# Devoir 1 - Optimisation d'une galerie d'art

**Remise le** 18 **février 2024 (avant minuit) sur Moodle pour tous les groupes.**

## Consignes

- Le devoir doit être fait par groupe de 2 au maximum. Il est fortement recommandé d'être 2.
- Lors de votre soumission sur Moodle, donnez votre rapport en format **PDF** ainsi que vos fichiers de code à la racine d'un seul dossier compressé nommé (`matricule1_matricule2_Devoir1.zip`).
- Indiquez vos noms et matricules en commentaires au dessus des fichiers .py soumis.
- Toutes les consignes générales du cours (interdiction de plagiat, etc.) s'appliquent pour ce devoir.
- Il est permis (et encouragé) de discuter de vos pistes de solution avec les autres groupes. Par contre, il est formellement interdit de reprendre le code d'un autre groupe ou de copier un code déjà existant (StackOverflow ou autre). Tout cas de plagiat sera sanctionné de la note minimale pour le devoir.

## Enoncé du devoir

Votre très bon ami Célestin, magnat de l'immobilier et peintre à ses heures perdues, participe à sa première exposition d'oeuvres d'art.

Pour respecter une certaine harmonie, chaque pan de mur de la galerie où se déroulera l'exposition ne doit porter des tableaux que d'un seul artiste. L'organisateur, un philantrope méconnu, fait payer un dollar symbolique aux exposants pour chaque mur qu'ils décident de réserver.

Célestin, bien que riche comme Crésus, est radin comme un pou et ne veux pas dépenser un dollar de trop. Il fait donc appel à vous pour savoir, compte tenu des dimensions des toiles qu'il souhaite exposer, le nombre minimum de murs qu'il devra réserver.

Voyant les choses en grand, votre ami souhaite exposer dans toutes les sections de la galerie (20 sections exactement), chacune correspondant à un courant artistique différent (impressionnisme, vorticisme, lumisime, etc.) et possédant des murs aux dimensions différentes. Le nombre de murs dans chaque section n'est selon l'organisateur, pas un problème tant que l'on paie le dollar symbolique. Chaque section correspond à une instance du problème.

Votre ami vous remet donc une liste pour chaque thème avec les dimensions de ses oeuvres et vous demande pour chacune de fournir le nombre minimum de murs à réserver ainsi qu'un plan de ses toiles sur ces derniers. Il est bien sûr clair que vous ne pouvez en aucun cas découper les toiles ou placer une rotation de celles-ci.

Ces listes contiennent $k+2$ lignes où $k$ est le nombre de cadres qu'il souhaite placer dans cette section. La première ligne liste deux entiers : $W$ la largeur et $H$ la hauteur des murs de la section où il souhaite placer ses oeuvres. La seconde contient l'entier $k$, et les $k$ lignes suivantes contiennent trois entiers $i$, l'identifiant du cadre ainsi que $w$ et $h$, les dimensions du cadre.

À titre d'exemple, cette instance (`small/in_01.txt`) vous demande de placer 10 cadres sur un nombre minimal du murs de 10 mètres par 10. Une solution possible est visible sur la figure 1.

```
10 10
10
1  3  7
2  3  7
```

```
3   8   2
4   8   2
5   10  2
6   5   4
7   5   4
8   5   4
9   2   9
10  2   9
```

