

Développement initiatique

SAE 1 : Jeu de Scrabble (version du 05/12)

1 Présentation du jeu de Scrabble et du projet à réaliser

Le but de ce projet est de programmer une partie du jeu de Scrabble entre humains dans un premier temps, puis entre humains et ordinateur.

Le jeu de Scrabble se compose d'un plateau, d'un sac de jetons-lettres et de chevalets. Le plateau contient une grille à compléter par des mots à la manière des grilles de mots croisés. Les cases de la grille ont des couleurs différentes selon la valorisation des nombres de points des lettres ou mots posés sur ces cases. Les jetons-lettres sont des jetons carrés contenant chacun une lettre majuscule, ainsi que dans un coin le nombre de points rapportés par la pose de ce jeton sur le plateau (les lettres les plus rares sont les plus chères !). Les chevalets permettent aux joueurs de ranger leurs jetons.

Chaque joueur dispose à tout moment de 7 jetons (ou moins quand le sac est vide), et pose à tour de rôle un mot sur le plateau, qui lui fait gagner un certain nombre de points selon les points des jetons du mot formé et la valorisation donnée par les couleurs des cases sur lesquelles il a posé ses jetons. De plus, si le joueur "fait un scrabble", c'est-à-dire pose un mot utilisant ses 7 jetons, il gagne 50 points supplémentaires.

Vous trouverez tous les détails du contenu et de la règle du jeu sur internet, par exemple sur les sites <https://fr.wikipedia.org/wiki/Scrabble> et https://fr.wikipedia.org/wiki/Lettres_du_Scrabble.

Vous allez d'abord programmer une version de base du jeu, avec une règle du jeu un peu simplifiée (par exemple, absence de prise en compte des jetons "touchant sur les côtés" le mot ajouté). On vous demande d'écrire le programme en programmation à objets, plus appropriée pour gérer les différents éléments du jeu. Vous pourrez ensuite améliorer cette première version à l'aide de plusieurs extensions optionnelles permettant de corriger les simplifications faites dans la version de base, et d'améliorer la présentation visuelle du jeu et les algorithmes mis en oeuvre dans certaines fonctions. Vous pourrez aussi inclure l'ordinateur parmi les joueurs. L'algorithme d'IA proposé pour faire jouer l'ordinateur est rudimentaire, mais libre à vous d'en utiliser un autre si vous avez des idées !

L'évaluation prendra en compte la qualité du code produit. C'est pourquoi, il est conseillé de prendre le temps de bien écrire la version de base avant d'aborder les extensions. Si vous souhaitez ajouter des extensions, vous pouvez choisir celles que vous souhaitez programmer, dans l'ordre que vous préférez.

Barème

La version de base rapportera au maximum 14 points sur 20, si elle est correcte et très bien écrite.

Vous rendrez donc au minimum les classes de la version de base, et le cas échéant une ou plusieurs classes codant les extensions choisies. Vous indiquerez en commentaires au début des classes additionnelles les numéros des extensions choisies, ainsi que toutes les explications nécessaires si vous avez choisi des extensions personnalisées (variantes de celles proposées, ou complètement différentes).

Remarque sur le passage de paramètres des objets et des tableaux

Pour faire évoluer les différentes structures représentant l'état du jeu au cours d'une partie, on va parfois définir des fonctions qui vont modifier les tableaux ou les objets passés en paramètres. Ce qui enlève les premières hypothèses posées en début d'année sur le passage de paramètres, et confirme le *passage de paramètres par référence* en Java qui sera étudié lors du dernier cours (sur les références et les listes chaînées). *En Java, un tableau ou un objet passé en paramètre d'une fonction et modifié dans cette fonction est aussi modifié dans la fonction appelante*, ce qui est justement l'effet souhaité ici.

2 Version de base

Le modèle objet de ce jeu est basé sur plusieurs classes détaillées ci-dessous. Le chef d'orchestre est la classe `Scrabble` qui gère le déroulement d'une partie. On va définir un objet `Plateau` comprenant une matrice de `Case` et une classe `Joueur`. Une classe `MEE` permettra de gérer des ensembles de jetons présents sur les chevalets ou le sac de jetons.

Dans cette version très guidée, les méthodes dont la signature est précisée dans cette section doivent être obligatoirement écrites. Vous pouvez éventuellement ajouter des méthodes supplémentaires, à condition d'en donner les spécifications complètes, mais notez qu'il n'est pas nécessaire de le faire pour faire fonctionner ce programme.

