Programmation orientée objet

Version 1.0

enseignants dpt info iuto

févr. 02, 2017

Les sujets de Cours

Contents :

CHAPITRE 1

Les documents de cours.

1.1 Cours 1

Ce n'est pas encore en ligne.

1.2 Cours 2

Ce n'est pas encore en ligne.

1.3 Cours 3

Ce n'est pas encore en ligne.

CHAPITRE 2

Semaine 1

2.1 TD 1

2.1.1 Comment transformer du code python en code java ?

Observez les traductions suivantes. Que signifie l'instruction **new ArrayList<String>() ;** en java ?

Python	Java

`x = 3`
`if x > 3 :`
`    y = 7`
`else :`
`    y = 8`
 	
`print(y)`
	`int x = 3;`
	`int y ;`
	`if( x > 3)`
	`   y = 7;`
	`else`
	`   y = 8;`
	`System.out.println(y);`

Python :

```
liste = ["bonjour", "salut", "encore", "bien", "sur", "zorro", "oui", "non"]
motPlusGrand = None
for mot in liste :
        if motPlusGrand == None or mot > motPlusGrand :
                motPlusGrand = mot
print("Le mot le plus grand dans l'ordre alphabétique est ", motPlusGrand)
```

Java :

```
import java.util.ArrayList;
class Executable {
  public static void main(String [] args) {
     ArrayList<String> liste = new ArrayList<String>() ;
     liste.add("bonjour");
     liste.add("salut");
```

```
        liste.add("encore");
        liste.add("bien");
        liste.add("sur");
        liste.add("zorro");
        liste.add("oui");
        liste.add("non");
        String motPlusGrand = null;
        for(String mot : liste)
            if(motPlusGrand == null || mot.compareTo(motPlusGrand) > 0)
                motPlusGrand = mot;
        System.out.println("Le mot le plus grand dans l'ordre alphabétiques est " +␣
↪motPlusGrand );
    }
}
```

2.1.2 Traduire les codes python en codes java

Pour chacun des codes python suivant :

— Trouvez ce qu'il affiche,

— traduisez le en java.

1.
```
print("Les nombres de 0 à 9")
for nombre in range(10):
        print(nombre)
```

2.
```
print("Somme des nombres de 0 à 15")
somme=0
for nombre in range(16):
    somme=somme+nombre
print(somme)
```

3.
```
for ligne in range(5) :
    for etoile in range(ligne):
        print('*', end = '')
    print()
```

4.
```
liste = ["zorro", "salut", "bonjour", "au revoir", "lundi", "jeudi", "merci"]
lettre = "l"
res = ["","","","","","","",""]
indiceDebut = 0
indiceFin = len(liste) - 1
for mot in liste :
   if mot >= lettre:
        res[indiceFin] = mot
        indiceFin = indiceFin - 1
  else :
        res[indiceDebut] = mot
        indiceDebut = indiceDebut + 1

print("res = ", res)
```

5.
```
liste = ["salut", "encore", "bonjour", "lundi", "samedi", "oui", "non"]
motCherche = "bonjour"
```

```
trouv = False
continuer = True
i = 0;
while continuer :
   if i >= len(liste)-1:
       continuer = False
   if liste[i] == motCherche :
       trouv = True
       continuer = False
   i = i + 1

print("trouvé ", tr)
```

6.
```
liste = ["oui", "non", "de", "alors", "bien", "sur", "or"]
for i in range(len(liste)):
   for j in range(len(liste)-1, i)
       if liste[j] < liste[j - 1] :
           aux = liste[j]
           liste[j] = liste[j - 1]
           liste[j - 1] = aux
print(liste)
```

7.
```
mot = "eluparcettecrapule"
palindrome = True
continuer = True
i = 0;
while continuer :
   if i >= len(mot)-1:
       continuer = False
   if mot[i] != mot[-i-1]:
       palindrome = False
       continuer = False
   i = i + 1
if palindrome:
    print("Cette phrase est un palindrome")
else:
    print("Cette phrase n'est pas un palindrome")
```

Recherchez dans la documentation fournie comment s'utilise l'égalité de chaînes en java.

2.1.3 Documentation :

Méthode de String	Description
char **charAt(int index)**	Returns the char value at the specified index.
int **compareTo(String anotherString)**	Compares two strings lexicographically.
boolean **equals(Object anObject)**	Compares this string to the specified object.
int **indexOf(int ch)**	Returns the index within this string of the first occurrence of the specified character.
int **length()**	Returns the length of this string.

2.2 TP 1

2.2.1 Correction de codes java et messages d'erreur

Dans les classes suivantes, des erreurs se sont glissées (*de compilation ou d'exécution*).

Notez les messages d'erreur obtenus (*lors de la compilation et/ou exécution*), expliquez les et proposez une correction.

```
class CoupleEntiers {
       int x = 5, y = -8;
       int aux;
       aux = x;
       x = y;
       y = aux;
}
```

— Nombre de chiffres

```
class Entier {
         int valeur;
         int nbChiffres() {
              int n = valeur;
              int nb = 0;
              while(n > 0)
                     n = n/10;
                     ++nb;
              return nb;
         }
 }
```

— Valeur absolue

```
class Entier {
         int valeur;
         int absolue() {
             if( valeur > 0)
                 return valeur
             else
                 return -valeur;
         }
}
```

— Fraction

```
class CoupleEntiers {
         int x,y;
         CoupleEntiers() {
             x = 0;
             y = 0;
         }
         int fraction() {
             return x/y;
         }
}
```

— Classes Essai et Executable

```
class Essai {
         int x;
         double y;
         void modifx(int a) {x += a;}
         void modify(int b) {y -= b;}
         int genere() {
             if(x > y) return x;
             else return y;
         }
}

class Executable {
         public static void main(String [] args) {
              Essai e = new Essai();
              e.x = 5;
              e.y = -12.3;
              e.modifx(e.y);
              e.modify(e.x);
              System.out.println(e.genere());
         }
}
```

