Jeu de l'anagramme [tb06] - Examen

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 19 mai 2018

Table des matières

Java - Jeu de l'anagramme (Solution)

Mots-Clés Tableau unidimensionnel, Jeu ■
Requis Structures de base, Structures conditionnelles, Algorithmes paramétrés, Structures répétitives, Schéma itératif ■
Difficulté ●●○ (2 h) ■

Objectif
Cet exercice réalise le jeu de l'anagramme.

...(énoncé page suivante)...

1 Jeu de l'anagramme / pgjeugram (20 points)

1.1 Présentation du jeu

Anagramme

Une **anagramme** (le mot est féminin) – du grec *anagramma* : renversement de lettres – est une construction qui permute les lettres d'un mot pour en extraire un mot nouveau. Exemple : crane, écran, nacre, carne, rance, ancre. (Source : Wikipédia)

Jeu de l'anagramme

Il se joue à deux. L'un pense à un mot, par exemple :

```
potence (la longueur du mot est de 7 caractères)
```

mélange les lettres de ce mot initial et l'affiche :

```
eteconp (on ne vérifie pas que le mot existe!)
```

L'autre doit deviner le mot initial en proposant des anagrammes. Après chacun des anagrammes proposés, l'autre joueur donne le nombre de lettres bien placées. Par exemple, pour :

```
tecnpoe: 1 lettre(s) bien placée(s) (dernier e bien placé)
petonce: 5 lettre(s) bien placée(s) (ici p.t.nce)
```

Au bout de sept échecs, le joueur a perdu s'il n'a pas réussi à deviner le mot initial.

Déroulement du programme

La machine organise le jeu.

1. Le premier joueur entre un mot de 3 à 10 lettres pendant que l'autre joueur ne regarde pas.
2. La machine mélange les lettres de ce mot puis affiche l'anagramme.
3. Le second joueur propose un anagramme.
4. La machine indique combien de lettres sont bien placées.
5. La machine reprend au point 3 sauf si le joueur a épuisé tous ses coups ou trouvé le mot initial.

A la fin de la partie, la machine affiche si le joueur a gagné ou non.

1.2 Saisie d'un mot (6 points)

L'idée est d'utiliser un tableau d'au plus dix caractères pour mémoriser le mot à deviner ainsi que pour proposer un mot. Pour indiquer qu'une lettre a été prise, on la remplacera par le caractère souligné qui n'est pas une lettre.

De plus, pour éviter d'avoir des mots dont certaines lettres seraient en minuscule et d'autres en majuscule, seules des lettres minuscules pourront être saisies.

(1 point) Définissez les constantes entières `LMIN=3` (longueur minimale) et `LMAX=10` (longueur maximale) des mots. Puis définissez le type `TabCarac` comme étant un tableau de `LMAX` caractères.

Validez vos définitions avec la solution.

Solution Java @[pgjeugram.java]

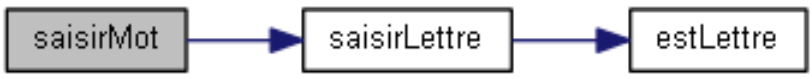
```
/**
  Longueur minimale des mots
*/
final static int LMIN = 3;

/**
  Longueur maximale des mots
*/
final static int LMAX = 10;
```

(1 point) Écrivez une fonction `estLettre(c)` qui teste et renvoie `Vrai` si un caractère `c` est une lettre minuscule (donc compris entre `'a'` et `'z'`), `Faux` sinon.

(2 points) Déduisez une fonction `saisirLettre` qui effectue la saisie contrainte d'une lettre puis la renvoie, c.-à-d. que la fonction doit demander un caractère jusqu'à ce que `estLettre` soit `Vrai`.

(2 points) Enfin écrivez une fonction `saisirMot(mot)` qui effectue la saisie d'un entier `n` (contraint entre `LMIN` et `LMAX`), saisit ensuite `n` lettres dans un `TabCarac mot` par appel à la fonction `saisirLettre` puis renvoie `n` (longueur de `mot`).



Validez vos fonctions avec la solution.

