SAE 1.02 - E3CETE

J. Renaud - M. Franceus-Cointrel

Pour le 14 janvier 2024

Table des matières

Introduction

Dans cette SAE 1.02 nous étudions les *Paquet* de *Carte*. Un *Paquet* est une collection de *Carte* tous distincts. Une *Carte* est définie par ses caractéristiques possibles, ici : *Couleur*, *nbFiguresMax*, *Figure* et *Texture*.

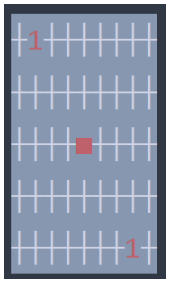

Figure 1: Affichage d'une carte *ROUGE* ayant *1 CARRE* et une texture *LIGNES_CROISEES*

En utilisant des outils informatiques, nous étudions des concepts mathématiques via la génération, le calcul, le stockage, le traitement, et l'affichage de données expérimentales ; pour les analyser et les comparer dans la mesure du possible aux valeurs théoriques. D'une part, nous nous intéressons au tri d'un *Paquet* mélangé ; et d'autre part aux probabilités. Nous nous focaliserons en particulier sur 3 algorithmes de tris en place : le tri par séléction, le tri par insertion et le tri à bulles. Nous étudions également les probabilités de 3 événements particuliers : 3CR, 3CR&2CL et E3C.

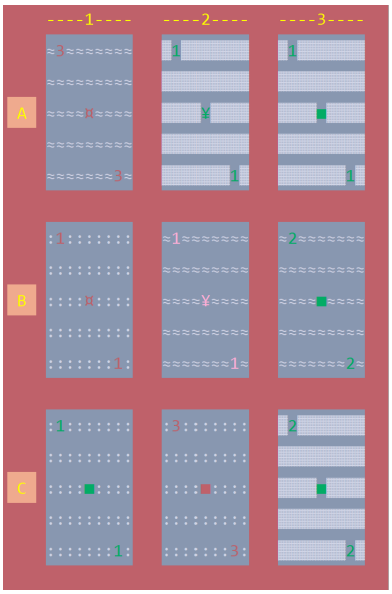

Figure 2: Affichage d'une *Table* 3x3 de *Carte* de notre jeu E3C (paramètres par default)

Chapitre 1

Analyse et comparaison des 3 méthodes de tris

1.1 Fonctions de tris et comptage du nombre d'opérations

Afin d'étudier la performance des 3 méthodes de tris, nous devons compter approximativement le nombre d'opération élémentaire effectués lors d'un tel tri. Pour cela, nous utilisons une variable de *Class* (globale) *nbOpApprox* que nous initialisons à 0 au début de chaque tri. Nous incrémentons cette valeur de 1 pour chaque opération élémentaire réalisé lors du tri (affectation, condition, lecture...). Nous décidons d'omettre les opérations liées à l'incrémentation de *nbOpApprox*. Cela importe peu puisque ce qui nous intéresse est l'ordre de grandeur de la valeur *nbOpApprox*.

1.1.1 Tri par sélection

```
1  public Paquet trierSelection() {
2      Paquet paqTri = new Paquet(this);
3      nbOpApprox = 0;
4
5      for (int i = 0; i < paqTri.nbCartesLeft-1; i++) {
6          int indiceMin = i;
7          for (int j = i+1; j < paqTri.nbCartesLeft; j++) {
8              if (paqTri.cartes[j].compareTo(paqTri.cartes[indiceMin]) < 0) {
9                  indiceMin = j;
10                 nbOpApprox += 1; // 1 pour l'affectation
11             }
12             nbOpApprox += 4 + 6 + 1 ; //4 pour le for, 6 pour le compareTo, 1 pour le if
13         }
14         paqTri.swap2Cartes(i, indiceMin);
15         nbOpApprox += 4 + 4 + 1; // 4 pour le swap, 4 pour le for, 1 pour l'affectation
16     }
17     return paqTri;
18 }
```

Code 1.1: *Fonction tri sélection*

1.1.2 Tri par propagation

```java
public Paquet trierBulles() {
    Paquet paqTri = new Paquet(this);
    boolean inOrder = false;
    nbOpApprox = 1;
    while (!inOrder) {
        inOrder = true;
        for (int i = 0; i < paqTri.nbCartesLeft - 1; i++) {
            if (paqTri.cartes[i+1].compareTo(paqTri.cartes[i]) < 0) {
                paqTri.swap2Cartes(i+1, i);
                inOrder = false;
                nbOpApprox += 1 + 5; // 1 pour l'affectation, 5 pour le swap
            }
            nbOpApprox += 4 + 6 + 2; // 4 pour le for, 6 pour le compareTo, 2 pour le if
        }
        nbOpApprox += 2; // 2 pour le while
    }
    return paqTri;
}
```

Code 1.2: *Fonction tri à bulles*

1.1.3 Tri par insertion

```java
public Paquet trierInsertion() {
    Paquet paqTri = new Paquet(this);
    nbOpApprox = 0;
    for (int i = 1; i < paqTri.nbCartesLeft; i++) {
        int j = i;
        while (j > 0 && paqTri.cartes[j].compareTo(paqTri.cartes[j-1]) < 0) {
            paqTri.swap2Cartes(j, j-1);
            j--;
            nbOpApprox += 6 + 2 + 1 + 5; // 6 pour le compareTo, 2 pour le while, 1 pour
                l'affectation, 5 pour le swap
        }
        nbOpApprox += 4 + 6 + 2; // 4 pour le for, (6+2) pour le while dans la cas on ne
            rentre pas
    }
    return paqTri;
}
```

Code 1.3: *Fonction tri insertion*

1.2 Protocole de test

1.2.1 Définition des variables

L'objectif de cette expérience est d'évaluer la performance de chacun des tris. Pour cela nous devons réaliser chacun des tris en variant le nombre de cartes N contenus dans le Paquet. Pour cette expérience les paquets triés seront toujours pleins, et le nombre de cartes N contenus est déterminé par les cardinalités des caractéristiques possibles. En principe nous fixons les cardinalités associés aux couleurs/figures/textures pour ne seulement varier le nombres de répétition des figures. Ainsi le nombre de cartes N est défini tel que :

$$N = cardCouleurs \times cardFigures \times cardTextures \times cardRepetFigures$$

Pour chacune des méthodes de tris nous calculons le nombre d'opération $nOpApprox$ et son temps d'éxecution (en ms) $tempsExec$. Ainsi, en répétant les tris $nbRepetTest$ nombre de fois pour N fixé, nous en déduisions les valeurs moyennes ainsi que leurs incertitudes (écartype) telles que :

$$\boldsymbol{nbOpMoy = \frac{1}{nbRepetTest} \times \sum_{i=1}^{nbRepetTest} nbOpApprox_i} \tag{1.1}$$

$$\boldsymbol{u\_nbOp = \sqrt{\frac{1}{nbRepetTest} \times \sum_{i=1}^{nbRepetTest} (nbOpApprox_i - nbOpMoy)^2}} \tag{1.2}$$

$$\boldsymbol{tempsExecMoy = \frac{1}{nbRepetTest} \times \sum_{i=1}^{nbRepetTest} tempsExec_i} \tag{1.3}$$

$$\boldsymbol{u\_tempsExec = \sqrt{\frac{1}{nbRepetTest} \times \sum_{i=1}^{nbRepetTest} (tempsExec_i - tempsExecMoy)^2}} \tag{1.4}$$

1.2.2 Protocole expérimental

1. Réaliser les 3 tris sur $nbRepetTest$ paquets ayant les mêmes caractéristiques mais chacun mélangés différemment, pour un même nombre de cartes N.
2. Récupérer le nombre d'opération $nbOpApprox$ et le temps d'éxécution $tempsExec$ de chacun des tris pour toutes les répétitions.
3. Calculer et stocker les valeurs moyennes $nbOpMoy$ (1.1) et $tempsExecMoy$ (1.3), ainsi que leurs incertitudes u_nbOp (1.2) et $u\_tempsExec$ (1.4).
4. Répéter l'expérience en variant N, en modifiant la valeur de $cardRepetFigures$.