— Tableaux

```
import java.util.ArrayList;
public class Executable {
   public static void main(String [] args) {
       ArrayList<String> tab = new ArrayList<String>();
       tab.add("lundi");
       tab.add("oui");
       tab.add("bonjour");
       tab.add("anticonstitutionnellement");
       tab.add("mot");
       tab.add("longueur");
       for(int i = 0; i <= 6; ++i)
           tab.set(i, tab.get(i) + "!");
       System.out.println(tab);
       String res = "";
       int taille = 0;
       for(int i = 0; i < tab.size(); ++i)
           if (tab.get(i).length() > = taille) {
               taille = tab.get(i).length();
               res = tab.get(i);
           }
       System.out.println(res);
   }
}
```

— Allocation et tableaux

```
class Tableau {
   ArrayList<String> tab = new ArrayList<String>();
   void ajout(String s) {
       if(s.length() >= 2)
           tab.add(s);
   }
   void affiche(int nb) {
       for(int i = 0; i < nb; ++i)
```

```
            System.out.print(tab.get(i)+" ");
        System.out.println();
    }
    int fonc(int nb) {
        int indiceMaximum = -1;
        int i = 0;
        while (i < nb) {
            if (indiceMaximum == -1 || (tab.get(i) > tab.get(indiceMaximum)))
                indiceMaximum = i;
            ++i;
        }
        return indiceMaximum;
    }
}
```

Après correction de ce code, expliquez et notez ce qu'il fait.

2.2.2 Classe Triplet

On veut définir une classe **Triplet** permettant de représenter un triplet de chaînes de caratères, dont on veut pouvoir trouver la plus grande chaine (au sens du nombre de caractères). Cette classe doit fonctionner avec l'exécutable suivant :

```
class ExecutableTriplet {
   public static void main(String [] args) {
          Triplet triplet = new Triplet();
          triplet.init("bleu", "rouge", "vert");
          triplet.affiche();
          triplet.modif("orange", 3);
          triplet.affiche();
   }
}
```

1. Écrivez l'interface de la classe **Triplet**, sans donner le code des méthodes.
2. Écrivez la méthode *init* permettant l'initialisation des trois chaînes du vecteur.
3. Écrivez la méthode *affiche* donnant l'affichage suivant :

```
Triplet : <bleu rouge vert >
Plus longue chaine : rouge
```

1. Écrivez la méthode *maximum* renvoyant la chaine la plus longue.
2. Écrivez la méthode *modif* permettant de modifier la ième composante du triplet (i variant de 1 à 3).

2.2.3 Classe Triplet et tableau

Définissez une nouvelle classe **TripletTab** en utilisant maintenant un tableau de 3 chaînes pour définir le triplet. Votre classe exécutable doit reprendre celle de l'exercice précédent en modifiant juste le nom de la classe.

2.2.4 Classe Complexe

On vous demande de définir une classe **Complexe**. Un complexe comporte une partie réelle et une partie imaginaire, il existe deux solution pour afficher un tel élément : soit sous forme cartésienne, soit sous forme complexe. Si l'on

définit un complexe dont la partie réelle est *5* et la partie imaginaire est *-6.3*, son affichage sous forme cartésienne sera **(5, -6.3)** son affichage sous forme complexe sera **-6.3 + 5i**.

On désire que cette classe satisfasse l'exécutable suivant :

```
class ExecutableComplexe{
   public static void main(String [] args) {
          Complexe c1 = new Complexe();
          c1.set(5, -7.5);
          c1.afficheCartesien();
          Complexe c2 = new Complexe();
          c2.set(5, 6.5);
          c2.afficheComplexe();
          Complexe c3 = new Complexe();
          c3 = c1.add(c2);
          c3.afficheCartesien();
   }
}
```

1. La méthode *set* permet d'initialiser ou modifier les deux composantes d'un complexe.
2. Les deux méthodes d'affichages affichent un complexe soit sous forme cartésienne, soit sous forme complexe.
3. L'instruction **c3 = c1.add(c2) ;** permet d'affecter au complexe *c3* la somme des deux complexes *c1* et *c2*.

2.2.5 Classe Date

Définissez une classe **Date**, ayant pour attributs le jour, le mois et l'année. Cette classe doit satisfaire l'exécutable suivant :

```
class ExecutableDate {
   public static void main(String [] args) {
          Date d = new Date();
          d.init(13, 12, 1940);
          if(d.valide())
              d.affiche();
          else System.out.println("Date invalide ");
          Date d2 = new Date();
          d2.init(4, -5, 1900);
          if(d2.valide())
              d2.affiche();
          else System.out.println("Date invalide ");
          Date d3 = new Date();
          d3.init(1, 3, 2012);
          if(d3.bissextile())
              System.out.println("année bissextile");
          else
              System.out.println("année non bissextile");
          d3.affiche();
          System.out.println("mois à "+d3.nbJourMois()+" jours");
   }
}
```

Vous devez pouvoir initialiser une date, déterminer si elle est valide, savoir si elle correspond à une année bissextile et indiquer le nombre de jours du mois de la date.

2.3 Exercices pour s'entrainer à faire en autonomie

2.3.1 Classe Jeu

On veut définir une classe **Jeu** qui doit fonctionner avec l'exécutable suivant : Vous identifierez correctement les attributs, le(s) constructeur(s) et les méthodes.

```
class ExecutableJeu {
	public static void main(String[] args){
		Jeu j1= new Jeu("Monopoly",31,false);
		System.out.println(j1.toString()); // doit afficher :
		// Nom du jeu : Monopoly
		// Prix neuf : 31.0 euros
		// En solde à moitié prix : false
		System.out.println(j1.prix());// doit afficher 31.0
		System.out.println(j1.getPrixNeuf());// doit afficher 31.0
		j1.setSolde(true);
		System.out.println(j1.toString()); // doit afficher :
		// Nom du jeu : Monopoly
		// Prix neuf : 31.0 euros
		// En solde à moitié prix : true
		System.out.println(j1.prix());// doit afficher 15.5
	}
```

