Introduction aux arbres et à la programmation graphique

Équipe Algorithmique Avancée

Semaine 1

1 Statistiques en python

L'union européenne collecte des données statistiques sur les pays membres. Ces statistiques permettent de connaître la population, le niveau de vie, le taux de chômage, le nombre de médecins et d'autres informations pour toute l'Europe. Ces statistiques ont pour finalité d'aider les parlementaires, gouvernements, entrerpises et autres institutions à prendre des décisions utiles. [1] On peut ainsi obtenir des synthèses comme celle de la figure 1.

Selon l'utilisation qui en est faite, on a besoin de données concernant soit toute l'Europe, soit un pays, soit une partie d'un pays, soit une ville. Pour cela, les données sont recueillies commune par commune, puis on peut les regrouper pour des unités plus grandes, pour obtenir par exemple la population par département ou le nombre de médecins par habitant dans chaque région.

1.1 Représentation avec des classes, laborieuse

Le fichier `dep_reg_pays.py` contient les classes qui sont nécessaires pour représenter ces statistiques dans un programme en python. Il s'exécute avec la commande suivante : `python3 dep_reg_pays.py allemagne_france.json`.

Question 1

Compléter le tableau suivant à partir du code qui vous est donné.

Classe	Membres avec leur type
`Commune`	`population:int, nom:`
`Département`	`communes:list(Commune)`
	`départements: list(Département)`
`Pays`	`régions:`

1. Ces statistiques peuvent aussi permettre de *justifier* des politiques décidées pour d'autres motifs, d'intérêt général ou non.

FIGURE 1 – Origine des glaces consommées dans l'union européenne

Question 2

Que renvoie la méthode `mystere` de la classe `Departement` ? Et la méthode `boule_de_gomme` de la classe `Region` ? Donnez-leur des noms pertinents.

Question 3

Compléter la méthode `population` de la classe `Pays`.

Question 4

Compléter la méthode `medecins_par_10_000_hab` de la classe `Region`.

Question 5

Ajouter une méthode `liste_nom_departements(self)` à la classe `Region`, et une méthode `liste_nom_regions(self)` à `Pays`.

Question 6

Ajouter une méthode `liste_noms_communes` pour la classe `Departement`, puis pour les classes `Region` et `Pays`.

Question 7

Ajouter la méthode `liste_noms_departements` à la classe `Pays`.

Question 8

Ajouter une méthode `nombre_communes` aux classes `Region` et `Pays`.

1.2 Utilisation d'une classe arborescente

Les classes `Pays`, `Region` et `Departement` se ressemblent beaucoup, nous allons tenter un petit *refactoring*, en les regroupant en une seule classe `Groupement`. En effet, les `Pays` sont des groupements de `Regions`, les `Regions` sont des groupements de `Departements`, et les `Departements` sont des groupempents de `Communes`.

Question 9

Le fichier `groupement.py` vous donne une ébauche de la classe `Groupement`. Compléter les méthodes de cette classe en vous inspirant des méthodes déjà écrites et des méthodes des classes `Pays`, `Region` et `Departement`.

Question 10

Le fichier `irlande_luxembourg.json` contient des statistiques pour deux pays européens. Représenter graphiquement le découpage de ces deux pays. Est-il possible de le modéliser avec les classes `Pays`, `Region` et `Departement` en respectant le tableau de la question 1 ?

À retenir

Définition 1 (Classe arborescente)**.** *Une classe `C` est* arborescente *si elle a un ou plusieurs membres qui sont des structures de données susceptibles de contenir une instance de la classe `C`. Très souvent, ce membre sera une liste ou un tableau d'instances de `C` ou d'une interface implémentée par `C`.*

Exemple 1. *Dans l'exercice 1, la classe `Regroupement` contient une liste `divisions`. Les éléments de cette liste sont soit des communes, soit des* ***instances de `Regroupement`****. La classe `Regroupement` est donc arborescente.*

Une *instance* d'une classe arborescente représente une hiérarchie. Cette hiérarchie est représentée par un *arbre*.