Les jetons

Dans cette version de base, il n'y a pas de jeton joker (sans lettre, pouvant remplacer n'importe quelle lettre). Un jeton est codé par le rang de 0 à 25 de la lettre qu'il contient dans l'ordre alphabétique (0 pour 'A' et 25 pour 'Z'). On stocke le nombre de points rapporté par chaque jeton (lettre) dans un tableau `nbPointsJeton` indicé de 0 à 25. Dans la suite, on identifiera un jeton à son code de 0 à 25.

Ce tableau constituera un attribut de classe (statique) de la classe principale `Scrabble`. Il sera déclaré et initialisé par une instruction de la forme :

```
private static int [] nbPointsJeton = {1, 3, 3, ... };
```

La classe MEE et les ensembles de jetons

La classe MEE décrit un Multi-Ensemble d'Entiers. Elle est utilisée pour représenter des ensembles de jetons, notamment :

- le sac,
- les jetons se trouvant sur le chevalet de chaque joueur.

Un ensemble de jetons est notamment représenté par un tableau d'entiers indicé de 0 à 25 contenant à l'indice i le nombre de jetons i (c'est-à-dire de code i) de l'ensemble. Plus précisément, les attributs de la classe MEE sont :

```
private int [] tabFreq; // tabFreq[i] est le nombre d'exemplaires
                        // (fréquence) de l'élément i

private int nbTotEx;    // nombre total d'exemplaires
```

Par exemple, un ensemble de 2 jetons 'A', un jeton 'D' et 3 jetons 'E' est représenté par un objet MEE dont l'attribut `tabFreq` est : [2, 0, 0, 1, 3, 0, 0, ..., 0] (`tabFreq.length` = 26, ou 27 si on tient compte des jetons joker) et l'attribut `nbTotEx` vaut 6. Prévoir un accesseur en lecture `getNbTotEx` de l'attribut `nbTotEx`. L'attribut `nbTotEx` n'est pas indispensable, mais il permet de savoir si un multi-ensemble est vide sans parcourir le tableau `tabFreq`.

La classe MEE permet de manipuler des multi-ensembles d'entiers compris entre 0 et `tabFreq.length-1`. Comme un ensemble, un multi-ensemble n'est pas ordonné, mais contrairement à un ensemble, il peut contenir plusieurs exemplaires du même élément (ce qui est bien le cas des ensembles de jetons). La classe MEE est similaire à la classe `EEbool` (sujet de TP sur Moodle), qui est elle-même une variante de la classe `EE` vue en TD.

On définit trois constructeurs :

```
/**
 * pré-requis : max >= 0
 * action : crée un multi-ensemble vide dont les éléments seront
 *          inférieurs à max
 */
public MEE (int max){

/**
 * pré-requis : les éléments de tab sont positifs ou nuls
 * action : crée un multi-ensemble dont le tableau de fréquences est
 *          une copie de tab
 */
public MEE (int[] tab){
```

```

/**
 * constructeur par copie
 */
public MEE (MEE e){

```

On définit également les méthodes suivantes :

```

/**
 * résultat : vrai ssi cet ensemble est vide
 */
public boolean estVide (){

```

```

/**
 * pré-requis : 0 <= i < tabFreq.length
 * action : ajoute un exemplaire de i à this
 */
public void ajoute (int i) {

```

```

/**
 * pré-requis : 0 <= i < tabFreq.length
 * action/résultat : retire un exemplaire de i de this s'il en existe,
 *                  et retourne vrai ssi cette action a pu être effectuée
 */
public boolean retire (int i) {

```

```

/**
 * pré-requis : this est non vide
 * action/résultat : retire de this un exemplaire choisi aléatoirement
 *                  et le retourne
 */
public int retireAleat () {

```

```

/**
 * pré-requis : 0 <= i < tabFreq.length
 * action/résultat : transfère un exemplaire de i de this vers e s'il
 *                  en existe, et retourne vrai ssi cette action a pu être effectuée
 */
public boolean transfere (MEE e, int i) {

```

```

/** pré-requis : k >= 0
 * action : tranfère k exemplaires choisis aléatoirement de this vers e
 *          dans la limite du contenu de this
 * résultat : le nombre d'exemplaires effectivement transférés
 */
public int transfereAleat (MEE e, int k) {

```

```

/**
 * pré-requis : tabFreq.length <= v.length
 * résultat : retourne la somme des valeurs des exemplaires des
 * éléments de this, la valeur d'un exemplaire d'un élément i
 * de this étant égale à v[i]
 */
public int sommeValeurs (int[] v){

```

Pour le choix aléatoire d'un jeton, vous pouvez utiliser la fonction `Ut.randomMinMax(int min, int max)` avec `min = 0` et `max = 25`, mais attention, l'entier retourné n'est pas forcément un jeton de l'ensemble...

