Rapport de projet mastermind

# Contexte

## Présentation

Ceci est mon 3eme projet dans le parcours « Développeur d’applications – JAVA » au sein d’OpenClassrooms. Ce rapport me sert à documenter mon travail ainsi qu’a en garder une trace. Il servira aussi de bas à ma présentation PowerPoint.

Le but de ce projet est de programmer une application proposant 2 jeux de logique : la recherche +/- et le mastermind.

## Objectifs de compétences

L’objectif en termes de compétences ciblé par ce projet est de **maîtriser les concepts de base de la programmation en Java.**

# Problématique

## Enoncé

### Pré-requis

Ce projet ne nécessite pas de pré-requis autres que ceux d'entrée dans le parcours de formation de Développeur d'Application Java JEE.

### Contexte

Vous voilà prêt à entrer dans le vif du sujet : programmer !

Dans ce premier projet, je vous propose de créer une application proposant des jeux de logique. Vous aurez à développer plusieurs variantes autour de la recherche de combinaisons secrètes :

* La recherche d'une combinaison à l'aide d'indications +/-
* Le célèbre Mastermind. (Si vous voulez, vous pouvez vous baser sur des chiffres au lieu des couleurs.)  
  Plus d'informations sur [*http://fr.wikipedia.org/wiki/Mastermind*](https://fr.wikipedia.org/wiki/Mastermind)

#### Recherche +/-

Le but : découvrir la combinaison à x chiffres de l'adversaire (le défenseur). Pour ce faire, l'attaquant fait une proposition.

Le défenseur indique pour chaque chiffre de la combinaison proposée si le chiffre de sa combinaison est plus grand (+), plus petit (-) ou si c'est le bon chiffre (=).

L'attaquant doit deviner la combinaison secrète en un nombre limité d'essais.

(Combinaison secrète : 1234)

Proposition : 4278 -> Réponse : -=--

Proposition : 2214 -> Réponse : -=+=

...

#### Mastermind

Le but : découvrir la combinaison à x chiffres/couleurs de l'adversaire (le défenseur). Pour ce faire, l'attaquant fait une proposition.

Le défenseur indique pour chaque proposition le nombre de chiffre/couleur de la proposition qui apparaissent à la bonne place et à la mauvaise place dans la combinaison secrète.

L'attaquant doit deviner la combinaison secrète en un nombre limité d'essais.

(Combinaison secrète : 1234)

Proposition : 4278 -> Réponse : 1 présent, 1 bien placé

Proposition : 6274 -> Réponse : 2 bien placés

...

### Travail demandé

Vous devez développer une application en Java, soit en mode console soit avec Swing, permettant de lancer les différents jeux.

Chaque jeu pourra être joué selon 3 modes :

1. Mode challenger où vous devez trouver la combinaison secrète de l'ordinateur
2. Mode défenseur où c'est à l'ordinateur de trouver votre combinaison secrète
3. Mode duel où l'ordinateur et vous jouez tour à tour, le premier à trouver la combinaison secrète de l'autre a gagné

Voici le fonctionnement normal attendu dans l'application :

1. Au démarrage, l'utilisateur doit choisir le jeu auquel il veut jouer parmi les choix proposés.
2. L'utilisateur sélectionne le jeu et le mode de son choix. L'application lance le jeu sélectionné.
3. L'utilisateur joue. S'il perd, l'application affiche la solution.
4. À la fin de la partie, l'utilisateur peut choisir :
   1. de rejouer au même jeu
   2. de lancer un autre jeu (retour à l'écran de choix des jeux du début)
   3. de quitter l'application

Il doit être possible de lancer l'application dans un mode "développeur". Dans ce mode la solution est affichée dès le début. Cela permet de tester le bon comportement de l'application en cas de bonne ou de mauvaise réponse de l'utilisateur. Ceci est à réaliser avec les mécanismes suivants :

* Passage d'un paramètre au lancement de l'application
* Propriété spécifique dans le fichier de configuration

Un fichier de configuration (config.properties) permettra de paramétrer l'application, notamment :

* Pour chaque jeu :
  + le nombre de cases de la combinaison secrète
  + le nombre d'essais possibles
* Pour le Mastermind :
  + le nombre couleur/chiffre utilisables (de 4 à 10)

Un fichier de configuration (log4j.xml) permettra de paramétrer les logs de l'application. La gestion des logs se fera avec Apache Log4j.

### Livrables attendus

Vous livrerez, sur [GitHub](https://github.com/) ou [GitLab](https://gitlab.com/) (dans un seul dépôt Git) :

* le code source de l’application
* le fichier de configuration
* une documentation succincte (un fichier README.md suffit) expliquant comment compiler et lancer l'application

Soignez particulièrement le code source de votre application : indentation, formatage, nom des variables, commentaires...

Le projet est validé tant sur le fond (application fonctionnelle) que sur la forme (organisation technique, présentation du code...).

Vous donnerez un accès en lecture à votre dépôt Git à votre mentor et au mentor qui vous fera passer la soutenance

### Soutenance

#### Avant la soutenance

Quelques jours avant la date de la soutenance (en général 3-4 jours avant), vous transmettrez tous vos livrables au mentor qui vous fera passer celle-ci, afin qu’il puisse en prendre connaissance en amont et éventuellement tester votre application.

