Collectionneur de vignettes

3ième année informatique et réseaux TD A
TALEB Nour Eddine
KIZIL Huseyin

ESIREM Dijon

Table des matières

Remerciements

Introduction

Dans le cadre de ce travail, nous explorons une activité populaire : la collection de vignettes. Cette activité consiste à rassembler un ensemble complet de vignettes uniques souvent distribuées aléatoirement dans des paquets de céréales par exemple. Notre étude se concentre sur le temps que peut mettre une personne pour compléter la collection. Combien de paquets de céréales une personne doit-elle acheter pour terminer sa collection de N vignettes différentes ?

Ce sujet nous intéresse pour son lien étroit avec les concepts mathématiques, en particulier la théorie des probabilités. En analysant les probabilités d'obtenir des vignettes déjà possédées ou nouvelles à chaque achat de paquet de céréales, en supposant que la personne achète 1 paquet par semaine, nous cherchons à établir des estimations théoriques pour le nombre moyen de paquets nécessaires pour achever la collection et donc la notion d'ésperances rentrera en jeu.

En outre, nous avons examiné des situations plus complexes, comme les échanges de doublons avec d'autres collectionneurs ou les offres spéciales des fabricants permettant l'acquisition de vignettes spécifiques en échange d'un certain nombre de vignettes (par exemple 10 vignettes contre 1 vignette spécifique). Il est clair que ces éléments vont influencer au niveau du temps de collecte des vignettes et c'est ce que nous allons démontrer durant notre étude.

L'objectif principal de notre étude est donc de développer un modèle mathématique solide pour estimer le temps moyen nécessaire à un collectionneur pour finaliser sa collection de vignettes. Nous analysons également diverses stratégies et conditions qui pourraient affecter ce temps.

Au fil de ce rapport, nous présentons notre approche mathématique, discutons des résultats théoriques obtenus à l'aide des lois de probabilités usuelles et analysons différentes situations pour mieux comprendre le problème du collectionneur de vignettes. Pour cela, nous avons réalisé un programme informatique en C++ et nous avons fait plusieurs simulations à l'aide de RStudio.

Au besoin, les différents codes sont disponibles en annexe, à la fin du document.

Modèle de base

2.1 Étude théorique

Lorsque l'on suppose que toutes les vignettes sont équiprobables, c'est-à-dire que chaque type de vignette a une probabilité de $\frac{1}{n}$ d'être obtenu lors de l'achat d'une boîte, on peut étudier la durée nécessaire pour compléter une collection de vignettes.

Soit T la variable aléatoire qui représente le nombre de paquets à acheter (et par extension, le nombre de semaines nécessaires) pour compléter la collection de vignettes.

L'obtention d'une nouvelle vignette dans un paquet k suit une loi géométrique de paramètre $p = \frac{n-(k-1)}{n}$, où n est le nombre total de vignettes à collecter. Ainsi, $T_{n,k}$, une variable aléatoire représentant le nombre de paquets à acheter pour obtenir une nouvelle vignette lorsque vous en avez déjà k différentes, suit également une loi géométrique de paramètre p.

L'espérance de $T_{n,k}$ est donnée par : $E(T_{n,k}) = \frac{1}{p} = \frac{n}{n-(k-1)}$

Par conséquent, la variable aléatoire T qui représente le temps nécessaire pour compléter la collection entière est la somme de toutes les $T_{n,k}$ où k varie de 1 à n,

$$T = \sum_{k=1}^{n} T_{n,k}$$

En utilisant la linéarité de l'espérance, l'espérance de T est :

$$\begin{aligned}
E(T_n) &= \sum_{k=1}^{n} E(T_{n,k}) \\
&= \sum_{k=1}^{n} \frac{n}{n-(k-1)} \\
&= n\left(\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \ldots + \frac{1}{2} + \frac{1}{1}\right) \\
&= n \cdot \left(1 + \frac{1}{2} + \ldots + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}\right) \\
&= n \cdot \sum_{k=1}^{n} \frac{1}{k}
\end{aligned}$$

D'où

$$E(T_n) = n \cdot H_n$$

En utilisant le développement asymptotique de H_n, on obtient :

$$E(T_n) = n \cdot H_n = n \cdot \ln(n) + \gamma \cdot n + \frac{1}{2} + o(1)$$

Où $\gamma \approx 0.5772156649$ est la constante d'Euler-Mascheroni.[1]

2.2 Simulations

2.2.1 Simulation en c++

Les simulations Monte Carlo [2] sont utilisées pour trouver des solutions (approximatives) à des problèmes de probabilité, surtout en inférence statistique lorsque des solutions exactes ne sont pas connues ou sont très difficiles à calculer. Le problème du collectionneur de vignettes, tel que nous le nommons, est particulièrement adapté à cela.