1.2.3 Code associé au protocole

```
1
2  /**
3   * FONCTION AJOUTEE
4   * Action : realise les 3 methodes de tris pour un paquet initialise a nbCartes nombre
        de carte.
5   * Pour cette fonction le nombre de Couleur/nbFigures/Figure/Texture sont fixes (
        variables de classes : cardCouleurs, cardRepetFigures, cardFigures, cardTextures).
6   * Nous repetons ces tris plusieurs fois (valeurs entres en params) sur des paquets
        presentant les memes caracteristiques mais melanges differremment ; pour obtenir
        des donnees significatifs :
7   * Stock les donees relatifs (nb Cartes, nombre OP et temps exec) aux tris dans un
        tableau de tableau.
8   * tabInfos[O a 2][0] --> nbCartes
9   * tabInfos[O a 2][1] --> nbOP moyen
10  * tabInfos[O a 2][2] --> ecart-type nbOp
11  * tabInfos[O a 2][3] --> tempsExec moyen
12  * tabInfos[O a 2][4] --> ecart-type tempsExec
13  *
14  * tabInfos[0][0 a 4] --> Infos de Methode SELECTION
15  * tabInfos[1][0 a 4] --> Infos de Methode BULLES
16  * tabInfos[2][0 a 4] --> Infos de Methode INSERTION
17  * @param nbRepetition
18  * @return un tableau de tableau (3x3) de double.
19  */
20 private static double[][] trisPaquet(int nbRepetition) {
21     double[][] tabInfos = new double[3][5];
22     double tempsExec;
23     double[][][] tabInterm = new double [3][2][nbRepetition];
24
25     Paquet paq;
26
27     for (int i = 0; i < nbRepetition; i++) {
28         paq = new Paquet(
29             Couleur.valuesInRange(0, cardCouleurs),
30             cardRepetFigures,
31             Figure.valuesInRange(0, cardFigures),
32             Texture.valuesInRange(0, cardTextures)
33             );
34
35         tempsExec = Ut.getTempsExecution(paq::trierSelection);
36         tabInterm[0][0][i] = nbOpApprox;
37         tabInterm[0][1][i] = tempsExec;
38
39         tempsExec = Ut.getTempsExecution(paq::trierBulles);
40         tabInterm[1][0][i] = nbOpApprox;
41         tabInterm[1][1][i] = tempsExec;
42
43         tempsExec = Ut.getTempsExecution(paq::trierInsertion);
44         tabInterm[2][0][i] = nbOpApprox;
45         tabInterm[2][1][i] = tempsExec;
46     }
47
48     for (int i = 0; i < 3; i++) {
49         tabInfos[i][0] = cardCouleurs * cardRepetFigures * cardFigures * cardTextures;
50     }
```

```
51      tabInfos[0][1] = Ut.moyenne(tabInterm[0][0]);
52      tabInfos[0][2] = Ut.ecarttype(tabInterm[0][0]);
53      tabInfos[0][3] = Ut.moyenne(tabInterm[0][1]);
54      tabInfos[0][4] = Ut.ecarttype(tabInterm[0][1]);
55
56      tabInfos[1][1] = Ut.moyenne(tabInterm[1][0]);
57      tabInfos[1][2] = Ut.ecarttype(tabInterm[1][0]);
58      tabInfos[1][3] = Ut.moyenne(tabInterm[1][1]);
59      tabInfos[1][4] = Ut.ecarttype(tabInterm[1][1]);
60
61      tabInfos[2][1] = Ut.moyenne(tabInterm[2][0]);
62      tabInfos[2][2] = Ut.ecarttype(tabInterm[2][0]);
63      tabInfos[2][3] = Ut.moyenne(tabInterm[2][1]);
64      tabInfos[2][4] = Ut.ecarttype(tabInterm[2][1]);
65
66      return tabInfos;
67  }
68
69  /**
70   * FONCTION AJOUTEE
71   * Prerequis : 0 < nStart < nEnd ET 0 < nStep
72   * Action : realise trisPaquet(int nbRepetition) en variant le nbCartes en modifiant la
         cardinalite de nbFigures, entre en parametres, de nStart a nEnd(exclu) en faisant
         des pas de nStep.
73   * Les cardinalites de couleurs/figures/textures sont fixees a des valeurs constantes
         par les variables de class.
74   * @param nStart premiere valeur de nbFiguresMax
75   * @param nEnd valeur limite (exclue)
76   * @param nStep pas
77   * @return un tableau contenant tout les resultats de chacun des trisPaquet().
78   */
79  private static double[][][][] testTrisVarN(int nStart, int nEnd, int nStep) {
80      int range = (int) Math.ceil((float) (nEnd - nStart)/nStep);
81      double[][][][] tabInfos = new double[range][3][5];
82      int count = 0;
83
84      for (int n = nStart; n < nEnd; n+=nStep) {
85          cardRepetFigures = n;
86          tabInfos[count] = trisPaquet(nbRepetTest);
87          count++;
88      }
89      return tabInfos;
90  }
```

Code 1.4: *Fonctions de calculs de données expérimentales*

1.3 Tests

1.3.1 Initialisation et spécificités

Les variables :

- int $nbRepetTest = 1000$
- int $cardCouleurs = 1$; int $cardFigures = 1$; int $cardTextures = 1$
- Test 1 : int $cardRepetFigures \in [10 - 500]$; step $= 1$
- Test 2 : int $cardRepetFigures \in [1000 - 10000]$; step $= 1000$

Specificités de l'appareil utilisés :

➤ CPU : 12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz

➤ RAM : 16.0 GB

➤ OS : Windows 11

1.3.2 Analyse graphique

Nous stockons les données des tests sous forme de fichier .csv, grâce à une fonction de conversion. Le fichier contient les informations des 3 tris, i.e. toutes les valeurs de N, $nbOpMoy$, u_nbOp, $tempsExec$, $u\_tempsExec$ pour chacune des méthodes de tri.

```
1   N, nbOpSEL, u_nbOPSEL, tExecSEL (ms), u_tExecSEL (ms), nbOpBUL, u_nbOPBUL, tExecBUL (ms), u_
2   10, 588, 2, 0.001, 0.003, 966, 196, 0.002, 0.005, 423, 80, 0.001, 0.002
3   11, 709, 3, 0.001, 0.001, 1171, 226, 0.001, 0.001, 498, 93, 0.001, 0.000
4   12, 841, 3, 0.001, 0.003, 1425, 261, 0.002, 0.005, 594, 104, 0.001, 0.000
5   13, 984, 3, 0.001, 0.002, 1691, 298, 0.002, 0.009, 690, 116, 0.001, 0.000
```

Figure 1.1: Exemple de données dans le fichier .csv en question

3 Nous choisissons de réaliser le graphique de $nbOpMoy = f(N)$ et de $tempsExecMoy = f(N)$ à l'aide du logiciel *Regressi*. Etant donées que ces 3 fonctions de tris utilisent chacune une double boucle imbriquée, nous nous attendons à une complexité quadratique $O(n^2)$.

Nous observons bien sur les graphiques ci-dessous une compléxité quadratique, puisque les courbes s'apparentent à des paraboles. Quant à la performance de ces 3 tris en place, le Tri à bulles est le moins performant et le moins stable (incertitudes qui augmentent avec N) ; ce qui est cohérent puisque ce-dernier compare répétitivement les cartes "consécutifs" du Paquet, et les permutent si "non triées". Ainsi, pour un paquet désordonné il réalise beaucoup de permutations de cartes. Ensuite, le Tri par sélection semble être le plus stable. Ceci s'explique du fait que pour un même N, il réalise toujours le même nombre d'opération et permutations quelque soit l'arrangement du Paquet. Finalement, le tri par Insertion est celui qui est le plus rapide (du moins pour $N < 500$). En effet, ce dernier est efficace lorsque le Paquet est partiellement trié, puisque pour chacune des cartes, il consiste à le placer au bon endroit dans le tableau parmi toutes les cartes précédentes qui sont déjà ordonnées. D'ailleurs, nous observons que pour N plus importants (Test n°2) le tri par séléction l'emporte sur le tri par insertion au niveau du temps d'éxecution. Nous avons choisi de réaliser un deuxième test pour N plus élevés pour avoir des valeurs plus significatifs concernant le temps d'éxécution.