#### Déroulement de la soutenance

La soutenance, d’une durée de 20-25 minutes, se déroulera en deux parties :

* Partie 1 – 15 minutes : **Présentation de l'application**
  + [~ 5 minutes] Vous ferez une démonstration rapide de votre application
  + [~ 10 minutes] Vous présenterez votre solution technique et les points particuliers de votre développement (à vous d'être pertinent sur les points à aborder)
* Partie 2 – 5-10 minutes : **Retour sur la soutenance**
  + Le mentor pourra vous demander d’approfondir certains aspects ou vous questionner sur vos livrables
  + Le mentor vous fera un debrief sur votre prestation en soutenance

### Étapes de réalisation proposées

1. Configurez votre EDI (Eclipse/IntelliJ/...) :
   1. Créez un nouveau projet Java (JDK 8).
   2. Ajoutez un répertoire src/main/java et déclarez-le en tant que source du projet.  
      Vous y déposerez vos fichier sources Java (\*.java).
   3. Ajoutez un répertoire src\main\resources et déclarez-le en tant que source du projet.  
      Vous y déposerez les autres resources de votre application (config.properties, log4j.xml, images...).
   4. Ajoutez un répertoire lib à la racine de votre projet et déclarez-le en tant que conteneur de bibliothèques. Vous y déposerez les bibliothèques (libraries) tierces nécessaires au bon fonctionnement de votre application (Apache Commons Lang...).
2. Créez le socle de votre application :
   1. Créez la classe  Main
   2. Créez les classes de base (classe abstraites, classes mères des classes principales représentant un jeu...)
3. Implémentez le premier jeu.
4. Prenez en compte la configuration du jeu :
   1. Créez le fichier de configuration (config.properties)
   2. Charger le fichier de configuration dans l'application
5. Implémentez les autres jeux
6. Rédigez la documentation de compilation et d'utilisation (fichier README.md).

# Sommaire

[Contexte 2](#_Toc504736895)

[Présentation 2](#_Toc504736896)

[Objectifs de compétences 2](#_Toc504736897)

[Problématique 2](#_Toc504736898)

[Enoncé 2](#_Toc504736899)

[Pré-requis 2](#_Toc504736900)

[Contexte 2](#_Toc504736901)

[Recherche +/- 3](#_Toc504736902)

[Mastermind 3](#_Toc504736903)

[Travail demandé 4](#_Toc504736904)

[Livrables attendus 5](#_Toc504736905)

[Soutenance 5](#_Toc504736906)

[Avant la soutenance 5](#_Toc504736907)

[Déroulement de la soutenance 5](#_Toc504736908)

[Étapes de réalisation proposées 6](#_Toc504736909)

[Sommaire 7](#_Toc504736910)

[Partie 1 10](#_Toc504736911)

[Vue d’ensemble 10](#_Toc504736912)

[Les packages 10](#_Toc504736913)

[Les classes 11](#_Toc504736914)

[12](#_Toc504736915)

[Prévu / Réel 13](#_Toc504736916)

[13](#_Toc504736917)

[Partie 2 14](#_Toc504736918)

[Packages & classes – Hiérarchie type 14](#_Toc504736919)

[JeuDeLogique 15](#_Toc504736920)

[JeuDeLogique 15](#_Toc504736921)

[Attributs 15](#_Toc504736922)

[Methodes 16](#_Toc504736923)

[RecherchePlusMoins 17](#_Toc504736924)

[Fonctionnement 17](#_Toc504736925)

[Attributs 17](#_Toc504736926)

[Méthodes 17](#_Toc504736927)

[Mastermind 18](#_Toc504736928)

[Fonctionnement 18](#_Toc504736929)

[Attributs 19](#_Toc504736930)

[Methodes 20](#_Toc504736931)

[Utilisateur 28](#_Toc504736932)

[Utilisateur 28](#_Toc504736933)

[Fonctionnement 28](#_Toc504736934)

[Attributs 28](#_Toc504736935)

[Méthodes 29](#_Toc504736936)

[Joueur 29](#_Toc504736937)

[Attributs 29](#_Toc504736938)

[Méthodes 29](#_Toc504736939)

[Ordi 30](#_Toc504736940)

[Attributs 30](#_Toc504736941)

[Méthodes 30](#_Toc504736942)

[Tables 30](#_Toc504736943)

[Tables 30](#_Toc504736944)

[Attributs 30](#_Toc504736945)

[Methodes 31](#_Toc504736946)

[ColonneTerminee 31](#_Toc504736947)

[Attributs 31](#_Toc504736948)

[Methodes 31](#_Toc504736949)

[MasterTable 31](#_Toc504736950)

[Attributs 31](#_Toc504736951)

[Methodes 31](#_Toc504736952)

[IndiceTab 31](#_Toc504736953)

[Attributs 31](#_Toc504736954)

[Methodes 32](#_Toc504736955)

[Propo 32](#_Toc504736956)