La solution Monte Carlo est simplement un algorithme : nous simulons un certain nombre de paquets de céréales dans lequels des vignettes sont présentes et comptons combien de paquets sont nécessaires pour remplir un une collection séquentiellement. Après avoir répété cette procédure un certain nombre de fois, on calcule simplement la moyenne du nombre de paquets nécessaires pour remplir les albums. Pour ce faire, nous avons codé en language C++ et nous avons réalisé les courbes à l'aide du logiciel Rstudio.

L'algorithme doit reproduire les conditions décrites dans le problème : ici, l'album comporte par exemple 500 vignettes, chaque paquet contient d'abord 1 vignette pour le cas simple puis nous verrons avec 5 vignettes distinctes, et les vignettes seront considérées comme réparties uniformément parmi les paquets. Ses étapes sont les suivantes :

1. Nous avons créer un tableau avec les numéros des vignettes que nous possédons. Au départ, sa longueur est de zéro.
2. Nous avons pris k = 1 où k est le nombre de paquets.
3. Nous avons simulé un paquet en prenant un échantillon de taille 1, sans remplacement, à partir d'une urne contenant 500 balles numérotées de 1 à 500.
4. Nous avons vérifié la vignette dans le paquet de l'étape 3 par rapport à celles dans le tableau de l'étape 1 et nous avons inclus toute vignette qui n'est pas encore présente. Nous Supprimons les vignettes que nous possédons déjà.
5. Si le tableau a une longueur de 500, alors on arrête. Sinon, on incrémente k de un et on retourne à l'étape 3.

En éxecutant notre programme, voici ce qu'on obtient :
Dans un premier temps, nous entrons nos différentes valeurs (voir Figure 1 1).

```
Entrez nombre de vignettes de votre collection (0 pour arreter programme)
Ici >  500
Entrez nombre de simulations (0 pour arreter programme)
Ici >  1000
Entrez nombre de vignettes (0 pour arreter programme)
Ici >  5
```

Figure 1 – Entrée valeurs simulation

Voici les résutats obtenus (voir les figures 2) :

```
-Simulation :
>> Il faut en moyenne 3402.73 semaines pour completer la collection de 500 vignettes sans echange.
```

```
>> Il faut en moyenne 685.576 semaines pour completer la collection de 500 vignettes avec 5 dans la boite de cereales
```

Figure 2 – Résultats de la simulation

```
-Valeur theorique :
>> Il faut en moyenne 3396.41 semaines pour completer la collection de 500 vignettes sans echange a l'aide de la formule
 theorique n*somme(1/k).
>> Il faut en moyenne 3396.41 semaines pour completer la collection de 500 vignettes sans echange a l'aide de l'approxim
ation de la forumule theorique.
```

FIGURE 3 – Résultats théoriques

Soit T la variable aléatoire représentant le nombre de paquets nécessaires pour compléter la collection. Nous nous intéressons à la valeur attendue de T, donc nous répétons simplement la procédure ci-dessus pour un grand nombre d'albums et prenons la moyenne arithmétique des valeurs finales de k. À titre d'exemple, nous avons exécuté l'algorithme pour 1000 "albums" et avons obtenu 3482,73 comme estimation du nombre attendu de paquets nécessaires pour compléter un album avec 1 vignette dans un paquet. Il convient de souligner que, en tant qu'approximation stochastique, l'algorithme donne généralement une réponse différente à chaque exécution (la réponse 3483 mentionnée précédemment vient de l'arrondi à l'entier supérieur de 3482,73). En comparant cela à la valeur théorique qui de 3397 en prenant l'arrondi supérieur (voir Figure 3), on voit une petite différence.

Et pour la simulation avec 5 vignettes dans un paquet, nécéssairement,le temps de collecte diminue (voir Figure 2).