}

2.3.2 Classe Maison

Ecrire une classe **Maison** qui répond à l'executable proposé. Votre classe comportera trois attributs : verb !ville !, verb !nbPieces !, et verb !pieces ! (une liste dans laquelle se trouve le nom de chaque pièce).

```
class ExecutableMaison {

	public static void main(String[] args){
		Maison m= new Maison("Orléans",3);
		m.affiche(); // doit afficher :
		// Localisation de la maison : Orléans
		// Nombre de pieces : 3
		// piece numero 1 : null
		// piece numero 2 : null
		// piece numero 3 : null
		System.out.println(m.getNbPieces());// doit afficher 3
		m.setVille("Tatooine");
		m.precisePiece(1,"Chambre");
		m.precisePiece(3,"Salle de billard");
		m.precisePiece(1,"Garage pour hélicoptère");
		m.affiche(); // doit afficher :
		// Localisation de la maison : Tatooine
		// Nombre de pieces : 3
		// piece numero 1 : Garage pour hélicoptère
		// piece numero 2 : null
		// piece numero 3 : Salle de billard
		System.out.println(m.cherche("Salle de billard"));
		// doit afficher true
	}
```

}

2.3.3 Classe Vaisseau

On veut définir une classe **Vaisseau** permettant de représenter un vaisseau défini par son **nom**, son **type**, le nom de son **proprietaire** ainsi que le nombre **passagers** qu'il peut embarquer et la liste des passagers à bord. Cette classe doit fonctionner avec l'exécutable suivant :

```
class ExecutableVaisseau {
   public static void main(String [] args) {
          Vaisseau faucon = new Vaisseau("Faucon millenium","epave","Lando",6);
          faucon.affiche(); // Doit afficher
          // Faucon millenium
          // Type : epave
          // Proprietaire : Lando
          // Passagers max : 6
          // Passagers à bord : -
          // Places restantes : 0

          faucon.setType("cargo");
          faucon.setProprietaire("Solo");
          faucon.addPassager("Luke");
          faucon.addPassager("Leila");
          faucon.affiche(); // Doit afficher
          // Faucon millenium
          // Type : cargo
          // Proprietaire : Solo
          // Passagers max : 6
          // Passagers à bord : Luke Leila
          // Places restantes : 4

   }
}
```

1. Écrivez l'interface de la classe **Vaisseau**, sans donner le code des méthodes.
2. Écrivez le constructeur permettant de construire un vaisseau à partir de 4 paramètres.
3. Écrivez la méthode *affiche*.

CHAPITRE 3

Semaine 2

3.1 TD 2

3.1.1 Week-end entre amis

Dans cet exercice, vous allez modéliser avec des objets un problème vu au semestre précédent et vous l'implémenterez en java.

On est à la fin d'un week-end entre amis, et on cherche à faire les comptes suite aux dépenses de chacun.

Par exemple, Pierre a acheté du pain pour 12 euros, Paul a dépensé 100 euros pour les pizzas, Pierre a payé l'essence et en a eu pour 70 euros, Marie a acheté du vin pour 15 euros, Paul a aussi acheté du vin et en a eu pour 10 euros, Anna n'a quant à elle rien acheté.

1. On vous propose de créer les classes : Personne, Depense et DepensesWeekEnd. Quels attributs doivent avoir les différentes classes pour pouvoir représenter les données ?
2. Quelles méthodes et avec quels profils devez vous ajouter pour pouvoir :

— créer une personne, créer une dépense, créer un week end. — ajouter une personne à un week end.
— ajouter une dépense à un week end.
— savoir combien a dépensé une personne,
— savoir combien chacun a dépensé en moyenne,
— savoir combien d'argent a été dépensé pour un produit donné,

3. Supposons que chaque personne veuille dépenser la même somme. Il faut donc que ceux qui ont dépensé trop récupèrent de l'argent et que les autres en re-versent. On désire donc connaître l'avoir par personne ; à chaque personne doit correspondre la somme qu'elle doit récupérer (en négatif) ou rembourser (en positif). Quelle méthode proposez vous, dans quelle classe et avec quel profil ?

3.1.2 Frigo et recettes

Dans cet exercice, nous allons considérer que l'on a une recette à faire et que l'on a un frigo rempli de différentes choses (ingrédients) ayant une quantité donnée. Une recette est exécutée avec des ingrédients selon certaines quantités et pour un nombre de personnes.

1. Comment proposez vous de *modéliser* ce problème ? On ne vous demande que les classes, les attributs ainsi que les méthodes nécessaires avec leur profil.
2. On désire savoir si le contenu d'un frigo permet de réaliser ou non une recette. *Que proposez-vous ?*
3. On veut cette fois-ci savoir, à partir du contenu d'un frigo et d'une recette, quel est le nombre maximum de personnes pour qui on peut cuisiner. *Que proposez-vous ?*

3.1.3 Pluviométrie

On désire dans cet exercice gérer la pluviométrie sur une année avec une granularité au jour (on connaît pour chaque jour la quantité d'eau qui est tombée). Pour pouvoir gérer les problèmes qui suivent, on vous demande quelle(s) classe(s) sont nécessaires, quels attributs doivent elles posséder, quelles méthodes sont nécessaires (vous donnerez les profils uniquement) ?

1. quelle est la quantité d'eau d'un jour donné ?
2. quelle est la quantité totale d'eau tombée en une année ?
3. quelle est la quantité moyenne d'eau tombée en une année ?
4. quelle est la quantité maximale d'eau tombée sur une année ?
5. quelle est la quantité d'eau tombée pour un mois donné ?
6. déterminer le tableau des pluie par mois.
7. quelle est la quantité totale qui est tombée tous les dimanches de l'année ?
8. (bonus) quel est le nombre maximal de jours d'affilés sans pluie ?
9. (bonus) quelle est la probabilité qu'il pleuve le lendemain si on a une pluviométrie donnée un jour ?

On veut maintenant gérer plusieurs années de pluviométrie.

1. Que proposez-vous comme solution ?
2. Quelles méthodes pourrait-on vouloir pour avoir les mêmes possibilités que précédemment ?

On veut également rajouter (bonus) :
— l'année la plus pluvieuse,
— le mois d'une année qui a maximisé la pluie.

3.1.4 D'une modélisation à une implémentation

On modélise ici le comptage de personnes dans un supermarché, on vous fournit l'extrait de classe suivant :

```
class Caisse {
   ArrayList<Integer> nombreDePassages;
   // chaque indice correspond à une heure de la journée
   // par ex. nombrePassages.get(8) correspond au nombre
   // de personnes passées à la caisse entre 8h et 9h.
   // en moyenne 250 passages
   ArrayList<Boolean> ouverte;
   // caisse ouverte ou non à cette heure
   Caisse() {...}
   // remplir aléatoirement les deux tableaux
   int nombreTotal() {...}
   int nombreTotalEntre(int hDebut, int hFin) {...}
   // heure où il y a le plus de passages
   int heureAffluence() {...}
}
```

— Donnez le code de chaque méthode.