La classe Case

Une case est définie par plusieurs attributs. Sa couleur (valorisation des lettres ou des mots occupant la case) sera un entier entre 1 et 5. Une case est libre ou recouverte et, dans le dernier cas, donne accès à la lettre recouvrant la case.

En plus de la couleur, vous pourrez définir les attributs de votre choix, mais vous devrez coder les méthodes suivantes.

```

/**
 * pré-requis : uneCouleur est un entier entre 1 et 5
 * action : constructeur de Case
 */
public Case (int uneCouleur){

/**
 * résultat : la couleur de this, un nombre entre 1 et 5
 */
public int getCouleur (){

/**
 * pré-requis : cette case est recouverte
 */
public char getLettre (){

/**
 * pré-requis : let est une lettre majuscule
 */
public void setLettre (char let){

```

```

/**
 * résultat : vrai ssi la case est recouverte par une lettre
 */
public boolean estRecouverte (){

public String toString (){

```

Les mots

Les joueurs saisissent des suites de lettres (représentant les mots proposés) sous forme de variables de type String.

Précision sur le type String : si mot est une variable de type String alors il peut être saisi par un appel à `Ut.saisirChaine()`, sa longueur est `mot.length()` et pour tout entier i de 0 à `mot.length() - 1`, son caractère de rang i est `mot.charAt(i)`.

Pour vérifier qu'un caractère saisi est une lettre majuscule, convertir une lettre majuscule en son code de 0 à 25, et inversement, vous pouvez ajouter à votre fichier `Ut` les méthodes `estUneMajuscule`, `majToIndex` et `indexToMaj` en vous inspirant des fonctions `Ut.alphaToIndex` et `Ut.indexToAlpha` réalisant ces conversions pour les lettres minuscules.

La classe Plateau

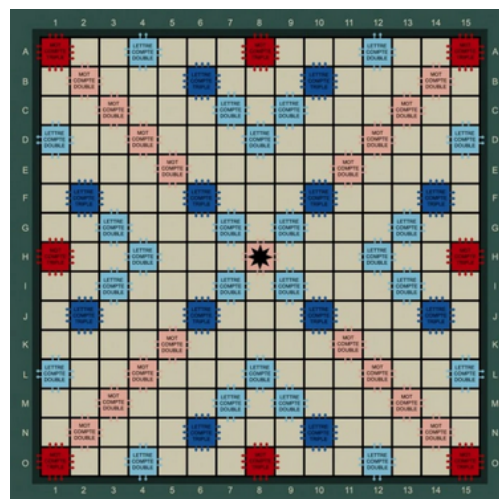


FIGURE 1 – Plateau de Scrabble (source tout pour lejeu.com)

La classe `Plateau` comprend un unique attribut représentant une grille de 15×15 cases de couleur :

```
private Case [][] g; // g pour grille
```

Les couleurs sont codées par des entiers de la façon suivante :

- gris (pas de valorisation particulière) : 1
- bleu clair (lettre compte double) : 2
- bleu foncé (lettre compte triple) : 3
- rose (mot compte double) : 4
- rouge (mot compte triple) : 5

Le constructeur de Plateau n'a pas d'argument et initialise l'attribut à partir de la déclaration d'une variable locale indiquant les couleurs de toutes les cases :

```
int [][] plateau = { {...} , {...} , ... };
```

(C'est un peu fastidieux, on essaiera de faire mieux dans une extension !)

Cette classe devra offrir une méthode permettant l'affichage :

```
/**
 * résultat : chaîne décrivant ce Plateau
 */
public String toString (){
```

Suggestions pour l'affichage (que vous êtes libre de personnaliser) :

- utiliser les caractères '|' et '_' pour dessiner la grille,
- indiquer les indices de lignes à gauche et les indices de colonnes en haut (pour faciliter la saisie de l'emplacement du mot proposé par un joueur),
- mettre dans chaque case recouverte la lettre majuscule correspondante,
- mettre dans chaque case non recouverte de couleur non grise son code de couleur (de 2 à 5),
- laisser vides les autres cases (non recouvertes, de couleur grise).

