TP0 – IFT2015 – H18 – Major – Conway's Game of Life

Certains d'entre vous connaissent peut-être déjà le "jeu" mythique inventé par le mathématicien anglais John Horton Conway : « Game of Life ». C'est ce jeu qui a donné naissance à ce qu'on appelle maintenant les automates cellulaires. Afin de vous familiariser avec le langage de programmation Python, vous allez réaliser l'implémentation d'une variante du jeu pour ce travail.

Game of Life : l'original

« Game of Life » est en fait un jeu qui se joue à zéro joueur : en effet, l'état initial détermine entièrement les états subséquents et donc il s'agit davantage d'une simulation que d'un jeu. Cette simulation se déroule sur une grille à deux dimensions, composée de cellules carrées. Une cellule peut être soit vivante, soit morte (occupée ou déserte, alternativement). À chaque génération, des cellules peuvent naître, mourir ou demeurer en vie selon des règles simples qui dépendent du nombre de voisins vivants de chaque cellule. Les règles d'origine ont été choisies de manière à éviter une croissance explosive tout en permettant une complexité intéressante. Les voici :

1- Une cellule morte « naît » si elle comporte exactement 3 voisins vivants
2- Une cellule vivante « meurt » si elle comporte moins de 2 ou plus de 3 voisins vivants
3- Une cellule vivante le demeure si elle comporte 2 ou 3 voisins vivants

N.B. Une cellule qui n'est pas sur une frontière a au total 8 cellules voisines. Les diagonales comptent! Une cellule frontière ne communique pas avec les autres frontières opposées, par exemple les quatre coins ont chacun 3 voisins.

Ces règles simples donnent lieu à une foule de constructions aux propriétés intéressantes, par exemple des planeurs qui glissent sur le plan en diagonale et des fusils à planeurs qui génèrent des planeurs périodiquement. Pour des animations et plus de détails sur le jeu, visitez sa page Wikipédia : https://en.wikipedia.org/wiki/Conway's_Game_of_Life

Game of Life en couleur!

Pour ce TP, vous allez implémenter une version modifiée de Game of Life dans laquelle il existe quatre types d'organismes pouvant occuper les cellules : les rouges (R), les verts (G), les bleus (B) et les jaunes (Y). Des ensembles de règles différents s'appliqueront à chacun des types, mais pour chaque cellule **les voisins de seront comptés indépendamment de leur type**. Vous pouvez vous imaginer des bactéries de quatre espèces différentes qui survivent à différents niveaux de population, mais qui ne distinguent pas les espèces environnantes.
Un ensemble de règles est défini par trois nombres :

a : le nombre de voisins pour qu'un organisme « naisse »
b : le nombre minimal de voisins qu'un organisme doit avoir pour survivre
c : le nombre maximal de voisins qu'un organisme peut avoir avant de mourir

On définit l'inégalité suivante pour notre version du jeu, ce qui est logique si on pense à l'analogie biologique :

$$b \leq a \leq c$$

Un organisme d'une couleur donnée restera toujours de cette couleur dans le cas où il survit. Pour les naissances, on définit un ordre de priorité dans le cas où il y ait des conflits dans les règles données :

$$B \gg Y \gg R \gg G$$

C'est-à-dire que pour une cellule vide, on vérifie d'abord si elle a le nombre de voisins approprié pour qu'un « bleu » naisse. Sinon, on vérifie pour un « jaune », ensuite pour un « rouge » et enfin pour un « vert ». Un « vert » ne peut donc naître que si les conditions pour la naissance d'un « bleu » , d'un « jaune » et d'un « rouge » ne sont pas remplies.

Représentation
Votre programme devra prendre en entrée les ensembles de règles pour chacune des quatre couleurs, la taille de la grille, la configuration initiale et le nombre d'étapes à simuler. Le format d'entrée sera décrit plus loin. Vous devrez produire en sortie une représentation de l'état final. Cette représentation se fera en ASCII directement en sortie standard. Pour avoir une représentation plus ou moins carrée, chaque case sera constituée d'un caractère **suivi d'un espace**. Voici les caractères à utiliser :

```
Cellule vide :          .
Organisme rouge :       R
Organisme vert :        G
Organisme bleu :        B
Organisme jaune :       Y
```

Par exemple, une représentation d'un état possible sur une grille 10 x 10 est :

```
. . . . . . . . . .
. . . . . . . . . .
. . . G G . . . . .
. . G . . . . . . .
. . G . R G . . . .
. . . . G R B G . .
. . . . . B Y G . .
. . . . . G G . . .
. . . . . . . . . .
. . . . . . . . . .
```