Nous modélisons maintenant le supermarché :

```
class SuperMarche {
   ArrayList<Caisse> caisses;
   SuperMarche() // constructeur
   //nombre total de passages dans le magasin
   int nombreTotal() {...}
   int heureAffluence() {...}
   int heureTranquille() {...}
   // tableau du nombre moyen de passages par caisse
   // par heure
   ArrayList<Integer> moyennePassageParCaisse() {...}
   ArrayList<Integer> nbCaissesOuvertes() {...}
}
```

— Donnez le code de chaque méthode.

3.2 TP 2

3.2.1 Week end entre amis

On reprend ici l'exercice de TD, avec la correction suivante pour la partie "modélisation" :

```
public class Personne{
   String nom;
   String prenom;
   Personne(String n, String p){...}
}
public class Depense{
   Personne pers;
   double montant;
   String produit;
   Depense(Personne pers, double mont, String prod ){...}
}

public class WeekEnd{
   ArrayList<Personne> listeAmis;
   ArrayList<Depense> listeDepenses;
   WeekEnd(){...}
   void ajouterPersonne(Personne p){...}
   void ajouterDepense(Depense d){...}
   // totalDepensesPersonne prend en entrée une personne
   // et renvoie la somme des depenses de cette personne.
   double totalDepensesPersonne(Personne p){...}
   // totalDepenses renvoie la somme de toutes les dépenses.
   double totalDepenses(){...}
   // depensesMoyenne renvoie la moyenne des dépenses pour une
   // personne
   double depensesMoyenne(){...}
   //depenseProduit prend en entrée un produit, et renvoie la
   // somme des dépenses pour ce produit.
   // (du pain peut avoir été acheté plusieurs fois...)
   double depenseProduit(String p){...}
   // avoirPersonne prend en entrée une personne et renvoie
   // son avoir pour le week end.
   double avoirPersonne(Personne p){...}
}
```

— Recopiez cette correction en remplaçant les ... par du code qui ne fait rien, mais qui est correct et permet de compiler.
— Compilez vos classes. *Note :* pour le moment, ces classes ne font rien.
— Créez une classe **Executable** contenant la modélisation de la situation suivante :

Pierre a acheté du pain pour 12 euros, Paul a dépensé 100 euros pour les pizzas, Pierre a payé l'essence et en a eu pour 70 euros, Marie a acheté du vin pour 15 euros, Paul a aussi acheté du vin et en a eu pour 10 euros, Anna n'a quant à elle rien acheté.

1. Ajoutez un appel à la méthode permettant de connaître les dépenses totales du week end.
2. Ajoutez les appels permettant de tester *toutes* les méthodes de vos classes.
3. Implémentez les méthodes dans cet ordre, en vérifiant à chaque fois que votre implémentation est correcte sur l'exemple donné :

— Les constructeurs : Personne, Depense et WeekEnd.
— void ajouterPersonne(Personne p) et void ajouterDepense(Depense d)

— flot totalDepenses()
— float depensesMoyenne(Personne p)
— float totalDepensesPersonne(Personne p)
— float depenseProduit(Produit p)
— avoirPersonne(Personne p)

3.2.2 Frigo et Recette

Reprenez l'exercice *Frigo et recettes* du TD et suivez la même démarche qu'à l'exercice précédent.

3.2.3 Qui est-ce

Dans cet exercice, on veut modéliser le jeu **Qui-est-ce**, dans lequel on a une liste de personnes et où on élimine les personnes qui ne répondent pas à certains critères jusqu'à n'avoir plus qu'une seule personne.

On vous propose la modélisation suivante :

```
class Barbe{
       String couleur;
       boolean existe; // true si la barbe n'est pas rasée
       Barbe(){...}    // crée une nouvelle barbe rasée
       Barbe(String s){...} // crée une nouvelle barbe de couleur s
       boolean estEgalA(Barbe b){...}
}

class Personne{
       Barbe barbe;
       boolean lunettes; // true si la personne porte des lunettes
       String yeux; // contient la couleur des yeux
       String nom;
       String prenom;

       Personne(String nom, String prenom, Barbe barbe, boolean lunettes, String yeux)
↪{...}
       // renvoie vrai si b est semblable à la barbe de la personne
       boolean verifie(Barbe b){...}
       // renvoie vrai si le booléen lun correspond au booléen lunettes de la personne
       boolean verifie(boolean lun){...}
       // renvoie vrai si la couleur des yeux de la personne est la même que s
       boolean verifie(String s){...}
       void affiche(){...}
}

class QuiEstCe{
       ArrayList<Personne> personnes;
       // crée une instance de QuiEstCe en utilisant les personnes de tab pour␣
↪initialiser personnes
       QuiEstCe(Personne[] tab){...}
       // supprime de personnes toutes les personnes dont les yeux ne sont pas de la␣
↪couleur yeux
       void elimine(String yeux){...}
       // idem avec les lunettes
       void elimine(boolean lunettes){...}
       // idem avec la barbe
       void elimine(Barbe b){...}
       // renvoie le nombre de personnes dans personnes
```

```
        int nbPossibilites(){...}
        void affiche(){...}
}
```

— Créez une exécutable :

1. crée un ensemble de 5 personnes différentes,
2. affiche cet ensemble
3. élimine les personnes sans lunettes
4. affiche à nouveau cet ensemble
5. recommence avec d'autres critères.

— Implementez les méthodes jusqu'à ce que l'exécutable fonctionne.

CHAPITRE 4

Semaine 3

4.1 TD3 - Space Invader

L'objectif de ce TD, qui se prolongera en TP puis en DM est de réaliser un jeu de Space Invader en ASCII art.