La classe Plateau comporte également des méthodes permettant de placer un mot proposé par un joueur. Un mot proposé est caractérisé par :

- le mot lui-même, de type String,
- les coordonnées (indices de ligne et de colonne du plateau) de la case devant contenir la première lettre du mot,
- le sens de placement du mot ('h' pour horizontal, 'v' pour vertical).

```
/**
 * pré-requis : mot est un mot accepté par CapeloDico,
 * 0 <= numLig <= 14, 0 <= numCol <= 14, sens est un élément
 * de {'h','v'} et l'entier maximum prévu pour e est au moins 25
 * résultat : retourne vrai ssi le placement de mot sur this à partir
 * de la case (numLig, numCol) dans le sens donné par sens à l'aide
 * des jetons de e est valide.
 */
public boolean placementValide(String mot, int numLig, int numCol,
                               char sens, MEE e) {
```

Le mot proposé doit d'abord satisfaire les règles du Scrabble (nom commun du dictionnaire, au singulier ou au pluriel, ou verbe à l'infinitif ou conjugué, etc.). Dans la version de base, cette validation sera assurée par un être humain, un utilisateur que nous appellerons CapeloDico¹, dont vous devrez gérer l'intervention à chaque fois qu'un joueur propose un mot. L'utilisation d'un "vrai" dictionnaire, stocké sous forme de fichier, fera l'objet d'une extension.

Ensuite, la méthode `placementValide` effectuera plusieurs contrôles selon que le plateau est vide (premier mot placé) ou non :

Dans le cas où le plateau est vide, la méthode `placementValide` doit vérifier que :

- le mot proposé a au moins 2 lettres,
- la *zone de placement du mot* (suite de cases consécutives - de la longueur du mot - à partir de la case de sa première lettre, dans le sens de placement du mot) contient la case centrale du plateau,
- l'ensemble `e`, correspondant au chevalet du joueur proposant le mot, contient les jetons permettant de former le mot.

Dans le cas où le plateau n'est pas vide, la méthode `placementValide` vérifie que :

- la zone de placement du mot ne dépasse pas de la grille,
- cette zone n'est ni précédée ni suivie d'une case recouverte par un jeton ; autrement dit, elle est précédée (respectivement suivie) du bord de la grille ou d'une case non recouverte,
- cette zone contient au moins une case non recouverte par un jeton,
- cette zone contient au moins une case recouverte par un jeton,
- pour chaque case recouverte par un jeton de la zone de placement du mot, la lettre du jeton est la même que celle du mot à placer dans cette case,
- l'ensemble `e`, correspondant au chevalet du joueur proposant le mot, contient les jetons permettant de former le mot.

La classe `Plateau` comprend enfin des méthodes pour comptabiliser le nombre de points rapportés par la pose du mot et pour modifier le plateau avec le mot posé.

```
/**
 * pré-requis : le placement de mot sur this à partir de la case
 * (numLig, numCol) dans le sens donné par sens est valide
 * résultat : retourne le nombre de points rapportés par ce placement, le
 * nombre de points de chaque jeton étant donné par le tableau nbPointsJet.
 */
public int nbPointsPlacement(String mot, int numLig, int numCol,
                             char sens, int[] nbPointsJet) {
```

Dans cette version de base, on ne tient pas compte de la présence éventuelle de jetons "touchant" le mot sur les côtés et formant des mots transversaux. C'est à l'avantage du joueur si ces mots transversaux ne sont pas tous des mots autorisés par CapeloDico, mais à son désavantage s'ils le sont car ils lui feraient gagner des points supplémentaires !

1. En hommage à Jacques Capelovici, dit maître Capelo, grand amateur des mots et des calembours, auteur de nombreuses grilles de mots-croisés, à la fin du 20^e siècle.


```

/**
 * pré-requis : le placement de mot sur this à partir de la case
 *      (numLig, numCol) dans le sens donné par sens à l'aide des
 *      jetons de e est valide.
 * action/résultat : effectue ce placement et retourne le
 *      nombre de jetons retirés de e.
 */
public int place(String mot, int numLig, int numCol, char sens, MEE e){

```

La classe Joueur

Un joueur sera caractérisé par trois attributs :

```

private String nom;
private MEE chevalet;
private int score;

```

Vous devez coder quelques méthodes utiles de la classe Joueur : un constructeur (public Joueur(String unNom)), une méthode permettant l'affichage, un accesseur en lecture de l'attribut score, et une méthode permettant d'augmenter le score de nb points (public void ajouteScore(int nb)).