[Attributs 32](#_Toc504736957)

[Methodes 32](#_Toc504736958)

[Config 33](#_Toc504736959)

[ConfigG 33](#_Toc504736960)

[Fonctionnement 33](#_Toc504736961)

[Attributs 33](#_Toc504736962)

[Méthodes 33](#_Toc504736963)

[ConfigMMD 34](#_Toc504736964)

[Attributs 34](#_Toc504736965)

[Méthodes 34](#_Toc504736966)

[ConfigRPM 34](#_Toc504736967)

[Fonctionnement 34](#_Toc504736968)

[Attributs 34](#_Toc504736969)

[Méthodes 35](#_Toc504736970)

[ConfigurationG 35](#_Toc504736971)

[Attributs 35](#_Toc504736972)

[ConfigurationMMD 35](#_Toc504736973)

[Attributs 35](#_Toc504736974)

[ConfigurationRPM 35](#_Toc504736975)

[Attributs 36](#_Toc504736976)

[Paramètres 36](#_Toc504736977)

[ParametresG 36](#_Toc504736978)

[Attributs 36](#_Toc504736979)

[Méthodes 36](#_Toc504736980)

[ParametresMMD 36](#_Toc504736981)

[Attributs 36](#_Toc504736982)

[Méthodes 36](#_Toc504736983)

[ParametresRPM 37](#_Toc504736984)

[Attributs 37](#_Toc504736985)

[Méthodes 37](#_Toc504736986)

[Bilan 38](#_Toc504736987)

[Compétences 38](#_Toc504736988)

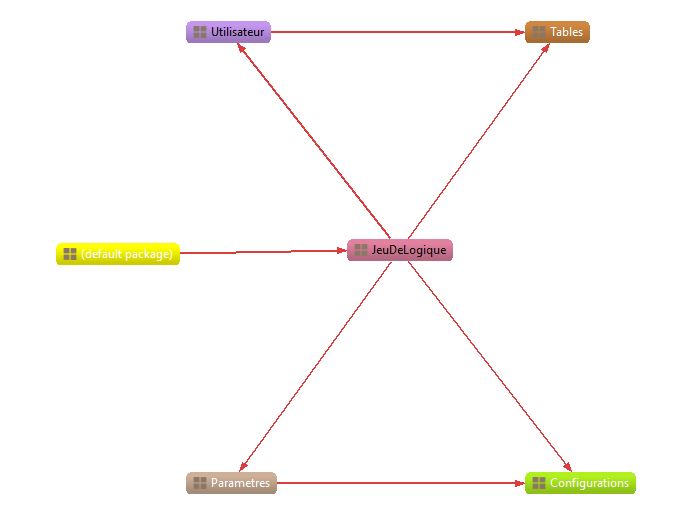
[Pour aller plus loin 38](#_Toc504736989)

[Impact projet professionnel 38](#_Toc504736990)

# Partie 1

## Vue d’ensemble

### Les packages



L’application contient 5 packages ainsi que celui par défaut :

- JeuDeLogique **(en rose)** contient les classes décrivant le fonctionnement des 2 jeux

- Utilisateur **(en violet)** contient les classes décrivant les différents utilisateurs : joueur et ordi

- Tables **(en marron)** contient différentes classes de tables nécessaires au fonctionnement de l’ « IA » de l’ordinateur pour trouver une solution au Mastermind.

- Configurations **(en vert)** contient les classes permettant d’interagir avec le fichier de configuration (config.properties) de l’application

- Paramètres **(en gris)** contient les classes décrivant le menu des paramètres depuis lequel on va pouvoir modifier les champs du fichier properties susnommé (nombre de cases, nombre d’essais, …)

### Les classes

## 

## Prévu / Réel

Graphiques représentant le temps prévu à chaque tache ainsi que le temps réel investi sur une durée totale de 3 mois.

## 

# Partie 2

## Packages & classes – Hiérarchie type

Classe Mère

Variable 1

Variable 2

Classe Fille 1

Variable 3

Variable 4

Classe Fille 2

Variable 5

Variable 6

PACKAGE 1

Les attributs et méthodes communes aux classes Filles sont placées dans la classe Mère

## JeuDeLogique

JeuDeLogique

RecherchePlusMoins

Mastermind

JeuDeLogique

### JeuDeLogique

#### Attributs

Integer rejouer :

variable qui stocke le choix du joueur après une fin de partie

Integer trouve :

permet de compter les valeurs exactes

Integer mode :

variable qui stocke la valeur correspondant au mode de jeu

Boolean gagneOrdi :

true si l’ordi a la bonne réponse

Boolean gagneJoueur :

true si le joueur a la bonne réponse

Ordi ordi :

objet ordi

Joueur joueur :

objet joueur

ConfigurationMMD configMMD :

permet de charger les paramètres de jeu

ConfigG configG :

permet de charger le paramètre « modeDéveloppeur »

#### Methodes

afficheTab(Integer[] tab)

affiche la table passée en paramètres dans la console

analyseTrouve(String[] tab)

analyse si le résultat est entièrement correct (tab[\*] == « = »)

choisirMode(Integer c)

Permet de choisir le mode de jeu (Challenger, Défenseur, Duel)

c : jeu -> 0 = RPM, 1 = MMD

compare(Integer a, Integer b)

compare a par rapport à b (si a>b -> ‘+’, si a<b -> ‘-‘ …)

compareTab(Integer[] tab1, Integer[] tab2, Integer a)

compare les valeurs de tab1 par rapport à celles de tab2 et compte le nombre de valeurs égales

a = 0 -> pas d’affichage

a = 1 -> affichage de la comparaison

reponse()

affiche la réponse si le mode développeur est activé

### RecherchePlusMoins

#### Fonctionnement

Se joue à 2 joueurs, l’un doit créer une combinaison secrète, l’autre doit la trouver.

