# Développement initiatique

# SAE 1 : Jeu de Master Mind

## 1 Présentation du jeu de Master Mind et du projet à réaliser

Les parties modifiées de ce sujet sont écrites en rouge. Elles se trouvent uniquement dans les extensions 3.1 (ajoutée) et 3.2, avec un décalage dans la numérotation des extensions.

Dans le squelette de code, 4 méthodes ont été modifiées. Chacune d'elles est précédée d'une ligne de la forme :

/**CHANGE : (commentaire sur la modification faite)

Le but de ce projet est de programmer une partie du jeu de Master Mind entre un humain et l'ordinateur (cf. https://fr.wikipedia.org/wiki/Mastermind).

Le Master Mind est un jeu à 2 joueurs. Lors d'une manche d'une partie de Master Mind, l'un des joueurs est le *codeur* (ou codificateur) et l'autre le *décodeur*. Le codeur choisit un code secret, qui est une séquence d'un certain nombre (4 ou 5 dans les versions classiques du jeu) de pions de couleur. Le nombre de couleurs possibles est 6 ou 8 dans les versions classiques du jeu. Le décodeur doit alors déterminer ce code secret.

Pour cela, il fait des propositions de codes au codeur, qui lui indique pour chacune d'elles le nombre de pions de la bonne couleur bien placés et mal placés. Par exemple, si le code secret est (Bleu, Rouge, Jaune, Rouge) et que le décodeur propose (Jaune, Rouge, Rouge, Vert), le codeur indique 1 bien placé (le pion rouge de rang 1, en considérant que le premier pion est de rang 0) et 2 mal placés (l'autre pion rouge et le pion jaune).

La manche se termine quand le décodeur propose un code égal au code secret. Le nombre de propositions de code faites par le décodeur est alors ajouté au score du codeur. La règle impose un nombre d'essais maximum ($nbEssaisMax$) au décodeur (égal au nombre de rangées sur le plateau). Quand le décodeur atteint le nombre maximum d'essais, $nbEssaisMax$ est ajouté au score, ainsi qu'un malus lié au dernier code proposé.

Une partie se compose d'un certain nombre de manches, chaque joueur étant alternativement codeur et décodeur. Le nombre de manches doit donc être pair pour équilibrer les chances des 2 joueurs. Le gagnant de la partie est le joueur ayant le plus grand score à la fin de la partie.

Vous allez d'abord programmer une version de base du jeu, entre un humain et l'ordinateur. Vous pourrez ensuite améliorer cette première version à l'aide de plusieurs extensions optionnelles permettant :

— d'évaluer la stratégie de l'ordinateur, et de la comparer éventuellement à d'autres stratégies ;
— d'améliorer la présentation visuelle du jeu ;
— de réécrire le programme en programmation objet, plus appropriée pour gérer les différents éléments du jeu ;
— de définir des contre-stratégies pour le joueur humain et l'ordinateur ;
— de définir de nouvelles stratégies pour l'ordinateur.

Il vous est demandé de programmer au minimum la version de base. Pour l'évaluation, il sera tenu compte des extensions réalisées, mais surtout de la qualité de programmation. Il vous est donc conseillé de prendre le temps de bien écrire la version de base avant d'aborder les extensions, parmi lesquelles vous pouvez choisir celles que vous souhaitez programmer, dans l'ordre que vous préférez.

Vous rendrez donc au minimum une classe `MasterMindBase` (version de base), et éventuellement une classe `MasterMindEtendu` et des classes correspondant à la version objet. Si vous avez une version étendue, vous indiquerez en commentaires au début de la classe `MasterMindEtendu` les numéros des extensions choisies, ainsi que toutes les explications nécessaires.

Un squelette de la classe `MasterMindBase` vous est donné. Ce squelette contient les en-têtes et spécifications de toutes les fonctions nécessaires à la version de base. Vous devez écrire le code de ces fonctions sans en modifier les en-têtes et en suivant scrupuleusement leurs spécifications, faute de quoi vos fonctions ne passeront pas les tests automatiques qui seront utilisés pour l'évaluation de votre projet ! Vous pouvez rajouter des fonctions à ce squelette si vous en éprouvez le besoin, à condition d'en écrire les spécifications et d'écrire aussi le code de toutes les fonctions demandées, puisque ce sont elles qui seront testées par les tests automatiques.

