

# Introduction aux arbres et à la programmation graphique

Équipe Algorithmique Avancée

Semaine 1

## 1 Un questionnaire en java

Dans cet exercice, nous allons créer une petite application de questionnaires. Un questionnaire se compose d'une série de questions qui s'enchaînent. Ces questions sont soit des questions avec réponse libre, telles que «Quel est votre nom?», «Quelle est votre quête», soit des questions à choix multiples, comme «Quelle est votre couleur favorite : rouge, ou bleu?». Le fichier `App.java` contient le code dont vous partirez pour cet exercice ; il utilise le fichier `json-simple-1.1.1.jar` que vous trouverez sur Célène.

### 1.1 Questionnaires linéaires

L'interface `interface Question` correspond à une question, elle est implémentée par les classes `class QuestionOuverte` et `class QuestionChoixMultiples`. La classe `class Questionnaire` correspond à une série de questions. La méthode `pose` de chacune de ces classes permet d'afficher les questions ou le questionnaire à l'écran et de récupérer les réponses. Le fichier `App.java` définit un exemple de questionnaire, puis pose les questions de ce questionnaire et permet d'y répondre. Le questionnaire est chargé à partir du fichier `exemple_lineaire.json`, avec les commandes suivantes :

```
javac -cp json-simple-1.1.1.jar App.java
java -cp json-simple-1.1.1.jar:. App exemple_lineaire.json
```

### Question 1

Ajouter une méthode `pose_et_sauve(Console console, Writer out)` à l'interface `Question`. Cette méthode affiche la question sur la console, attend la saisie d'une réponse, puis écrit la question et la réponse dans le fichier `fichier`. Implémenter cette nouvelle méthode dans les classes `QuestionOuverte` et `QuestionChoixMultiples`



FIGURE 1 – Un classique du film de questionnaire : *Usual Suspects*

## Question 2

Ajouter une méthode `pose_et_sauve(Console console, Writer out)` à la classe `class Questionnaire`

### 1.2 Questionnaires à plusieurs niveaux

La classe `class QuestionCMApprofondie` permet de définir des questions à choix multiples à *tiroirs*. Ces questions contiennent des sous-questions qui seront posées en fonction de la réponse à la question principale. On peut ainsi obtenir les interactions suivantes :

Q : Quelle est votre couleur préférée : (1) rouge ou (2) bleu ?  
 R : Bleu  
 Q : Merci vous pouvez passer  
 R : À bientôt.

Q : Quelle est votre couleur préférée : (1) rouge ou (2) bleu ?  
 R : Rouge  
 Q : Êtes vous sûr·e : (1) oui ou (2) non ?  
 R : Non

La classe `class QuestionCMApprofondie`, qui dérive de `class QuestionChoixMultiples` contient un membre supplémentaire, `List<Question> sous_questions`. Le sens de ce membre est que si l'on choisit la réponse `reponses.get(i)`, on doit ensuite poser la question `sous_questions.get(i)`.

## Question 3

Implémenter la méthode `pose` de la classe `class QuestionCMApprofondie`, et vérifier le résultat à l'aide du questionnaire contenu dans le fichier `exemple_approfondi.json`.

#### Question 4

Implémenter la méthode `pose_et_sauve` de la classe `class QuestionCMAprofondie`, et vérifier le résultat à l'aide du questionnaire `exemple_approfondi.json`.

#### Question 5

Écrire un fichier `mon_questionnaire.json` contenant des questions à *trois* niveaux. Vérifier si votre code fonctionne aussi sur ce questionnaire.

#### Question 6

Donner une représentation graphique du questionnaire de la question précédente.

## 2 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 médecin et d'autres informations pour toute l'Europe. Ces statistiques ont pour finalité d'aider les parlementaires, gouvernements, entreprises et autres institutions à prendre des décisions utiles.<sup>1</sup> On peut ainsi obtenir des synthèses comme celle de la figure 2.

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



FIGURE 2 – Origine des glaces consommées dans l'union européenne

## 2.1 Représentation avec des classes, laborieuse

Le fichier `com_dep_reg_pays.py` contient les classes qui sont nécessaires pour représenter ces statistiques dans un programme en python.

### Question 7

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

Classe	Membres avec leur type
Commune	<code>population: int</code> , <code>nom:</code>
Département	<code>communes: list(Commune)</code> <code>départements: list(Département)</code>
Pays	<code>régions:</code>

### Question 8

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

### Question 9

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

### Question 10

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

### Question 11

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

### Question 12

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

### Question 13

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

### Question 14

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

## 2.2 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 `Groupelement`. En effet, les `Pays` sont des groupements de `Regions`, les `Regions` sont des groupements de `Departements`, et les `Departements` sont des groupements de `Communes`.

### Question 15

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

### Question 16

Le fichier `statistiques_espagne_irlande_luxembourg.json` contient des statistiques pour trois pays européens. Représenter graphiquement le découpage de ces trois pays. Est-il possible de le modéliser avec les classes `Pays`, `Region` et `Departement` en respectant le tableau de la question 7 ?

### 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 2, 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.

**Exemple 2.** Dans l'exercice 1, la classe *class QuestionCMapprofondie* contient une liste *List<Question> sous\_questions* d'instances de *interface Question*. Or *class QuestionCMapprofondie* elle-même implémente *interface Question*, donc *sous\_questions* peut contenir des instances de *class QuestionCMapprofondie* (comme à la question 5). Cette classe est donc arborescente.

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

### 3 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 17 Échauffement

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

#### Question 18 Les goûts et les couleurs

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

#### Question 19 Une question de taille

Changer la taille de la balle.

#### Question 20

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

### Question 21 *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 22 *Un petit rafraîchissement*

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

## 4 Un micro-jeu

### 4.1 La base

Puisque la bibliothèque s'appelle **pygame**, 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 23 *Une question de classe*

Dans un nouveau fichier `balle.py`, créer une classe **Balle**, avec des attributs `position`, `vitesse`, `couleur` et `taille`.

Quels sont les types de ces attributs ?

### Question 24 *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 25 *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 26 *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 27** *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 28** *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.

## **4.2 Les suppléments**

### **Question 29** *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 30** *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 31** *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.