

PROJET TUTEURÉ DU SEMESTRE 3

DOSSIER DE PROGRAMMATION SYSTÈME RÉDUCTEUR DE MISE

Philippe BAUS
Elias HIETANEN
Antoine MILHAU
Cédric OLENDER

2019-2020

Client : M. ARNAULT Brice
Tuteur : M. FEHRENBACH Jérôme

I Présentation du projet.....	3
II Outils utilisés.....	4
1 Planification du projet.....	4
2 Partage de code.....	4
3 Langage de programmation.....	4
III Fonctionnalités attendues.....	5
IV Architecture du logiciel et fonctionnement.....	6
1 Préambule :.....	6
2 Développement du logiciel.....	6
a) Programme 1 : Trouver une couverture pour un système réducteur d'une taille donnée avec un nombre garanti donné.....	6
b) Programme 2 : pallier le facteur aléatoire de l'algorithme de génération en répétant plusieurs fois le premier programme.....	8
c) Programme 3 : le programme livré au client.....	12
V Améliorations potentielles.....	26
1 Interface.....	26
2 Taille du système réducteur.....	26
3 Nombre d'itérations.....	26
4 Protection de nos listes de grilles.....	26

I Présentation du projet

L'objectif de l'équipe était de développer une application d'un système réducteur de mise paramétrable. Le logiciel sera utilisé par M. ARNAULT, notre client, afin d'optimiser ses mises lorsqu'il joue à l'EuroMillions.

Les systèmes réducteurs sont une technique mathématique d'optimisation des mises pour les jeux de tirages. Le principe est de combiner plusieurs grilles de manière particulière et de restreindre l'ensemble des numéros sur lesquels on joue, afin de maximiser les chances de gain à un jeu de hasard.

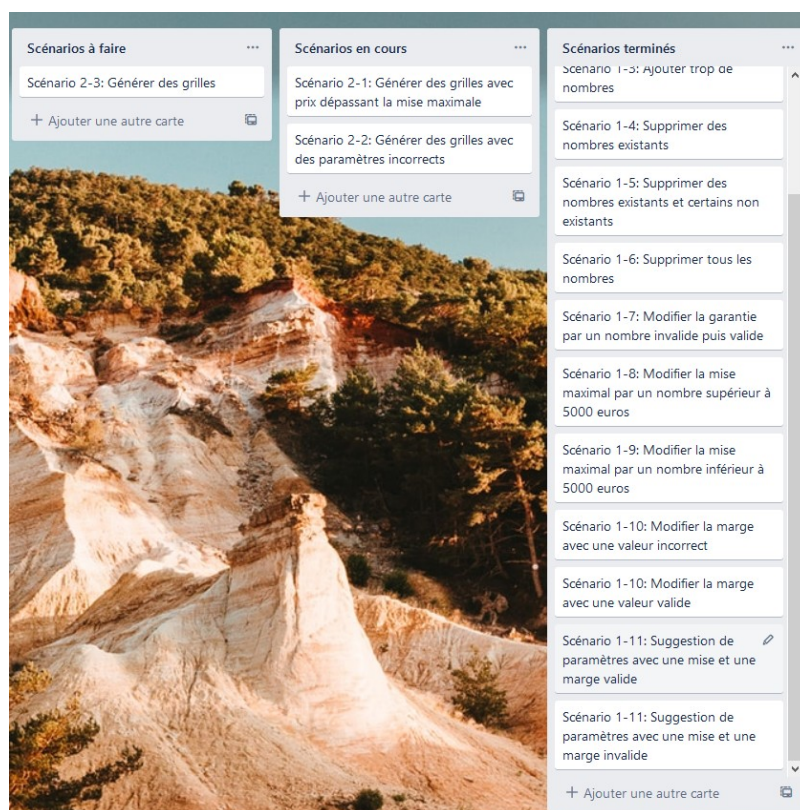
Lors de ce projet, nous avons donc conçu et réalisé une application d'un système réducteur de mise permettant à l'utilisateur de sélectionner les nombres qu'il souhaite jouer, le nombre de nombres garantis, de sélectionner une mise maximale ainsi qu'une marge s'appliquant à cette mise maximale. Ensuite, l'utilisateur peut générer et enregistrer les grilles dans un fichier qu'il est possible d'ouvrir dans un tableur. L'utilisateur peut également obtenir des suggestions de paramètres.

Nous décrivons dans ce dossier la démarche que nous avons effectuée pour développer cette application, ainsi que l'implémentation en JAVA des différentes fonctionnalités.

