TP3 - Eternity II

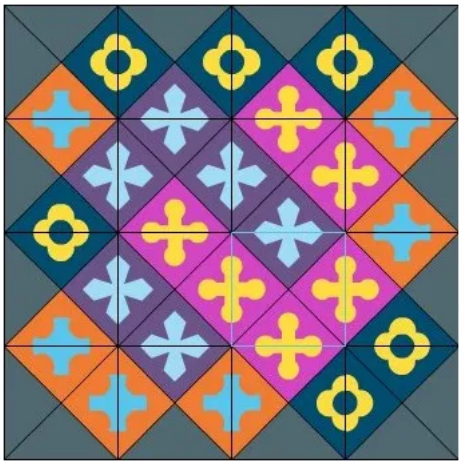
Remise le 28/03/21 sur Moodle.

Consignes

- Le TP peuvent être faits par groupe de 2 au maximum.
- Toutes les consignes générales du cours (interdiction de plagiat, etc.) s'appliquent pour ce TP.
- Votre rapport doit faire au maximum 2 pages (titre, page de garde, et figures non compris).

1 Énoncé du problème

Votre entreprise, `Mosaïque`, est experte dans la pose de carrelages, mais pas n'importe lesquels. Ces carrelages sont divisés en 4 zones contenant des motifs différents. Cependant, ces motifs sont incomplets et nécessite que le carrelage soit posé à coté d'un autre ayant le même motif pour être complété. Un exemple avec 16 carrelages vous est présenté ci-dessous.



Cependant, le stagiaire de l'équipe fabrication à perdu le plan de placement des carrelages chez un client. En temps que superviseur, c'est à vous de corriger son erreur et de reformer l'assemblage.

1.1 Eternity II

Cette mise en situation correspond à un célèbre puzzle nommé `Eternity II`[1], demandant de remplir un plateau de taille 16×16 avec 256 cases. La renommée de ce jeu vient du fait que son créateur, *Christopher Monckton*, confiant sur sa difficulté a promis une somme de 2 millions de dollar à la première personne trouvant une solution. Le défi a été lancé à la sortie du jeu (2007), et avait pour échéance la date du 31 décembre 2010. Personne n'a réussit ce défi. La meilleure solution trouvée avant l'échéance avait 467 connexions correctes, sur les 480. A l'heure actuelle, personne n'a encore réussi à résoudre ce puzzle.

Ce TP vous demandera de développer des algorithmes innovants basés sur la recherche locale (et méta-heuristiques) pour résoudre au mieux ce problème. Évidemment, résoudre le problème de base ne sera

1. `https://en.wikipedia.org/wiki/Eternity_II_puzzle`

pas requis pour obtenir la totalité des points (des instances plus simples seront proposées), mais vous permettra de pousser à l'extrême les capacités de vos algorithmes et à - *qui sait* - améliorer l'état de l'art pour ce puzzle. De plus, le meilleur résultat apparaîtra dans le *Hall of fame* du cours pour les prochaines années. :-)

1.2 Format

De manière plus complète, le problème est le suivant. Vous avez un certain nombre de tuiles à votre disposition. Celles-ci sont carrées. Elle sont également divisées en 4 triangles. Ces triangles sont colorés d'un certain motif. Le jeu consiste à placer toutes les tuiles sur le plateau de telle sorte que chaque pair de tuile voisine aie le même motif sur leur triangles adjacent. Une rotation peut également être effectuée sur chaque tuiles. Une couleur spéciale (gris dans notre exemple) correspond au bord du plateau.

Les fichiers contenant les instances du problème sont organisés en $n^2 + 1$ lignes. La première ligne contient un entier : la dimension n du plateau. Chacune des n^2 lignes suivante correspond à une tuile. La tuile est composée de 4 entiers. Chacun d'entre eu correspond à un identifiant de motif. Le premier motif est celui du nord, le second, celui du sud, puis celui de l'ouest et finalement celui de l'est.

```
n
c_1N c_1S C_1W C_1E
c_2N c_2S C_2W C_2E
...
c_nN c_nS C_nW C_nE
```

Le format d'une solution attendue comporte $n^2 + 2$ lignes. La première ligne contient un entier qui donne le nombre de conflits de la solution, la seconde donne la dimension n de l'instance, les autres n^2 lignes donnent les tuiles dans un ordre qui commence en bas à gauche, liste les tuiles de la première ligne, puis la deuxième (celle au dessus de la première), etc. jusqu'à la dernière ligne pour finir sur la dernière tuile en haut à droite. Le code de la tuile est dans le même ordre que pour le fichier d'entrée (nord, sud, ouest, est), tenant compte de la rotation que vous appliquer à la tuile.

```
totValue
n
C_1N C_1S C_1W C_1E
C_2N C_2S C_2W C_2E
...
C_nN C_nS C_nW C_nE
```