2.2.2 courbe sous Rstudio

Nous avons réalisé une simulation du problème du collectionneur de vignettes en modélisant le processus d'achat de paquets de céréales. Nous avons suivi l'évolution du nombre de vignettes collectées au fil du temps, et les résultats ont été représentés graphiquement.

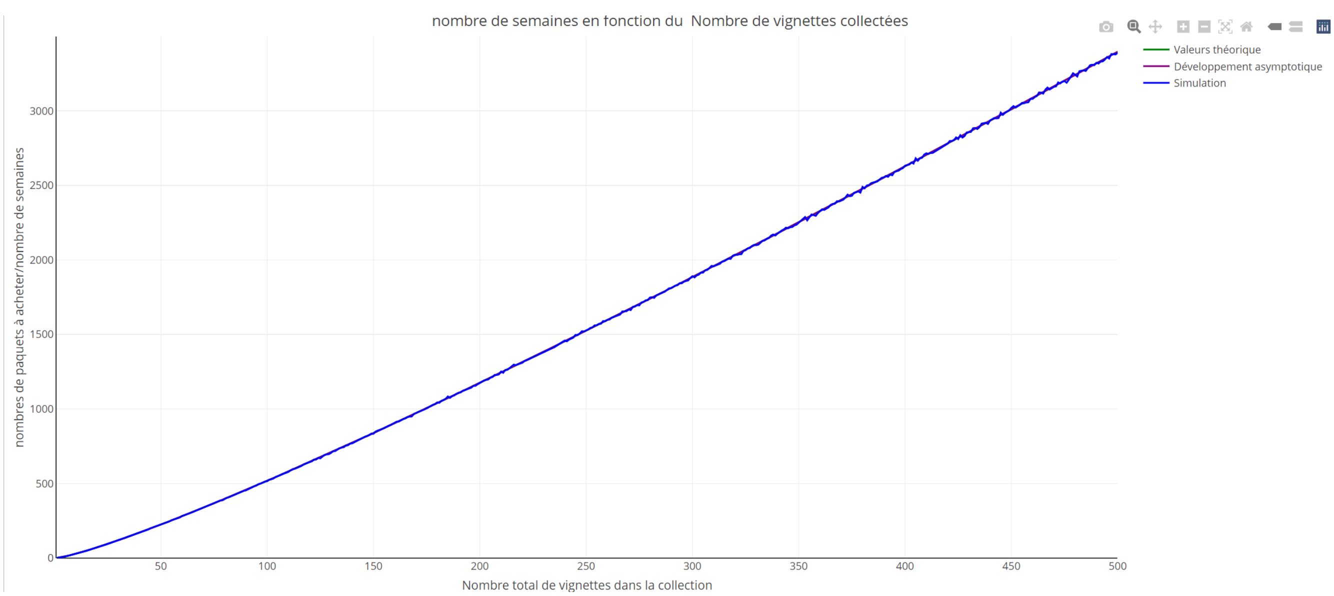

FIGURE 4 – Évolution du nombre de vignettes collectées au fil du temps

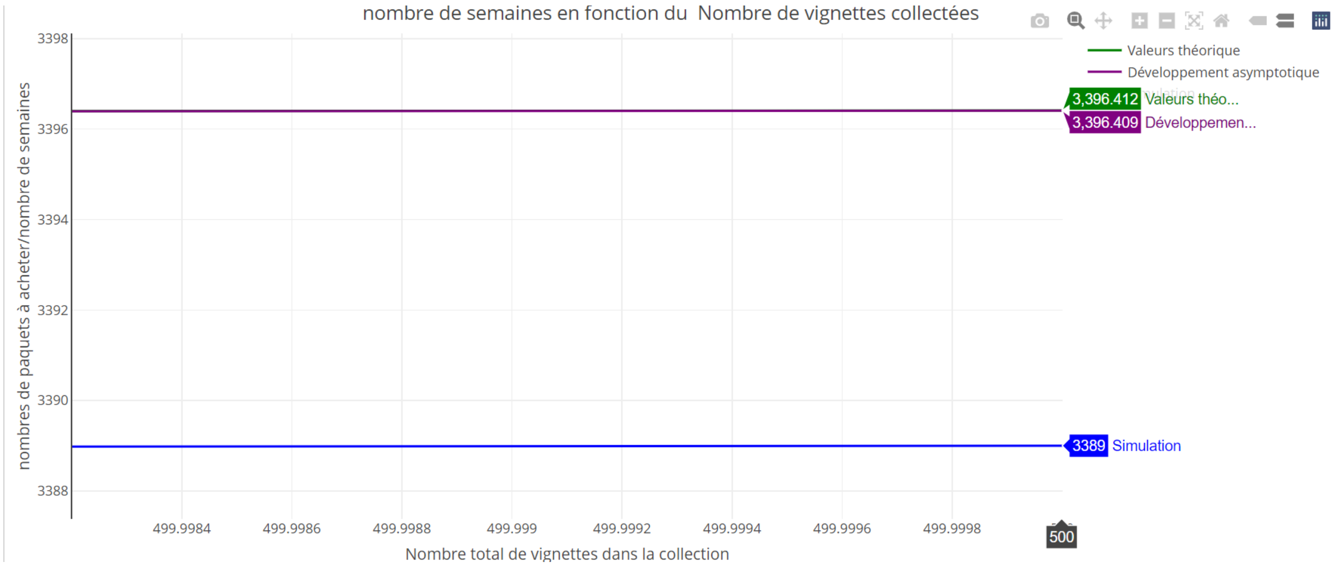

FIGURE 5 – Zoom

La Figure 4 illustre l'évolution du nombre de vignettes collectées en fonction du temps. A noter que dans cet exemple, il n'y a qu'une vignette dans le paquet de céréale. Nous pouvons tirer plusieurs observations et conclusions à partir de cette représentation graphique.

Tout d'abord, l'ascension initiale de la courbe indique une phase où de nombreuses vignettes uniques sont collectées rapidement au début de la période d'achat. Cela reflète la probabilité élevée d'obtenir des vignettes manquantes au début de la collection.

Ensuite, au fur et à mesure que le temps progresse, la courbe présente une pente moins prononcée, indiquant que la collecte de nouvelles vignettes devient plus lente. Cela pourrait correspondre à la phase où le collectionneur commence à accumuler des vignettes en double et trouve moins fréquemment de nouvelles vignettes.

Nous remarquons par ailleurs que la courbe est de la forme $n \cdot H_n$.

Ces observations préliminaires nous inciteront à approfondir notre analyse, à examiner les stratégies utilisées et à comparer les résultats obtenus. Cette analyse détaillée sera présentée dans la section suivante.