II Outils utilisés

1 Planification du projet

Pour travailler en équipe, planifier les tâches et toujours savoir où nous en sommes dans le projet, nous avons utilisé Trello. Trello permet de constater la progression du projet, et de découper le projet en sous tâches afin que tout le monde puisse travailler. Cela permet également de garder les idées claires, de savoir ce que l'on doit faire, et donc de pouvoir se concentrer sur la manière dont on souhaite le faire.



2 Partage de code

Pour pouvoir tous travailler sur le code le plus récent de manière simultanée, nous avons utilisé GIT



















3 Langage de programmation

Pour développer notre application, nous avons choisi d'utiliser JAVA, car il s'agit du langage que nous maîtrisons le mieux et sur lequel nous avons le plus d'expérience.



III Fonctionnalités attendues

Tableau reprenant les fonctionnalités exprimées dans le cahier des charges et leur réalisation effective

Fonctionnalité		Commentaire
Paramètres : - Numéros à jouer - Somme à miser - Nombre de numéros garantis - Taux de marge	   	
Affichage console		
Suggestion de paramètres en fonction de la mise maximale que le client est prêt à jouer		Fonctionnalité ajoutée après discussion avec le client
Gratuité pour M. Arnault		
Portabilité de l'application (pas d'installation)		
Multi-plateformes		
Temps nécessaire à l'exécution inférieur à 15 minutes		
Présence d'un avertissement envers les jeux d'argent		
Ergonomie : - Présence d'un manuel d'utilisation - Moins de 3 interactions de l'utilisateur pour accéder aux différentes fonctions du logiciel - Au moins deux tests utilisateurs	  	Nous n'avons eu le temps de faire tester notre interface qu'une seule fois par le client
Exportation des résultats : - Possibilité d'enregistrer au format .csv - Possibilité d'enregistrer ses résultats sous forme d'une page web	 	Nous avons utilisé format .txt au détriment du format .csv. Toutefois, il est quand même possible d'importer le fichier.txt dans un tableur. Nous n'avons pas eu le temps de réaliser l'enregistrement dans une page format web

IV Architecture du logiciel et fonctionnement

1 Préambule :

Remarques importantes : nous avons réalisé un système réducteur pour le jeu EuroMillions, cela implique que :

- Il est uniquement possible de jouer des entiers allant de 1 à 50
- Les grilles sont composées de 5 entiers
- Nous ne traitons pas les numéros chances

Explication du fonctionnement d'un système réducteur :

1. Choisir t entiers à jouer entre 1 et 50 ; ces nombres constituent notre ensemble e de nombre à jouer
2. Décider d'un nombre n de nombres garantis. Ce nombre signifie que si les 5 nombres gagnants tombent dans notre ensemble e , je dois avoir la garantie d'avoir au moins une grille jouée contenant au moins n nombres gagnants.
3. Générer un minimum de grilles à partir des entiers contenus dans e et garantissant qu'au moins l'une d'entre elles contienne au moins n nombres gagnants si les 5 nombres gagnants tombent dans notre ensemble e .

2 Développement du logiciel

Au cours de notre processus de développement, 3 programmes ont en réalité été écrits.

a) Programme 1 : Trouver une couverture pour un système réducteur d'une taille donnée avec un nombre garanti donné

Dans un premier temps, nous avons cherché à mettre en place un algorithme capable, à partir d'une taille de système réducteur et d'un nombre de nombres garantis, de calculer une couverture minimale de grille répondant à ces paramètres. Ce problème s'est révélé compliqué, nous n'avons pas trouvé de solution sur internet. De plus, nous nous sommes rendus compte que d'un point de vue mathématique, ce problème n'était, à priori, pas résolu.

A force d'essais, nous avons réussi à développer un algorithme efficace.

A partir d'une taille de système réducteur t et d'un nombre de nombres garantis n , l'algorithme génère des grilles d'entiers allant de 1 à t et garantissant n nombres garantis, quelle que soit la combinaison gagnante tirée à l'intérieur du système réducteur.

Le principal problème de cet algorithme est que le nombre de grilles générées pour faire la couverture est variable. Il y a une part d'aléatoire dans cet algorithme.

Un autre problème de cet algorithme est qu'il utilise uniquement des entiers de 1 à t , il est donc impossible de jouer les nombres que l'on veut.