Vous devez également coder les méthodes suivantes :

```

/**
 * pré-requis : nbPointsJet indique le nombre de points rapportés par
 *      chaque jeton/lettre
 * résultat : le nombre de points total sur le chevalet de ce joueur
 * suggestion : bien relire la classe MEE !
 */
public int nbPointsChevalet (int[] nbPointsJet){

/**
 * pré-requis : les éléments de s sont inférieurs à 26
 * action : simule la prise de nbJetons jetons par this dans le sac s,
 *      dans la limite de son contenu.
 */
public void prendJetons (MEE s, int nbJetons) {

```

Quand c'est au tour d'un joueur, il peut soit passer son tour, soit échanger tout ou partie de ses jetons, soit proposer de placer un mot sur le plateau. La méthode suivante donne le choix au joueur this :

```

/**
 * pré-requis : les éléments de s sont inférieurs à 26
 *               et nbPointsJet.length >= 26
 * action : simule le coup de this : this choisit de passer son tour,
 *           d'échanger des jetons ou de placer un mot
 * résultat : -1 si this a passé son tour, 1 si son chevalet est vide,
 *            et 0 sinon
 */
public int joue(Plateau p, MEE s, int[] nbPointsJet) {

```

Les deux actions principales possibles du joueur sont décrites par les méthodes suivantes :

```

/** pré-requis : les éléments de s sont inférieurs
 *               à 26 et nbPointsJet.length >= 26
 * action/résultat : simule le placement d'un mot de this
 *                   et retourne vrai ssi ce coup est valide
 *                   (accepté par CapeloDico et satisfaisant les
 *                   règles détaillées plus haut)
 */
public boolean joueMot(Plateau p, MEE s, int[] nbPointsJet) {

```

Si le placement du mot est *valide*, le joueur place son mot sur le plateau, reprend des jetons dans le sac pour avoir de nouveau 7 jetons (si le contenu du sac le permet). Le nombre de points du mot avec ses valorisations éventuelles (sans oublier le bonus de 50 points si le joueur a fait un scrabble) est ajouté à son score.

```

/**
 * pré-requis : les éléments de sac sont inférieurs à 26
 * action : simule l'échange de jetons de ce joueur
 */
public void echangeJetons(MEE sac) {

```

La méthode demande de saisir la suite de lettres qu'il souhaite échanger (sous forme de String) et réalise cet échange. Si la saisie est incorrecte (c'est-à-dire si le joueur ne saisit pas que des lettres majuscules ou n'a pas sur son chevalet les jetons correspondant à sa saisie), il doit recommencer sa saisie jusqu'à ce qu'elle soit correcte. Si le contenu du sac est insuffisant, les premiers jetons du mot saisi sont échangés dans la limite du contenu du sac.

La classe Scrabble

Cette classe définit toutes les structures de données utiles à une partie et comporte une méthode (partie) qui effectue une partie du début à la fin.

La classe Scrabble comporte les attributs suivants :

```
private Joueur[] joueurs;  
private int numJoueur; // joueur courant (entre 0 et joueurs.length-1)  
private Plateau plateau;  
private MEE sac;  
private static final nbPointsJeton = {1,3,3, ...};
```

Le constructeur de cette classe prend comme paramètre un tableau de chaînes de caractères indiquant les noms de tous les joueurs. Il permet d'initialiser toutes les structures du jeu. Dans cette version de base, le numéro du joueur qui commence est choisi aléatoirement.

La méthode `toString` permet d'afficher à chaque tour de jeu l'état du plateau et le joueur qui a la main.

La méthode `partie` orchestre une partie de Scrabble :

1. la distribution initiale des jetons aux joueurs,
2. des itérations sur les différents tours de jeu jusqu'à la fin de la partie,
3. le calcul des scores finaux,
4. l'affichage du ou des gagnants.

La partie peut se terminer pour deux raisons.

La partie se termine généralement quand le joueur courant n'a plus de jetons (et que le sac est vide). Ce joueur récupère alors les points des jetons restants des autres joueurs et, pour chaque autre joueur, le nombre de points de ses jetons restants est retiré de son score.

L'autre cas de terminaison (rare) se produit lors d'un "blocage", c'est-à-dire quand tous les joueurs ont passé leur tour². Dans ce cas, le nombre de points des jetons restants est retiré du score de chaque joueur.

Le gagnant de la partie est le joueur ayant le nombre maximum de points. Il peut y avoir des ex aequo.

La classe `MainScrabble`

La procédure principale (*main*) de cette classe commence par la saisie du nombre de joueurs. Cet entier doit évidemment être supérieur ou égal à 1, et ne doit pas être trop grand pour permettre à chaque joueur de commencer la partie avec 7 jetons. Les joueurs sont numérotés à partir de 0. La fonction procède ensuite à la saisie des noms des différents joueurs, avant de créer un objet `Scrabble` et d'effectuer une partie.

2. La règle officielle prévoit en fait que tous les joueurs passent plusieurs (trois) fois leur tour consécutivement pour arrêter la partie, mais vous coderez la version simplifiée où tous les joueurs passent une fois leur tour à la suite.