1.3.3 Graphiques : Test 1

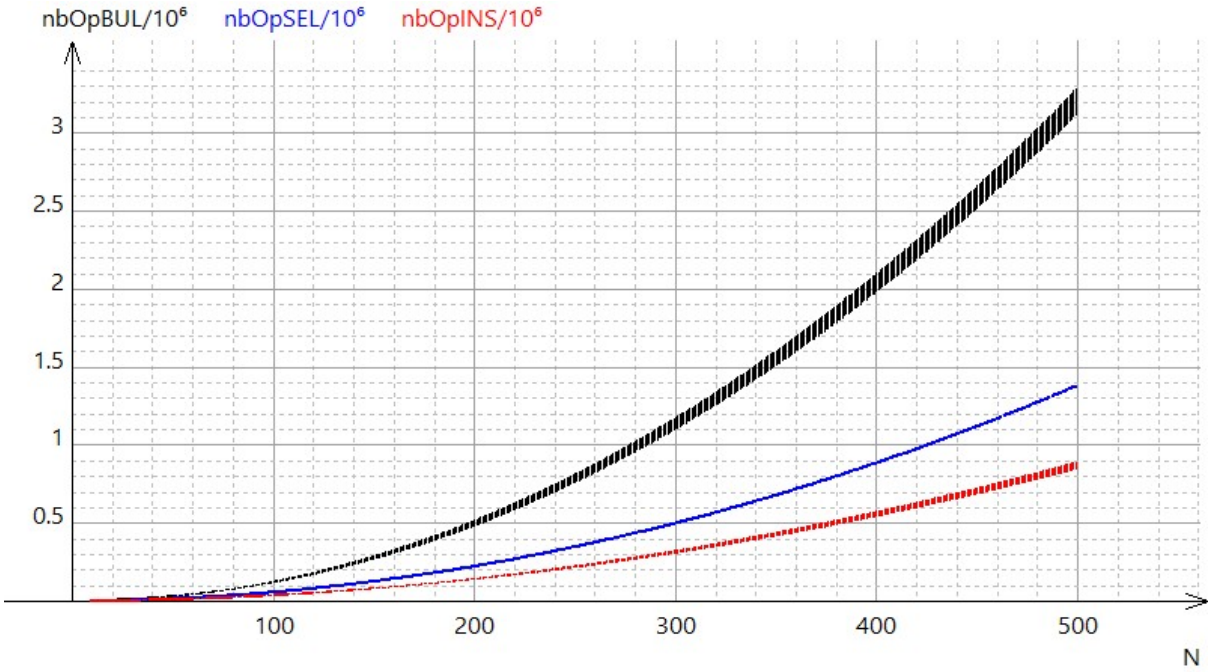

Figure 1.2: Graphique de $nbOpMoy = f(N)$ avec les barres d'incertitudes sur $nbOpMoy$ pour les 3 méthodes de tris : en noir le tri à bulles, en bleu le tri sélection et en rouge le tri insértion.

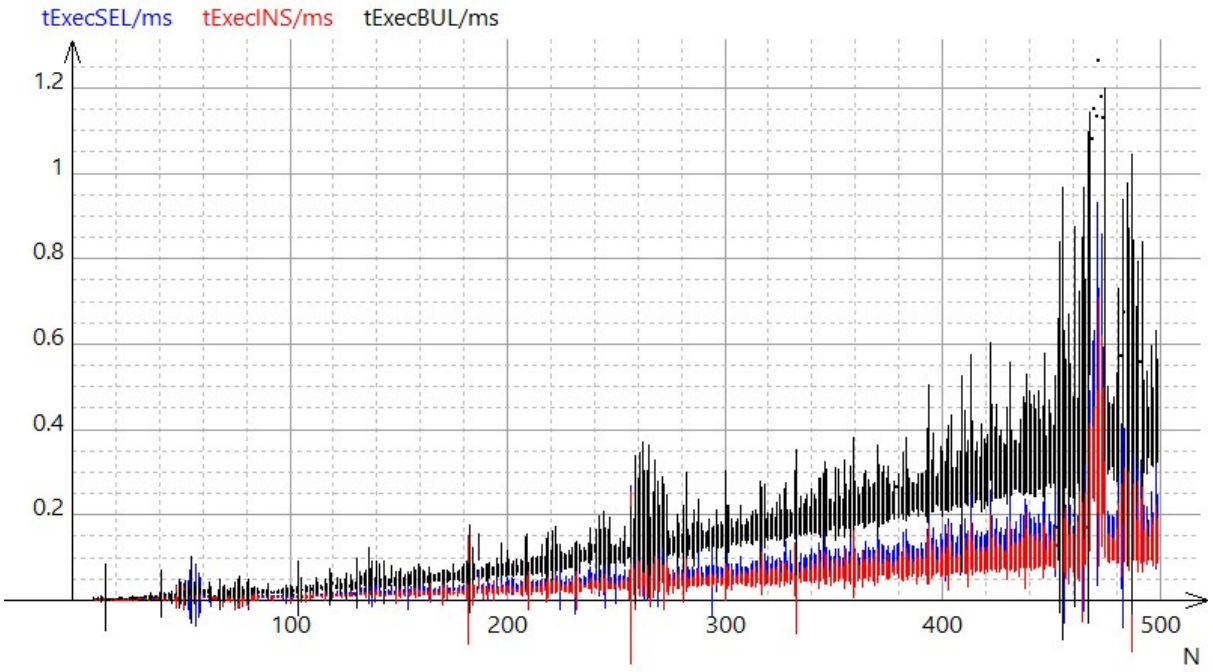

Figure 1.3: Graphique de $tempsExecMoy = f(N)$ avec les barres d'incertitudes sur $tempsExecMoy$ pour les 3 méthodes de tris : en noir le tri à bulles, en bleu le tri sélection et en rouge le tri insértion.

1.3.4 Graphiques : Test 2

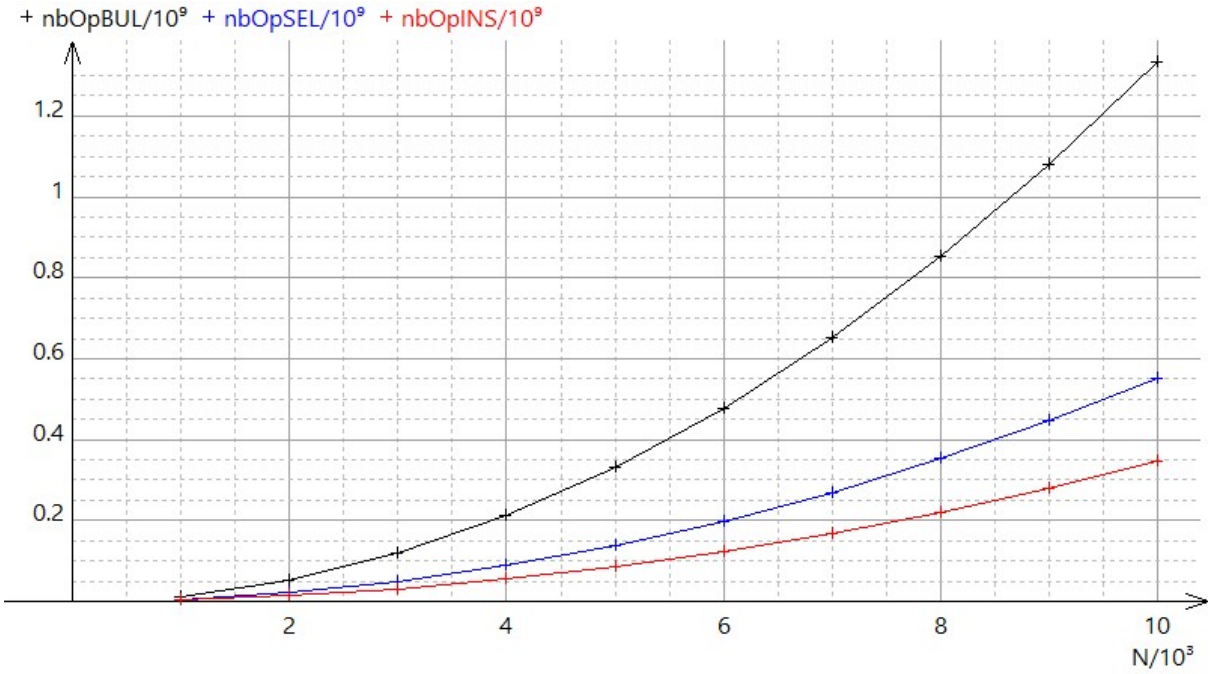

Figure 1.4: Graphique de $nbOpMoy = f(N)$ avec les barres d'incertitudes sur $nbOpMoy$ pour les 3 méthodes de tris : en noir le tri à bulles, en bleu le tri sélection et en rouge le tri insértion.

