20/02/2018

Projet Mastermind

B2 Ingesup

Jonathan DALBERTO
Mathias LAFONTAN
Fanny LAJEUNESSE
Simon ROBERT
YNOV AIX-EN-PROVENCE

Table des matières

1. Présentation du projet

a. Objectifs du projet

L'objectif de ce projet est de réaliser le développement d'un programme Java reproduisant le fonctionnement du jeu Mastermind. Appliquer les principes de la programmation orientée objet en réalisant le design complet de l'architecture et en mettant en œuvre les méthodes de conception objet. Fournir une documentation exhaustive et cohérente rendant compte de la démarche adoptée et des différentes étapes d'analyse, de conception, de développement, de test et de déploiement de l'application.

Connecter Java à une base de données (ici, MySQL).

Implémenter quelques-uns des patrons de conception (design patterns) indispensables dans l'architecture objet.

b. Acteurs

L'équipe projet est composée de quatre étudiants :

- DALBERTO Jonathan (développeur BDD)
- LAFONTAN Mathias (lead développeur)
- LAJEUNESSE Fanny (chef de projet, développeur BDD)
- ROBERT Simon (développeur)

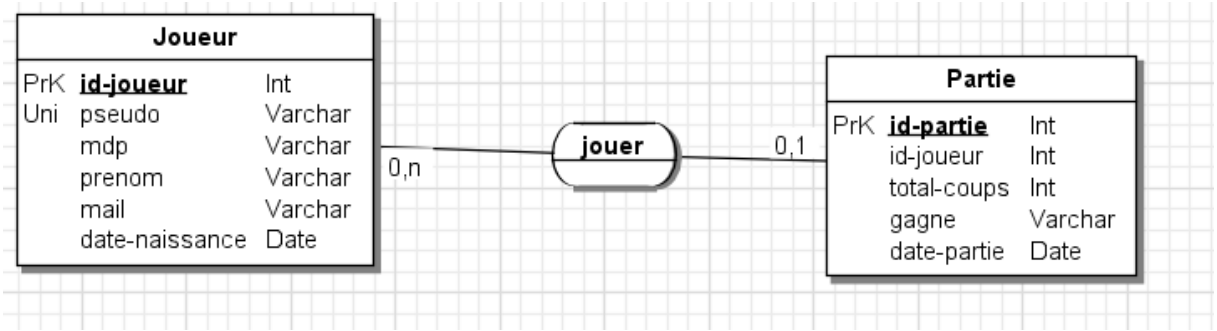
2. Spécifications fonctionnelles

a. Base de données

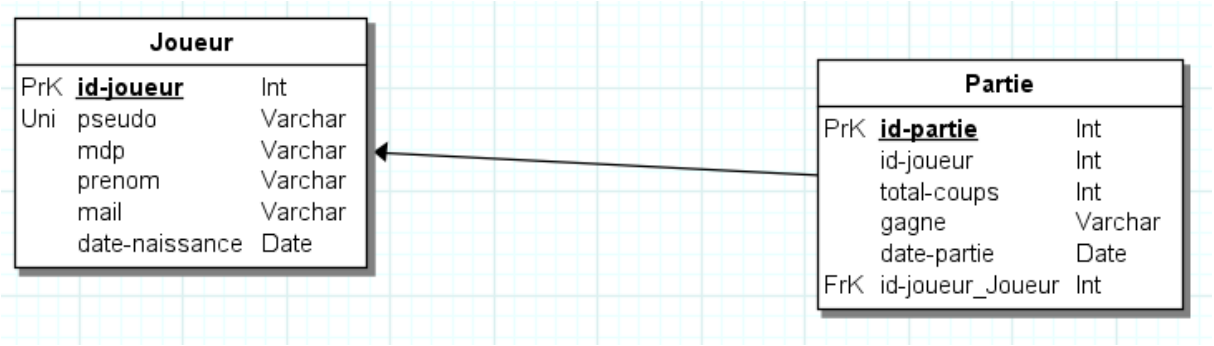
i. Dictionnaire de données

NOM	DESCRIPTION	FORMAT	LONGUEUR	TYPE	GESTION	CALCUL	SOURCE
id_joueur	Identifiant du joueur	int	10	E	auto_increment		B2 - projetDev -1718.pdf
pseudo	Pseudo	varchar	25	E	unique		
mdp	Mot de passe	varchar	8	E	1> et <8		
nom	Nom	varchar	50	E	N/A		
prenom	Prénom	varchar	50	E	N/A		
date_naissance	Date de naissance	date		E	N/A		
mail	Adresse mail	varchar	50	E	regex		
nb_parties	Nombre de parties jouées	int	10	C	count	count all games from id_joueur	
nb_gagne	Nombre de parties gagnées	int	10	C	count	count all won games from id_joueur	
nb_perdu	Nombre de parties perdues	int	10	C	count	count all lost games from id_joueur	
meilleure_partie	Meilleure partie réalisée				N/A		
id_partie	Identifiant de la partie	int	10	E	auto_increment		
date_partie	Date de la partie	date		E	N/A		
nb_coups	Nombre de coups joués	int	2	E	<11		
gagne	Indique si la partie a été gagnée ou non	Bool		E	N/A		

