IN200: projet sur la simulation de l'écoulement d'un tas de sable

4 février 2022

Ce premier projet a pour objectif de vous apprendre à écrire un programme de manière autonome. Le sujet vous donne des indications sur la manière de construire votre programme et il doit être lu attentivement. Ce premier projet n'est pas noté, mais il vous prépare au deuxième projet du semestre qui sera évalué lors d'une soutenance. Ces deux projets se font par groupe de 3 ou 4. Les membres de votre groupe sont choisis par votre chargé de TD, et vous ne devez pas en changer durant le semestre.

Le projet doit être écrit en Python. Les compétences techniques attendues sont celles du cours IN100 du premier semestre. Quelques compléments seront apportés en cours sur des points précis de programmation en Python. Vous devez utiliser les outils de programmation étudiés en IN100, à savoir le gestionnaire de version `git` associé au dépôt `GitHub`, le linter `flake8` qui doit être installé avec votre environnement `VSCode`, et le debugger fourni avec `VSCode`.

1 Avant de commencer

Toutes les étapes de cette partie doivent être réalisées en TD avant de commencer la programmation en tant que telle.

1. Votre chargé de TD vous répartit en groupes de 3 ou 4. Une fois cela fait, regroupez vous pour les questions qui suivent.

2. Vérifier que votre environnement de programmation est correctement installé. Sinon cliquez sur le lien suivant pour retrouver les instructions d'installation. Pour savoir si votre environnement vous permet de faire ce qui vous sera demandé durant le semestre, vérifiez que, dans le répertoire `examples` du projet `l1-python`, les programmes des dossiers `hello` et `gui` s'exécutent sans erreur. Entraidez-vous pour tenter de résoudre les éventuels problèmes des membres de votre groupe.

3. Dès maintenant, choisissez 2 responsables pour les tâches suivantes :
   - responsabilité du dépôt `GitHub` : vous hébergez le projet sur votre compte et devez donc inviter les autres membres à participer au projet ; vous vous assurez que les commits qui sont "poussés" sur le dépôt sont bien fonctionnels (rappel : on ne doit pas faire de commit d'un programme qui n'a pas été scrupuleusement testé) ; de plus, le dépôt ne doit contenir que les sources du projet et non les éventuels fichiers générés par le projet (fichier de sauvegarde par exemple).
   - responsabilité de la mise en forme du code : vous vous assurez que le code satisfait bien les règles de style de la PEP8 (le linter `flake8` ne doit retourner aucun warning) et que les bonnes pratiques de programmation sont respectées (nommage cohérent des variables et des fonctions, documentation du code avec des docstrings pour les fonctions, etc.)

4. Le responsable du dépôt `GitHub` créée un nouveau projet intitulé `projet_tas_de_sable` sous `GitHub`, en se connectant à son compte, puis en choisissant `New repository` dans le menu accessible via le signe + en haut à droite de `GitHub` (attention à ne pas choisir `New project` dans ce menu). Le projet doit impérativement être **public**. En créant un nouveau projet, `GitHub` propose d'initialiser le dépot avec un `README`. Faites le et remplissez ce fichier avec une brève description du projet. Plus tard, vous ajouterez des précisions, notamment sur la manière d'utiliser le projet. N'ajoutez pas de `gitignore` pour le moment, ni de licence.

5. Toujours sous `GitHub`, le responsable invite les autres membres du groupe à participer au projet. Pour cela, sur la page du projet, aller dans le menu `Settings`, puis choisir `Collaborators`, puis le bouton vert

`add people`, et enfin, inscrire les noms d'utilisateur `GitHub` (devant être sous la forme uvsqXXX, où XXX est le numéro d'étudiant).

6. Les autres membres du groupe doivent accepter l'invitation depuis leur boîte mail ou bien les notifications sur leur compte `GitHub` (cloche en haut a gauche). À cette étape, tous le monde a les droits d'écriture sur le dépôt du projet.