Les dessins sont tirés de la page http://textart4u.blogspot.fr/2014/04/space-invaders-copy-paste-ascii-text-art.html

4.1.1 Diagramme de classes

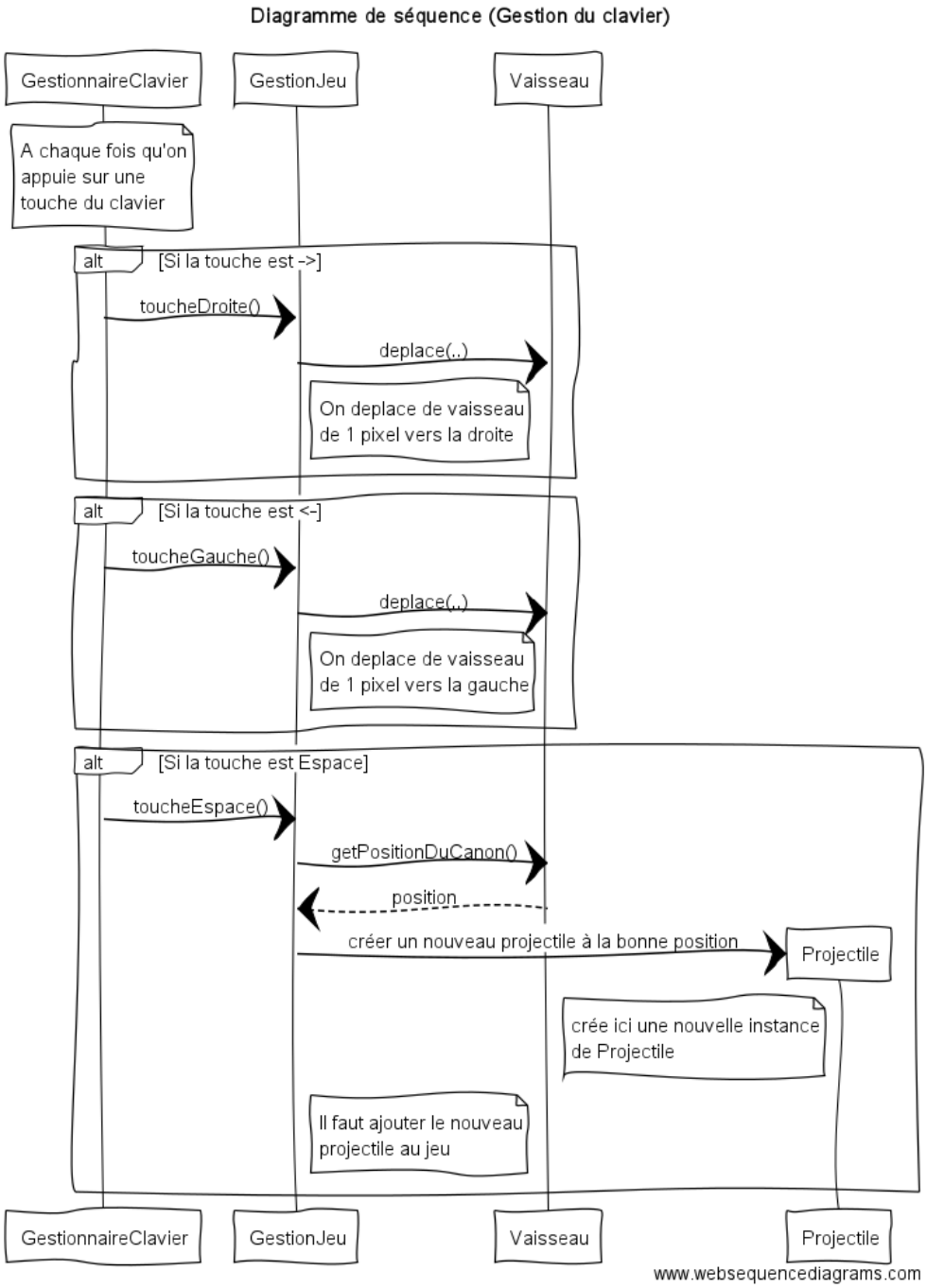
Ce projet va nécessiter plusieurs classes :
- ChainePositionnee
- Dessin
- Alien
- GestionJeu

— Projectile
— Score
— Vaisseau
— Vue
— GestionnaireClavier
— GestionnaireTemps

1. D'après vous, à quoi vont servir chacune de ces classes ?
2. Réclamez un diagramme de classes (vide) à votre professeur. Vous devrez le compléter au fur et à mesure de votre travail et le rendre à la fin du projet.

Remarque : vous devrez, à chaque fois, préciser le type des attributs et le profil des méthodes (type de retour, nombre et type des paramètres)

4.1.2 Diagramme de séquence (Gestion des touches du Clavier)



Le diagramme de séquences ci-dessus décrit ce qu'il se passe à chaque fois qu'on appuie sur une touche du clavier.

Le début du diagramme se lit comme suit :

Si on appuie sur la touche −> du clavier :

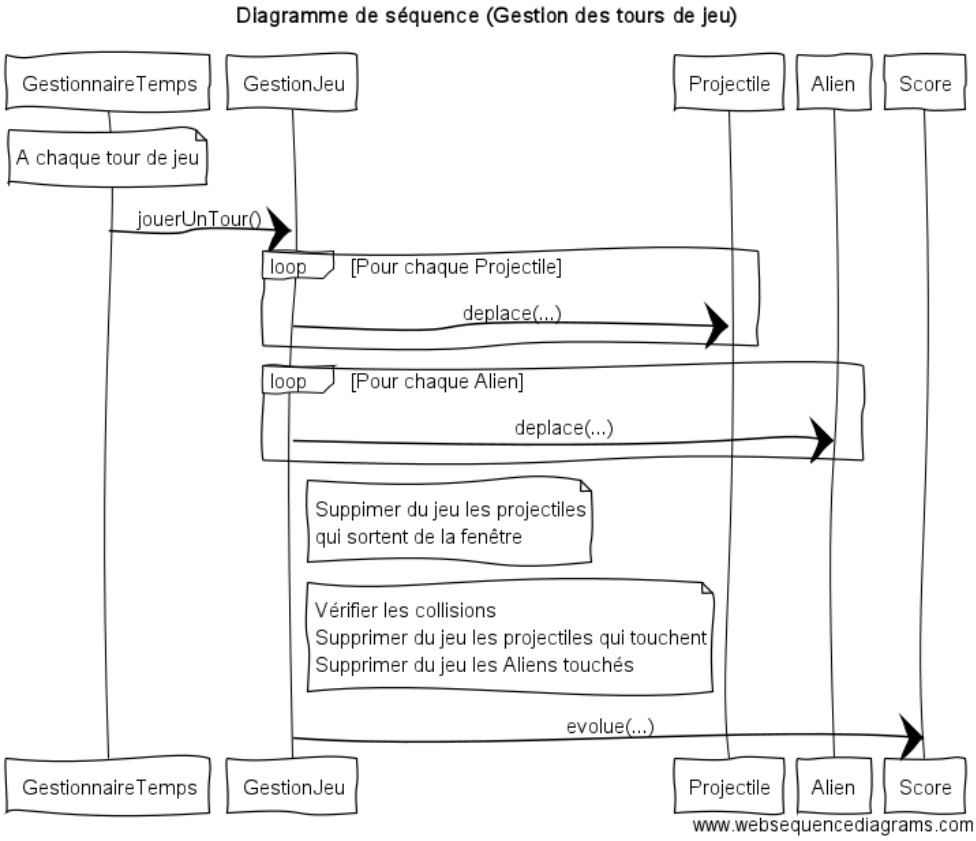
- le GestionnaireClavier appelle la méthode *toucheDroite()* qui se trouve dans la classe GestionJeu. Pour que cela soit possible :
  - la classe GestionnaireClavier doit posséder un attribut de type GestionJeu ;
  - la classe GestionJeu doit posséder une méthode *toucheDroite()*