A chaque tentative, celui qui cherche obtient un retour du programme lui indiquant le positionnement de chacun des chiffres à trouver par rapport aux siens.

*Exemple :*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **combinaison** | **0** | **7** | **5** | **6** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **5** | **5** | **5** | **5** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **indice** | **-** | **+** | **=** | **+** |

Si la proposition n’est pas égale à la combinaison, le joueur perd 1 essai, sinon il gagne.

#### Attributs

ConfigurationRPM configRPM :

Permet de charger les paramètres du Recherche+/-

Integer NB\_CASES\_COMBI

Paramètre « taille de la combinaison » (dans l’exemple ci-dessus : 4)

Integer NB\_ESSAIS

Paramètre « nombre d’essais »

#### Méthodes

challengerMode()

Démarre le mode challengeur : le joueur doit trouver la combinaison de l’ordinateur

defenseurMode()

Démarre le mode défenseur : l’ordinateur doit trouver la combinaison du joueur

duelMode()

Démarre le mode duel : le joueur et l’ordinateur jouent l’un contre l’autre

reponse(Utilisateur u, int a)

affiche la réponse de u

a  = 1 : fait appel à sa méthode mère

a = 0 : rien

tourDeLOrdi(Ordi o, Joueur j, Integer a)

o cherche la combinaison de j, si elle n’est pas bonne, j perd 1 essai

a = 1 : affiche la comparaison

a = 0 n’affiche pas la comparaison

Fonctionnement de la recherche

Pour chaque case de la proposition, l’ordinateur propose un nombre aléatoire compris entre des limites.

Au début, ces limites sont **[0 ; 10]** et au fur et à mesure que le programme lui donne des indices, l’ordinateur **adapte** **ses limites** de propositions.

Pour la 1ere case de l’exemple ci-dessus (proposition : 5, indice : ‘-‘) ses limites changeront : [0 ; 5].

Le tour suivant, il ne pourra proposer qu’un nombre compris entre 0 et 5, et ainsi de suite.

À noter que la génération du nombre est **aléatoire** mais une génération non aléatoire serait tout autant possible.

tourDuJoueur(Ordi o, Joueur j)

j cherche la combinaison de o, si elle n’est pas bonne, j perd 1 essai

### Mastermind

#### Fonctionnement

Se joue à 2 joueurs, l’un doit créer une combinaison secrète, l’autre doit la trouver.

A chaque tentative, celui qui cherche obtient un retour du programme lui indiquant si des chiffres sont bien placés 0 ou mal placés 0.

*Exemple :*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **combinaison** | **0** | **7** | **5** | **6** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **5** | **5** | **5** | **5** |

Bien placés : 1

Mal placés : 0

#### Attributs

ConfigurationMMD configMMD :

Permet de charger les paramètres du Mastermind

Integer NB\_CASES\_COMBI :

Paramètre « taille de la combinaison » (dans l’exemple ci-dessus : 4)

Integer NB\_ESSAIS

Paramètre « nombre d’essais »

Integer NB\_COULEURS

Paramètre « nombre de couleurs » indiquant le nombre de couleurs (ou de chiffres) disponibles pour jouer

Int BPj :

Permet de stocker le nombre de chiffres bien placés (rond noir) du joueur

Int MPj :

Permet de stocker le nombre de chiffres mal placés (rond blanc) du joueur

Int BPo :

Permet de stocker le nombre de chiffres bien placés (rond noir) de l’ordi

Int MPo :

Permet de stocker le nombre de chiffres mal placés (rond blanc) de l’ordi

Logger logger :

Permet d’écrire dans le fichier de logs

IndiceTab :

Table permettant de répertorier les nombre connus de la combinaison

ColonneTerminee :

Table permettant d’indiquer les colonnes qui sont « terminees » : chiffre de la combinaison et son indice dans la combinaison trouvés.