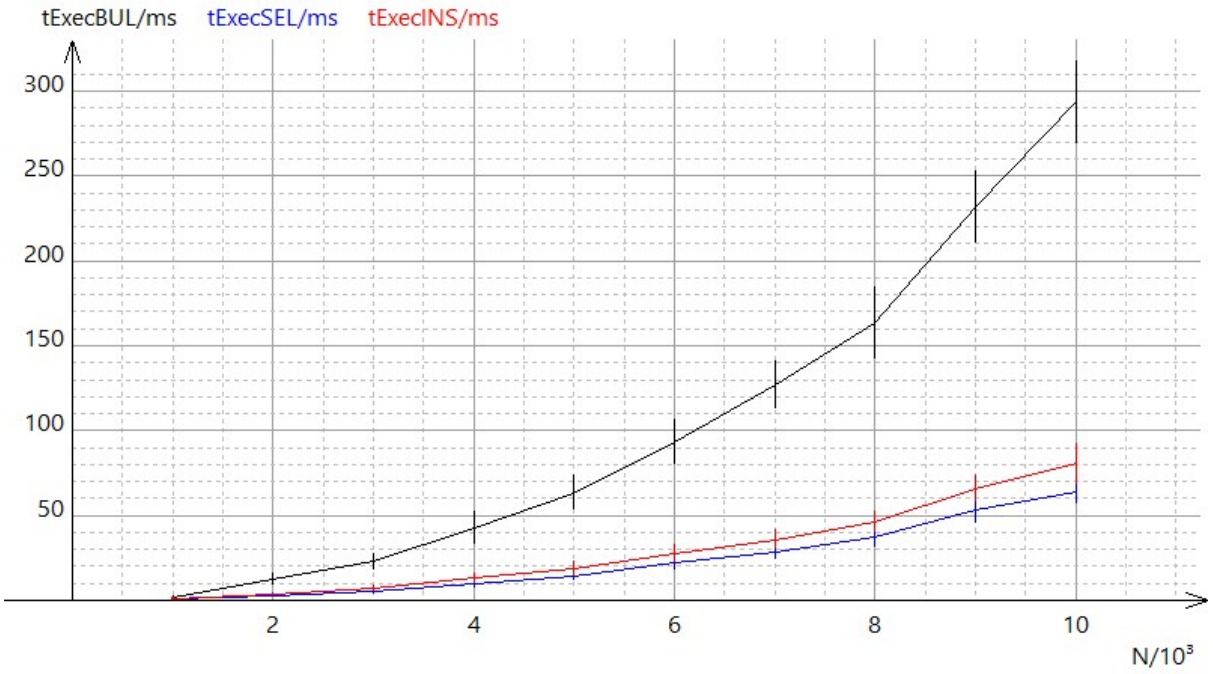

Figure 1.5: Graphique de $tempsExecMoy = f(N)$ avec les barres d'incertitudes sur $tempsExecMoy$ pour les 3 méthodes de tris : en noir le tri à bulles, en bleu le tri sélection et en rouge le tri insértion.

Chapitre 2

Théorie

2.1 Représentation mathématique de la Class *Table*

4 En termes de structure de données, une *Table* est un tableau de *Carte*. Les *Carte* d'une *Table* sont indicés par rapport à leur position dans le tableau, ce qui détermine leurs positions sur la *Table* au niveau de l'affichage graphique. Ainsi, une *Table* est une liste ordonnée de *Carte* tous différentes (pas de répétition), parmi toutes les *Carte* du jeu contenues dans un *Paquet*. Une *Table* est donc un arrangement.

5 Pour une *Table* de 9 *Cartes* et un jeu de 81 *Cartes*, le nombre de *Tables* différentes possibles est tel que :

$$A_{81}^{9} = \frac{81!}{(81-9)!} = 81 \times 80 \times ... \times 74 \times 73 = 94670977328928000$$

2.2 Démarche expérimental

L'expérience consiste à répeter $nbEssai$ fois l'action suivante : piocher 9 cartes d'un paquet mélangé de 81 cartes. Nous obtenons sur la table un arrangement de ces 9 cartes. L'événement 3CR consiste à détecter si parmi ces 9 cartes, on a exactement 3 cartes rouges. L'événement 3CR&2CL consiste à détecter si parmi ces 9 cartes, on a exactement 3 cartes rouges et 2 cartes ayant au moins un losange. L'événement E3C consiste à détecter si parmi ces 9 cartes, on a au moins un E3C.

Pour chacun de ces événements :
On notera $Freq$ la fréquence de cet evenement durant l'experience. De la $Freq$ on en deduit la probabilité individuelle empirique P de l'événement associé à cette experience : $P = \frac{Freq}{nbEssai}$. De cela nous pouvons en déduire le pourcentage d'erreur $Erreur$ de cette probabilité par rapport à la valeur théorique calculée (si c'est le cas).

Ensuite, nous répétons l'experience en variant $nbEssai$, et nous regardons l'évolution de la $Freq$, de P et de $Erreur$ en fonction de $nbEssai$.

```java
/**
 * FONCTION AJOUTEE
 * Prerequis : 0 <= startEssai < endEssai ET 0 <= stepEssai
 * experience = 1 pour probabilite de 3CR
 * experience = 2 pour probabilite de E3C
 * experience = 3 pour probabilite de E3C&2CL
 * Action : Realise les fonctions proba3CR(int nbEssai) ou probaE3C(int nb) en
 * faisant varier le nombre d'essai de startEssai a endEssai en faisant des pas
 * de stepEssai.
 * Stock les donnees dans un tableau de double de taille 4 :
 *  tabStats[0] --> Nombre d'essai
 *  tabStats[1] --> Frequence de cas favorables de l'evenement
 *  tabStats[2] --> Probablilite individuelle : P = frequence/nbEssai
 *  tabStats[3] --> Pourcentage d'Erreur de la probabilite individuelle de l'evenement
     par rapport a la valeur theorique ; calculable seulement en connaissance de la
     valeur theorique
 * Convertion de ces donnees en String adaptes au format .csv et ecrit le String dans un
     fichier : "proba3CR.csv" ou "probaE3C.csv" ou "proba3CR&2L"
 */
public static void probaVarEssai(int startEssai, int endEssai, int stepEssai, int
    experience) throws FileNotFoundException {
    int range = (int) Math.ceil((float) (endEssai - startEssai)/stepEssai);
    double[][] tabStats = new double[range][4];
    int count = 0;
    for (int i = startEssai; i < endEssai; i+=stepEssai) {
        tabStats[count][0] = i;
        if (experience == 1) {
            tabStats[count][1] = proba3CR(i);
            tabStats[count][2] = tabStats[count][1]/i; //proba individuelle
            tabStats[count][3] = Ut.pourcentageErreur(0.28956680871386137, tabStats[
                count][2]);
        }
        else if (experience == 2) {
            tabStats[count][1] = probaE3C(i);
            tabStats[count][2] = tabStats[count][1]/i; //proba individuelle
            //na calcul pas erreur car pas de val theorique
        }
        else if (experience == 3) {
            tabStats[count][1] = proba3CRAnd2CL(i);
            tabStats[count][2] = tabStats[count][1]/i; //proba individuelle
            tabStats[count][3] = Ut.pourcentageErreur(0.06884362550959248, tabStats[
                count][2]);
        }
        count++;
    }
    if (experience == 1) {
        stringInfosToCsv(probaDatasToString(tabStats), "proba3CR");
    }
    else if (experience == 2) {
        stringInfosToCsv(probaDatasToString(tabStats), "probaE3C");
    }
    else if (experience == 3) {
        stringInfosToCsv(probaDatasToString(tabStats), "proba3CR&2L");
    }
}
```

Code 2.1: *Fonction pour réaliser les expériences en variant nbEssai avec l'événement à choisir en paramètre*