Entrée
Votre programme doit comprendre un fichier `tp0_matricule1_matricule2.py` qui sera **exécuté dans un dossier** comprenant les fichiers **rules.txt** et **config.txt**. Si vous êtes seul vous n'incluez évidemment qu'un matricule. Votre programme prendra en **unique argument** le **nombre d'étapes** de simulation. Le fichier **rules.txt** comprendra quatre lignes, avec la lettre de la couleur, un séparateur « : » et les valeurs des nombres *a, b* et *c* définis précédemment, séparées par des virgules. Exemple de ligne :

```
R:3,2,3
```

Le fichier **config.txt** comprendra en première ligne la taille de la grille, avec la largeur en premier suivie de la hauteur. Ensuite, chaque ligne sera un triplet C,i,j où C est la couleur, i la rangée et j la colonne de chaque organisme présent dans l'état initial. Exemple sur quelques lignes :

```
10,10
R,1,1
R,1,2
B,2,1
G,9,5
Y,8,8
```

Ce fichier **config.txt** donne l'état initial suivant :

```
. . . . . .
. R R . . . .
. B . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . Y .
. . . . G . .
```

Sortie
Votre sortie doit être exactement telle que décrite, incluant les espaces séparant les caractères. Votre programme sera évalué sur différents ensembles de règles et différentes configurations initiales.

Tests unitaires
Une semaine avant la remise, un ensemble de tests unitaires et un script pour les effectuer vous seront remis. Vous êtes aussi encouragés à écrire vos propres tests unitaires afin de vous assurer du fonctionnement correct de votre programme.

Structure du code
Avertissement : vous devez développer VOTRE PROPRE CODE et tout plagiat détecté entraînera automatiquement un échec.

Votre code doit être clair et bien documenté. Vous êtes encouragés à consulter PEP 8 qui est un guide de style pour Python à la page suivante : https://www.python.org/dev/peps/pep-0008/

Il n'est pas exigé que vous utilisiez un type de programmation en particulier, mais nous vous recommandons fortement la programmation orientée objet pour ce TP. En plus de vous simplifier la tâche, votre code sera certainement plus lisible. Voici quelques idées de classes utiles :

- *grille* qui contient le jeu complet sous forme de tableau 2D de cellules
- *cellule* qui contient l'état d'une cellule (morte ou contenant un organisme)
- *organisme* qui contient les informations d'un organisme particulier
- *règles* qui définit les règles s'appliquant aux différents organismes

Pour les arguments en ligne de commande, ils se trouvent dans la liste `sys.argv` et comme il n'y a qu'un argument vous ne devriez pas avoir besoin de modules externes pour le traiter.

Version de Python
Il est fortement recommandé d'utiliser Python 3.6.4 ou toute autre version 3.6.*X*. N'utilisez pas Python 2 car des problèmes de compatibilité pourraient survenir.

Équipes
Vous pouvez faire votre travail seul ou en équipes de **deux** personnes **maximum**. Vous pouvez discuter avec les autres équipes évidemment, mais tout code dupliqué entre deux équipes sera considéré comme un plagiat des deux côtés.

Remise
Vous avez jusqu'au **4 février à 23h55** pour remettre votre travail sur studium. Remettez votre travail soit sous forme d'un seul fichier `tp0_matricule1_matricule2.py` ou sous la forme d'une archive `tp0_matricule1_matricule2.tar.gz` ou `tp1_matricule1_matricule2.zip` (**aucune** autre forme de compression ne sera tolérée) si vous avez plus d'un fichier. Dans ce cas, l'archive sera décompressée dans le dossier contenant les fichiers de spécifications et devra contenir entre autres le fichier `tp0_matricule1_matricule2.py` qui sera exécuté.

Correction négative

- 50% des points peuvent être perdus si votre code ne s'exécute pas ou ne produit pas de sortie

- Une pénalité de **20% par jour de retard** sera appliquée dès la première seconde de chaque période de 24h, en commençant à 23h56 le 4 février

Correction positive

- 30% : clarté du code
- 60% : exécution correcte sur un ensemble de tests
- 10% : sortie telle que spécifiée

Points boni

Bonus 1 : pour 5% de bonus, ajoutez un mode animation auquel on accède en passant l'argument `—animation` à votre programme au lieu du nombre d'étapes à simuler. Dans ce mode, chaque appui de la touche **<Enter>** simule une étape. Ajoutez des lignes blanches avant l'impression pour ne voir que l'état actuel sur la console.

Bonus 2 : pour un autre 5% de bonus, ajoutez un mode couleur qui est comme le mode animation mais avec le caractère # pour identifier les organismes vivants, d'une des quatre couleurs appropriées. On accède au mode couleur en passant `—couleur` en argument à votre programme.

Ajoutez au nom de votre programme les boni que vous avez ajouté ainsi :
`tp0_b1_b2_`*`matricule1_matricule2`*`.py`

Questions

Envoyez vos questions sur le travail à omailhot92@gmail.com. Les questions sur les points boni seront simplement ignorées (ce sont des points boni après tout).
Bon travail !!!