Rapport

Projet « bases de la programmation impérative »

1 – Justesse du code

Le premier objectif, qui est de calculer une valeur approximative de pi à l'aide d'une simulation de Monte-Carlo, est rempli par la fonction simulate de simulator.py. En effet, cette fonction prend en paramètres le nombre de points de la simulation ainsi qu'une option permettant de définir le nombre de valeurs intermédiaires de pi à renvoyer (1 par défaut), et retourne un générateur où chaque valeur est un couple avec la valeur de pi et une liste des points générés à chaque stade de la simulation.

Le second objectif, qui est de générer une image animée représentant visuellement cette simulation est rempli par le programme approximate_py : pour chaque étape de la simulation, une image PPM est créée et chacune de ces images permet de créer une image animée GIF, soit grâce au programme convert pré-installé sur plusieurs distributions Unix, soit grâce au module convert.py pour les systèmes qui n'ont pas le programme requis (comme sur Windows par exemple).

2 – Qualité du code

Deux scripts Python, simulate.py et convert.py sont validés à 100 % par le fichier de configuration de pylint fourni pour ce projet. Pour ce qui est de approximate.py, une recommandation n'est pas respectée, nommée « too-many-branches ». En effet la fonction pour afficher chaque caractère de la valeur de pi représente chacun de ces caractères comme un afficheur 7-segments ; chacun de ces segments est dessiné ou non pour représenter le caractère souhaité. Pour cela, j'ai choisi de créer plusieurs blocs if les uns à la suite des autres pour chaque segment de l'afficheur, mais pylint ne peut pas savoir, sans connaître l'entrée de la fonction, combien de ces blocs s'exécuteront. J'aurais pu conditionner pour chaque caractère plutôt que pour chaque segment, pour qu'un seul de ces blocs ne soit exécuté et n'avoir aucun avertissement sur pylint, mais cela aurait engendré une redondance importante du code (un segment s'affiche pour plusieurs caractères) et aurait donc nuit à la maintenance future du programme.

3 – Performance du programme

La performance de la simulation et de la génération d'images dépends directement du nombre de points que l'on souhaite générer ; ni la taille de l'image ni la précision de la valeur de pi n'influent sur la vitesse d'exécution du programme. Ainsi, comme présenté dans le sujet, la génération d'une image de taille 800 avec une simulation d'un million de points a pris environ 9 secondes sur un ordinateur portable récent assez haut de gamme.

4 – Utilisation mémoire

D’après le module Python memory_profiler, 87 mégaoctets de mémoire vive sont utilisés au maximum par le programme pour générer une image animée de taille 800, avec un million de points pour la simulation et avec un précision de 5 chiffres après la virgule et en utilisant le module convert.py.

5 – Poids des images

Comme décrit sur le sujet du projet, une image de taille 800 x 800 générée pèse environ 1,9 mégaoctets, ce qui peut être difficilement plus léger étant donné que nous avons trois octets par pixel : 3 x 800 x 800 = 1 920 000 octets.

6 – Respect des consignes

Si l’utilisateur ne fournit pas le bon nombre d’arguments, une exception est levée avec un message de rappel de l’utilisation du programme. Pour chacun des trois arguments demandés, une exception TypeError est levée si l’entrée fournit par l’utilisateur n’est pas un nombre entier et une exception ValueError est levée si cette entrée est bien un nombre entier mais qu’il est inférieur ou égal à zéro.

Des f-strings sont utilisés pour composer le nom des images avec la valeur de l’approximation de pi, créer l’entête de ces images en fonction de la taille de l’image fournie par l’utilisateur, créer le nom de l’image animée finale, ou encore rappeler l’utilisation du programme à l’utilisateur.

Le module subprocess a bien été utilisé pour générer l’image animée à partir des images, et les potentielles erreurs qui pourraient survenir lors de son exécution sont capturées pour être affichées sur la sortie standard.

Conclusion

Pour conclure, les programmes joints à ce rapport remplissent les consignes du projet et semblent optimisés tant en complexité temporelle que spatiale. Le code est également volontairement construit de manière à être relu et maintenu, au détriment des recommandations de pylint.