2 Découverte de pygame

Au cours de ce module d'algorithmique, nous allons utiliser la bibliothèque `pygame` pour réaliser des interfaces graphiques. Dans ce TP, nous allons nous initier à son maniement.

La page web de `pygame` se trouve à `http://www.pygame.org`. `pygame` fournit un *canevas*, c'est à dire une fenêtre dans laquelle on peut dessiner des formes (rectangles, ellipses, droites, courbes), et un moyen de récupérer les clics et autres actions de l'utilisateur. Dans `pygame`, le contenu du canevas n'a pas de structure, contrairement à ce qui se passe en html, mais uniquement des pixels sur lesquels on peut opérer.

Créer une application `pygame` demande quelques étapes ; on vous fournit le fichier `squelette.py` qui contient une application minimale.

Question 11 *Échauffement*

Lancer l'application squelette avec la commande `python3 squelette.py`.

Question 12 *Les goûts et les couleurs*

Changer la couleur de la balle. Indication : ce magnifique cyan est la couleur rgb `123,234,222`.

Question 13 *Une question de taille*

Changer la taille de la balle.

Question 14

Que représentent les paramètres de la fonction `pygame.draw.circle` ? Vérifier dans votre réponse dans la documentation.

Question 15 *Revoyons l'action au ralenti*

On veut voir la balle se déplacer étape par étape. Pour cela, il faut afficher l'animation à une vitesse de 2 images par secondes (2 FPS) au lieu de 30. Effectuer la modification nécessaire.

Quelle est l'instruction qui provoque l'attente entre deux images ?

Passer l'animation en 60 FPS pour plus un confort de visionnage incomparable.

Question 16 *Un petit rafraîchissement*

À quoi sert la fonction `refresh` ? Que se passe-t-il si on ne l'appelle pas ?

3 Un micro-jeu

3.1 La base

Puisque la bibliothèque s'appelle py**game**, nous allons créer un micro-jeu à partir de squelette.py.

Commençons par entraîner le joueur à cliquer sur la balle. Pour cela, il faut d'abord savoir où se trouve la balle, donc avoir un objet qui la représente.

Question 17 *Une question de classe*

Dans un nouveau fichier `balle.py`, créer une classe `Balle`, avec des attributs `position_x`, `position_y`, `vitesse_x`, `vitesse_y`, `couleur` et `taille`.

Quels sont les types de ces attributs ? Pour les attributs numériques, en quelles *unités* sont-ils exprimés ?

Question 18 *On avance*

Ajouter à la classe `Balle` une méthode `avance(t)` qui modifie la position de la balle pour refléter le fait que "t" millisecondes se sont écoulées.

Question 19 *Méthode de dessin*

Ajouter à la classe `Balle` une méthode `dessine(s)` qui dessine la balle sur le canevas pygame `s` passé en argument.

Question 20 *La question à deux balles*

Utiliser la classe `Balle` depuis squelette.py pour créer une application avec deux balles qui se déplacent (avec des vitesses différentes).

Question 21 *Dans le mille*

Ajouter à la classe `Balle` une méthode `contient(position)` qui indique si la position passée en argument est située à l'intérieur de la balle.

Question 22 *La récompense*

Le score du joueur est le nombre de fois où il ou elle a cliqué sur l'une des balles. Ajouter le score dans l'application et l'afficher.

3.2 Les suppléments

Question 23 *Niveau 2*

Ajouter une nouvelle balle (avec une vitesse et une couleur aléatoires) à chaque fois que le joueur clique sur une balle existante.

Question 24 *Niveau hardcore*

Ajouter une balle (et augmenter le score) uniquement si le joueur clique sur chaque balle sans cliquer deux fois sur la même. Quand le joueur re-clique sur une balle qui a déjà été cliquée, on oublie quelles balles ont été cliquées précédemment.

Question 25 *Moteur physique dernière génération*

Que peut-on modifier pour que les balles rebondissent sur les bords de l'écran ? *Indication :* on peut commencer par se contenter de les faire rebondir quand leur *centre* atteint le bord de l'écran.