Retranscriptions algorithmiques du TP 3 : Mastermind

Fonction pour compter les couleurs bien placées

Pour compter les couleurs bien placées, il nous suffit que d'une seule passe dans la chaîne de la combinaison et la chaîne de la proposition. En effet, si on trouve la même couleur à la même position dans ces 2 chaînes, cela veut dire que c'est une couleur bien placée.

```
@Fonction compterPlacees C: lettre[], P: lettre[] -> int
    @DebutBloc
    result: nombre <- 0

    @Pour i: nombre <- 0, i < C.longueur, i <- i +1
        @DebutBloc
        @Si C[i] = P[i]
            @DebutBloc
            result <- result +1
            @FinBloc
    @FinBloc

    @Résultat: result
    @FinBloc
```

```
 private static int compterPlacees(char[] C, char[] P)
{
    int result = 0;

    for (int i = 0; i < C.length; ++i) {
        if (C[i] == P[i]) {
            ++result;
        }
    }

    return result;
}
```

Fonction pour compter les couleurs mal placées

Pour compter les couleurs mal placées, d'abord, nous sommes pouvons filtrer les couleurs bien placées. C'est à dire que dès qu'on voit la même couleur à la même position à la fois dans la chaîne des de la combinaison et la chaîne de la proposition, on peut passer à la position suivante. Donc, si ce n'est pas une couleur bien placée, c'est peut-être une couleur mal placée. Pour pouvoir savoir si c'est une couleur mal placée, nous allons la comparer avec les autres couleurs du tableau. J'ai eu une idée en off d' optimiser cet algorithme en utilisant les nombres premiers, mais je vous laisse voir haha

```
@Fonction compterMalPlacees C: lettre[], P: lettre[] -> int
    @DebutBloc
    result: nombre <- 0

    @Pour i: nombre <- 0, i < P.longueur, i <- i +1
        @DebutBloc

        @Si P[i] = C[i]
            @DebutBloc
            @PasserIteration
            @FinBloc

        @Pour j: nombre <- 0, j < C.longueur, j <- j +1
            @DebutBloc
            @Si P[j] != C[j] && P[i] = C[j]
                @DebutBloc
                r <- r +1esult
                @SortirBoucle
                @FinBloc
            @FinBloc

        @FinBloc

    @Résultat: result
    @FinBloc
```

```
 private static int compterMalPlacees(char[] C, char[] P)
{
    int result = 0;

    for (int i = 0; i < P.length; ++i) {

        if (P[i] == C[i]) {
            continue;
        }

        for (int j = 0; j < C.length; ++j) {
            if (P[j] != C[j] && P[i] == C[j]) {
                ++result;
                break;
            }
        }

    }

    return result;
}
```

Procédure principale

Enfin, sur cette procédure principale, elle va surtout demander à l'utilisateur de rentrer une combinaison et de rentrer une proposition et d'évaluer si l'utilisateur à gagner. Dans le cas où il n'aurait pas gagné, on lui affiche le nombre de couleurs bien placées et le nombre de couleurs mal placées. À savoir que dans cet algorithme, nous faisons une boucle pour bien vérifier que l'utilisateur ne dépasse pas le nombre d'essais maximal autorisé.

```
@Fonction mastermind Rien -> Rien
    @DebutBloc
    nbEssaiMax: nombre <- 6
    String combinaison <- Demander "Combinaison?"

    @Pour i: nombre <- 0, i < nbEssaiMax, i <- i +1
        @DebutBloc
        String proposition <- Demander "Proposition?"

        C: lettre[] <- combinaison
        P: lettre[] <- proposition
        nbPlacees: nombre <- @Appeler compterPlacees @Avec C, P
        nbMalPlacees: nombre <- @Appeler compterMalPlacees @Avec C, P

        @Si nbPlacees = C.longueur
            @DebutBloc
                Afficher "Gagné !"
                @Résultat: Rien
            @FinBloc

        Afficher "Placees: " + nbPlacees + ", mal placées: " + nbMalPlacees
        @FinBloc

    Afficher "On ne peut pas tout le temps gagné !"
    @FinBloc
```

```
public static void mastermind() {
    int nbEssaiMax = 6;
    String combinaison = JOptionPane.showInputDialog("Combinaison?");

    for (int i = 0; i < nbEssaiMax; ++i) {
        String proposition = JOptionPane.showInputDialog("Proposition?");

        char[] C = combinaison.toCharArray();
        char[] P = proposition.toCharArray();
        int nbPlacees = compterPlacees(C, P);
        int nbMalPlacees = compterMalPlacees(C, P);

        if (nbPlacees == C.length) {
            System.out.println("Gagné !");
            return;
        }

        System.out.println("Placees: " + nbPlacees + ", mal placées: " + nbMalPlacees);
    }

    System.out.println("On ne peut pas tout le temps gagné !");
}
```

Exécution

Enfin, pour exécuter cet algorithme, il suffit d'appeler l'algorithme principal qu'on a créé juste avant.

```
@Appeler mastermind @Avec Rien
```

```
public static void main(String[] args) {
    mastermind();
}
```