Code de l'algorithme qui trouve une couverture pour un système réducteur d'une taille donnée avec un nombre garanti donné

```
/**
 * Méthode permettant de générer un ensemble de grille répondant au contraintes
 * passées en paramètre
 *
 * @param tailleSysReduc taille du système réducteur
 * @param nbGarantis      nombre de nombre garantis
 * @return ensemble de grilles généré
 */
public static List<Grille> genererCouverture(int tailleSysReduc, int nbGarantis) {
    // Ensemble de grilles que l'on va renvoyer
    List<Grille> bonnesGrilles = new ArrayList<>();

    // Ensemble de toutes les grilles possibles avec des entiers allant de 1 à
    // tailleSysReduc
    List<Grille> ensemble = Grille.enumererGrilles(tailleSysReduc);

    Collections.shuffle(ensemble);
    while (ensemble.size() > 0) {
        int i = 0;
        Grille gTemp = ensemble.get(0);
        ensemble.remove(0);
        bonnesGrilles.add(gTemp);
        while (i < ensemble.size()) {
            if (gTemp.nbNombresEnCommun(ensemble.get(i)) >= nbGarantis) {
                ensemble.remove(i);
            } else {
                i++;
            }
        }
    }
    Collections.sort(bonnesGrilles);
    return bonnesGrilles;
}
```

b) Programme 2 : pallier le facteur aléatoire de l'algorithme de génération en répétant plusieurs fois le premier programme

Pour pallier le problème d'aléatoire de notre algorithme de génération de grilles, nous avons décidé de créer un programme qui permet de :

- Lancer notre algorithme 1000 fois pour tous les paramètres, c'est-à-dire avec une taille de système réducteur variant de 11 à 30, et un nombre de nombres garantis variant de 2 à 4.
- Garder en mémoire l'ensemble de grille le plus petit par combinaison de paramètres
- Enregistrer dans des fichiers texte les ensembles de grilles précédemment générés, pour pouvoir les réutiliser.

Ainsi, on augmente les chances d'obtenir une solution performante.

En plus de pallier le problème d'aléatoire de notre algorithme, ce programme permet une exécution rapide du programme que nous livrons au client, puisque celui-ci ne fera que des correspondances entre les paramètres saisis par le client et les grilles pré générées.

Code du programme de génération des grilles

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

    long startTime = System.nanoTime();

    // Vérification de la présence du dossier
    // Ressources et création le cas échéant
    Path folder = Paths.get("Ressources");
    if (Files.notExists(folder)) {
        File folde = new File("Ressources");
        boolean bool = folde.mkdir();
        if (bool) {
            System.out.println("Répertoire créée");
        } else {
            System.out.println("Impossible de créer le répertoire");
        }
    }

    // Liste dans laquelle on va récupérer
    // les prix en fonction des paramètres
    List<String> strPrix = new ArrayList<>();
    int nbIterations = 1000;

    for (int garantis = 2; garantis < 5; garantis++) {
        for (int tailleSysReduc = 11; tailleSysReduc < 31; tailleSysReduc++) {
            if (garantis == 4 && tailleSysReduc == 26)
                break;
            // Génération des sets de grilles et conservation
            // du meilleur pour chaque paramètres
            List<Grille> grillesAJouerRetenues = Outils
                .genererCouverture(tailleSysReduc, garantis);
            for (int i = 0; i < nbIterations; i++) {
                List<Grille> grillesAJouerActuelles = Outils
                    .genererCouverture(tailleSysReduc, garantis);
                if (grillesAJouerActuelles.size() < grillesAJouerRetenues.size()) {
                    grillesAJouerRetenues = new ArrayList<>(grillesAJouerActuelles);
                }
            }
        }
    }
}
```

```

// Affichage
System.out.println("\nParamètres : " + tailleSysReduc + " "
    + garantis);
System.out.println("Taille de l'ensemble de grilles retenu : "
    + grillesAJouerRetenues.size());

// Création du nom du fichier en fonction des paramètres actuels
String nFic = "Ressources/data_";
nFic += tailleSysReduc + "_" + garantis + ".txt";
System.out.println("Nom du fichier : " + nFic);

strPrix.add(tailleSysReduc + "_" + garantis + ","
    + new Float(grillesAJouerRetenues.size() * 2.5F).toString());

// Transformation de la liste de grilles retenue en liste de String
List<String> strGrilles = new ArrayList<>();
for (Grille g : grillesAJouerRetenues) {
    strGrilles.add(g.toString());
}

// Ecriture des grilles dans le fichier
File f = new File(nFic);
try {
    f.createNewFile();
} catch (IOException e1) {
    e1.printStackTrace();
}
Path fichier = Paths.get(nFic);
try {
    Files.write(fichier, strGrilles, Charset.forName("UTF-8"),
        StandardOpenOption.APPEND);
} catch (IOException e) {
    e.printStackTrace();
}
}