2.3 Possibilité d'échange

2.3.1 Algorithme

A comple
ter

2.3.2 Comparaison entre cas normal et possibilité d'échanger 10 vignettes

A compé
ter

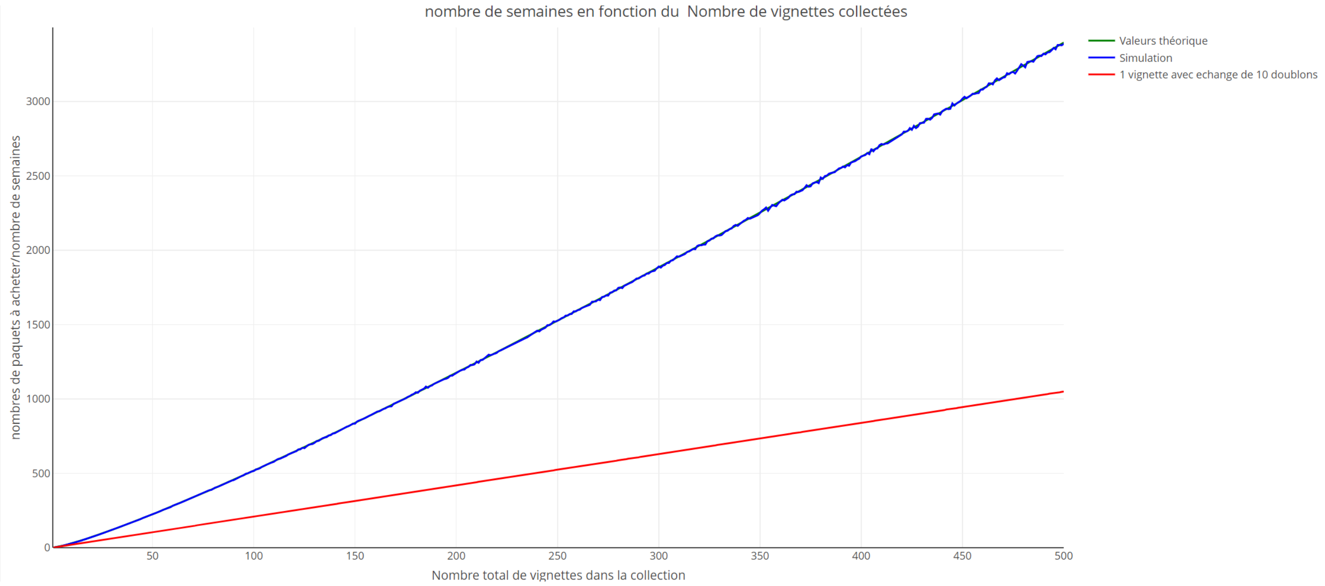

FIGURE 6 – Nombre de semaines en fonction du nombre de vignettes collectées

A compéter

2.3.3 influence du prix d'échange

A complé ter

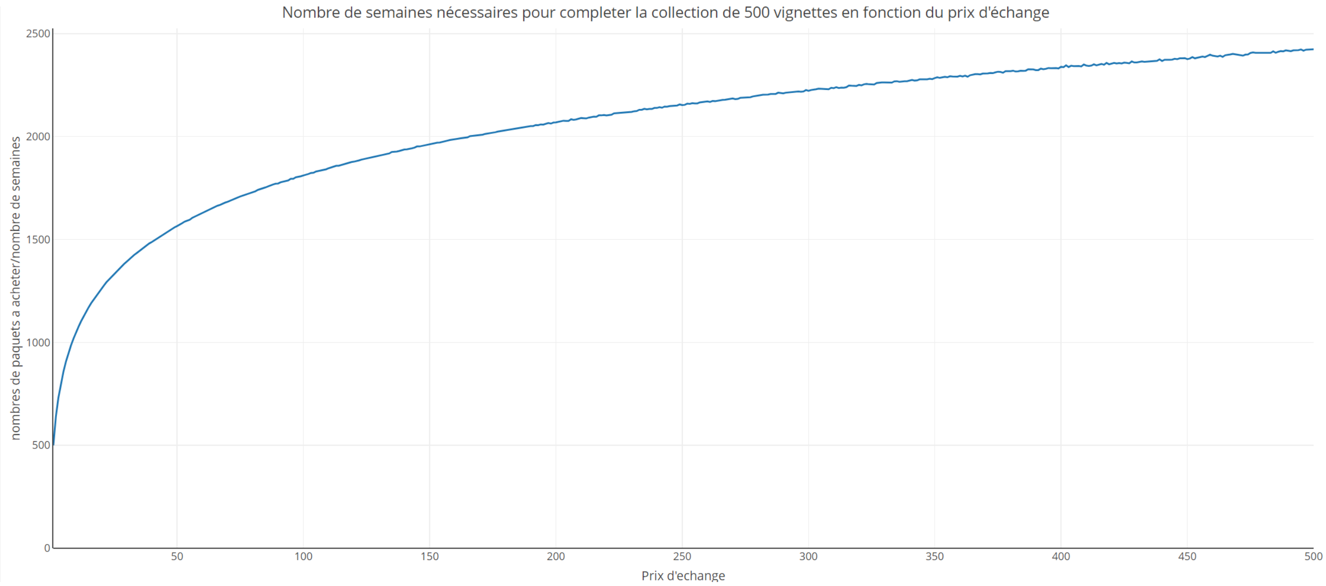

FIGURE 7 – Influence prix d'échange

Généralisation : Plusieurs vignettes

3.1 Etude théorique et approximation

A modifier

Dans le problème original du collectionneur de coupons, chaque paquet a une seule vignette. Ainsi, le nombre de paquets T nécessaires pour compléter la collection est une somme de variables aléatoires géométriques avec une probabilité variable de trouver une nouvelle vignette. La valeur attendue de cette variable est simplement la somme des valeurs attendues de telles variables aléatoires géométriques,