Solution Java @[pgjeugram.java]

```
/**
  Prédicat de lettre minuscule
  @param[in] c - un caractère
  @return Vrai si c est une lettre minuscule, Faux sinon
*/

public static boolean estLettre(char c)
```

```
{
  return (('a' <= c) && (c <= 'z')) || (c == '_');
}
```

```
/**
  Saisie d'une estLettre
  @return une estLettre
*/

public static char saisirLettre()
{
  Scanner input = new Scanner(System.in);
  char c;
  c = input.next().charAt(0);
  while (!estLettre(c))
  {
    System.out.println(c + " n'est pas une lettre valide");
    c = input.next().charAt(0);
  }
  return c;
}
```

```
/**
  Saisie d'un TabCarac
  @param[out] mot - un TabCarac
  @return la taille de mot
*/

public static int saisirMot(char[] mot)
{
  System.out.println("Tapez votre mot (lettres en minuscule) -- Finissez par _");
  int n = 0;
  char c = saisirLettre();
  while ((n < LMAX) && (c != '_'))
  {
    mot[n] = c;
    ++n;
    c = saisirLettre();
  }
  return n;
}
```

(0.5 point) Écrivez le début d'un programme qui déclare un `TabCarac` puis effectue sa saisie.

Testez.

1.3 Procédures et fonctions utilitaires (6 points)

(2 points) Écrivez une procédure `melanger(mot,n)` qui mélange les `n` caractères d'un `TabCarac mot` comme suit :

- Pour `j` variant de `n-1` à 1 (pas **décroissant**)

1. Tirez au hasard un entier `k` entre 0 et`j` **inclus**.
2. Permutez les caractères en position `k` et `j` de `mot`.

Outil Java

La classe `Random`, définie dans le paquetage `java.util`, fabrique des générateurs de valeurs entières ou réelles calculées de façon pseudo-aléatoire. La méthode `nextInt()` renvoie un entier pseudo-aléatoire. Utilisez le modulo, **éventuellement** la valeur absolue, pour projeter l'entier dans l'intervalle souhaité.

(1 point) Écrivez une procédure `afficherMot(mot,n)` qui affiche les `n` premiers caractères d'un `TabCarac mot`.

(1 point) Écrivez une procédure `copierMot(mot,copie,n)` qui recopie les `n` caractères d'un `TabCarac mot` dans un `TabCarac copie`.

Validez vos procédures avec la solution.

Solution Java @[pgjeugram.java]

```
/**
  Mélange les lettres d'un TabCarac
  @param[in,out] mot - un TabCarac
  @param[in] n - taille de mot
*/

public static void melangerMot(char[] mot, int n)
{
  Random rnd = new Random();
  for (int j = n - 1; j > 0; --j)
  {
    int k = rnd.nextInt(j + 1);
    char tmp = mot[k];
    mot[k] = mot[j];
    mot[j] = tmp;
  }
}
```

```
/**
  Affichage d'un TabCarac
  @param[in] mot - un TabCarac
  @param[in] n - taille de mot
*/

public static void afficherMot(final char[] mot, int n)
{
  for (int j = 0; j < n; ++j)
  {
    System.out.print(mot[j]);
  }
  System.out.println();
}
```

```java
/**
  Copie d'un TabCarac
  @param[in] mot - un TabCarac
  @param[out] copie - un TabCarac
  @param[in] n - taille de mot
*/

public static void copierMot(final char[] mot, char[] copie, int n)
{
  for (int j = 0; j < n; ++j)
  {
    copie[j] = mot[j];
  }
}
```

(2 points) Écrivez une fonction `rechlin(c,mot,n)` qui recherche et renvoie l'indice d'un caractère `c` dans les `n` premiers caractères d'un `TabCarac mot`, -1 en cas de recherche infructueuse.

Validez votre fonction avec la solution.

Solution Java @[pgjeugram.java]

```java
/**
  Recherche linéaire d'un caractère dans un TabCarac
  @param[in] c - un caractere
  @param[in] mot - un TabCarac
  @param[in] n - taille de mot
  @return Position de c dans les n caractères de mot, -1 si n'existe pas
*/

public static int rechlin(char c, final char[] mot, int n)
{
  int j = 0;
  while (j < n && mot[j] != c)
  {
    ++j;
  }
  return (j < n ? j : -1);
}
```

1.4 Proposition d'un mot (3 points)

Pour la saisie contrainte de `n` caractères dans un `TabCarac mot` étant donné l'anagramme `TabCarac ana`, on utilisera l'algorithme suivant :

- Recopiez l'anagramme `ana` dans un `TabCarac copie`.
- Tant que les `n` caractères n'ont pas été saisis :
  - Saisissez une lettre dans `c` (par exemple).
  - Recherchez `c` dans la `copie`.