MasterTable :

Table permettant de stocker les chiffres connus de la combinaison et leurs indices

Integer[] jat :

Table (1x2) permettant de stocker :

- la valeur d’un chiffre de la combinaison déjà trouvée

- l’indice possiblement associé et à tester

#### Methodes

bienMalPlace(Integer[] tab1, Integer[] tab2)

compare tab2 à tab1 et incrémente BP et MP en fonction de cette comparaison

combiTrouvee()

renvoie true si la table ‘colonneTerminee’ est remplie de true ET si la 1ere ligne de la masterTable est complète

compteCombien(int i, int[] tab)

compte combien de fois i apparaît dans tab

indiceBon()

renvoie true si tous les indices ont étés trouvés (si la masterTable est complète)

initBPMP()

Initialise BPj, BPo, MPj, MPo à 0

initTables()

fait appel aux fonctions d’initialisation des tables indiceTab, colonneTerminee, et masterTable

MAJTables()

Fait appel aux fonctions de mise à jour des tables indiceTab, colonneTerminee et masterTable

propoChercheY()

Fonction indiquant la marche à suivre si la proposition précédente était de la forme « propoChercheY »

propoX()

Fonction indiquant la marche à suivre si la proposition précédente était de la forme « propoX»

propoXY()

Fonction indiquant la marche à suivre si la proposition précédente était de la forme « propoXY»

reponse(Utilisateur u, int a)

affiche la réponse de u

a  = 1 : fait appel à sa méthode mère

a = 0 : rien

tourDeLOrdi()

L’ordinateur cherche la combinaison

Fonctionnement de la recherche

Afin de trouver la combinaison, l’ordinateur possède 3 tables à remplir, un peu comme un sudoku.

|  |  |  |  |
| --- | --- | --- | --- |
| **MasterTable** | | | |
| Y1 | Y2 | Y3 | Y4 |
|  |  |  |  |
| iY1 |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 1 |
| 3 | 2 |
| 0 | 3 |
| 0 | 4 |
| 1 | 5 |
| 0 | 6 |
| 0 | 7 |
| 0 | 8 |
| 0 | 9 |
| **IndiceTab** | |

|  |  |  |  |
| --- | --- | --- | --- |
| **V** | **F** | **F** | **F** |
| **ColonneTerminee** | | | |

La 1ere ligne de la **MasterTable** contiendra les chiffres qui composent la combinaison (appelés Y dans la suite).

Les autres lignes de la **MasterTable** contiendront les indices des chiffres de la 1ere ligne.

Ils indiqueront la place de chaque chiffre de la 1ere ligne dans la combinaison, ainsi, il ne peut pas y avoir plusieurs chiffres sur la même ligne ou la même colonne.

L’**IndiceTab** répertoriera les chiffres connus de la combinaison et rempliera donc la 1ere ligne de la **MasterTable**.

Sa taille est définie par le « nombre de couleurs » (ici 10).

Enfin **ColonneTerminee** se verra attribuer une valeur « vrai ou faux » (V / F) en fonction de l’état de la colonne de la **MasterTable** correspondant.

Si jamais la 1ere case de la 1ere ligne de la **MasterTable** contient un chiffre et que la 1ere colonne contient l’indice correspondant, la valeur sera V.

*Exemple :*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **combinaison** | **1** | **5** | **3** | **2** |

Le principe des propositions faites par l’ordi est de tester soit 1 chiffre soit 2 mais pas plus, ce qui abouti à 2 formes de proposition :

- La propositionX a pour but de déterminer si X fait parti de la combinaison et si oui, combien de fois

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **X** | **X** | **X** | **X** |

- la propositionXY qui a pour but de déterminer à la fois le nombre de X et la position de Y (il faut connaître Y on ne peut donc commencer que par une propoX)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **X** | **X** | **Y** | **X** |

*Tour 1*

La 1ere proposition de l’ordinateur est toujours :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **0** | **0** | **0** | **0** |

Bien placés : 0

Mal placés : 0

Le seul tableau mis à jour est IndiceTab : on lui indique qu’il y a zéro fois 0 dans la combinaison.

|  |  |
| --- | --- |
| 0 | 0 |

*Tour 2*

On ne connaît pas de Y, on ne peut donc faire qu’une propositionX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **1** | **1** | **1** | **1** |

Bien placés : 1

Mal placés : 0

On a un bien placé et on sait que c’est un 1 mais on ne connaît pas son emplacement dans la

combi, on peut donc mettre à jour nos tables

|  |  |  |  |
| --- | --- | --- | --- |
| **MasterTable** | | | |
| 1 |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 1 |

*Tour 3*

On va maintenant tester l’appartenance de 2 (X) et la position de 1 (Y) (On teste les positions dans un ordre croissant)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **1** | **2** | **2** | **2** |

Bien placés : 2

Mal placé : 0

Bingo ! on sait que le 1 est bien placé et qu’en plus il y a un 2, on peut mettre à jour nos tables :

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 1 |
| 1 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **MasterTable** | | | |
| 1 | 2 |  |  |
| 1 | X | X | X |
| X |  |  |  |
| X |  |  |  |
| X |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| V | F | F | F |
| **ColonneTerminee** | | | |

On indique l’indice de 1, on raye la ligne et la colonne associée et on met à jour **ColonneTerminee.**

*Tour 4*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **3** | **2** | **3** | **3** |

Bien placé : 1

Mal placé : 1

On sait maintenant qu’il y a un 3 dans la combi et que le 2 n’est pas placé en 2eme position (on sait que le mal placé correspond au 2 car si il correspondait au 3 cela voudrait dire qu’il se trouve à la place du 2, il y aurait donc 0 bien placé et 2 mal placés)

|  |  |  |  |
| --- | --- | --- | --- |
| **MasterTable** | | | |
| 1 | 2 | 3 |  |
| 1 | X | X | X |
| X | X |  |  |
| X |  |  |  |
| X |  |  |  |