// Ecriture des correspondances paramètres / prix dans le fichier prix.txt
String ficPrix = "Ressources/prix.txt";
File fic = new File(ficPrix);
try {

```

```

        fic.createNewFile();
    } catch (IOException e1) {
        System.out.println("Erreur creation du fichier prix");
        e1.printStackTrace();
    }
    Path fichierPrix = Paths.get(ficPrix);
    try {
        Files.write(fichierPrix, strPrix, Charset.forName("UTF-8"),
            StandardOpenOption.APPEND);
    } catch (IOException e) {
        System.out.println("Erreur écriture dans le fichier prix");
        e.printStackTrace();
    }
    long endTime = System.nanoTime();
    long timeElapsed = endTime - startTime;
    System.out.println("Nombre de générations : " + nbIterations + 1);
    System.out.println("\n\nGénération terminée avec succès en "
        + timeElapsed / 1000000000 + " s");
    System.out.println("Temps pour 1 itérations : "
        + (timeElapsed / 1000000000) / (nbIterations + 1) + " s");
    System.out.println("Temps estimé pour 1000 itérations : "
        + ((timeElapsed / 1000000000) / (nbIterations + 1)) * 1000 + " s");
}

```

c) Programme 3 : le programme livré au client

Une fois que nous avons généré nos grilles, il nous a fallu concevoir le programme final, celui qui allait être livré au client et qui répondrait aux exigences définies avec lui dans le cahier des charges.

Voici les fonctionnalités du programme que nous avons réalisé:

- Le programme propose une interface console permettant d'accéder aux fonctions décrites ci-dessous
- Le programme permet à l'utilisateur de saisir les paramètres
 - Ensemble e d'entiers à jouer
 - Nombre g de nombres garantis
 - Mise maximale que souhaite jouer le client
 - Marge de la mise
- Le programme affiche des avertissements en cas d'incompatibilités des paramètres
- Le programme affiche des messages d'erreurs en cas de paramètres erronés
- Le programme peut faire des suggestions de paramètres, calculées à partir de la mise maximale saisie et de la marge.
- Lorsque l'utilisateur valide ses paramètres et demande la génération des grilles, le programme :
 - Ouvre le fichier .txt contenu dans le dossier "Ressources" correspondant aux paramètres saisis par l'utilisateur.
 - Fait la correspondance entre les nombres saisis par l'utilisateur et les grilles stockées dans les fichiers .txt
 - Affiche et enregistre les grilles à jouer sous forme de fichiers .txt dans le dossier "Resultats"
- Le programme enregistre les résultats dans un format qui permet une ouverture dans un tableur

En plus de proposer une interface utilisateur complète, ce programme pallie le second problème de notre algorithme initial, puisqu'il permet à l'utilisateur de jouer les nombres qu'il désire.

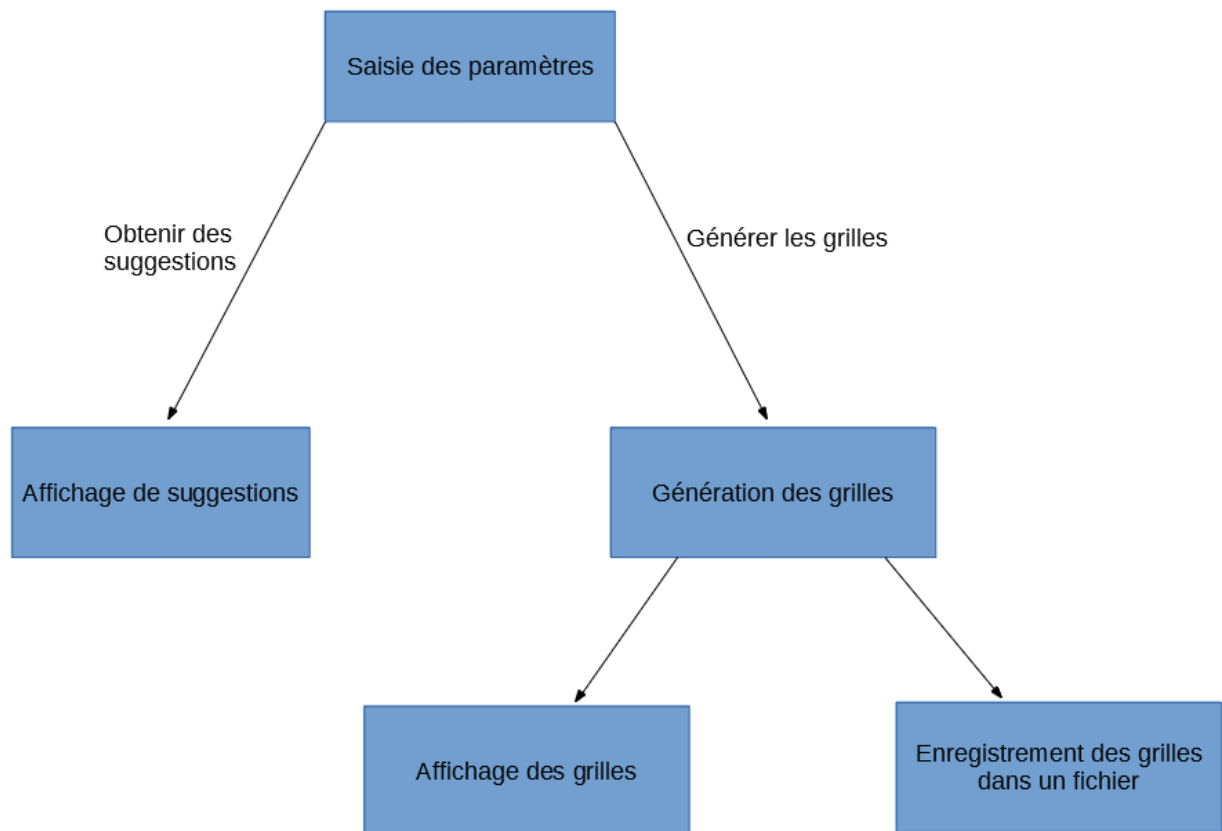


Schéma de fonctionnement du programme

Code du programme client