2.3 Etude du cas 3CR

2.3.1 Calculs théoriques

6 Sachant que pour une *Carte* il y a 3 *Couleurs*, 3 répétitions maximales de figures, 3 *Figures* et 3 *Textures* possibles ; on en déduit qu'il existe $1 \times 3 \times 3 \times 3 = 27$ cartes rouges distinctes. Pour déterminer les arrangements possibles contenant exactement 3 cartes rouges : comme l'ensemble des cartes rouges et l'ensemble des cartes non rouges sont disjoints, nous comptons le nombre de combinaisons possibles de 3 cartes rouges parmi les 27 cartes rouges du paquet ; et le nombre de combinaisons possibles de 6 cartes non rouges parmi les $81-27=54$ cartes non rouges. A chacune de ces combinaisons possibles de 3 cartes rouges + 6 cartes non rouges (jusque là pas d'ordre), nous pouvons réaliser 9! permutations possibles (relation d'ordre). Ainsi, le nombre de *Tables* différentes contenant exactement 3 *Cartes* rouges est tel que :

$$C_{27}^{3} \times C_{81-27}^{6} \times 9! = \frac{27!}{3!(27-3)!} \times \frac{54!}{6!(54-6)!} \times 9! = 2925 \times 25827165 \times 362880 = 27413572782960000$$

7 On en déduit la probabilité P_{3CR} d'obtenir une *Table* contenant exactement 3 cartes rouges :

$$P_{3CR} = \frac{casFavorables}{casTotal} = \frac{C_{27}^{3} \times C_{81-27}^{6} \times 9!}{A_{81}^{9}} = \frac{27413572782960000}{94670977328928000} = 0.28956680871386137$$

2.3.2 Fonction *proba3CR*

8 Fonction qui retourne le fréquence de l'événement 3RC parmi *nbEssai* entré en paramètre.

```java
/**
 * FONCTION AJOUTEE
 * Action : Pour un paquet de 81 cartes distincts (3 Couleurs/ nbFiguresMax=3/ 3 Figures / 3 Textures possibles par carte) :
 * calcul experimentalement et retourne la frequence d'obtenir une Table (arrangement de 9 cartes distincts parmi les 81 du Paquet) contenant exactement 3 Cartes Rouges.
 * Valeur theorique : 0.28956680871386137
 * @param nbEssai nombre d'essai
 */
public static double proba3CR(int nbEssai) {
    double countCasFavorable = 0;
    Paquet paq;
    for (int j = 0; j < nbEssai; j++) {
        paq = new Paquet(
            Couleur.valuesInRange(0, 3),
            3,
            Figure.valuesInRange(0, 3),
            Texture.valuesInRange(0, 3)
            );
        Carte[] tab = paq.piocher(9);
        double countRed=0;
        for (int k = 0; k < tab.length; k++) {
            if (tab[k].getCouleur() == Couleur.ROUGE) {
                countRed++;
```

```
23                  }
24              }
25              if (countRed==3) {
26                  countCasFavorable++;
27              }
28          }
29          return countCasFavorable;
30      }
```

Code 2.2: *Fonction proba3CR*

2.3.3 Traitements de données et analyse graphique

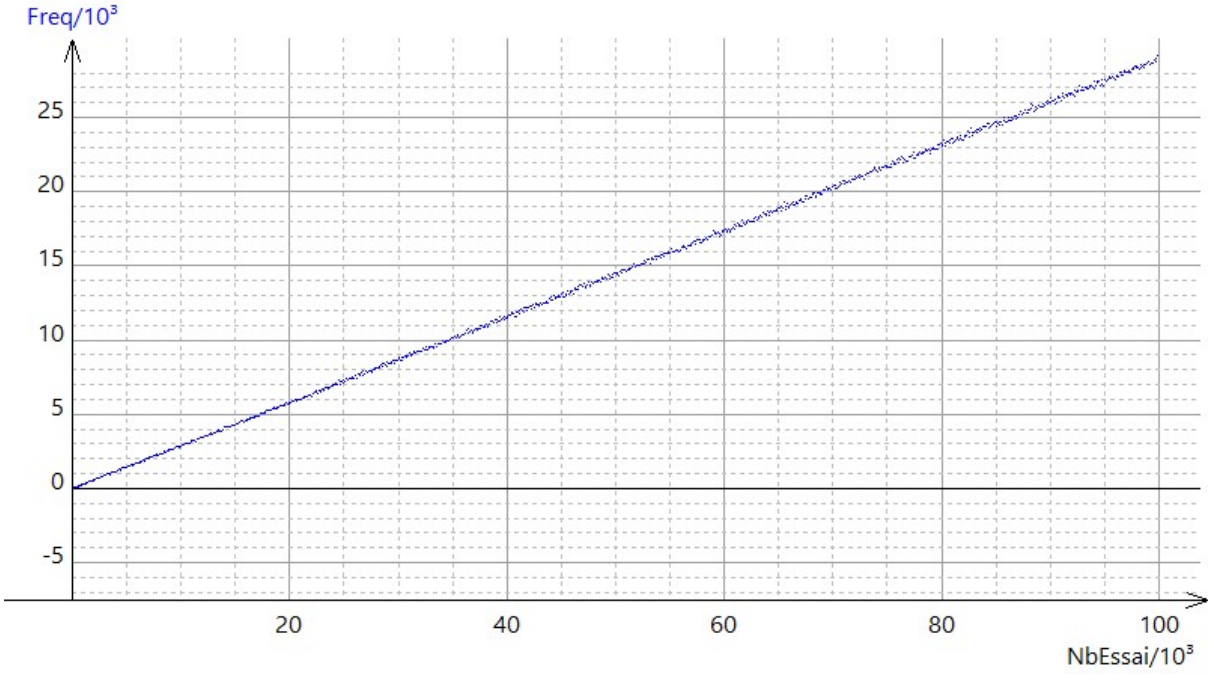

Figure 2.1: Graphique de $Freq = f(nbEssai)$ en variant $nbEssai$ de 100 à 100 000 par pas de 100

Les graphiques obtenus à partir des données générées par cette fonction permettent de déterminer une valeur expérimental de $P_{3CR}exp$. En effet, en traçant $Freq = f(nbEssai)$ nous apercevons que la courbe varie linéairement. Ceci est en accord avec la théorie puisque $P_{3CR}theo = \dfrac{casFavorables}{casTotal}$ soit $casFavorables = P_{3CR} \times casTotal$ ce qui équivaut à $Freq = a \times nbEssai$. En réalisant une modélisation linéaire grâce à l'outils de modélisation sur le logiciel *Regressi* ; nous en déduisons :

$$P_{3CR}exp = a = 0.28939 \pm 0.00015$$

Nous pouvons calculer le pourcentage d'erreur Err telle que :

$$Err = abs\left(\frac{P_{3CR}theo - P_{3CR}exp}{P_{3CR}theo}\right) \times 100 = 0.061\%$$

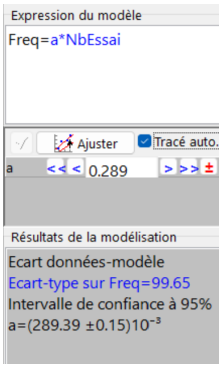

Figure 2.2: Modélisation linéaire de $Freq(nbEssai)$ sur *Regressi*

Pour déterminer à partir de combien d'essais il faut pour obtenir un résultat fiable, nous avons choisi de tracer $Err = f(nbEssai)$ à partir des probabilités propres à chacun des événements (associé au nombre d'essais) : $P = Freq/nbEssai$.

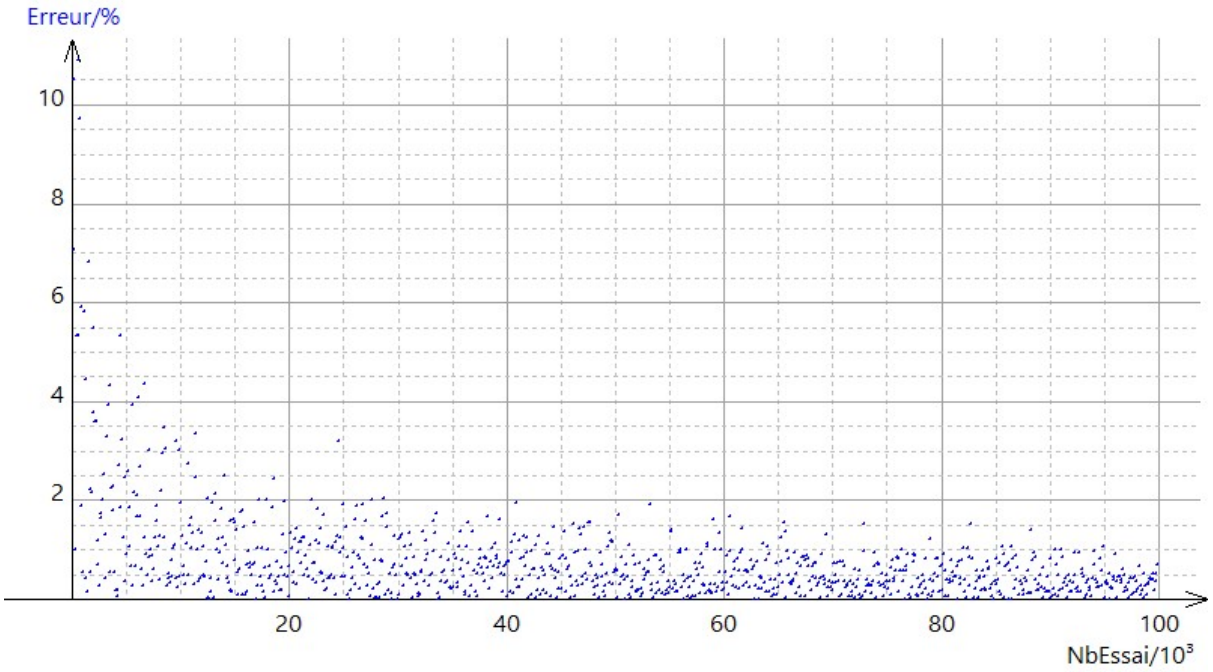

Figure 2.3: Graphique de $Erreur = f(nbEssai)$ en variant $nbEssai$ de 100 à 100 000 par pas de 100