7. Tous les membres réalisent le premier **clône** du projet.

8. Le responsable du dépôt ajoute alors le fichier `tas_de_sable.py` en y indiquant au début le groupe de TD, les noms des membres du projet et l'url du dépôt `GitHub` sous le format suivant :

```
########################################
# groupe MPCI 3
# Toto LEHERO
# Titi LEVAINQUEUR
# etc...
# https://github.com/uvsq-info/l1-python
########################################
```

Le fichier `README` doit aussi contenir ces informations.

9. Ensuite, le responsable ajoute ce fichier sur le dépôt avec la commande `sync` après avoir fait un `commit`.

10. Tous les autres font alors un `pull` pour récupérer la dernière version du dépôt.

11. Chaque membre du groupe à tour de rôle fait une modification mineure sur le programme, par exemple en ajoutant un espace après son nom (et en sauvegardant!), puis publie sa modification sur le dépôt `GitHub`. Puis, les autres membres récupèrent cette modification.

A partir de maintenant, vous pourrez travailler séparément sur le projet. A chaque fois que vous le faites, pensez à faire un `pull` du dépôt pour récupérer la dernière version du projet.

2 Description du projet

2.1 Contraintes de programmation

Votre programme doit satisfaire les contraintes suivantes :

- le programme doit être écrit dans un unique fichier qui s'appelle `tas_de_sable.py`;
- le programme ne doit pas définir de classes d'objets;
- vous devez utiliser la librairie graphique `tkinter`;
- quand ce n'est pas possible de faire autrement (notamment pour les widgets graphiques mais pas uniquement), vous pouvez utiliser des variables globales;
- le programme doit être découpé de la manière suivante (chaque partie devant apparaître bien distinctement en utilisant des blocs de commentaires) :
  - informations liées au groupe
  - import des librairies
  - définition des constantes (écrites en majuscule)
  - définition des variables globales
  - définition des fonctions (chaque fonction devra contenir une docstring)
  - programme principal contenant la définition des widgets et des événements qui leur sont liés et l'appel à la boucle de gestion des événements

2.2 Présentation du sujet

L'objectif de ce projet est de simuler l'écoulement d'un tas de sable et d'implémenter des opérations qui lui sont lié. La modélisation que l'on utilise repose sur un automate cellulaire appelé, en anglais, abelian sandpile model.

Nous définissons cet automate cellulaire sur une grille carrée. Chaque case de la grille peut contenir un nombre arbitraire de grains de sable. La donnée du nombre de grains de sable pour chaque case de la grille est appelée **configuration**. Voici, par exemple, une configuration sur une grille carrée de taille 3 :

```
  ♯ ♯ ♯
♯ 4 5 2 ♯
♯ 4 3 0 ♯
♯ 1 5 4 ♯
  ♯ ♯ ♯
```

Chaque case de la grille possède quatre **cases voisines** qui sont placées à gauche, à droite, au-dessus et en-dessous de celle-ci. En particulier, les cases qui sont au bord de la grille admettent pour voisines une ou deux (s'il s'agit d'une case dans le coin de la grille) cases représentées par le caractère dièse. Les cases dièses peuvent recevoir des grains de sable, mais leur quantité est ignorée : cette information ne fait donc pas partie de la configuration. Si une case possède au moins 4 grains de sable, alors elle est dite **instable**, autrement elle est dite **stable**. Dans notre exemple, 5 cases sont instables, et 4 sont stables. Faire une **avalanche** consiste, pour une case instable, à transmettre 1 grain de sable à chacune de ses voisines. La case perd donc 4 grains de sable, et toutes ses voisines en gagnent un. Il se peut qu'une avalanche rende les cases voisines instable et génère d'autres avalanches. Une **étape** de l'automate consiste à faire une avalanche sur chaque case instable du quadrillage en parallèle. Sur notre exemple précédent, une étape de l'automate donne la configuration suivante :

```
  ♯ ♯ ♯
♯ 2 2 3 ♯
♯ 1 6 1 ♯
♯ 3 2 1 ♯
  ♯ ♯ ♯
```

La configuration obtenue contient encore une case instable, et l'on peut **itérer** l'automate pour obtenir la configuration suivante :

```
  ♯ ♯ ♯
♯ 2 3 3 ♯
♯ 2 2 2 ♯
♯ 3 3 1 ♯
  ♯ ♯ ♯
```

On voit que cette configuration contient uniquement des cases stables. A ce moment l'automate s'arrête.