```
public static void main(String[] args) {
    final float PRIXGRILLE = 2.5F;
    df.setMaximumFractionDigits(2);
    Scanner entree = new Scanner(System.in);
    String choix;
    // Avant toute chose, on vérifie que le répertoire de ressources est présent,
    // sinon on ne fait rien
    if (Files.notExists(Paths.get("Ressources"))) {
        System.out.println("ERREUR : DOSSIER RESSOURCES INTROUVABLE");
        System.exit(0);
    }

    // message de mise en garde et demande +18
    Menu.clear();
    Menu.avertissementMenu();

    choix = entree.nextLine();
    // On quite le programme si le message n'est pas accepté
    if (!choix.equalsIgnoreCase("oui") && !choix.equalsIgnoreCase("o")) {
        System.exit(0);
    }

    /// VARIABLES "GLOBALES"
    final int NB_NB_JOUABLES = 50; // pour generer une array de 1 a 50
    float margeMise = 1.15F;
    int nbGarantis = 2; // garantie que veut l'utilisateur
    float miseMax = 0; // mise max de l'utilisteur
    List<Integer> reducteurNombres = new ArrayList<>(); // nombres de l'utilisateur
    List<Integer> list50 = new ArrayList<>(NB_NB_JOUABLES); // nombres ajoutables

    // generation des 50 nombres dispos
    for (int i = 1; i <= NB_NB_JOUABLES; i++) {
        list50.add(i);
    }

    List<Integer> nombresDispo = new ArrayList<>(list50);
```

```

// boucle principale
do {
    // variable de confirmation des actions de l'utilisateur
    Boolean confirm = null;
    // choix sur le menu principal
    switch (choix) {
    case "1":
        do {

            // Menu selection actions nombres du réducteur
            Menu.nombresSystemeMenu(reducteurNombres);
            System.out.println(">> Saisir une action :");
            choix = entree.nextLine();

            // choix ajout ou suppression
            switch (choix) {
            case "1":
                List<String> splitted;
                List<Integer> toModif;
                do {
                    // Menu ajout nombres
                    ajouterNombresSystemeMenu(reducteurNombres);
                    if (confirm != null)
                        if (confirm)
                            System.out.println("|| + Nombre(s) ajouté(s)");
                        else
                            System.out.println("|| /!\ \ Nombre(s) "
                                + "non ajouté(s)");
                    System.out.println(" ");
                    System.out.println(">> Saisir nombre(s) à ajouter");
                    System.out.println("séparés par un espace "
                        + "ou une virgule (ou 0 pour quitter) :");
                    choix = entree.nextLine();
                    System.out.println(" ");
                    if (!choix.equals("0")) {
                        toModif = new ArrayList<>();
                        confirm = null;
                        splitted = new ArrayList<String>(
                            Arrays.asList(choix.split(" |, ")));
                        for (String s : splitted) {

```