# 2 Version de base (12 pts)

## Les paramètres d'une partie

Avant de commencer la partie, joueur (humain) saisit le nombre de pions du code secret, noté $lgCode$, le nombre de couleurs possibles, noté $nbCouleurs$, l'identité de ces couleurs, le nombre de manches de la partie, noté $nbManches$, et le nombre maximum de codes à proposer (correspondant au nombre de rangées sur le plateau), noté $nbEssaisMax$.

Les couleurs sont stockées dans un tableau de caractères $tabCouleurs$ de longueur $nbCouleurs$ contenant les initiales des noms de couleurs saisis par le joueur. Ce sont ces initiales qui seront utilisées dans les codes. Les noms de couleurs doivent donc avoir des initiales distinctes. Les entiers $lgCode$, $nbCouleurs$, $nbManches$ et $nbEssaisMax$ doivent être strictement positifs, et $nbManches$ doit être pair. Ces conditions sont à vérifier au moment de la saisie, et comme indiqué dans le squelette de code, ce sont des pré-requis implicites de toutes les fonctions ayant ces données en paramètres.

Dans les exemples donnés dans le sujet et le squelette, on suppose que $nbCouleurs = 6$,

$lgCode = 4$ et que les couleurs saisies sont "Rouge", "Bleu", "Jaune", "Vert", "Orange" et "Noir" dans cet ordre, de sorte que le tableau $tabCouleurs$ contient ('R', 'B', 'J', 'V', 'O', 'N').

## Les deux représentations d'un code

Un code est représenté soit par un mot (de type String) de longueur $lgCode$ dont chaque caractère est un élément de $tabCouleurs$ (pour la saisie ou l'affichage du code), soit par un tableau d'entiers obtenu à partir de ce mot en remplaçant chaque caractère par son indice dans $tabCouleurs$, appelé "numéro" de la couleur correspondante (pour les calculs sur ce code).

Pour l'exemple ci-dessus, le code (Bleu, Rouge, Jaune, Rouge) est représenté, à l'écran, par le mot "BRJR" et, en machine, par le tableau d'entiers (1, 0, 2, 0).

## Précisions sur la fin d'une manche

Rappelons qu'une manche se termine quand le décodeur propose un code égal au code secret ou qu'il a fait $nbEssaisMax$ propositions.

Dans le premier cas (code trouvé), le nombre de propositions de code faites par le décodeur est ajouté au score du codeur.

Dans le deuxième cas, on ajoute $nbEssaisMax$ (qui est aussi le nombre de propositions de code) au score, plus un $malus$ qui dépend de la réponse au dernier code proposé : $rep[nbEssaisMax - 1] = (nbBienPlaces, nbMalPlaces)$.

On considèrera dans le programme (dans les fonctions `mancheHumain` et `mancheOrdinateur`) :

$$malus = nbMalPlaces + 2 \times (lgCode - (nbBienPlaces + nbMalPlaces))$$

Ce qui incitera le décodeur à faire une dernière proposition avec un maximum de pions de la bonne couleur...

## Outils de base

Vous allez écrire des fonctions utiles pour votre programme de jeu : d'abord des fonctions classiques sur les tableaux, puis des fonctions plus spécifiques pour la gestion des codes sous forme de mots ou de tableaux d'entiers.

Le nombre de pions mal placés dans un code par rapport au code secret est le nombre d'éléments communs aux deux tableaux d'entiers indépendamment de leur position, qui ne sont pas bien placés. Le nombre d'éléments communs aux deux tableaux se calcule efficacement à l'aide de leurs tableaux de fréquence (le tableau de fréquence d'un code contient à l'indice $i$ le nombre de pions de ce code de la couleur numéro $i$).

Ecrire et tester les fonctions de la rubrique *Outils de base*.