Un résultat mathématique affirme que, pour n'importe quelle configuration, le processus des avalanches s'arrête au bout d'un nombre fini d'étapes. De plus l'ordre dans lequel les avalanches sont faites ne modifie pas le nombre d'étapes et la configuration obtenue à la fin du processus. Cette configuration qui ne contient plus de cases instables est le résultat de l'opération de **stabilisation** de la configuration initiale.

2.3 Travail demandé

2.3.1 Pour démarrer

Dans un premier temps, vous écrirez un programme qui permet juste d'afficher une configuration aléatoire. Pour cela, vous pouvez procéder de la sorte :

- organiser votre programme en séparant les différentes parties (import des modules, variables globales, fonctions, etc.) par des blocs de commentaire ;
- créer une interface graphique qui permette d'afficher un canevas et un bouton qui créera la configuration aléatoire ;

- la configuration qui s'affiche dans le canevas est appelée la **configuration courante** et peut être stockée dans une liste à deux dimensions ayant le statut de variable globale ; de même pour les éléments graphiques du canevas qui sont associés à l'affichage de la grille ;
- programmer une fonction qui initialise la configuration courante et l'affichage de la grille avec une configuration vide de grains de sable ;
- programmer une fonction qui met à jour l'affichage de la grille à partir de la configuration courante ; en particulier il faut associer une couleur à chaque valeur d'une configuration (à choisir) ;
- programmer le calcul d'une configuration aléatoire (chaque case reçoit entre 0 et 3 grains de sable de manière équiprobable) : quand on clique sur le bouton, la configuration courante doit être modifiée ainsi que l'affichage de la grille ;

2.3.2 Suite du travail

Dans la suite du sujet, on n'impose pas le choix de l'interface. C'est à vous de décider du widget et/ou de la gestion d'événement clavier ou souris à utiliser pour réaliser ce qui est demandé. Vous devez préciser dans le fichier `README.md` associé au projet le mode d'emploi de votre programme.

Votre projet doit maintenant permettre de faire les opérations suivantes :

- créer une configuration qui est :
  - soit choisie dans une liste de configurations (voir le détail dans la liste après) ;
  - soit construite par l'utilisateur, par exemple en utilisant la souris ;
- programmer les opérations suivantes dont le résultat doit s'afficher dans le canevas :
  - **addition** de deux configurations : le résultat est la configuration qui est la somme case par case des deux configurations ; par exemple la somme de la configuration stabilisée précédente avec elle-même est la configuration :

    ♯ ♯ ♯
    ♯ 4 6 6 ♯
    ♯ 4 4 4 ♯
    ♯ 6 6 2 ♯
    ♯ ♯ ♯

    l'addition se fait entre la configuration courante (celle qui s'affiche) et une configuration choisie dans la liste des configurations ;
  - **soustraction** de deux configurations : pareil que pour l'addition, il s'agit d'une soustraction case par case ; le résultat doit cependant rester positif : on impose que si la différence de valeurs entre 2 cases est négative, alors le résultat est zéro ;
  - faire une étape de l'automate ;
  - calculer la stabilisation d'une configuration ;
- itérer les étapes de l'automate en les affichant jusqu'à ce que la configuration soit stabilisée ; avoir la possibilité d'interrompre l'animation ;
- sauvegarder une configuration dans un fichier ;

La liste des configurations proposées à l'utilisateur dans l'interface est la suivante :