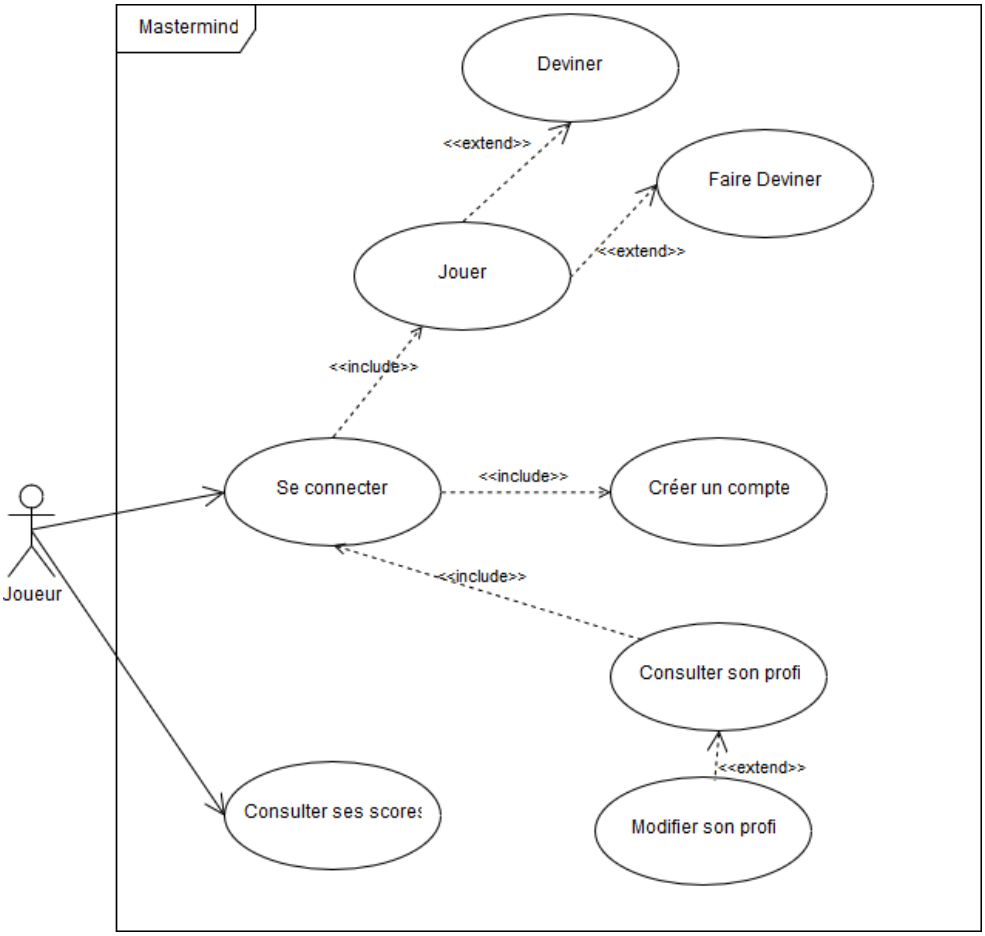
ii. Modèle conceptuel de données



iii. Modèle logique de données

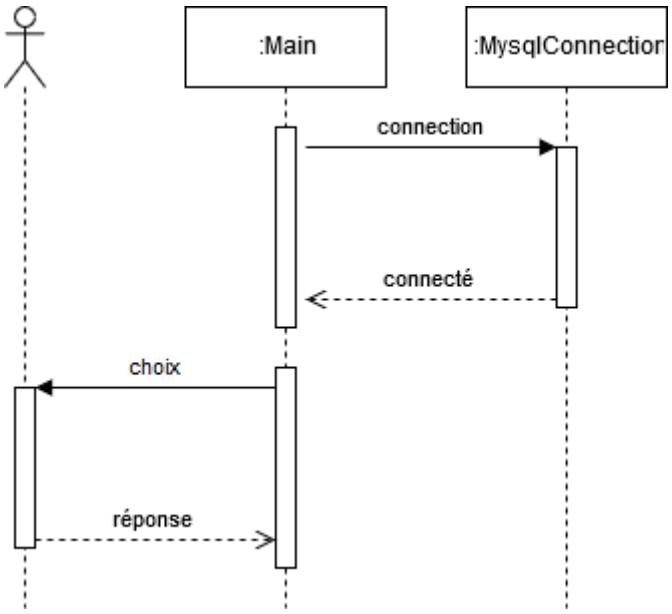


b. Use Case

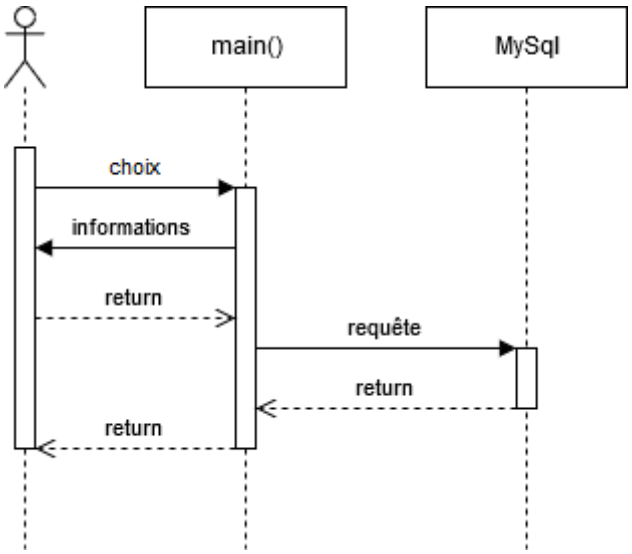


c. Diagrammes de séquence

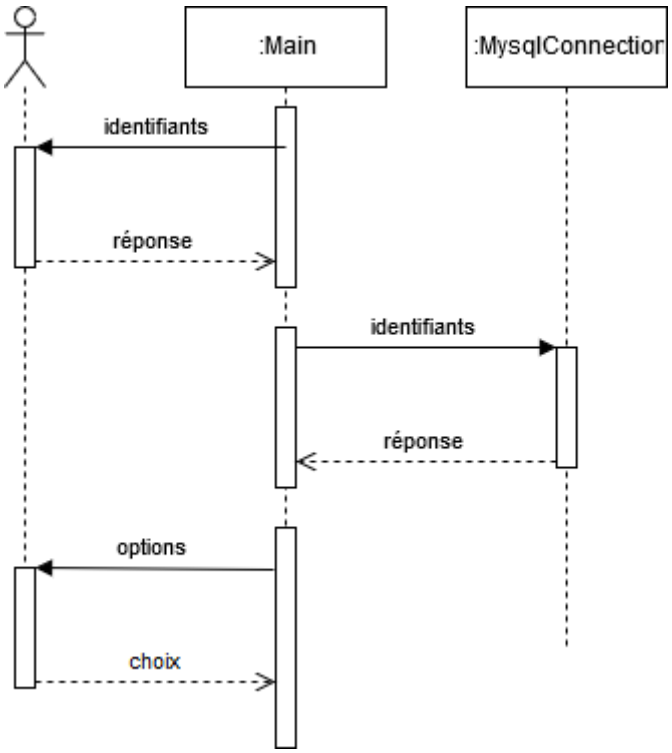
i. Lancement du jeu



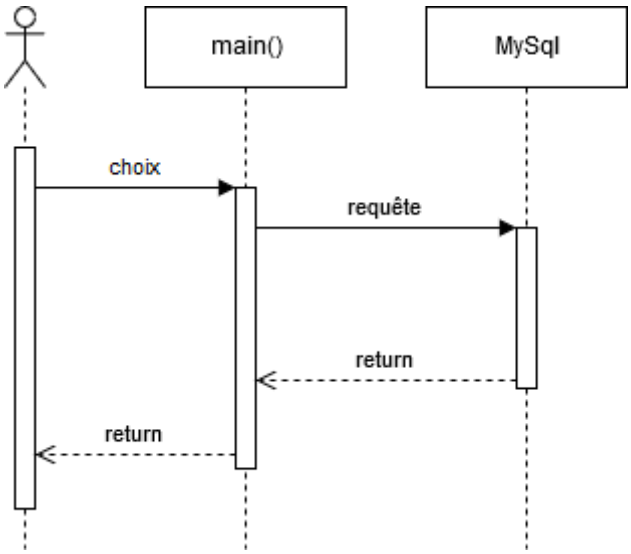
ii. Inscription



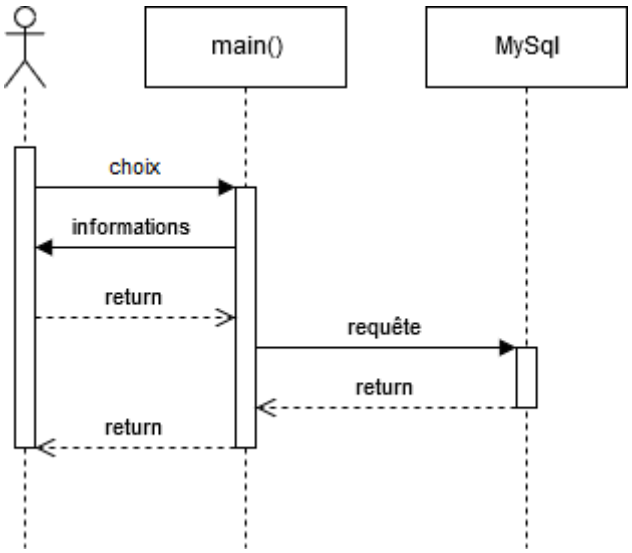
iii. Authentification



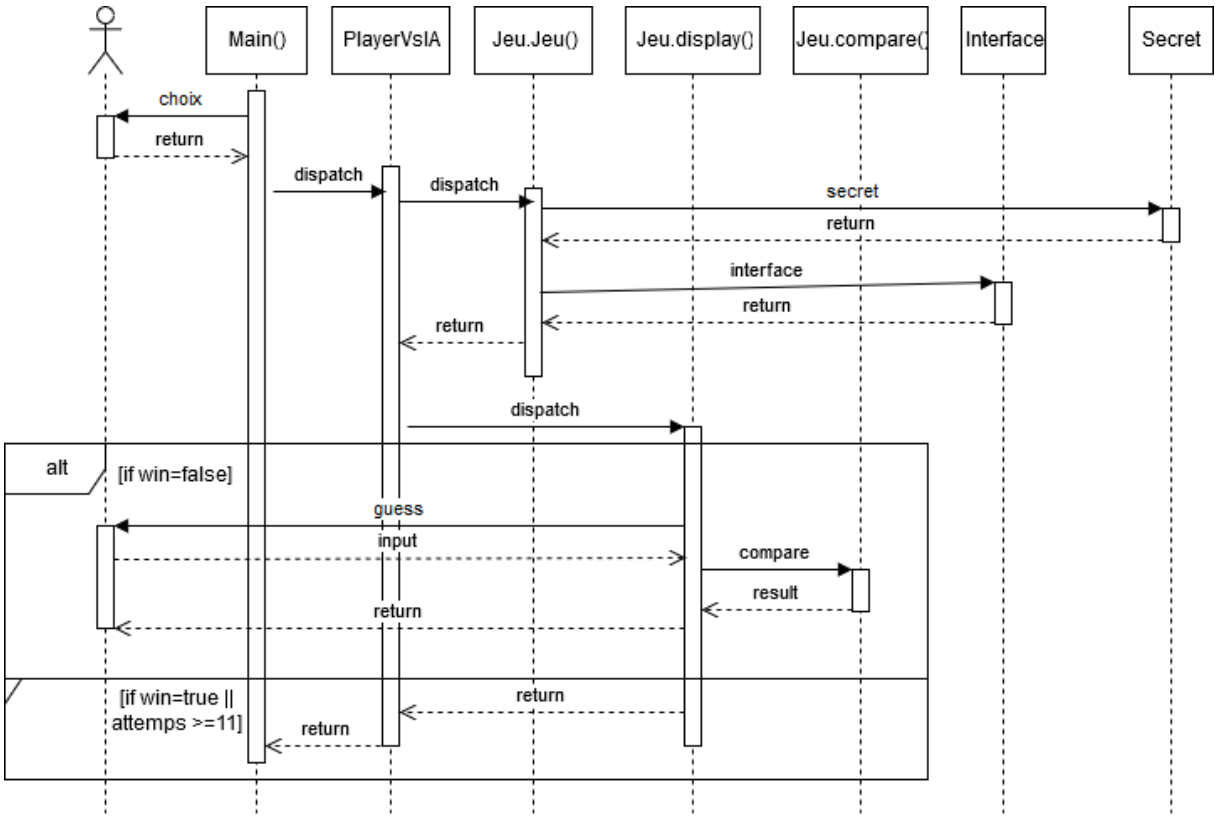
iv. Afficher le profil



v. Modifier le profil



vi. Choix Joueur VS Ordinateur



d. Algorithmes

i. Jeu 1 (le joueur devine)

```
Guess = LIRE entrée
Secret = [a,b,c,d,e]
BP=0
MP=0
Resultat = [BP, MP]

POUR i ALLANT DE 0 A 4 PAR PAS DE 1
     SI   Guess[i] == Secret[0] ||
          Guess[i] == Secret[1] ||
          Guess[i] == Secret[2] ||
          Guess[i] == Secret[3] ||
          Guess[i] == Secret[4]

          SI Guess[i] == Secret[i]
               BP = BP + 1
          SINON
               MP = MP + 1
          FIN SI
     FIN SI
FIN POUR

RETOURNER RESULTAT
```

ii. Jeu 2 (le programme devine)

```