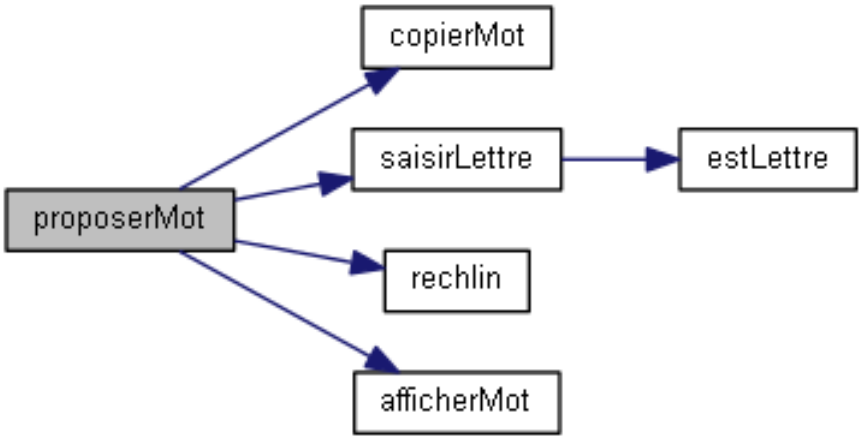
— En cas de recherche fructueuse, placez `c` dans `mot`, incrémentez le nombre de caractères saisis et remplacez la lettre dans `copie` par le blanc souligné (`_`).

— Sinon, affichez :

```
Lettre [c] non presente dans l'anagramme
```

- Finalement, affichez le mot saisi.

(3 points) Écrivez une procédure `proposerMot(mot,n,ana)` qui réalise l'algorithme spécifié.



Validez votre procédure avec la solution.

Solution Java @[pgjeugram.java]

```
/**
  Proposition d'un TabCarac
  @param[out] mot - un TabCarac
  @param[in] n - taille de ana et de mot
  @param[in] ana - un TabCarac anagramme
*/

public static void proposerMot(char[] mot, int n, final char[] ana)
{
  char[] copie = new char[LMAX];
  copierMot(ana,copie,n);
  int j = 0;
  while (j < n)
  {
    char c = saisirLettre();
    int k = rechlin(c,copie,n);
    if (k != -1)
    {
      mot[j] = c;
      ++j;
      copie[k] = '_';
      afficherMot(copie,n);
    }
    else
    {
```

```
      System.out.println("Lettre " + c + " non presente dans l'anagramme...");
    }
  }
  System.out.print("==> Votre Mot = ");
  afficherMot(mot,n);
}
```

1.5 Divination (5 points)

Reste à définir la procédure qui permet à un joueur de proposer des mots jusqu'à ce qu'il trouve le mot de `n` caractères d'un `TabCarac adeviner` en au plus sept coups. Cette procédure devra :

- Recopiez `adeviner` dans un `TabCarac ana`.
- Mélangez les lettres de l'anagramme `ana`.
- Déclarez un `TabCarac mot`, un entier `ncoups` du nombre de coups effectués et un entier `nbp` du nombre de lettres bien placées.
- Initialisez `ncoups` et `nbp`.
- Tant que les sept coups n'ont pas été joués et que le mot n'a pas été trouvé :
  - Incrémentez le nombre de coups effectués.
  - Affichez le message suivant (où `[x]` désigne le contenu de `x`) :

    ```
    coup [ncoups] -- Anagramme du mot a trouver = [ana]
    ```

  - Proposez un `mot` de `n` caractères parmi `ana`.
  - Calculez le nombre de lettres bien placées `nbp`.
  - Affichez le message suivant :

    ```
    [nbp] lettre(s) bien placee(s)
    ```

- Affichez l'un des messages, selon que le joueur a ou non trouvé le mot :

  ```
  Super gagne en [ncoups] coups
  Perdu -- le mot a trouve etait [adeviner]
  ```

(1 point) Écrivez une fonction `bienPlaces(mot,n,adeviner)` qui calcule et renvoie le nombre de caractères bien placés parmi les `n` premiers caractères d'un `TabCarac mot` et `TabCarac adeviner`.

Validez votre fonction avec la solution.