```

try {
    Integer olala = Integer.parseInt(s);
    if (reducteurNombres.indexOf(olala) == -1
        && nombresDispo.indexOf(olala) != -1
        && !toModif.contains(olala)) {
        toModif.add(olala);
    } else {
        confirm = false;
    }
} catch (Exception e) {
    confirm = false;
}

if (confirm == null)
    confirm = true;
Collections.sort(toModif);
List<String> filtre = new ArrayList<>();
for (Integer i : toModif) {
    filtre.add(String.valueOf(i));
}
splitted.removeAll(filtre);

if (!confirm && toModif.size() > 0) {
    System.out.println("||Ne peuvent pas être ajoutés : "
        + splitted
        + " Peuvent être ajoutés : " + toModif);
    System.out.println("|| Voulez-vous ajouter le(s) "
        + " nombre(s) possible(s) ? (oui/non)");
    choix = entree.nextLine();
}

if (choix.equalsIgnoreCase("o")
    || choix.equalsIgnoreCase("oui")
    || confirm) {
    reducteurNombres.addAll(toModif);
    nombresDispo.removeAll(toModif);
    Collections.sort(reducteurNombres);
    confirm = true;
}

toModif = null;

```



```

    }

    } while (!choix.equals("0"));
    choix = "";
    confirm = null;
    break;

case "2":
    do {
        // Menu suppression nombres
        supprimNombresSystemeMenu(reducteurNombres);
        if (confirm != null)
            if (confirm)
                System.out.println("|| - Nombre(s) supprimé(s)");
            else
                System.out.println("|| /!\ Nombre(s) "
                    + "non supprimé(s)");
        System.out.println(" ");
        System.out.println(">> Saisir nombre(s) à supprimer");
        System.out.println("séparés par un espace ou une virgule"
            + " (ou 0 pour quitter) :");
        choix = entree.nextLine();
        System.out.println(" ");
        if (!choix.equals("0")) {
            toModif = new ArrayList<>();
            confirm = null;
            splitted = new ArrayList<String>(
                Arrays.asList(choix.split(" |,")));
            for (String s : splitted) {
                try {
                    Integer olala = Integer.parseInt(s);
                    if (reducteurNombres.indexOf(olala) != -1
                        && nombresDispo.indexOf(olala) == -1
                        && !toModif.contains(olala)) {
                        toModif.add(olala);
                    } else {
                        confirm = false;
                    }
                } catch (Exception e) {

```

```

        confirm = false;
    }
}
if (confirm == null)
    confirm = true;
Collections.sort(toModif);
List<String> filtre = new ArrayList<>();
for (Integer i : toModif) {
    filtre.add(String.valueOf(i));
}
splitted.removeAll(filtre);

if (!confirm && toModif.size() > 0) {
    System.out.println("|| Ne peuvent pas "
        + "être supprimé : "
            + splitted
            + " Peuvent être supprimé : "
            + toModif);
    System.out.println("|| Voulez-vous supprimer "
        + "le(s) possible(s) ? (oui/non)");
    choix = entree.nextLine();
}
if (choix.equalsIgnoreCase("o")
    || choix.equalsIgnoreCase("oui")
    || confirm) {
    reducteurNombres.removeAll(toModif);
    nombresDispo.addAll(toModif);
    Collections.sort(nombresDispo);
    confirm = true;
}
toModif = null;

    }
} while (!choix.equals("0"));
choix = "";
confirm = null;
break;

case "3":
do {

```

```

        // Menu tout supp
        Menu.clear();
        System.out.println(" ");
        System.out.println(">> Voulez-vous vraiment supprimer "
            + "tous les nombres ? (oui/non)");
        System.out.println(" ");
        choix = entree.nextLine();
        System.out.println(" ");

        if (choix.equalsIgnoreCase("o")
            || choix.equalsIgnoreCase("oui")) {
            reducteurNombres = new ArrayList<>();
            nombresDispo = new ArrayList<>(list50);
            System.out.println("Nombre(s) supprimé(s)");
        } else {
            System.out.println("Nombre(s) non supprimé(s)");
        }
        System.out.println(" ");
        System.out.println(">> Appuyez sur n'importe quelle touche "
            + "pour revenir au menu précédent");
        choix = entree.nextLine();
        choix = "0";

    } while (!choix.equals("0"));
    choix = "";
    confirm = null;
    break;
}

} while (!choix.equals("0"));
break;

case "2":
    confirm = null;
    do {
        // menu garantie
        Menu.garantieMenu(nbGarantis);
        if (confirm != null)
            if (confirm)
                System.out.println("|| # Garantie modifiée");
    }

```