- la configuration sauvegardée s'il y en a une ;
- la configuration `Random` qui choisit une valeur uniformément entre 0 et 3 pour chaque case de la grille ;
- la configuration `Pile centrée` qui attribue N grains de sables à la case du milieu et 0 aux autres ; la valeur N est choisie par l'utilisateur ;
- la configuration `Max Stable` qui attribue 3 grains de sable à chaque case ;
- la configuration `Identity` : on appelle *double max stable* la configuration obtenue en additionant `Max Stable` à elle-même ; alors `Identity` est obtenue en retranchant à double max stable la configuration stabilisée de double max stable, puis en stabilisant le résultat. Pour une grille de taille 100, la configuration `Identity` est représentée sur la figure plus bas ;

Attention à ce que la taille de la configuration sauvegardée soit la même que celle de la configuration courante si elle est utilisée pour l'addition ou la soustraction.

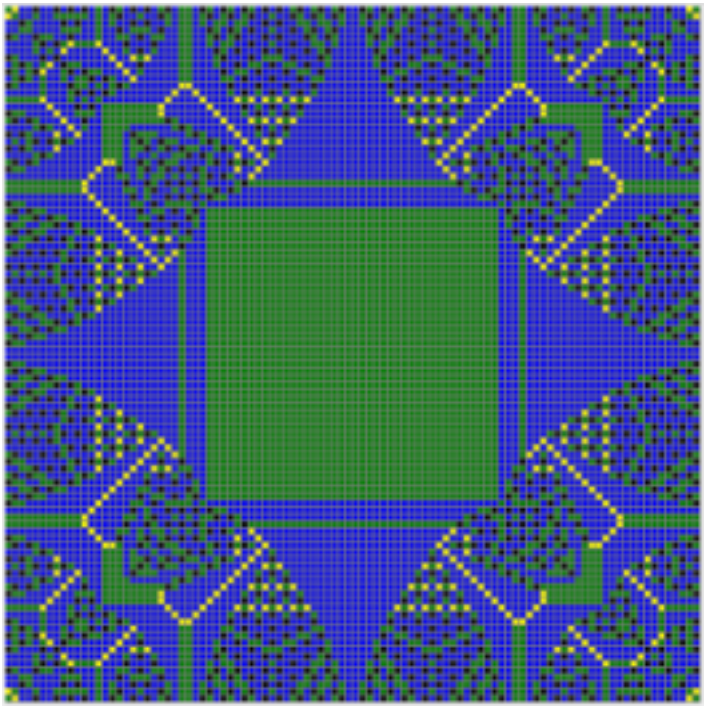

FIGURE 1 – Configuration **Identity** pour une grille de taille 100 ; noir, jaune, vert, bleu codent pour 0, 1, 2 et 3 grains de sable respectivement

Vous pouvez maintenant jouer avec votre programme : par exemple, calculer la somme de `Max Stable` avec une autre configuration puis stabiliser la configuration. Ensuite ajouter la configuration `Identity` puis stabiliser de nouveau. Qu'observez-vous ?

2.3.3 Pour continuer

Faites preuve d'initiative. Vous pouvez proposer d'autres fonctionnalités et améliorations à votre simulateur, par exemple :

- optimiser votre code pour que le calcul de la configuration `Identity` soit le plus rapide possible ;
- modifier le voisinage des cases (passer de 4 cases voisines à 8 par exemple) ;
- pouvoir sauvegarder plusieurs configurations dans des fichiers ;

2.3.4 Et maintenant ?

Vous pouvez (devez !) commencer votre projet dès maintenant, entamer avec les membres de votre groupe une réflexion sur la façon dont vous allez le mettre en œuvre, comment vous répartir les rôles…

N'hésitez surtout pas à interroger vos enseignants si vous avez des questions. Le forum mis en place dans l'espace Moodle du cours permet aussi de partager vos questions avec les autres étudiants.