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |

*Tour 5*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **4** | **4** | **2** | **4** |

Bien placé : 0

Mal placé : 1

Par élimination, on sait que le 2 est placé en dernière position.

|  |  |  |  |
| --- | --- | --- | --- |
| **MasterTable** | | | |
| 1 | 2 | 3 |  |
| 1 | X | X | X |
| X | X |  |  |
| X | X |  |  |
| X | 4 | X | X |

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 0 | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| V | V | F | F |
| **ColonneTerminee** | | | |

*Tour 6*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **5** | **3** | **5** | **5** |

Bien placé : 0

Mal placés : 2

Le fait qu’il y ait 2 mal placés nous indique que le dernier chiffre à trouver est un 5 et qu’il se trouve à la place du 3.

On peut donc compléter l’**IndiceTab**.

On peut aussi rayer les lignes et colonnes de l’indice du 5.

Enfin il ne nous reste qu’une case pour l’indice du 3.

On finit par compléter **ColonneTerminee.**

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 0 | 4 |
| 1 | 5 |
| 0 | 6 |
| 0 | 7 |
| 0 | 8 |
| 0 | 9 |

|  |  |  |  |
| --- | --- | --- | --- |
| **MasterTable** | | | |
| 1 | 2 | 3 | 5 |
| 1 | X | X | X |
| X | X | X | 2 |
| X | X | 3 | X |
| X | 4 | X | X |

|  |  |  |  |
| --- | --- | --- | --- |
| V | V | V | V |
| **ColonneTerminee** | | | |

*Tour 7*

Enfin on peut proposer la solution via une propositionFinale qui est construite en fonction de la MasterTable : on place le Y de l’indice 1 en premier puis le Y de l’indice 2 etc.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **proposition** | **1** | **5** | **3** | **2** |

tourDuJoueur()

Le joueur cherche la combinaison

challengerMode()

Démarre le mode challengeur : le joueur doit trouver la combinaison de l’ordinateur

defenseurMode()

Démarre le mode défenseur : l’ordinateur doit trouver la combinaison du joueur

duelMode()

Démarre le mode duel : le joueur et l’ordinateur jouent l’un contre l’autre

## Utilisateur

### Utilisateur

#### Fonctionnement

Classe mère des Utilisateurs permettant de capitaliser les attributs/méthodes communes

#### Attributs

Boolean[] choixSur :

(Recherche+/-)

Table qui stocke pour chaque nombre proposé :

- true si il est bon (=)

- false sinon (+/-)

Integer[] combiTab :

Table qui stocke la combinaison à trouver

String[] compraisonTab :

Table qui stocke les indices renvoyés par l’ordinateur(+,-,=)

Integer essais :

Variable qui stocke le nombre d’essais de l’utilisateur

Integer[] maxiBorneTab :

Table qui stocke les valeurs des limites **max** de chaque chiffre

Integer[] miniBorneTab :

Table qui stocke les valeurs des limites **min** de chaque chiffre

String[] propositionString :

Table qui stocke la valeur de la proposition en String (utile quand le joueur entre sa proposition/combinaison)

Integer[] propositionTab :

(Recherche+/-)

Table qui stocke une proposition

Integer[] propoTab :

(Mastermind)

Table qui stocke une proposition

#### Méthodes

cherche(Integer x)

Demande une proposition de réponse de taille x

Combi(Integer x)

Demande la création d’une combinaison de taille x

DecoupageAleatoire(Integer x)

Renvoie une table de x chiffres aléatoires

DecoupageAleatoire(Integer x, Integer[] mini, Integer[] maxi)

Renvoie une table de x chiffres aléatoires (mais aléatoires dans les limites [mini, maxi])

Initialisation(Integer i)

Initialise miniBorneTab, maxiBorneTab, choixSur, combiTab, propositionTab à la taille i

initMiniMaxiChoixSur(Integer j)

initialise miniBorneTab[\*] à 0, maxiBorneTab[\*] à 9 et choixSur[\*] à false

nombreAleatoire(Integer min, Integer max, Random hasard)

Renvoie un nombre aléatoire compris entre min et max

### Joueur

#### Attributs

Integer tailleCombi :

Variable qui stocke le paramètre « NB\_CASES\_COMBI »

Logger logger :

Permettant d’écrire dans le fichier de logs

String entreeUtil :

Permettant de stocker l’entrée de l’utilisateur au clavier

Boolean entreeCorrecte :

Permettant de vérifier si l’entrée de l’utilisateur est conforme aux attentes

#### Méthodes

Cherche(Integer x)

Demande une proposition de réponse de taille x

Combi(Integer x)

Demande la création d’une combinaison de taille x

isNumeric(String str)

Renvoie true si str est uniquement composé de nombre/chiffres

testLongueur(String str, Intger taille)

Teste si la longueur de str est égale à taille, renvoie true si c’est le cas, false sinon