$$E(T) = n\left(1 + \frac{1}{2} + \cdots + \frac{1}{N}\right).$$

La série à l'intérieur des parenthèses est la N-ième somme harmonique. Dans notre problème, chaque paquet ayant cinq vignettes différents, une approximation est $E(T)/5$. Quelles sont les approximations pour la somme harmonique N-ième ? Souvent, elle est approximée par $\ln(n)$. Cependant, une approximation différente est donnée par

$$N\left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right) \approx n\ln(n) + \gamma n + \frac{1}{2},$$

où $\gamma \approx 0.57721$ est la constante d'Euler–Mascheroni. Pour , cette approximation donne $E(T)/2 \approx$., beaucoup plus proche des solutions trouvées ci-dessus.

Dans la suite de ce chapitre nous allons prendre le cas de deux vignettes dans le céréale comme exemple pour étude de plusieurs vignettes.

3.2 Simulation

3.2.1 Algorithme et code C++

A comple-
ter

3.2.2 Courbe sous R

A comple-
ter

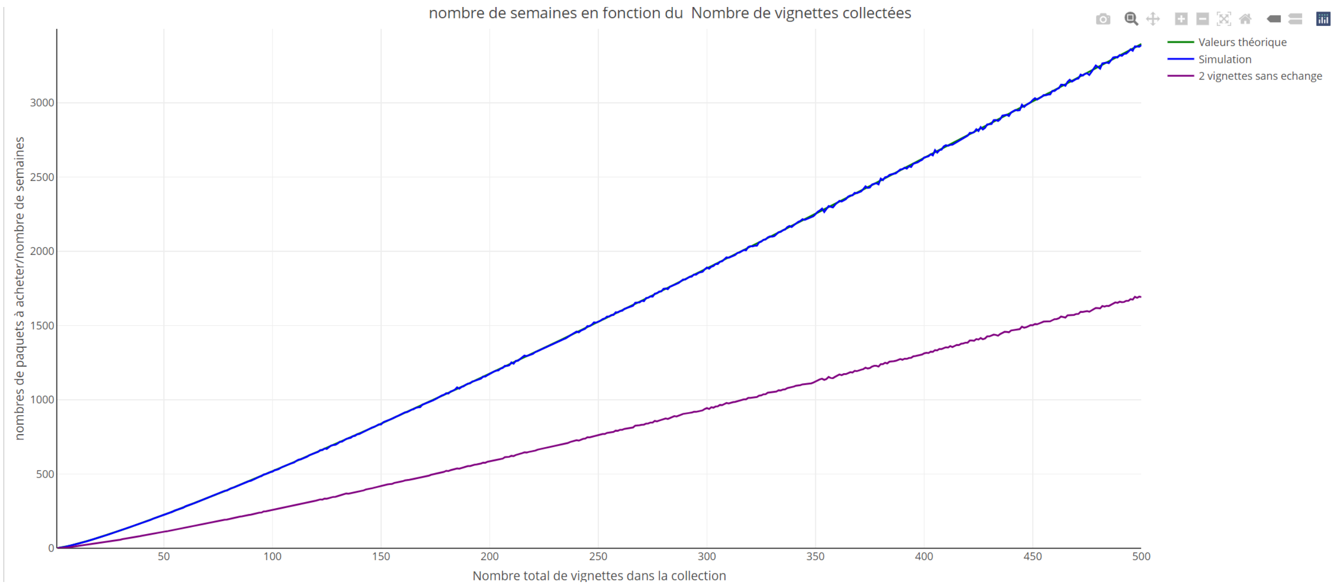

FIGURE 8 – Nombre de semaines en fonction du nombre de vignettes collectées.