FONCTION Combinaisons()
     // On génère toutes les combinaisons possibles
          Possibilité = [‘’]   //tableau de chaines de caractères
          a, b, c, d, e =1
          isOver = Faux

          TANT QUE isOver != Vrai

               Si e < 9
                    e = e + 1
                    Possibilité = Possibilité + a + ‘’ + b + ‘’ + c
                    + ‘’ + d + ‘’ + e

               SINON SI d < 9
                    e = 1
                    d = d + 1
                    Possibilité = Possibilité + a + ‘’ + b + ‘’ + c
                    + ‘’ + d + ‘’ + e

               SINON SI c < 9
                    e = 1
                    d = 1
                    c = c + 1
```

```
                Possibilité = Possibilité + a + '' + b + '' + c
                + '' + d + '' + e

            SINON SI b < 9
                e = 1
                d = 1
                c = 1
                b = b + 1
                Possibilité = Possibilité + a + '' + b + '' + c
                + '' + d + '' + e

            SINON SI a < 9
                e = 1
                d = 1
                c = 1
                b = 1
                a = a + 1
                Possibilité = Possibilité + a + '' + b + '' + c
                + '' + d + '' + e

            SINON
                isOver = Vrai
            FIN SI
        FIN TANT QUE
        RETOURNER Possibilité
FIN FONCTION

FONCTION Demander()
        // On demande et un récupère les entrées utilisateur
        Result=['','']
        AFFICHER « Entrer le nombre de chiffres bien placés »
        Result = Result + LIRE ENTREE

        AFFICHER « Entrer le nombre de chiffres mal placés »
        Result = Result + LIRE ENTREE

FIN FONCTION

FONCTION Comparaison(B, C)
        bpCompare, mpCompare = 0
        tabcompare = ['','']

        POUR i ALLANT DE 0 A 5 PAR PAS DE 1
            SI   C[0] == B[i] ||
                 C[1] == B[i] ||
                 C[2] == B[i] || //Si C contient l'index  de
                      C[3] == B[i] || la chaine B
                 C[4] == B[i]

                 SI B[i] == C[i]
                      bpCompare = bpCompare + 1
                 SINON
```

```
                        mpCompare = mpCompare + 1
                    FIN SI
                FIN SI
            FIN POUR

            tabcompare = [bpCompare, mpCompare]
            RENVOYER tabcompare

        FIN FONCTION


        FONCTION Tri_possibilités(Possibilité, Result, combinaison)

            bpR = Result[0]
            mpR = Result[1]
            Rez = [‘’, ‘’]

            POUR i ALLANT DE 0 à (TAILLE Possibilité) PAR PAS DE 1
                Rez = Comparaison(Possibilité[i], combinaison)

                Si Rez[0] =! bpR || Rez[1] =! mpR

                    RETIRER Possibilité[i]
                FIN SI
            FIN POUR
        FIN FONCTION

        nbTours = 1

        AFFICHER « Ecrivez votre combinaison sur une feuille de
papier »

        Combinaisons()

        TANT QUE nbTours < 11

            SI nbTours = 1

                AFFICHER Possibilité[0]
                Demander()

            SINON SI nbTours < 11

                Tri_possibilités()
                AFFICHER Possibilité[0]
                Demander()

            FIN SI

        FIN TANT QUE
```