1.3 Exemple

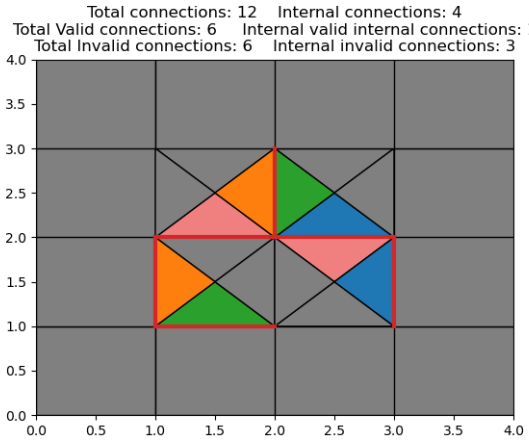
L'instance suivante a un plateau de 2 sur 2, avec donc 4 tuiles.

```
2
1 0 0 2
3 0 0 1
4 0 0 3
2 0 0 4
```

Une possible solution au problème est l'ensemble des tuiles suivantes qui possède 6 conflits.

```
6
2
0 4 3 0
1 0 0 2
0 1 0 3
0 2 4 0
```

Visuellement, cela donne ceci :



2 Implémentation

Vous avez a votre disposition un projet python ainsi qu'un environnement anaconda.

2.1 Instances de test

Vous avez à votre disposition 6 instances (dont 2 triviales, et une correspondant au puzzle complet) sur lesquelles vous pouvez vous entrainer.

2.2 Description du projet à votre disposition

Trois fichier python sont mis à votre disposition. Le fichier `eternity_puzzle.py` contient une classe modélisant une instance qui permet d'écrire la solution, de dessiner la solution, de calculer le coût, de vérifier une solution, etc. Le fichier `solver.py` contient une première version aléatoire de l'algorithme. Elle contient également la signature de la fonction de résolution plus avancée que vous devez implémenter. Vous pouvez utiliser la résolution naïve comme point de départ. Si vous avez besoin de fonction/classe annexe, veuillez les mettre dans le fichier `solver.py` ou modifier la classe dans `eternity_puzzle.py`. Le fichier `main.py` permet de lancer le programme.

3 Ce qui vous est demandé

Votre mission consiste à compléter la fonction `solve_advance(eternity_puzzle)` afin de résoudre une instance du jeu Eternity II en utilisant la recherche locale. Vous pouvez modifier le fichier `eternity_puzzle.py` si vous voulez pour y ajouter des fonctions d'aide. Le temps d'exécution est limité à 10 minutes. Inspirez vous des techniques déjà vue en cours. Notez qu'il existe une multitudes de façons pour résoudre ce problème.

Nous vous demandons d'utiliser au moins l'une des métaheuristiques suivantes : *tabu search*, algorithmes génétiques, *GRASP*, *iterated local search* ou *ant colony optimization*. Vous êtes libre de choisir celle de votre choix et d'y importer n'importe quelle amélioration (solution initiale heuristique, restarts, beamsearch, etc.).

4 Evaluation

Vous serez évalués en fonction de plusieurs critères : la clarté de votre implémentation (commentaire, etc.), la capacité de votre code à résoudre les instances données (plus vous êtes proche d'une solution faisable, plus vous aurez de points) et votre rapport (explication claire de votre algorithmes et de vos choix conceptuels). La répartition précise des points est la suivante :

- 6 points sur 10 seront attribués pour la qualité de votre code. Donne t-il une solution faisable meilleure qu'une solution naïve ? Donne t-il une solution d'une bonne qualité ? Chaque instance (sauf les deux triviales et le puzzle complet) rapporte 2 points. **Attention : un code qui ne compile pas ou qui retourne une solution incohérente sur une instance ne vous donnera aucun point pour l'instance en question.**
- 3 points sur 10 seront attribués pour la qualité de votre rapport. Est-ce que votre modèle et vos choix de conception sont clairement expliqués ?
- 1 point sur 10 sera attribué à la clarté de votre implémentation.
- 2 points bonus seront attribués au groupe ayant le meilleur résultat pour le puzzle initial d'Eternity II. En cas d'égalité pour une instance, les points seront répartis. De plus, le meilleur résultat apparaîtra dans le *Hall of fame* du cours pour les prochaines années. :-)

5 Remise

Vous devez remettre une archive zip contenant :

- vos fichiers `solver.py` et `eternity_puzzle.py`, contenant votre algorithme ainsi que vos fonctions et classes annexes
- les solutions obtenues sur chacune des instances à votre disposition (les fichiers solution doivent être nommés "solutionX" avec X la lettre correspondant à l'instance)
- un rapport de 2 pages (titre et images non compris) expliquant votre solution

Vous pouvez, pour vous motiver entre groupe, partager sur slack le nombre de conflits (pas les solutions exactes) de la meilleure solution que vous avez.