### Ordi

#### Attributs

Logger logger :

Permet d’écrire dans le fichier de logs

Integer tailleCombi :

Variable qui stocke le paramètre « NB\_CASES\_COMBI »

#### Méthodes

Analyse()

(Recherche+/-)

Définit les nouvelles bornes dans lesquelles chercher la combinaison à trouver

Cherche()

Demande de proposition de réponse aléatoire mais bornée

Combi(Integer x)

Demande la création d’une combinaison de taille x

Init(Integer x)

Initialise l’objet ordi de taille x

## Tables

### Tables

#### Attributs

Integer taille :

Taille de la table

T[] t :

Table de type T

#### Methodes

Affiche()

Affiche la table

cherchePremier(Integer a)

Renvoie l’indice du premier a dans tab

cherchePremierNull()

Renvoie le premier null (-2) dans tab

compteCombien(T x, T[] tab)

Renvoie le nombre de x dans tab

indexOf(T nbr, T[] tab)

Renvoie le ou les indice(s) de nbr dans tab

### ColonneTerminee

#### Attributs

Logger logger :

Permet d’écrire dans le fichier de logs

#### Methodes

afficheCT()

Affiche la table colonneTerminee

### MasterTable

#### Attributs

Logger logger :

Permet d’écrire dans le fichier de logs

#### Methodes

Init()

Initialise les valeurs de CT à false

Terminee()

Renvoie true si toutes les cases sont « true », false sinon

### IndiceTab

#### Attributs

Logger logger :

Permet d’écrire dans le fichier de logs

Integer premierNullIT :

Permet de stocker le premier null(-2) de la table IndiceTab

Integer total :

Variable qui stocke le paramètre « NB\_CASES\_COMBI »

#### Methodes

afficheIT()

Affiche l’IndiceTab

Init()

Initialise l’indiceTab à null(-2) partout

majIT()

complète IT si possible

### Propo

#### Attributs

Logger logger :

Permet d’écrire dans le fichier de logs

structurePropo :

Permet de stocker la valeur correspondante à une structure de proposition :

- 1 : propoX

- 2 : propoXY

- 3 : propoXYZ

Integer[][] XouXY :

Décrit la composition de propoXY

Permet de stocker :

[0][0] : valeur de X , [0][1] : nombre de X

[1][0] : valeur de Y , [1][1] : indice de Y

#### Methodes

affichePropo()

Affiche la propo

indexOf(Integer x)

Renvoie l’indice du premier x de propo

Init()

Initialise l’objet propo à -2 partout

Init(Integer x)

Initialise l’objet propo à x partout

propoChercheY(Integer[][] masterTable, Tables it)

Revoie la proposition finale en testant la valeur du dernier Y à trouver

propoFinale(Integer[][] masterTable)

Renvoie la proposition finale via la masterTable

propoX(Integer x)

Renvoie une table remplie de x

propoXouXY()

Analyse la proposition en cours et renvoie le tableau XouXY associé

propoXY(Integer x, Integer y, Integer iy, Integer[][] masterTable)

Renvoie un tableau rempli de X et un Y à la position iY

## Config

### ConfigG

#### Fonctionnement

Classe mère des Config qui permet de lire/écrire le paramètre « modeDeveloppeur » dans le fichier config.properties et le stocker/récupérer dans ConfigurationG.modeDeveloppeur .

#### Attributs

String CHEMIN\_FICHIER :

Chemin du fichier de configuration

String MODE\_DEVELOPPEUR :

Adresse de la valeur du mode développeur dans le fichier de configuration

Properties prop :

Permet de communiquer avec le fichier de configuration

#### Méthodes

loadConfigG()

Récupère la valeur « modeDeveloppeur » du fichier config.properties et la met dans la variable modeDeveloppeur de ConfigurationG

saveConfigG(ConfigurationG configG)

Ecrit la valeur « modeDeveloppeur » dans le fichier config.properties

### ConfigMMD

Classe fille de ConfigG spécialisée dans la lecture/écriture des paramètres du Mastermind

#### Attributs

String CHEMIN\_FICHIER :

Chemin du fichier de configuration

String ESSAIS :

Adresse de la valeur du nombre d’essais (Mastermind) dans le fichier de configuration

String NB\_CASES\_COMBI :

Adresse de la valeur du nombre de cases de la combinaison (Mastermind) dans le fichier de configuration

String NB\_COULEURS :

Adresse de la valeur du nombre de couleurs (Mastermind) dans le fichier de configuration

#### Méthodes

loadConfigMMD()

Récupère les valeurs « nombre d’essais », « nombre de cases de la combinaison », « nombre de couleurs », et « mode développeur » relatives au mastermind et les stocke dans l’objet ConfigurationMMD

saveConfigMMD(ConfigurationMMD configMMD)

Ecrit les valeurs « nombre d’essais », « nombre de cases de la combinaison », « nombre de couleurs », et « mode développeur » relatives au mastermind

### ConfigRPM

#### Fonctionnement

Classe fille de ConfigG spécialisée dans la lecture/écriture des paramètres du Recherche+/-

#### Attributs

String CHEMIN\_FICHIER :