3. Spécifications techniques

a. Langages et environnement de développement

Pour le développement de cette application, nous utiliserons le langage de programmation Java (version 1.8). Ce langage, étant orienté objet, est particulièrement adapté en raison de sa solidité et de sa maintenabilité. Il nous a été imposé dans le cadre du projet.

Pour la mise en place de la base de données, nous utiliserons le système de gestion open-source MySQL, lié à l'application Java grâce à l'interface JDBC (Java DataBase Connectivity). Ce système aura été privilégié pour son aspect open source, ainsi que par le fait qu'il soit déjà maîtrisé par la majorité de l'équipe.

La totalité du projet est déployée dans l'environnement Java 1.8, et nécessitera une machine virtuelle Java (JVM) dans une version 1.8 ou supérieure.

b. Outils

Le développement se fera sur l'IDE IntelliJ (licence étudiante JetBrains). Maîtrisé par une grande partie du groupe, il offre des fonctionnalités intéressantes qui rendent le codage plus agréable. IntelliJ IDEA possède sa propre extension permettant de créer et/ou de générer les diagrammes de classe.

Logiciel de modélisation JMerise (ou SQL Power Architect) pour le modèle de la base de données. Cet outil étant déjà maîtrisé par les membres du groupe c'est celui-ci qui a été privilégié.

Mise en place d'un répertoire Git pour la collaboration sur le code, ainsi que sur la documentation.

Mise en place d'un Google Drive pour le partage des informations et des divers documents, images…

c. Charte de codage/nommage

i. Nombre de caractères par ligne

Le nombre de caractères par ligne est limité à 120, indentations incluses.

ii. Indentation du code

Le code est indenté de quatre caractères d'espace. Les blocs de code sont limités par des accolades. Le retour à la ligne s'effectue après l'ouverture de l'accolade au début du bloc, et avant la fermeture en fin de bloc.

iii. Nommage

Les classes, variables et fonctions sont nommées en anglais. Le type de casse à respecter est *lowerCamelCase*, excepté pour les noms de classe en *CamelCase*. Tous les noms doivent être clairs, et énoncer au plus près leur fonctionnalité ainsi que la raison de leur présence dans le code, à l'exception des variables locales (principalement de type numérique) qui peuvent prendre comme noms : *a*, *b*, *i*, etc.

Concernant les classes, elles doivent être des noms simples et posséder en suffixe le nom du package auquel elles sont rattachées. Par exemple, pour une classe appartenant au package **Action** et permettant l'action de s'authentifier, on aura **AuthenticationAction**.

iv. Commentaires

Le maximum des commentaires doit être des annotations servant à la génération de la JavaDoc.

Les autres commentaires seront à placer avant le début d'un bloc, et non pas sur la même ligne que les instructions.

d. Règles de gestion

Sachant qu'il s'agit d'un jeu solo, un et un seul joueur pourra participer à autant de partie qu'il voudra.

4. Règles du jeu

a. Joueur contre l'Ordinateur

Dans le cas d'une partie contre l'ordinateur, c'est-à-dire que c'est au joueur de deviner le code secret, le joueur doit proposer un code à 5 chiffres. L'ordinateur lui répondra en lui donnant des indications sur sa proposition, BP pour indiquer le nombre de bons chiffres aux bons endroits, MP pour le nombre de bons chiffres au mauvais endroit.