— l'objet GestionJeu appelle alors la méthode *deplace()* qui se trouve dans la classe Vaisseau. Pour que cela soit possible :
  — la classe GestionJeu doit posséder un attribut de type Vaisseau ;
  — la classe Vaisseau doit posséder une méthode *deplace()*

1. Compléter le diagramme de classes fourni avec ces informations.
2. Analysez de la même façon la deuxième partie du diagramme (Si on appuie sur la touche <− du clavier) et compléter le diagramme de classes.
3. Analysez de la même façon la troisième partie du diagramme (Si on appuie sur la touche Espace du clavier) et compléter le diagramme de classes.

Remarque : cette première analyse n'est pas définitive et certaines choses peuvent évoluer au fur et à mesure de votre travail

4.1.3 Diagramme de séquence (Gestion du temps)



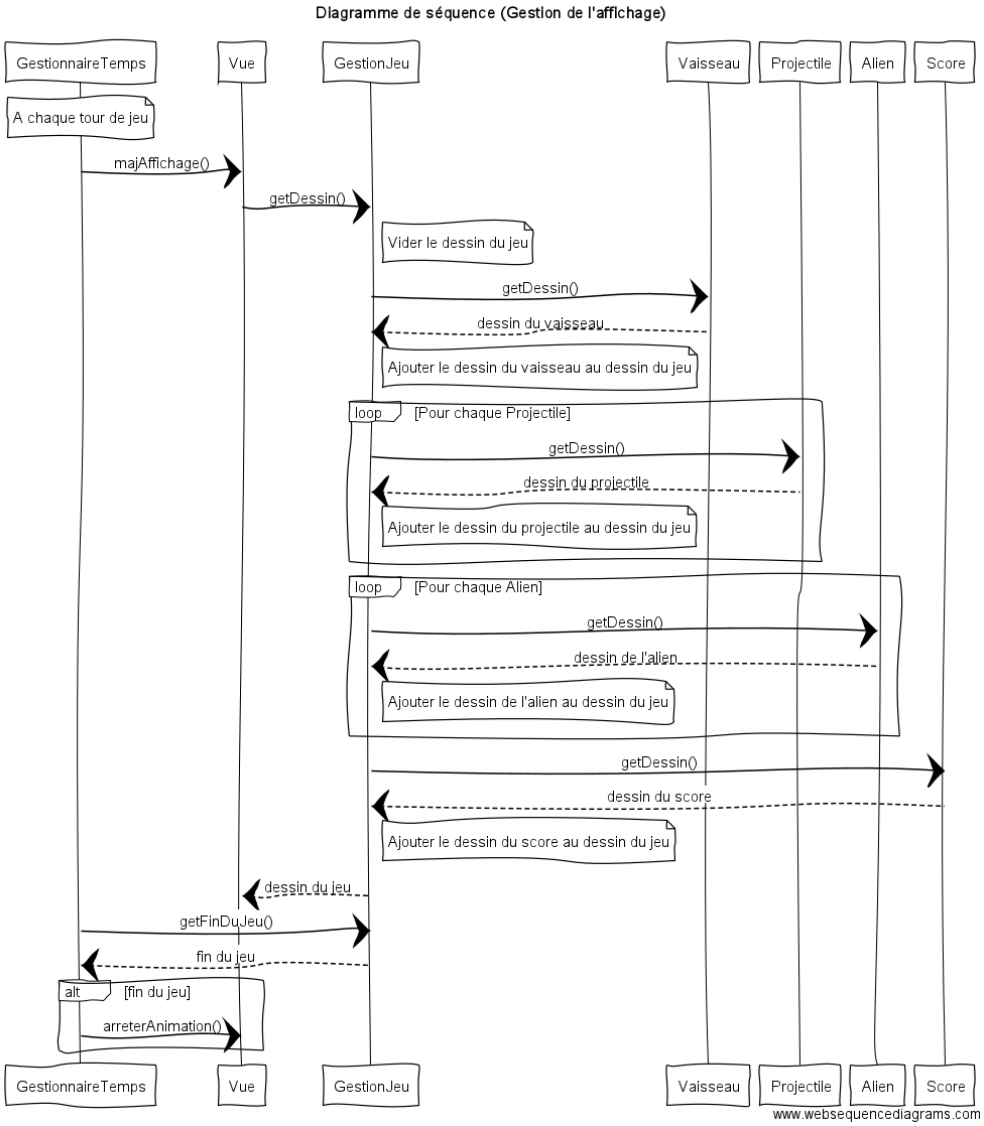
Le diagramme de séquences ci dessus décrit ce qu'il se passe à fréquence constante (qui correspond à un tour de jeu).

Analysez ce diagramme pour répondre aux questions suivantes et compléter le diagramme de classes au fur et à mesure :

1. Quels attributs doivent posséder les différentes classes ?
2. Quelles méthodes doivent posséder les différentes classes ?
3. Ajoutez-les à votre diagramme des classes.

