrée	sortie
`4 3` `10 1 2 3 4 5 5 4 3 2 1`	`11 24`

TABLE 1 – Exemple d'entrée avec 10 formes à placer dans un rectangle de 4×3, et la sortie attendue.

On remarque dans l'exemple 1 que le rectangle n'est pas totalement rempli (11 au lieu de 12) et qu'il y a un nombre limité de bonnes solutions (24). Un humain peut produire facilement ce résultat, même s'il y 770 placements corrects (i.e. sans chevauchements entre pièces, ni de débordements en dehors du tableau).

ATTENTION : même si plusieurs formes à placer sont du même type, elles sont considérées comme distinctes pour le calcul du nombre de placements optimaux. Par exemple, les deux cas de la figure 2 sont considérés comme deux solutions distinctes, malgré le fait que les rectangles soient au même endroit.

Pour illustrer les limites de l'humain, le tableau 2 donne un autre exemple avec un « petit » rectangle de 8×5 avec douze formes. Si on résout ce problème avec une solution

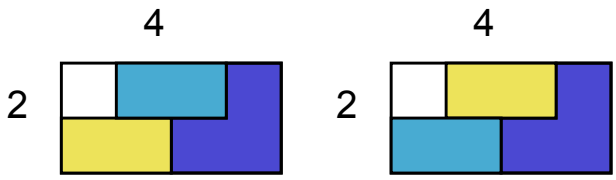

FIGURE 2 – Deux solutions optimales distinctes sur un rectangle 4×2

entrée	sortie
8 5 12 1 2 3 4 5 1 2 3 4 5 1 2	37 2016

TABLE 2 – Exemple d'entrée avec 12 formes à placer dans un rectangle de 8×5, et la sortie attendue.

Java très naïve (sans aucune optimisation), l'exécution dure environ 22 minutes, avec 10471978251 placements corrects évalués, et comme on le voit, 2016 qui sont optimaux, remplissant 37 des 40 cases.

La figure 3 donne un exemple de placement optimum sur le rectangle 8×5. Les numéros en italique correspondent aux ids des pièces. A droite sont données les pièces qui n'ont pas été retenues pour cette solution.

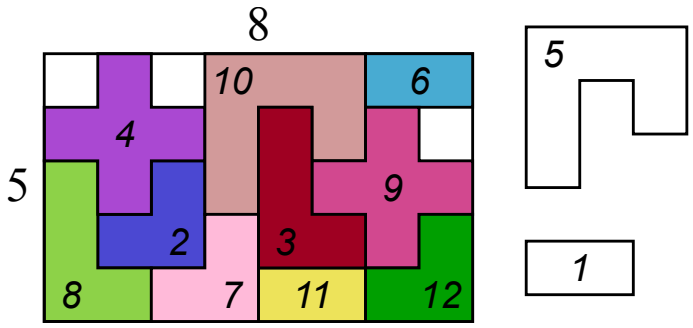

FIGURE 3 – Exemple de solution optimale avec 12 formes sur un rectangle 8×5

3 les ressources

Pour vous aider dans la réalisation du programme, vous trouverez sur
`http://cours-info.iut-bm.univ-fcomte.fr`
un article dans la section `hackaton` de l'année courante, portant le même titre que l'exercice. Il contient un lien permettant de télécharger un canevas de code, ainsi que le fichier d'entrée donné ci-dessus.

Bien entendu, vous êtes libres d'utiliser ou non ce canevas, mais c'est un gain de temps que de s'en servir comme base.