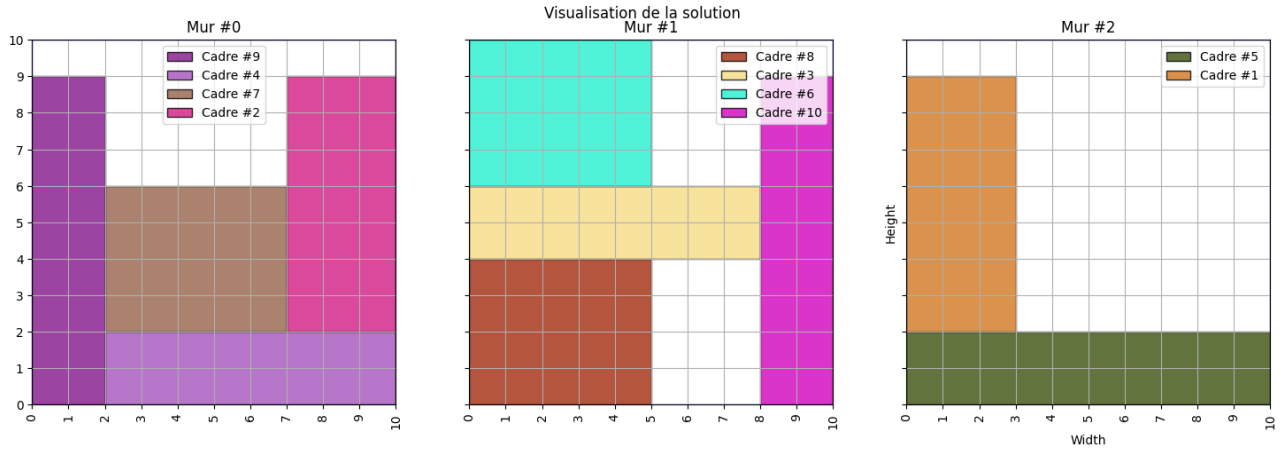


FIGURE 1 – Visualisation d'une solution possible pour l'instance *small/in_01.txt*

Le format attendu par Célestin est un fichier de $k$ lignes pour chaque section sous la forme suivante :

```
id_cadre_1 id_mur position_x position_y
id_cadre_2 id_mur position_x position_y
...
id_cadre_k id_mur position_x position_y
```

où le couple (`position_x`,`position_y`) correspond **au coin inférieur gauche** du tableau.

## Implémentation

Vous avez à votre disposition un projet python. Plusieurs fichiers vous sont fournis :

- `utils.py` qui implémente la classe `Instance` pour lire les instances, sauvegarder les solutions, les valider et les visualiser.
- `main.py` vous permettant d'exécuter votre code sur une instance donnée. Ce programme enregistre également votre meilleure solution dans un fichier au format texte et sous la forme d'une image.
- `solver_naive.py` : implémentation d'un solveur naïf qui réserve un mur par oeuvre d'art.
- `solver_advanced.py` : implémentation de votre méthode de résolution qui sera évaluée pour ce devoir.

Notez bien qu'un vérificateur de solutions est disponible. Vous êtes également libres de rajouter d'autres fichiers au besoin. De plus, 15 instances sont mises à votre disposition et votre merveilleux chargé de laboratoire vous a séparé les instances en deux groupes. L'un contient des instances arbitrairement plus faciles qui vous permettront de tester vos stratégies de remplissage et l'autre plus compliqué qui contient des ins-

tances pour lesquelles une recherche locale sera nécessaire.

Votre code sera également évalué sur **5 instances cachées qui comportent moins de 21 tableaux**. Pour vérifier que tout fonctionne bien, vous pouvez exécuter le solveur naïf comme suit.

```
python3 main.py --agent=naive --group=easy --number=1
```

Un fichier `solutions/<group>/in<number>.txt` et une image `visualization.png` seront générés.

## Production à réaliser

Vous devez compléter le fichier `solver_advanced.py` avec votre méthode de résolution. Au minimum, votre solveur doit contenir un algorithme de recherche locale. C'est à dire que votre algorithme va partir d'une solution complète et l'améliorer petit à petit via des mouvements locaux. Réfléchissez bien à la définition de votre espace de recherche, de votre voisinage, de la fonction de sélection et d'évaluation. Vous devez également intégrer un mécanisme de votre choix pour vous s'échapper des minima locaux. Vous êtes ensuite libres d'apporter n'importe quelle modification pour améliorer les performances de votre solveur. Une fois construit, votre solveur pourra ensuite être appelé comme suit.

```
python3 main.py --agent=advanced --group=easy --number=1
```

Un rapport **succinct** (2 pages de contenu, sans compter la page de garde, figures, et références) doit également être fourni. Dans ce dernier, vous devez présenter votre algorithme de résolution, vos choix de conceptions, et vos analyses de complexité. Reportez-y également les résultats obtenus pour les différentes instances.