```

        else
            System.out.println("|| /\ \ Nombre invalide");
System.out.println(" ");
System.out.println(">> Saisir la grantie désirée"
        + " (ou 0 pour quitter) :");
choix = entree.nextLine();
if (!choix.equals("0")) {
    try {
        Integer nb = Integer.valueOf(choix);
        if (confirm = (nb > 1 && nb < 5))
            nbGarantis = nb;
    } catch (Exception e) {
        confirm = false;
    }
}

} while (!choix.equals("0"));
break;

case "3":
    confirm = null;
    do {
        // menu mise max
        Menu.miseMaxMenu(miseMax);
        if (confirm != null)
            if (confirm)
                System.out.println("|| # Mise max modifiée");
            else
                System.out.println("|| /\ \ Nombre invalide");
System.out.println(" ");
System.out.println(">> Saisir la mise max (ou 0 pour quitter) :");
choix = entree.nextLine();
if (!choix.equals("0")) {
    float nb;
    try {
        nb = Float.valueOf(choix);
        if (confirm = (nb > 0.0f && nb <= 5000.0f))
            miseMax = nb;
    } catch (Exception e) {
        confirm = false;
    }
}

```

```

        }
    }

    } while (!choix.equals("0"));
    break;

case "4":
    confirm = null;
    do {
        // menu marge mise
        Menu.miseMargeMaxMenu((margeMise - 1.0F) * 100.0F);
        if (confirm != null)
            if (confirm)
                System.out.println("|| # Marge mise modifiée");
            else
                System.out.println("|| /!\ Nombres invalides");
        System.out.println(" ");
        System.out.println(">> Saisir la marge en pourcents(%) "
            + " (ou 0 pour quitter) :");
        choix = entree.nextLine();
        if (!choix.equals("0")) {
            float nb;
            try {
                nb = Float.valueOf(choix);
                if (confirm = (nb >= 1.0f && nb <= 99999.0f))
                    margeMise = (nb / 100.0F) + 1.0F;
            } catch (Exception e) {
                confirm = false;
            }
        }

    } while (!choix.equals("0"));
    break;

case "5":
    do {
        // suggestion parametres
        Menu.suggestionParamsMenu(margeMise, miseMax);
        System.out.println(">> Saisir une action :");
        choix = entree.nextLine();
    }

```

```

    } while (!choix.equals("0"));
    break;

case "9":
    do {
        // Menu de génération des grilles
        Menu.clear();
        // Vérification que l'utilisateur a saisi un système réducteur assez
        // grand
        if (reducteurNombres.size() < 11 || reducteurNombres.size() > 30
            || (reducteurNombres.size() > 25 && nbGarantis == 4)) {
            System.out.println("=====");
            System.out.println("/!\ Impossible de générer les grilles, "
                + "les paramètres saisis sont incorrects");
            System.out.println("=====");
            System.out.println(" ");
            System.out.println("\n0 >> Revenir au menu");
            System.out.println(" ");
            choix = entree.nextLine();
        } else {

            final int TAILLESYSREDUC = reducteurNombres.size();
            final int GARANTIS = nbGarantis;

            // Code permettant de récupérer le chemin du bon fichier
            String nFicLoad = "Ressources/data_";
            nFicLoad += TAILLESYSREDUC + "_" + GARANTIS + ".txt";
            Path path = Paths.get(nFicLoad);
            BufferedReader monFic = null;
            ;
            try {
                monFic = Files.newBufferedReader(path,
                    Charset.forName("UTF-8"));
            } catch (IOException e) {
                System.out.println("Erreur, fichier de données non trouvé");
                e.printStackTrace();
            }

            // code permettant de transformer les infos du fichier en objets

```

```

// de type grille :
List<Grille> grillesRecuperees = new ArrayList<>();
String ligne = "";
try {
    ligne = monFic.readLine();
} catch (IOException e) {
    System.out.println("Erreur lecture ligne");
    e.printStackTrace();
}
while (ligne != null) {
    grillesRecuperees.add(new Grille(ligne));
    try {
        ligne = monFic.readLine();
    } catch (IOException e) {
        System.out.println("Erreur lecture ligne");
        e.printStackTrace();
    }
}