3.3 Plusieurs vignettes avec échange

3.3.1 Algorithme et code C++

A comple ter

3.3.2 Simulation et comparaison entre les différentes cas et possibilités

A comple ter

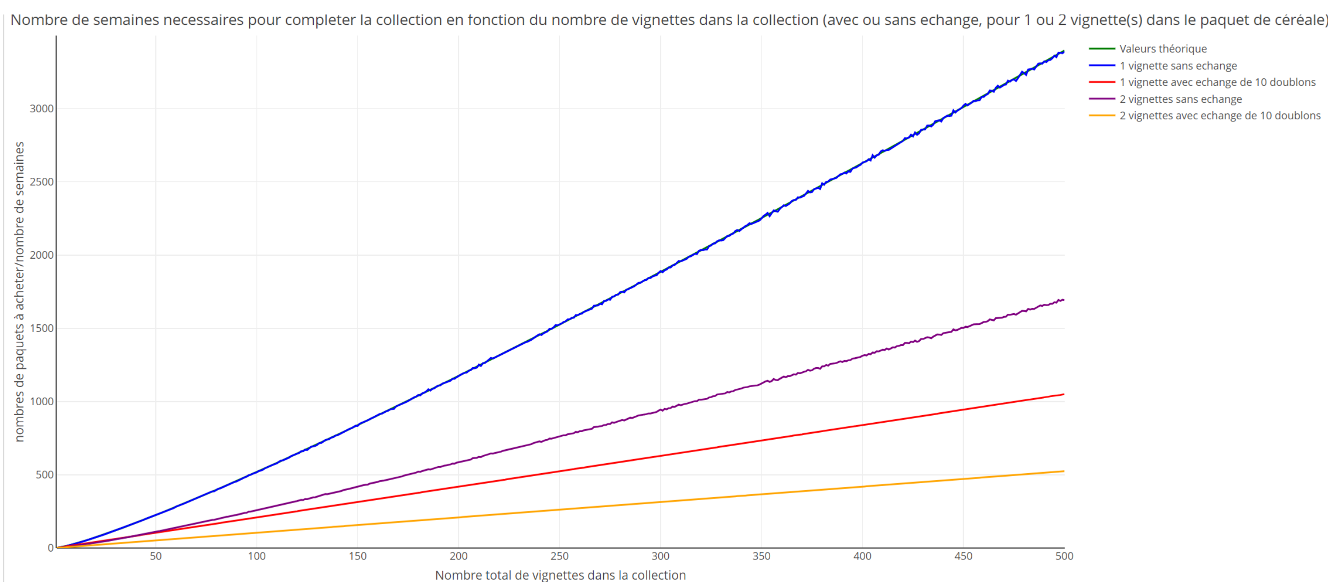

FIGURE 9 – Comparaison

Ajout d'un tableau et utiliser premier code simulation

3.4 Determiner point de croisement

A amélio rer

Dans cette partie nous allons s'interesser à l'influence du prix d'échange sur le temps de collecte. En particulier, pour le cas ou on a 1 vignette dans la boite de céréale avec possibilité d'échange (10 doublons quelconque contre 1), quel est le prix d'échange à prendre afin de pourvoir finir la collection dans la meme temps si on a 2 vignettes dans la boite de céréale sans la possibilité d'échange....

3.4.1 Exemple avec 2 vignettes sans et echange et 1 vignette avec echange

A amélio
rer

Remarque : afin de determiner le point de croissement nous allons ajouter comme conditions, en plus de la condition d'égalité, le cas ou la courbe en dessous (rouge) dépasse la courbe violette vu qu'on a une distribution ponctuelle des valeurs...