## Une manche Humain

On appelle manche Humain une manche au cours de laquelle c'est le joueur humain qui "joue", c'est-à-dire qui est le décodeur. L'ordinateur choisit un code secret sous forme d'un tableau d'entiers choisis aléatoirement entre 0 et $nbCouleurs - 1$. Ensuite, pour chaque code proposé par le joueur sous forme de mot, il vérifie que ce code est correctement formé (s'il ne l'est pas, le joueur doit le re-saisir jusqu'à ce qu'il le soit), puis il transforme le mot en un tableau d'entiers et affiche à l'écran les nombres de pions bien et mal placés dans le code proposé par le joueur.

Ecrire et tester la fonction *mancheHumain*.

## Fonctions complémentaires sur les codes pour la manche Ordinateur

Après quelques fonctions utiles pour l'affichage et la saisie, on vous propose d'écrire les fonctions gérant la stratégie de l'ordinateur.

La stratégie de l'ordinateur pour trouver le code secret $S$ est la suivante. Il propose tous les codes candidats possibles dans un certain ordre, sauf ceux qui ne sont pas compatibles avec les propositions et les réponses précédentes du joueur, c'est-à-dire telles que si le code à proposer était le code secret, le joueur n'aurait pas répondu exactement les mêmes réponses aux propositions précédentes. Autrement dit, si on compare un code à proposer candidat $C$ avec un code déjà proposé $O$ et que la réponse (nombre de pions bien et mal placés) est différente de la réponse obtenue en comparant $S$ et $O$, alors on sait que le code $C$ ne peut pas être le code secret $S$ et l'ordinateur ne le propose pas.

Une telle stratégie peut sembler assez inefficace a priori, ce sera à vous d'en juger par expérimentation (et calcul dans l'extension 3.3). Pour pouvoir déterminer la compatibilité d'une proposition de code, vous définirez la matrice $cod$, de dimensions $nbEssaisMax \times lgCode$, qui contient l'historique des propositions de codes. Vous définirez également la matrice $rep$ de dimensions $nbEssaisMax \times 2$ contenant les réponses du joueur. Pour toute ligne $i$ :

- $cod[i]$ contient la $(i+1)^{\text{ème}}$ proposition de code ;
- $rep[i][0]$ contient le nombre de pions bien placés à cette proposition ;
  $rep[i][1]$ contient le nombre de pions mal placés à cette proposition.

L'ordre choisi sur l'ensemble des codes représentés par des tableaux d'entiers est l'ordre *lexicographique*. Késako ?

Etant donnés deux tableaux $t_1$ et $t_2$ de longueur $lgCode$ contenant des entiers naturels inférieurs à $nbCouleurs$, $t_1$ *est inférieur à* $t_2$ *dans l'ordre lexicographique* si $t_1[i] < t_2[i]$, où $i$ est le plus petit indice tel que $t_1[i] \neq t_2[i]$. Dans l'exemple ci-dessus, si $t_1 = (3,2,3,4)$ et $t_2 = (3,2,4,0)$, $t_1$ est inférieur à $t_2$ car le plus petit indice $i$ tel que $t_1[i] \neq t_2[i]$ est 2 et $t_1[2] = 3 < 4 = t_2[2]$. Cet ordre correspond à l'ordre usuel sur les entiers naturels inférieurs à $nbCouleurs^{lgCode}$ si on voit un code comme l'écriture d'un tel entier en base $nbCouleurs$ sur $lgCode$ chiffres, en complétant par des 0 à gauche. C'est aussi en ordre lexicographique que sont rangés les mots dans un dictionnaire, en considérant l'ordre alphabétique entre les lettres.

**Remarque importante :** La fonction `passeCodeSuivantLexico` modifie le paramètre $cod1$ en le remplaçant par le code suivant. De même, la fonction `passePropSuivante` remplit une ligne de la matrice $cod$ donnée en paramètre. Donc, *contrairement à ce qui était souvent fait en cours et en TD*, vous allez définir des fonctions *modifiant* les tableaux (ou matrices) donnés en paramètres. En Java, un tableau transmis en paramètre d'une fonction $f_2$ et modifié dans $f_2$ est aussi modifié dans la fonction $f_1$ appelant $f_2$, ce qui est justement l'effet souhaité ici.