Solution Java @[pgjeugram.java]

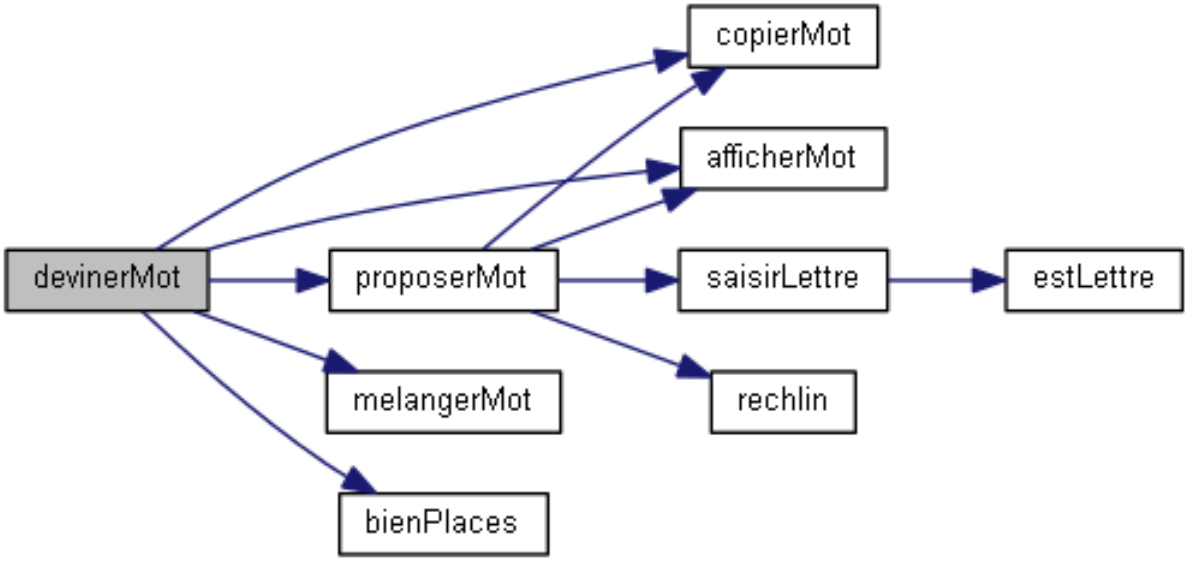
```
/**
  Nombre de lettres bien placées
  @param[in] mot - un TabCarac
  @param[in] n - taille de mot
  @param[in] adeviner - mot a-deviner
```

```
  @return le nombre de lettres bien-placées
*/

public static int bienPlaces(final char[] mot, int n, final char[] adeviner)
{
  int nbp = 0;
  for (int j = 0; j < n; ++j)
  {
    if (mot[j] == adeviner[j])
    {
      ++nbp;
    }
  }
  return nbp;
}
```

Définissez la constante `MAXCOUPS=7` (nombre de coups maximum).

(3 points) Écrivez une procédure `devinerMot(adeviner,n)` qui réalise l'algorithme spécifié ci-avant.



Validez votre définition et votre procédure avec la solution.

Solution Java @[pgjeugram.java]

```
/**
  Nombre maximum de coups
*/
final static int MAXCOUPS = 7;
```

```
/**
  Jeu de l'anagramme
  @param[in] adeviner - mot a-deviner
  @param[in] n - taille de adeviner
```

```
*/

public static void devinerMot(final char[] adeviner, int n)
{
  char[] ana = new char[LMAX];
  copierMot(adeviner,ana,n);
  melangerMot(ana,n);
  int ncoups = 0;
  int nbp = 0;
  while (ncoups < MAXCOUPS && nbp != n)
  {
    ++ncoups;
    System.out.print("coup " + ncoups + " -- Anagramme du mot a trouver = ");
    afficherMot(ana,n);
    char[] mot = new char[LMAX];
    proposerMot(mot,n,ana);
    nbp = bienPlaces(mot,n,adeviner);
    System.out.println(nbp + " lettres bien placees");
  }
  if (nbp == n)
  {
    System.out.println("Super Gagne en " + ncoups + " coups");
  }
  else
  {
    System.out.println("Perdu");
  }
}
```

(0.5 point) Complétez votre programme en appelant la procédure précédente.

Validez votre programme avec la solution.

Solution Java @[pgjeugram.java]

```
public static void main(String[] args)
{
  char[] adeviner = new char[LMAX];
  int n = saisirMot(adeviner);
  devinerMot(adeviner,n);
}
```

2 Références générales

Comprend [] ■