On vous demande finalement de fournir un dossier `solutions` avec vos solutions au format attendu par Célestin (décrit plus haut) respectant la structure `solutions/<easy|hard>/in<number>.txt`.

## Critères d'évaluation

L'évaluation portera sur la qualité du rapport et du code fournis, ainsi que sur les performances de votre solveur sur les différentes instances. Concrètement, la répartition des points (sur 20) est la suivante :

— **10 points sur 20** sont attribués à l'évaluation des solutions fournies par votre solveur. Vous commencez avec 10 points. **Vous perdez 0.5 points** pour chaque instance où Célestin devra réserver plus de murs que nécessaire. La note du solveur est limitée vers le bas à zéro. Les tableaux ci-dessous listent les valeurs à atteindre pour obtenir la note maximale si vous ne faites pas perdre d'argent a Célestin sur les instances mystères. Votre algorithme doit fournir une solution sous la limite de temps visible dans ces mêmes tableaux (5 minutes seront accordées pour chaque instance cachée).

| **Instance (easy)** (max 1 min) | **in1** | **in2** | **in3** | **in4** | **in5** |
| --- | --- | --- | --- | --- | --- |
| Nombre de murs acceptable | 2 | 3 | 3 | 2 | 3 |

| **Instance (hard)** (max 5 min) | **in1** | **in2** | **in3** | **in4** | **in5** | **in6** | **in7** | **in8** | **in9** | **in10** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nombre de murs acceptable | 5 | 8 | 15 | 3 | 12 | 14 | 3 | 9 | 17 | 2 |

— **10 points sur 20** sont attribués à l'évaluation de votre rapport, les éléments indispensables pour obtenir une bonne note sont évidemment :
  — Une formalisation de **votre espace de recherche** (est-il connecté ?).
  — **Une quantification chiffrée** de l'impact des mécaniques implémentée : vous échappez-vous bien des extrema locaux? Si oui, prouvez-le, si non, montrez *formellement* pourquoi. Avez-vous une stratégie qui permet de réduire la complexité d'une routine ? Prouvez que ce que vous faites est utile avec des *mesures de temps*, de *mémoire* ou en *donnant une notation asymptotique*.
  — Privilégiez graphes, tableaux et formules pour transmettre l'information de manière concise.
  — Évitez à tout prix les explications de nature qualitatives, rapporter qu'un algorithme "va plus vite", est "plus performant" ou "donne de meilleures solutions" demande peu de travail, arriver à le prouver permet d'obtenir une bonne note.

⚠**Il est attendu que vos algorithmes retournent une solution et des valeurs correctes. Par exemple, il est interdit de modifier artificiellement les dimensions des murs, tableaux ou autre. Un algorithme retournant une solution non cohérente est susceptible de ne recevoir aucun point.**

## Conseils

Voici quelques conseils pour le mener le devoir à bien :

1. Tirez le meilleur parti des séances de laboratoire encadrées afin de demander des conseils.
2. Inspirez vous des techniques vues au cours, et ajoutez-y vos propres idées.
3. Tenez compte du fait que l'exécution et le paramétrage de vos algorithmes peut demander un temps considérable. Dès lors, ne vous prenez pas à la dernière minute pour réaliser vos expériences.

## Remise

Vous remettrez sur Moodle une archive zip nommée `matricule1_matricule2_Devoir1` contenant :
— Votre code commenté au complet.
— Un dossier `solutions` avec vos solutions au format attendu par Célestin (décrit plus haut) respectant la structure `solutions/<easy|hard>/in<number>.txt`.
— Votre rapport de présentation de votre méthode et de vos résultats, d'au maximum 2 pages.