Remarque : à ce stade, certaines méthodes peuvent vous paraître encore un peu "floues". C'est normal et vous affinerez votre compréhension au fur et à mesure de votre travail

4.1.4 Diagramme de séquence (Gestion de l'affichage)



Le diagramme de séquences ci-dessus décrit comment notre application va gérer l'affichage

1. Analysez ce diagramme et compléter le diagramme de classes.

4.1.5 Retour sur le Diagramme de classes

Parmi toutes les méthodes de votre diagramme de classes, d'après vous :

1. Quelles sont celles qui vont nécessiter une boucle ?
2. Quelles sont celles qui vous semblent simples à coder (moins de 10 lignes de code) ?
3. Quelles sont celles qui ne vous semblent pas encore claires ?

4.2 TP3 - Space Invader

L'objectif de ce TP, qui se prolongera en DM est de réaliser un jeu de Space Invader en ASCII art.

Les dessins sont tirés de la page http://textart4u.blogspot.fr/2014/04/space-invaders-copy-paste-ascii-text-art.html

4.2.1 Prise en main

Un dessin ASCII est la donnée de plusieurs chaines de caractères, chacune venant avec sa position. Par exemple :

```
 XXX °°
 uu
oXXo
```

est donné par la liste `(1,2,"XXX"), (5,2,"°°"),(1,1,"uu"),(0,0,"oXXo")` . Remarquez qu'on utilise ici des caractères qui ne sont pas sur votre clavier. Je les ai obtenus par copier/coller.

Pour représenter un tel dessin (plusieurs chaines positionnées), on utilisera deux classes : ChainePositionnee et Dessin :

ChainePositionnee.java

```
// CLASSE DONNEE AUX ETUDIANTS
// NE PAS MODIFIER

/**
 * l'origine du repère, c'est à dire le point de coordonnées (0, 0),
 * est le point situé en bas à gauche de la fenêtre de jeu
 */

class ChainePositionnee{
        /* ======== Attributs ======== */
    double x,y;
    String chaine;

    /* ======== Constructeur ======== */
    ChainePositionnee(double x, double y, String str){
                this.x=x;
                this.y=y;
                this.chaine=str;
        }

        /* ======== Getteurs ======== */
        double getx(){ return this.x; }
        double gety(){ return this.y; }
        String getChaine(){ return this.chaine; }

        /* ======== Setteurs ======== */
        void setx(double x){ this.x=x; }
        void sety(double y){ this.y=y; }
        void setChaine(String c){ this.chaine=c; }

        /* ======== Autres méthodes ======== */
        /**
         * Cette méthode renvoie true si le point de coordonnées (tx, ty)
         * coincide avec l'un des caractères de la chaine positionnée
         */
        boolean contient(double tx, double ty){
                return  (this.x <=tx  && (int) this.y == (int) ty && tx < this.
↪x+chaine.length());
        }
 }
```

Dessin.java

```
// CLASSE DONNEE AUX ETUDIANTS
// NE PAS MODIFIER

import java.util.ArrayList;
/**
 * Dessin est une classe qui permet de représenter un dessin
 * Un dessin contient un attribut qui est une liste de ChainePositionnee
 */

class Dessin {
        /* ======== Attributs ======== */
    ArrayList<ChainePositionnee> listeChaines;

    /* ======== Constructeur ======== */
    Dessin(){
                this.listeChaines= new ArrayList<ChainePositionnee>();
        }

        /* ======== Getteurs ======== */
        ArrayList<ChainePositionnee> getChaines(){
                return this.listeChaines;
        }

        /* ======== Autres méthodes ======== */

    public void ajouteChaine(double x, double y, String chaine){
        this.listeChaines.add(new ChainePositionnee(x,y,chaine));
    }


    public void ajouteDessin(Dessin autreDessin){
        for(ChainePositionnee cp : autreDessin.getChaines())
            this.listeChaines.add(cp);
    }

        /**
         * indique si une position (x,y) coincide avec l'un des caractères
         * de l'ensemble des chaines de caractères
         */
    public boolean contient(double posx, double posy){
        for(ChainePositionnee cp: listeChaines){
            if (cp.contient(posx, posy))
                return true;
        }
        return false;
    }

    /**
         * permet de vider le dessin
         */
    public void vider(){ this.listeChaines.clear();}

}
```

Pour la gestion de l'affichage, on vous donne une classe `Vue` dont *nul n'est besoin de lire le code* et téléchargeable `ici`.

Deux autres classes orchestreront la gestion du clavier et la gestion du temps : `GestionnaireClavier` et `GestionnaireTemps` téléchargeable `ici` et `ici`

Pour que cela fonctionne, vous devez "seulement" coder le jeu, en ayant une classe `GestionJeu` fournissant les méthodes suivantes :

- `int getHauteur()` et `int getLargeur()` renvoyant la largeur et la hauteur de la zone de jeu.
- `void toucheGauche()`, `void toucheDroite()` et `void toucheEspace()` ne renvoyant rien et qui sont appellées automatiquement lorsque l'on appuie sur la flèche gauche, droite ou touche espace du clavier.
- `Dessin getDessin()` renvoyant le dessin à afficher.
- `void jouerUnTour()` qui est appellée à fréquence constante (qui correspond à un tour de jeu) et que vous remplirez pour faire évoluer le jeu.

Remarque : l'origine du repère, c'est à dire le point de coordonnées (0, 0), est le point situé en bas à gauche de la fenêtre de jeu

Mise en place

1. Modifiez le contenu de la classe *GestionJeu* pour obtenir une zone de jeu de largeur 100 et de hauteur 60, et affichant dans la fenêtre jeu le dessin `XX` en position X=0, Y= 30 (ce dessin très simple contient une liste qui ne contient qu'une seule ChainePositionnee)
2. Modifiez votre classe *GestionJeu* pour que lorsque l'utilisateur appuie sur la touche espace, le texte "Appui sur la touche espace" soit affiché **dans le terminal**.
3. Modifiez votre classe pour que le dessin `XX` soit déplacé vers la droite (dans la fenêtre de jeu) à chaque fois que l'utilisateur appuie sur la flèche droite. Pour cela, votre classe devra contenir un attribut "positionX" contenant la position en X de votre dessin.