Le joueur dispose de 10 tentatives. Passé ses 10 tentatives, il a perdu.

Exemple :

```
A vous de jouer !
1 : 12345 > 3BP/1MP
2 : 12364 > 2BP/1MP
3 : 12547 > 1BP/2MP
4 : 15348 > 2BP/2MP
5 : 18395 > 4BP/0MP
6 : 18325 > 5BP/0MP
BRAVO
```

b. Ordinateur contre le joueur

Dans le cas d'une partie où c'est l'ordinateur qui va tenter de deviner le code imaginé par le joueur. Le joueur va choisir un code composé de 5 chiffres, il peut contenir des doublons, et l'écrire sur un papier (pour ne pas l'oublier). L'ordinateur va alors proposer un premier code, le joueur devra lui répondre en respectant la syntaxe suivante : []BP / []MP. Avec BP le nombre d'éléments bien placés (peut être égal à 0), et MP le nombre d'éléments mal placés, mais présents dans le code (peut être égal à 0).

Attention, l'ordinateur signalera une erreur de syntaxe, vous obligeant à saisir les informations telles qu'il l'attend.

L'ordinateur dispose de 10 tentatives. Si au bout des 10 tentatives il n'a toujours pas trouvé le code, il vous le demandera, et vérifiera que vos vérifications étaient conformes.

Exemple de partie :

```
Prêt ? A moi de jouer !
1 : 12364 > 2BP/1MP
2 : 12547 > 1BP/2MP
3 : 15348 > 2BP/2MP
4 : 18395 > 4BP/0MP
5 : 18325 > 5BP/0MP
Je suis heureux d'avoir trouvé
```

5. Bilan

a. Logiciel

i. Fonctionnalités

Une certaine partie des fonctionnalités attendue n'aura pas pu être mise en place en raison du retard dans le développement. L'équipe a fait le choix de se consacrer plus particulièrement à la documentation du projet. Ainsi, les fonctionnalités suivantes n'apparaissent pas dans le code source final :

- Consultation du profil
- Modification du profil
- Consultation des scores/statistiques
- Persistance en BDD des objets représentant une partie

ii. Sécurisation du code

Étant données les difficultés rencontrées lors du développement des fonctionnalités, celles-ci ne sont pas sécurisées et ne gèrent pas toutes les erreurs possibles.

Ne sont pas pris en charge, lors d'une saisie utilisateur :

- Les erreurs de syntaxe
- Les erreurs de type

Lors de la connexion :

- Mauvais mot de passe
- Mauvais pseudonyme

b. Équipe

i. Difficultés rencontrées

Au cours de ce projet, l'équipe se sera heurtée principalement à deux obstacles.

En premier lieu, une grande partie du retard a été causée par une difficulté à mettre en place une organisation efficace. Outre le fait que les membres de l'équipe travaillent chacun sur des projets parallèles, à des rythmes différents, ils ont également été confrontés à des temps d'autoformation variables d'une personne à l'autre. Dans ces conditions, il n'a pas été possible de définir un planning optimal – ni de le suivre.

En second lieu, le langage imposé lors du projet ne fait partie des spécialités des membres de l'équipe. Aucune ressource ne pouvait vraiment mettre en avant des compétences intéressantes, ni donc venir en aide aux autres. Comme dit plus haut, le temps consacré à l'autoformation et aux recherches a été très long. Si cela a permis aux membres d'acquérir de nouvelles connaissances, l'impact n'a pas été bénéfique concernant la réalisation du projet même.

ii. Compétences acquises

- Conception objet
- Langages et outils de connexion à un SGBD en Java
- Javadoc

- Mise en place de Design Patterns

Annexes

Design Patterns

Singleton

Ce design pattern, très simple à implémenter, est traditionnellement utilisé pour permettre l'instanciation unique d'un objet. Il a été mis en œuvre ici afin de n'autoriser qu'une seule connexion à la base de données à la fois.

Factory

Afin de faciliter l'instanciation des classes gérant l'accès aux objets (DAO), le design pattern Factory a été utilisé.

Strategy

Le design pattern Strategy a été utilisé pour permettre de simuler un système de routage reliant le choix de l'utilisateur à une action spécifique. Il fait partie du modèle d'architecture MVC.