Nous observons que pour avoir un résultat fiable à 5% d'erreur près, il faut au moins environ 10 000 essais. Le graphique ci-dessous de $P = f(nbEssai)$ illustre bien la répartition des probabibilités qui converge vers la valeur théorique en augmentant le nombre d'essais.

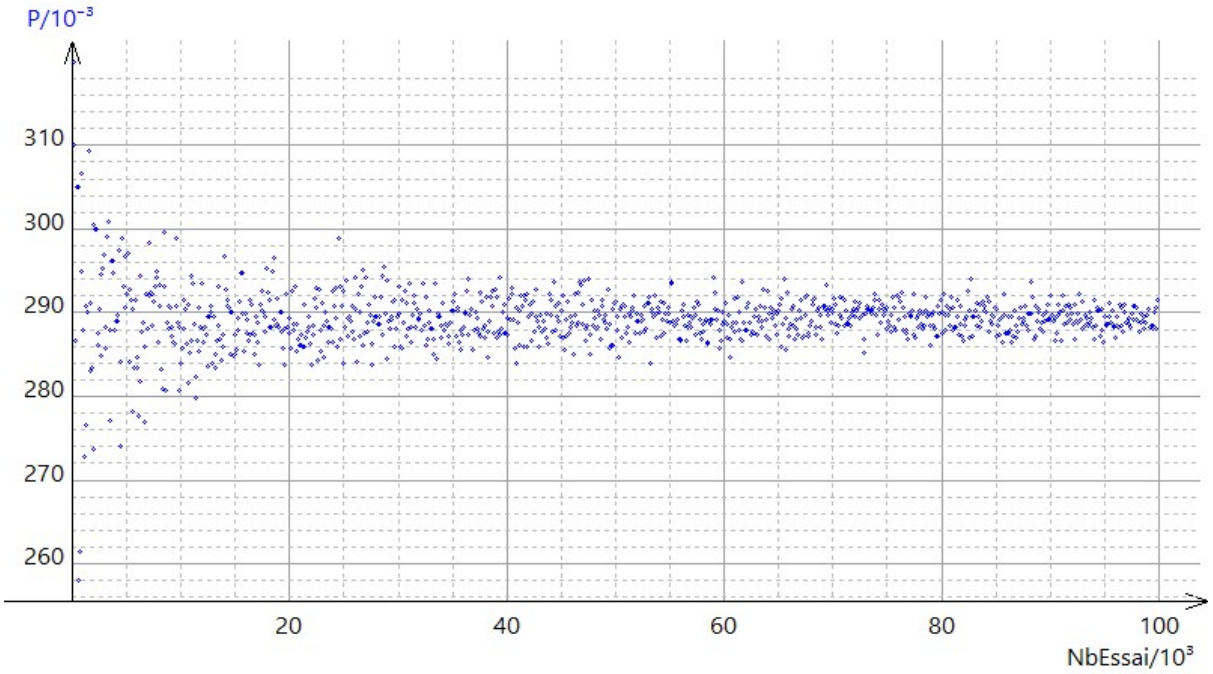

Figure 2.4: Graphique de $P = f(nbEssai)$ en variant $nbEssai$ de 100 à 100 000 par pas de 100

2.4 Etude du cas 3CR&2CL

2.4.1 Caluls théoriques

8 L'intersection de l'ensemble des cartes rouges et des cartes ayant au moins un losange n'est pas vide. Il faut étudier chacun des cas et diviser les ensembles en ensembles disjoints. Il y a 27 cartes rouges distincts et 27 cartes ayant au moins un losange. Parmi ces cartes rouges seulement $1 \times 3 \times 2 \times 3 = 18$ n'ont pas de losange. De même, parmi les cartes ayant au moins un losange, $2 \times 3 \times 1 \times 3 = 18$ ne sont pas rouges. Ainsi, $1 \times 3 \times 1 \times 3 = 9$ cartes sont à la fois rouges et ont au moins un losange.

Carte	R∧L	R∧¬L	L∧¬R	¬(R∨L)	SUM
Total	9	18	18	36	81
Cas 1	2	1	0	6	9
Cas 2	1	2	1	5	9
Cas 3	0	3	2	4	9

Table 2.1: Tableau récapitulatif des différents cas possibles pour obtenir une combinaison de 9 cartes avec exactement 3 cartes rouges (R) et 2 cartes ayant au moins un losange (L) :

On en déduit le nombre d'arrangements possibles de *Table* contenant exactement 3 cartes rouges et 2 cartes ayant au moins un losange tel que :

$$\begin{aligned} &9! \times (C_9^2 \times C_{18}^1 \times C_{18}^0 \times C_{36}^6 \\ &+ C_9^1 \times C_{18}^2 \times C_{18}^1 \times C_{36}^5 \\ &+ C_9^0 \times C_{18}^3 \times C_{18}^2 \times C_{36}^4) \\ &= 362880 \times 17960464368 = 6517493309859840 \end{aligned}$$

On en déduit la probabilité $P_{3CR\&2CL}$ de cet événement :

$$P_{3CR\&2CL} = \frac{casFavorables}{casTotal} = \frac{6517493309859840}{94670977328928000} = 0.06884362550959248$$

2.4.2 Fonction *proba3CR&2CL*

```java
/**
 * FONCTION AJOUTEE
 * Action : Pour un paquet de 81 cartes distincts (3 Couleurs/ nbFiguresMax=3/ 3 Figures / 3 Textures possibles par carte) :
 * calcul experimentalement et retourne la frequence d'obtenir une Table (arrangement de 9 cartes distincts parmi les 81 du Paquet) contenant exactement 3 Cartes Rouges et 2 cartes ayant au moins 1 Losange.
 * Valeur theorique : 0.06884362550959248
 * @param nb nombre d'essai
 */
public static double proba3CRAnd2CL(int nb) {
    double countCasFavorable = 0;
    Paquet paq;
    for (int i = 0; i < nb; i++) {
        paq = new Paquet(
            Couleur.valuesInRange(0, 3),
            3,
            Figure.valuesInRange(0, 3),
            Texture.valuesInRange(0, 3)
            );
        Carte[] tab = paq.piocher(9);
        double countRed=0;
        double countLosange=0;
        for (int j = 0; j < tab.length; j++) {
            if (tab[j].getCouleur() == Couleur.ROUGE) {countRed++;}
            if (tab[j].getFigure() == Figure.LOSANGE) {countLosange++;}
        }
        if (countRed==3 && countLosange==2) {
            countCasFavorable++;
        }
    }
    return countCasFavorable;
}
```

Code 2.3: *Fonction proba3CRAnd2CL*

2.4.3 Traitements de données et analyse graphique

De la même manière que pour le cas 3CR, nous obtenons via une modélisation linéaire :

$$P_{3CR\&2CL}exp = 0.06883 \pm 0.00006$$

Nous pouvons calculer le pourcentage d'erreur Err telle que :

$$Err = abs\left(\frac{P_{3CR\&2CL}theo - P_{3CR\&2CL}exp}{P_{3CR\&2CL}theo}\right) \times 100 = 0.020\%$$