4.2.2 Vaisseau et projectiles

Vaisseau

1. Écrivez une classe `Vaisseau` qui :

- contient des attributs (voir analyse faite en TD)
- contient une méthode *déplace()* prenant en entrée un double (dx) et modélisant le déplacement du vaisseau.
- contient une méthode *getDessin()* renvoyant le Dessin permettant d'afficher le vaisseau dans la fenêtre de jeu en position (posX, 0) :

```
      o
     XXX
oXXXXXXXXXXXo
XXXXXXXXXXXXX
```

2. Faites afficher ce vaisseau. Pour cela, ajoutez un attribut de type Vaisseau au gestionnaire de jeu.
3. Faites en sorte que lorsque l'on appuie sur les flèches du clavier, le vaisseau se déplace.
4. Ajoutez une méthode *positionCanon* qui renvoie la position en x du canon (on utilisera cette méthode plus tard pour savoir d'où faire partir le projectile).

Projectile(s)

1. Écrivez une classe `Projectile` qui :

— contient deux attributs positionX et positionY de type double.
— a un constructeur prenant en argument deux *double* : la position en X et la position en Y.
— contient une méthode *getDessin()* renvoyant le dessin du projectile (ici, la liste de ChainePositionnee ne contiendra que la chaine positionnée `(positionX, positionY,"^")`.
— contient une méthode `void evolue()` qui fait incrémenter la position en y de 0.2 .

2. Ajoutez à votre classe GestionJeu un attribut de type Projectile. Ajoutez le code nécessaire pour que ce projectile s'affiche et évolue à chaque tour de jeu.
3. Comment faire pour avoir plusieurs projectiles (éventuellement autant qu'on veut) ? Modifiez votre classe GestionJeu dans ce sens.
4. Modifiez votre classe GestionJeu pour que lorsque l'on appuie sur la touche espace, un nouveau projectile parte du canon du vaisseau (pensez à utiliser la méthode *positionCanon* de la classe Vaisseau).

4.2.3 Score et Aliens

Score

1. Écrivez une classe Score permettant d'afficher un score en haut à gauche de l'écran. (cette classe contient donc une méthode getDessin)
2. Ajoutez un attribut de type Score à votre gestionnaire de jeu.
3. Modifiez votre code pour que le score soit incrémenté à chaque tour de jeu. Pour cela, ajoutez une méthode *ajoute(int valeur)* à votre classe Score, et ajoutez dans GestionJeu un appel à cette méthode à chaque tour de jeu.

Aliens

1. Écrivez une classe Alien représentant un Alien (les *super vilains* qu'il faut *dégommer* dans Space Invader). Reprenez par exemple un dessin de la page http://textart4u.blogspot.fr/2014/04/space-invaders-copy-paste-ascii-text-art.html .
2. Modifiez la classe GestionJeu pour qu'elle contienne un ArrayList d'Aliens initialisé comme sur le dessin du début du sujet.
3. Modifiez vos classes pour que les Aliens évoluent de la façon suivante : Pendant les premiers 100 tours de jeu, leur position en X augmente de 0.1 à chaque tour. Puis on décrémente leur position en Y de 1. Pendant les 100 tours de jeu suivants, on décrémente leur position en X pendant 100 tours de suite. Puis on décrémente de leur position en Y, et ainsi de suite.

Vous devriez maintenant avoir quelque chose comme ça (avec le score en plus...) :

4.2.4 Supprimer les projectiles qui sortent de la zone de jeu

On va maintenant gérer le fait que lorsqu'un projectile sort de la zone de jeu, il est supprimé de la liste (sinon, on risque de saturer rapidement la mémoire).

On va décomposer par étapes :

Suppression des projectiles qui sortent

1. Pour chaque projectile du jeu, vérifier s'il est encore dans la fenêtre de jeu (il suffit de comparer sa position en Y avec la hauteur de la zone de jeu). Vous allez remplir une ArrayList des projectiles qui sortent. Affichez cette ArrayList à chaque tour de jeu.
2. Plutot que d'afficher cette ArrayList, on va appeler la méthode *remove()* de l'ArrayList de projectile pour supprimer chaque projectile qui sort de la fenêtre.

4.2.5 Collisions

On va maintenant gérer le fait que lorsqu'un projectile touche un alien, ce dernier disparaît. Pour cela, on va d'abord devoir être capable de détecter qu'un projectile touche un alien, c'est-à-dire que la position (x,y) du projectile correspond à la position d'un des caractères représentant l'alien.

On va décomposer par étapes :

Appartenance

1. En utilisant la méthode *contient()* de la classe ChainePositionnee, créer une méthode *contient()* dans la classe Dessin prenant en argument deux double x et y et renvoyant true si le point à ces coordonnées appartient à une des chaines et faux sinon.
2. Créer une méthode *contient()* dans la classe Alien prenant en argument deux double x et y et renvoyant true si le point à ces coordonnées touche l'Alien.
3. Vous pouvez maintenant modifier votre classe GestionJeu pour que lorsqu'un alien est "touché" par un projectile, le texte "TOUCHE" s'affiche **dans le terminal**. *Remarque* : vous allez devoir imbriquer deux boucles : Pour chaque projectile faire : Pour chaque alien faire : tester alien.contient(projectile.getx(),projectile.gety()) .

Suppression des objets

1. Maintenant, au lieu d'afficher "TOUCHE", vous allez remplir deux ArrayList : l'un contenant les projectiles qui ont touché des aliens, l'autre contenant les aliens qui ont été touchés par un projectile. Affichez ces deux ArrayList à chaque tour.
2. Plutôt que d'afficher ces deux ArrayList, on va appeler la méthode *remove()* de l'ArrayList de projectiles pour supprimer chaque projectile ayant touché un alien. De même avec l'ArrayList d'aliens pour supprimer tous les aliens qui ont été touchés.

4.2.6 A vous de jouer

Implémentez maintenant une ou plusieurs des fonctionnalités suivantes :

- Gérer la fin du jeu.
- Faire en sorte que les Aliens soient “animés” comme sur l’image.
- Faire en sorte que la vitesse des Aliens augmente au fur et à mesure.
- Ajouter des explosions quand un projectile touche un alien.
- Ajouter de la couleur
- Ajouter une fonctionnalité “Pause” quand on appuie sur la touche P
- d’autres idées ? ?