// Calcul du prix des grilles
float prix = grillesRecuperees.size() * PRIXGRILLE;

choix = "oui";
// Vérification du respect de la mise maximum, avec
// une marge de 15%
if (prix > miseMax * margeMise) {
    Menu.prixDepasseMiseMenu(prix, miseMax, margeMise);
    choix = entree.nextLine();
}

if (choix.toLowerCase().equals("oui")) {
    // On fait la correspondance
    Outils.convertir(reducteurNombres, grillesRecuperees);
    // Trouver le chemin du fichier ou on va stocker les
    // résultats, le fichier est
    // cree a partir de la date courante
    DateFormat format = new SimpleDateFormat(
        "yyyy_MM_dd-HH_mm_ss");
    Date dateCourante = new Date();
    // Vérification de la présence du dossier Resultats et

```

```

// création le cas échéant
Path folder = Paths.get("Resultats");
if (Files.notExists(folder)) {
    File folde = new File("Resultats");
    boolean bool = folde.mkdir();
    if (bool) {
        System.out.println("Répertoire créée");
    } else {
        System.out.println("Impossible de "
            + "créer le répertoire");
    }
}

// Ecriture des grilles dans les fichiers de resultat
String nFic = "Resultats/";
nFic += format.format(dateCourante) + ".txt";
File f = new File(nFic);
try {
    f.createNewFile();
} catch (IOException e1) {
    e1.printStackTrace();
}

// Enregistrement des grilles
List<String> strGrilles = new ArrayList<>();
for (int i = 0; i < grillesRecuperees.size(); i++) {
    if (i > 0 && i % 5 == 0)
        strGrilles.add("");
    strGrilles.add(grillesRecuperees.get(i).toString());
}

Path fichier = Paths.get(nFic);
try {
    Files.write(fichier, strGrilles,
        Charset.forName("UTF-8"),
        StandardOpenOption.APPEND);
} catch (IOException e) {
    e.printStackTrace();
}

// Affichahe des grilles jouées
Menu.clear();
for (int i = 0; i < grillesRecuperees.size(); i++) {
    if (i % 3 == 0 && i != 0)

```



```

        System.out.print("\n");
        if (i % 15 == 0 && i != 0)
            System.out.print("\n");
        System.out.print(grillesRecuperees.get(i) + " | ");
    }
    System.out.println("\n\n" + grillesRecuperees.size()
        + " grilles générées dans le fichier " + nFic);
    System.out.println("\nMONTANT A "
        + "MISER : " + prix + " euros\n");
    // Possibilite de retour au menu principal
    System.out.println("\n0 >> Revenir au menu");
    System.out.println(" ");
    choix = entree.nextLine();
} else if (choix.toLowerCase().equals("non")) {
    choix = "0";
}
}
} while (!choix.equals("0"));
break;
}

// on affiche le menu principal
Menu.accueilMenu(reducteurNombres.size(), nbGarantis, miseMax,
    (margeMise - 1.0F) * 100.0F);
System.out.println(">> Saisir une action :");
choix = entree.nextLine();

} while (!choix.equals("0"));
entree.close();
}

```

V Améliorations potentielles

1 Interface

Pour l'instant l'intégralité de l'affichage se fait dans une console, une interface graphique pourrait être une première amélioration. Une navigation au sein d'une interface graphique rendrait le logiciel plus facile d'utilisation et plus intuitif.

2 Taille du système réducteur

A l'heure actuelle notre logiciel ne peut être utilisé qu'avec des systèmes réducteurs allant de 11 à 30 nombres car c'est la fourchette de taille la plus commune pour un système réducteur. Il pourrait être intéressant de faire en sorte que les systèmes réducteurs puissent prendre des tailles allant de 5 à 48 nombres (5 étant la taille minimale d'un système réducteur car une grille est un ensemble de 5 chiffres et un système réducteur de 49 chiffres n'est pas réducteur sachant que l'ensemble de départ est constitué de 49 éléments).

3 Nombre d'itérations

Enfin notre algorithme nous permettant de trouver les couvertures les plus optimales est basé sur un système itératif : plus on fait d'itérations, plus on est susceptible de trouver des résultats intéressants, à savoir des couvertures avec un nombre réduit de grilles. Nous avons par souci de temps de calcul réalisé un nombre d'itérations un peu bas pour les paramètres les plus exigeants en ressources. Nous pourrions exécuter l'algorithme pendant des périodes de temps plus étendues pour trouver des couvertures encore plus petites et que l'utilisation du logiciel soit encore plus profitable à l'utilisateur.

4 Protection de nos listes de grilles

Pour l'instant, nos listes de grilles sont enregistrées sous forme de fichier texte. Elles ne sont en aucun cas protégées. Or, ce sont ces listes qui constituent notre "richesse", et nous devons les protéger si nous voulons envisager une exploitation commerciale de notre programme.