Chemin du fichier de configuration

String ESSAIS :

Adresse de la valeur du nombre d’essais (RPM) dans le fichier de configuration

String NB\_CASES\_COMBI :

Adresse de la valeur du nombre de cases de la combinaison (RPM) dans le fichier de configuration

#### Méthodes

loadConfigRPM()

Récupère les valeurs « nombre d’essais », « nombre de cases de la combinaison », et « mode développeur » relatives au Recherche+/- et les stocke dans l’objet ConfigurationRPM

saveConfigRPM(ConfigurationRPM configRPM)

Ecrit les valeurs « nombre d’essais », « nombre de cases de la combinaison », et « mode développeur » relatives au Recherche+/-

### ConfigurationG

Objet permettant de stocker la valeur du paramètre « modeDeveloppeur » du fichier de config

#### Attributs

Boolean modeDeveloppeur

True : mode développeur activé

False : mode développeur désactivé

### ConfigurationMMD

Objet permettant de stocker la valeur des paramètres « nbrCouleurs », « nbrCasesCombi » et « nbrEssais » du fichier de config

#### Attributs

Integer nbrCouleursMMD :

Permet de stocker la valeur du nombre de couleurs utilisables (de 4 à 10)

Integer nbrEssaisMMD :

Permet de stocker la valeur du nombre d’essais

Integer nbrCasesCombiMMD :

Permet de stocker la valeur de la taille de la combinaison

### ConfigurationRPM

Objet permettant de stocker la valeur des paramètres « nbrCasesCombi » et « nbrEssais » du fichier de config

#### Attributs

Integer nbrEssaisRPM :

Permet de stocker la valeur du nombre d’essais

Integer nbrCasesCombiRPM :

Permet de stocker la valeur de la taille de la combinaison

## Paramètres

### ParametresG

Classe mère des Parametres qui propose un menu permettant de modifier les paramètres du fichier de config

#### Attributs

ConfigurationG configG :

Objet qui fait le lien avec le fichier de config

#### Méthodes

affichageMenu()

Affiche le menu proposant de changer les paramètres :

- Taille de la combinaison

- Nombre d’essais

- Nombre de couleurs utilisables

- Mode développeur

changeModeDeveloppeur()

Permet d’activer/désactiver le mode développeur

### ParametresMMD

Classe fille de ParametresG qui fait appel aux fonctions relatives aux modifications des paramètres du Mastermind

#### Attributs

ConfigurationMMD configMMD :

Objet qui fait le lien avec le fichier de config

#### Méthodes

menuMMD()

Fait appel aux différentes méthodes en fonction de choix de l’utilisateur

afficheConfig()

Affiche la configuration courante

changeNombreCombi()

Permet de changer la taille de la combinaison

changeNombreEssais()

Permet de changer le nombre d’essais

changeNombreCouleurs()

Permet de changer le nombre de couleurs disponibles

### ParametresRPM

Classe fille de ParametresG qui fait appel aux fonctions relatives aux modifications des paramètres du Recherche+/-

#### Attributs

ConfigurationRPM configRPM :

Objet qui fait le lien avec le fichier de config

#### Méthodes

menuRPM()

Fait appel aux différentes méthodes en fonction de choix de l’utilisateur

afficheConfig()

Affiche la configuration courante

changeNombreCombi()

Permet de changer la taille de la combinaison

changeNombreEssais()

Permet de changer le nombre d’essais

# Bilan

## Compétences

Ce projet m’a permis d’apprendre de nouvelles facettes du développement Java, malgré le fait que tous les éléments que j’ai découvert grâce à mon mentor ne soient pas inclus dans ce projet, j’ai pu enrichir mon panel de développeur et ma « boîte à outils » Java.

Cela m’a aussi permis de me rendre compte que je n’avais pas une bonne méthode de développement de projet. Avec du recul, je m’aperçois que j’ai perdu du temps car je n’avais pas bien posé les bases et les limites, ce qui a abouti à des divergences non nécessaires durant mon travail.

Je pense que le fait de réaliser cela va me permettre de mieux m’organiser et d’être plus efficace pour les projets à venir.

## Pour aller plus loin

Au fur et à mesure de l’avancement dans le projet, j’ai eu différentes idées qui pourraient améliorer l’application.

Pour commencer, le plus évident est de faire une interface graphique (swing) afin de donner une meilleure expérience à l’utilisateur.

Ensuite, l’introduction de plusieurs joueurs plutôt qu’un seul me semble une bonne idée pour ouvrir à de nouveaux modes de jeu (2 joueurs contre l’ordi par exemple)

Enfin une optimisation des fonctions me semble souhaitable, non pour la performance mais pour la lisibilité du code.

## Impact projet professionnel

L’impact sur mon projet professionnel est positif, ce projet a ouvert le champ des possibilités que je me faisais du développement Java. J’ai donc envie de continuer et d’approfondir mes connaissances en JEE.

Les problèmes rencontrés dans ce projet au niveau de l’organisation m’a donné quelques idées d’applications que je pourrait développer dans mon dernier projet, le projet libre. (CV, k-lendar)