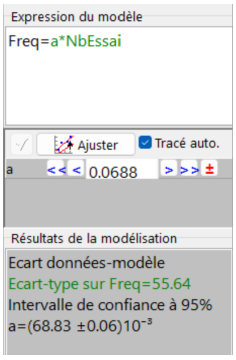

Figure 2.5: Modélisation linéaire de $Freq(nbEssai)$ sur *Regressi*

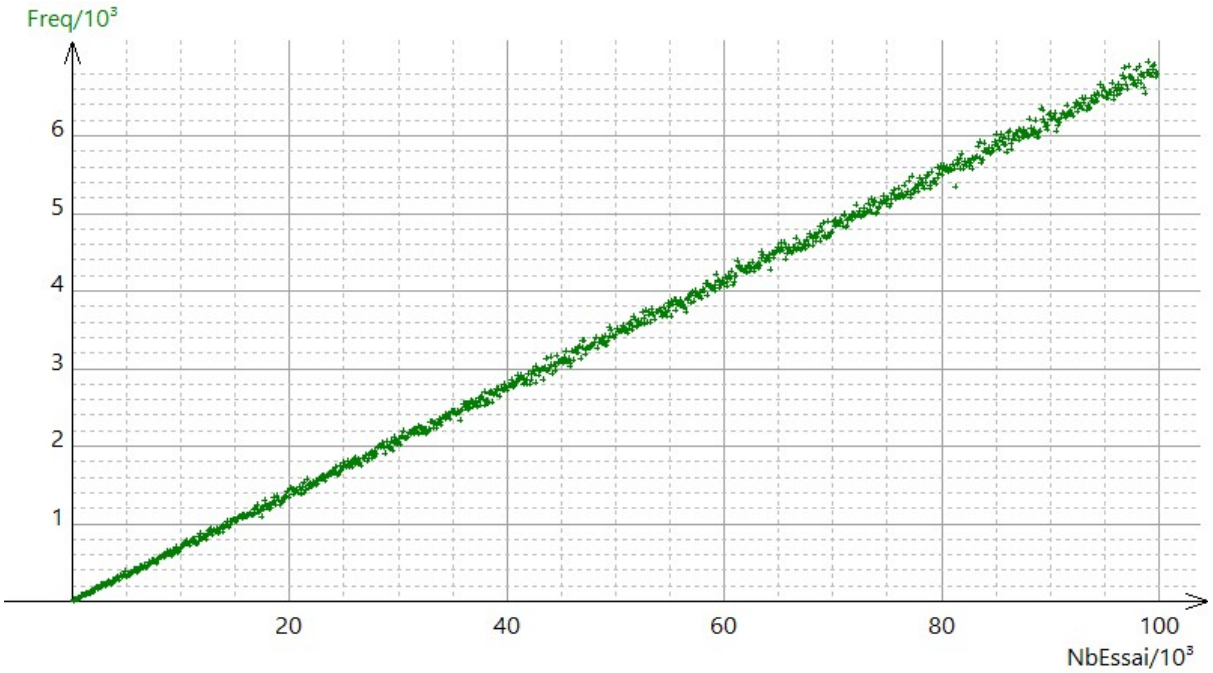

Figure 2.6: Graphique de $Freq = f(nbEssai)$ en variant $nbEssai$ de 100 à 100 000 par pas de 100

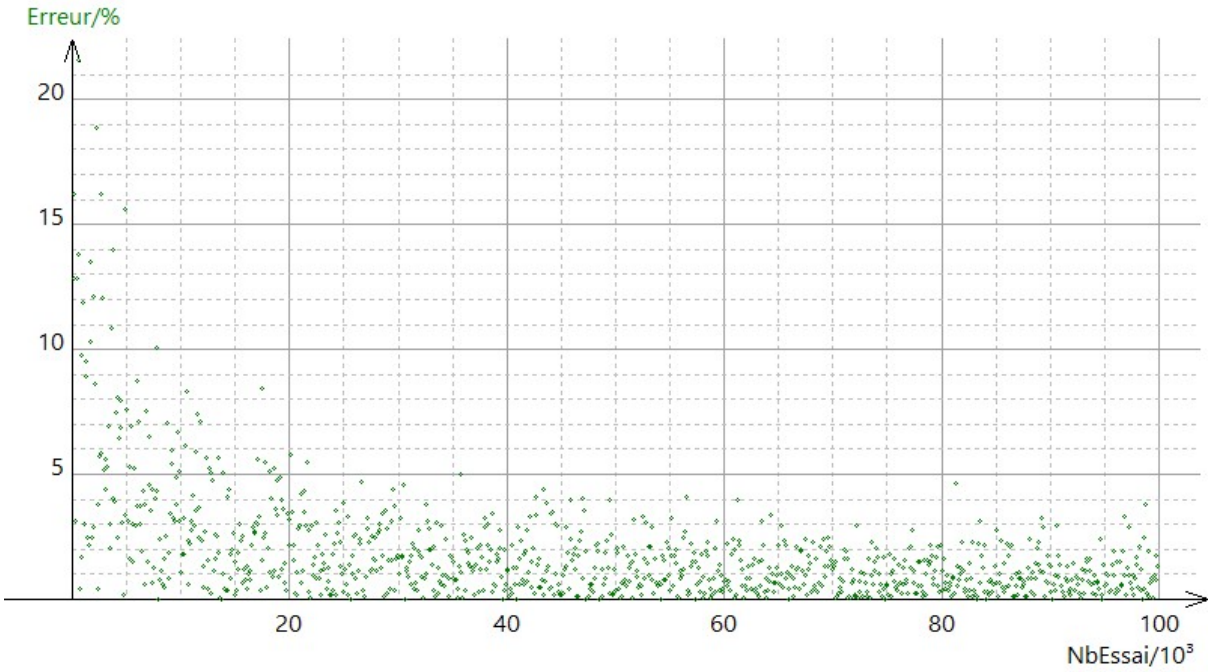

Figure 2.7: Graphique de $Erreur = f(nbEssai)$ en variant $nbEssai$ de 100 à 100 000 par pas de 100

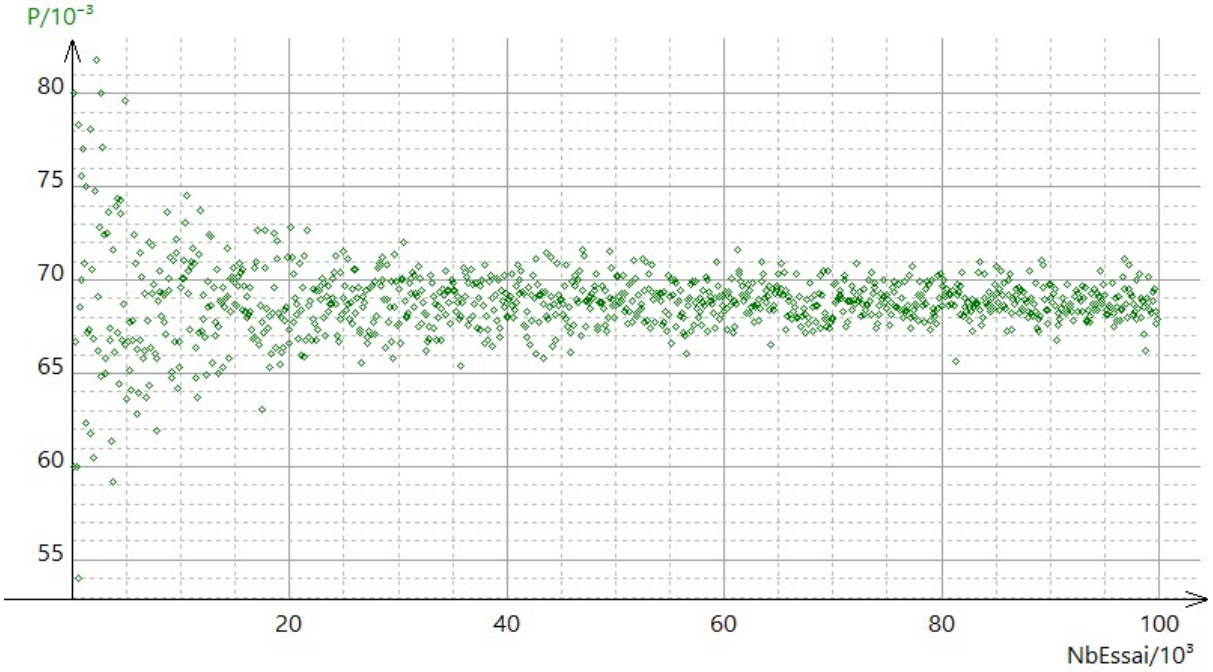

Figure 2.8: Graphique de $P = f(nbEssai)$ en variant $nbEssai$ de 100 à 100 000 par pas de 100