Ecrire et tester les fonctions complémentaires sur les codes pour la manche Ordinateur.

## Une manche Ordinateur

Une manche Ordinateur est une manche au cours de laquelle c'est l'ordinateur qui "joue" (décode). Le joueur humain choisit un code "dans sa tête", mais ne le saisit pas au clavier. L'ordinateur fait une suite de propositions de codes affichées sous forme de mots, et pour chacune d'elles, il demande au joueur humain le nombre de pions bien et mal placés et vérifie que les réponses sont correctement formées (si elles ne le sont pas, le joueur doit les re-saisir jusqu'à ce qu'elles le soient). Les propositions de code de l'ordinateur suivent la stratégie définie ci-dessus.

Si l'ordinateur atteint le dernier code possible dans l'ordre lexicographique sans trouver le code secret, c'est que le joueur s'est trompé dans au moins une réponse, ou a volontairement triché. Dans ce cas, le programme n'ajoute aucun point au score du joueur pour cette manche.

Voici un exemple d'exécution d'une manche Ordinateur. Le joueur a choisi le code secret VJBR. Les propositions de l'ordinateur et les réponses du joueur (nombre de pions bien placés (BP) et mal placés (MP) respectivement), sont les suivantes :

| Code | BP | MP |
|---|---|---|
| RRRR | 1 | 0 |
| RBBB | 1 | 1 |
| JRBJ | 1 | 2 |
| JBRV | 0 | 4 |
| VRJB | 1 | 3 |
| VJBR | 4 | 0 |

La fonction retourne 6, qui sera ajouté au score du joueur.

On remarque sur cet exemple (et vous pourrez remarquer sur vos propres exécutions de cette fonction) que la stratégie fait que l'ordinateur détermine le code secret en deux phases : une première phase où il détermine le nombre de pions de chaque couleur du code secret, dans l'ordre croissant des numéros de couleur (4 premières propositions dans l'exemple ci-dessus) et une deuxième phase où il détermine leur position par permutations des pions dans la proposition précédente. Cette remarque permet d'améliorer l'efficacité de l'algorithme de recherche de la proposition de code suivante (sans modifier la stratégie) ce qui fera l'objet d'une extension.

Ecrire et tester la fonction `mancheOrdinateur`.

### Le programme principal

Ecrire et tester les fonctions de saisie pour le programme principal, puis la fonction *main*.

# 3 Extensions

## 3.1 Affichage du plateau (1 pt)

Le plateau de jeu est composé de lignes contenant chacune une proposition de code du décodeur et la réponse du codeur. Il est pratique pour le décodeur de visualiser le contenu du plateau.

Ecrire la fonction suivante d'affichage du plateau de jeu.

```
/** pré-requis : cod est une matrice, rep est une matrice à 2 colonnes,
        0 <= nbCoups < cod.length, nbCoups < rep.length et
        les éléments de cod sont des entiers de 0 à tabCouleurs.length -1
    action : affiche les nbCoups premières lignes de cod (sous forme
         de mots en utilisant le tableau tabCouleurs) et de rep
*/
public static void affichePlateau(int [][] cod, int [][] rep,
                                  int nbCoups, char[] tabCouleurs){
}
```

La fonction `affichePlateau` est appelée par les fonctions `mancheHumain` et `mancheOrdinateur` après chaque réponse (nombre de pions bien et mal placés) du codeur. Vous devrez donc déclarer et mettre à jour dans la fonction `mancheHumain` des matrices $cod$ et $rep$, comme vous l'avez déjà fait dans la fonction `mancheOrdinateur`.

## 3.2 Affichage des erreurs de réponse (1 pt)

Vous allez d'abord rajouter le fait que dans le cas où le décodeur n'a pas réussi à trouver le code secret en $nbEssaisMax$ essais ou qu'il a détecté qu'aucun code n'est compatible avec les réponses du codeur, le codeur lui dévoile son code secret (sous forme d'affichage si le codeur est l'ordinateur et sous forme de saisie si le codeur est le joueur humain).

Quand le joueur humain code et que l'ordinateur décode, il est possible que le joueur se trompe dans l'une ou plusieurs de ses réponses (le décompte des pions bien et mal placés pour les propositions de l'ordinateur).

Ecrire la fonction suivante d'affichage des différentes réponses erronées commises par le joueur humain.

```
/** pré-requis : cod est une matrice, rep est une matrice à 2 colonnes,
```

```
      0 <= nbCoups < cod.length, nbCoups < rep.length,
      les éléments de cod sont des entiers de 0 à tabCouleurs.length -1
      et codMot est incorrect ou incompatible avec les nbCoups
      premières lignes de cod et de rep
   action : affiche les erreurs d'incorrection ou d'incompatibilité
*/
public static void afficheErreurs (String codMot, int [][] cod,
        int [][] rep, int nbCoups, int lgCode, char[] tabCouleurs) {
 }
```

La fonction `afficheErreurs` est appelée par la fonction `mancheOrdinateur` quand l'IA a épuisé toutes les combinaisons de code à proposer sans avoir trouvé la solution (ce qui est en théorie impossible, sauf erreur dans les réponses). Elle est aussi appelée quand l'ordinateur n'a pas trouvé le code secret en $nbEssaisMax$ essais et que le code secret saisi par le joueur humain n'est pas compatible avec les propositions et réponses précédentes. Dans les 2 cas, le paramètre $codMot$ est le code secret saisi par le joueur humain, et celui-ci n'obtient aucun point pour cette manche.

## 3.3 Quelques statistiques pour évaluer et comparer les stratégies (1pt)

Pour évaluer la qualité de la stratégie de l'ordinateur de la version de base et la comparer éventuellement avec celle de l'extension 3.7, écrire une fonction `statsMasterMindIA` calculant :

- le nombre maximum de propositions de codes pour tous les codes secrets possibles (ainsi que les codes secrets réalisant ce maximum), et
- la moyenne des nombres de propositions de codes pour tous les codes secrets possibles.

## 3.4 Interface graphique (2 pts)

Vous pouvez faire un affichage plus réaliste du jeu de Master Mind avec des pions de couleur (voir `UTILE/graphismeUpdate.zip` sur Moodle).

## 3.5 Version objet (3 pts)

Comme son nom l'indique, cette version se fait en programmation objet. Vous devrez définir les classes suivantes : `UtMM`, `Couleur`, `Code`, `Plateau`, `MancheOrdinateur`, `MancheHumain`, `Partie` et `MainMasterMind`.

L'idée est de déplacer toutes les fonctions que vous avez écrites dans la version de base vers les classes ci-dessus. `UtMM` est une classe "utile pour le Master Mind" qui contient des méthodes statiques effectuant des traitements de base.

### Les classes `Code` et `Couleur`

Les codes étant surtout utilisés sous forme de tableaux d'entiers pour la stratégie de l'ordinateur, il vous est suggéré de les définir dans la classe `Code` comme des tableaux d'entiers plutôt que comme des tableaux d'objets `Couleur`.

Ainsi, la classe `Couleur` n'aurait pas d'instance, elle n'aurait qu'un seul attribut, $tabCouleurs$, qui serait un attribut de classe (*static*). Elle n'aurait donc aucun constructeur et aurait quelques méthodes de classe permettant d'initialiser $tabCouleurs$, de récupérer le nombre de couleurs et de convertir une couleur donnée sous forme d'entier en caractère et inversement.

La classe `Code` aurait comme attributs :

— un tableau d'entiers représentant un code (attribut d'instance) et
— l'attribut de classe $lgCode$ indiquant la longueur des codes.