2.5 Etude du cas E3C

2.5.1 Fonctions *probaE3C* et *trouverE3C*

```java
/**
 * FONCTION AJOUTEE
 * Action : Pour un paquet de 81 cartes distincts (3 Couleurs/ nbFiguresMax=3/ 3 Figures / 3 Textures possibles par carte) :
 * calcul experimentalement et retourne la frequence d'obtenir une Table (arrangement de 9 cartes distincts parmi les 81 du Paquet) contenant exactement 3 Cartes Rouges et 2 cartes ayant au moins 1 Losange.
 * Valeur theorique : non connue
 * @param nb nombre d'essai
 */
public static double probaE3C(int nb) {
    double countCasFavorable = 0;
    Paquet paq;
    for (int i = 0; i < nb; i++) {
        paq = new Paquet(
            Couleur.valuesInRange(0, 3),
            3,
            Figure.valuesInRange(0, 3),
            Texture.valuesInRange(0, 3)
            );
        Carte[] tab = paq.piocher(9);
        if (trouverE3C(tab)) {
            countCasFavorable++;
        }
    }
    return countCasFavorable;
}

/**
 * FONCTION AJOUTEE
 * Action : verifie si un ensemble de cartes contient un E3C
 * @param cartes
 * @return vrai si oui, false sinon;
 */
private static boolean trouverE3C(Carte[] cartes) {
    if (cartes.length < 3) {
        return false;
    }
    for(int i = 0; i < cartes.length; i++)
        for(int j = i+1; j < cartes.length; j++)
            for(int k = j+1; k < cartes.length; k++)
                if(Jeu.estUnE3C(new Carte[]{cartes[i], cartes[j], cartes[k]}))
                    return true;
    return false;
}
```

Code 2.4: *Fonction probaE3C et trouverE3C*

2.5.2 Traitements de données et analyse graphique

De la même manière que pour les cas précédents, nous obtenons via une modélisation linéaire :

$$P_{E3C}exp = 0.70280 \pm 0.00011$$

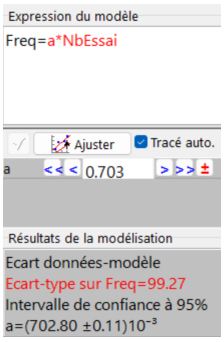

Figure 2.9: Modélisation linéaire de $Freq(nbEssai)$ sur *Regressi*

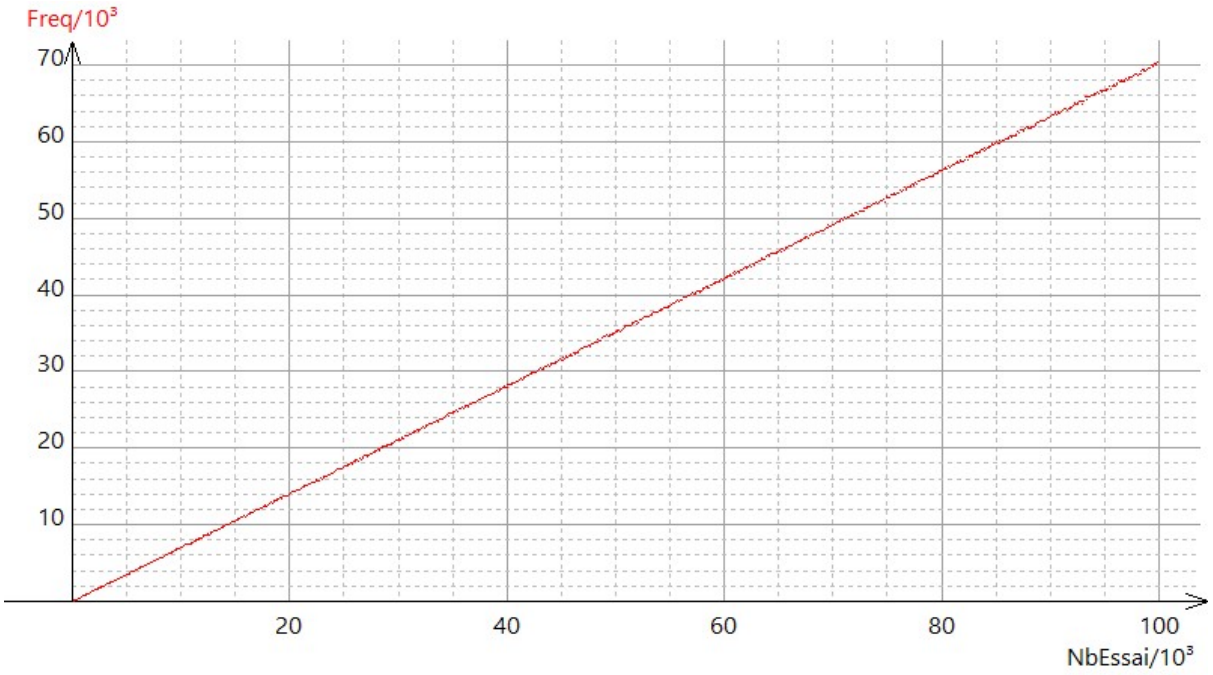

Figure 2.10: Graphique de $Freq = f(nbEssai)$ en variant $nbEssai$ de 100 à 100 000 par pas de 100

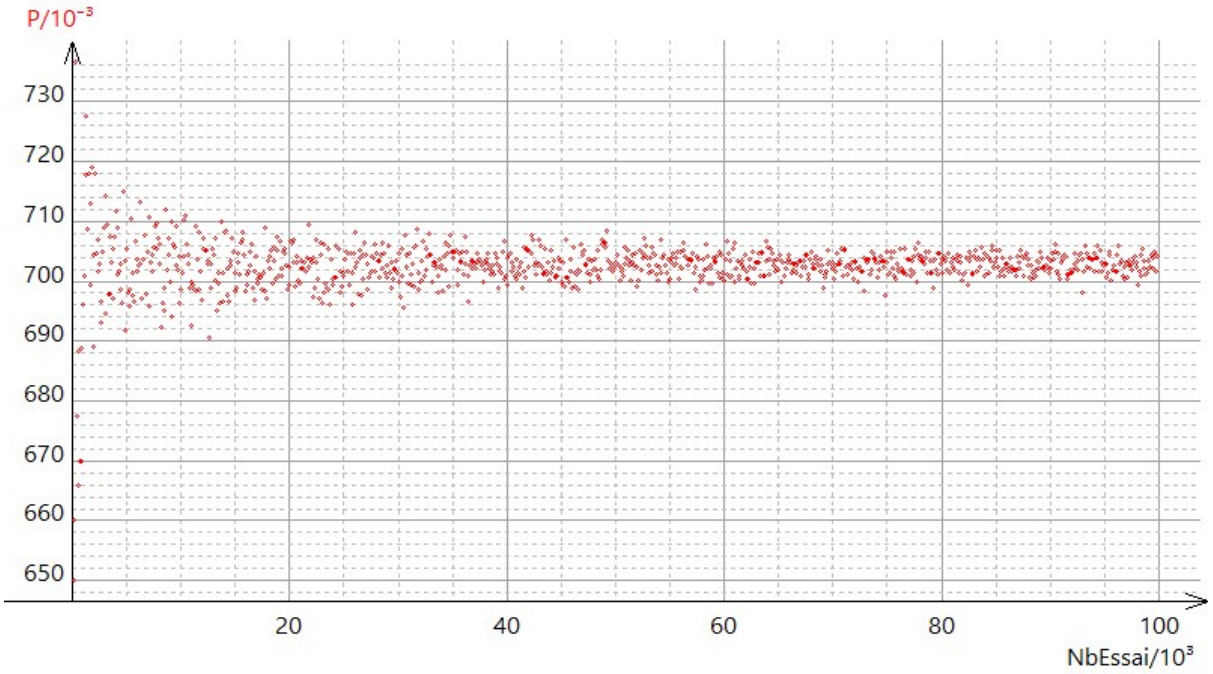

Figure 2.11: Graphique de $P = f(nbEssai)$ en variant $nbEssai$ de 100 à 100 000 par pas de 100

Annexe A

Langages, IDE et logiciels utilisés