### La classe `Plateau`

Pour une manche donnée, un objet `Plateau` stocke les différents codes placés sur le plateau par le décodeur ainsi que les réponses du codeur. On trouve ainsi dans la classe `Plateau` les attributs suivants :

```
private static int nbEssaisMax;  // >= 0
private Code[] cod;
private int[][] rep;
private int nbCoups;  // >= 0
```

### Les classes `MancheHumain` et `MancheOrdinateur`

Ces classes définissent un attribut : `private Plateau p` et comprennent une méthode `joue` effectuant une manche.

### La classe `Partie`

Un objet `Partie` orchestre, grâce à sa méthode `joue`, une partie de MasterMind, en alternant les manches où l'ordinateur et l'humain sont tour à tour codeurs et décodeurs. La classe `Partie` possède les attributs suivants :

```
private int nbManches;   // pair > 0
private int numManche;   // de 1 à nbManches
private int scoreJoueur; // >= 0
private int scoreOrdi;   // >= 0
```

#### La classe `MainMasterMind`

Cette classe contient le programme principal qui lance une partie. Elle effectue au préalable les saisies de tous les paramètres de la partie et initialise le nombre d'essais maximum dans la classe Plateau.

## 3.6 Contre-stratégies du joueur et de l'ordinateur (1 pt)

Pour contrer la stratégie de l'ordinateur, le joueur peut utiliser le résultat de l'extension 3.3 et choisir systématiquement un code secret pour lequel l'ordinateur devra proposer le nombre maximum de codes.

Pour contrer cette contre-stratégie du joueur, l'ordinateur peut ajouter de l'aléatoire à sa stratégie :

- choisir aléatoirement entre les ordres lexicographiques croissant et décroissant [1], ou
- choisir un code de départ aléatoire, puis suivre l'ordre lexicographique croissant à partir de ce code de départ (et revenir au premier code après le dernier), ou
- à vous de choisir une façon de mettre de l'aléatoire dans la stratégie de l'ordinateur !

## 3.7 Stratégie CFC de l'ordinateur (3 pts)

Dans cette extension, l'ordinateur utilise une autre stratégie, plus proche de l'intuition humaine et qui peut même être utilisée facilement par le joueur humain car cette stratégie n'utilise pas la compatibilité avec les propositions et réponses précédentes, ce qui pourrait la rendre moins performante que la stratégie de base. Ce sera à vous de tester (avec une évaluation plus précise par la fonction de l'extension 3.3).

Cette stratégie est appelée CFC (pour Couleur Fond Curseur [2]) car elle se compose de 2 phases :

1. la phase C où l'ordinateur détermine le nombre de pions de chaque couleur, et
2. la phase FC où il détermine leur position à l'aide d'une couleur de fond et d'une couleur de curseur.

La phase C est similaire à la première phase de la stratégie de base, sauf qu'il suffit d'ordonner les pions en ordre croissant de leurs numéros de couleurs puisqu'on ne tient pas compte de la compatibilité avec les différents essais.

Voici les propositions de code de la phase C pour notre exemple où le joueur a choisi le code secret VJBR :

Dans la phase FC, chaque proposition choisit seulement 2 couleurs parmi celles déterminées à l'étape C :

1. Le nombre maximum de propositions de codes est-il le même pour ces 2 ordres ? Y a-t-il des codes secrets pour lesquels le nombre de propositions est maximum pour ces 2 ordres ?

2. La page suivante ne vous aidera peut-être pas, mais vous donnera un peu de culture générale : https://fr.wikipedia.org/wiki/Chlorofluorocarbure.

| Code | BP | MP | Déduction |
|---|---|---|---|
| RRRR | 1 | 0 | code secret comprend 1 pion rouge |
| RBBB | 1 | 1 | code secret comprend 1 pion bleu |
| RBJJ | 0 | 3 | code secret comprend 1 pion jaune |
| RBJV | 0 | 4 | code secret comprend 1 pion vert |

— une couleur étant choisie pour un seul pion, appelé pion "curseur" ;
— l'autre couleur, appelée couleur de "fond", étant choisie pour tous les autres pions.

Pourquoi faire de telles propositions bicolores me direz-vous ? En fait, avec une telle proposition, le *nombre de pions mal placés* permet de déterminer si *la position du pion curseur* est occupée dans le code secret par :

— un pion de la couleur de fond,
— un pion de la couleur du curseur, ou
— ni l'un ni l'autre.

Ah oui, et comment me direz-vous ? A vous de réfléchir, mais les exemples suivants, notamment le deuxième, vont vous y aider !

Résumons la stratégie de la phase FC. Cette phase travaille avec le dernier code proposé à la phase C (RBJV pour l'exemple ci-dessus). Rappelons que ce code de référence contient tous les pions de couleur du code secret, mais dans un mauvais ordre. On va faire évoluer deux indices $iFond$ et $iCurs$ sur ce code de référence définissant respectivement les couleurs de fond et du curseur de chaque proposition. L'indice $iFond$ est initialisé à $0$ et l'indice $iCurs$ est initialisé à $lgCode - 1$. Ainsi, dans la première proposition, la couleur du premier pion du code de référence servira de couleur de fond et la couleur du dernier pion donnera la couleur du pion curseur. La boucle suivante va faire se rapprocher les deux indices l'un de l'autre, $iCurs$ étant incrémenté et $iFond$ étant décrémenté. La position du curseur va évoluer à chaque proposition et est initialisée à 0.

Après l'initialisation des indices et de la position du curseur, la phase FC effectue une boucle jusqu'à ce que les deux indices $iFond$ et $iCurs$ indiquent la même couleur, ce qui implique que tous les pions restant à positionner dans le code secret sont de cette couleur. A chaque itération/proposition :

— on choisit les couleurs de fond et du curseur à l'aide des indices $iFond$ et $iCurs$, comme expliqué ci-dessus ;
— on en déduit des informations sur la couleur de la position du curseur dans le code secret ;
— on incrémente la position du curseur (modulo $lgCode$), en sautant les positions où la couleur est déjà déterminée ;
— $iFond$ est incrémenté quand on a trouvé la position dans le code secret d'un pion de la couleur de fond ;
— $iCurs$ est décrémenté quand on a trouvé la position dans le code secret d'un pion de la couleur du curseur.

Voici les propositions de code de la phase FC pour l'exemple ci-dessus :

| Code | BP | MP | Déduction |
|---|---|---|---|
| VRRR | 2 | 0 | le $1^{er}$ pion est vert |
| RJRR | 2 | 0 | le $2^{\text{ème}}$ pion est jaune |
| RRBR | 2 | 0 | le $3^{\text{ème}}$ pion est bleu, et comme les couleurs de fond et du curseur sont toutes les 2 égales à rouge, le $4^{\text{ème}}$ pion est rouge |
| VJBR | 4 | 0 | c'est gagné ! |

Voici un deuxième exemple, où le joueur a choisi le code secret BBRV :

Phase C :

| Code | BP | MP | Déduction |
|---|---|---|---|
| RRRR | 1 | 0 | le code secret comprend 1 pion rouge |
| RBBB | 1 | 2 | le code secret comprend 2 pions bleus |
| RBBJ | 1 | 2 | le code secret ne comprend pas de pion jaune |
| RBBV | 2 | 2 | le code secret comprend 1 pion vert |

Phase FC :

| Code | BP | MP | Déduction |
|---|---|---|---|
| VRRR | 1 | 1 | le $1^{\text{er}}$ pion n'est ni rouge ni vert |
| RVRR | 1 | 1 | le $2^{\text{ème}}$ pion n'est ni rouge ni vert |
| RRVR | 0 | 2 | le $3^{\text{ème}}$ pion est rouge |
| BBBV | 3 | 0 | le $4^{\text{ème}}$ pion est vert, et comme les couleurs de fond et du curseur sont toutes les 2 égales à bleu, les $1^{\text{er}}$ et $2^{\text{ème}}$ pions sont bleus |
| BBRV | 4 | 0 | c'est gagné ! |

Il y a plusieurs façons d'améliorer cette stratégie. Vous pouvez bien sûr en proposer une version améliorée (basée sur les couleurs de fond et de curseur) à condition de l'expliquer dans les spécifications.