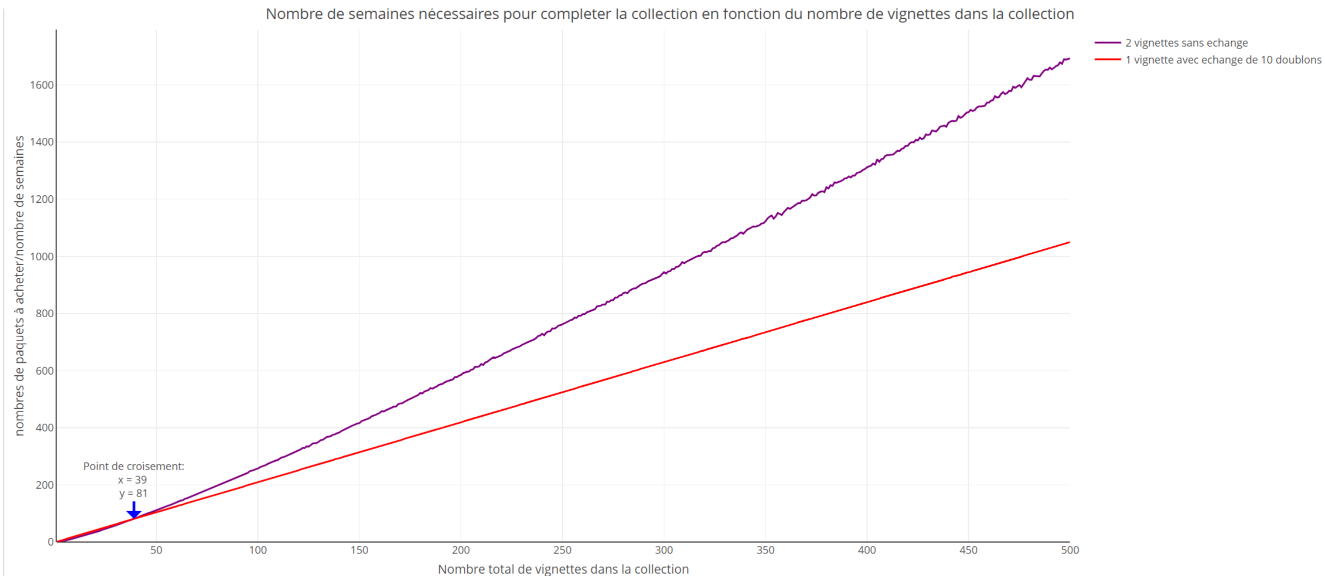

FIGURE 10 – Point de croissement

Pour les différentes valeurs du prix d'échange on observe que le point de croissement s'éloigne du 0 si on augmente le prix d'échange, et inversement il s'approche de 0 si on dimunue le prix d'échange....

par la suite nous allons

3.4.2 Différente point de croissement

A compe
ter

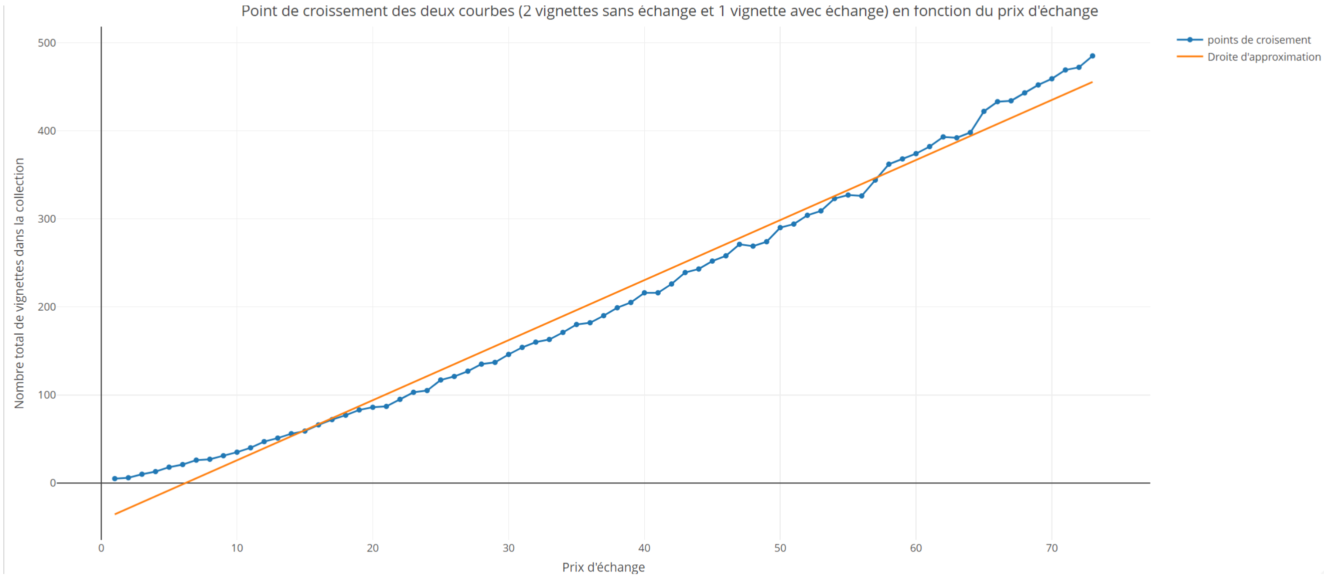

FIGURE 11 – Ensemble des points

3.5 Influence du prix d'echange sur le nombre de semaines

A complé
ter

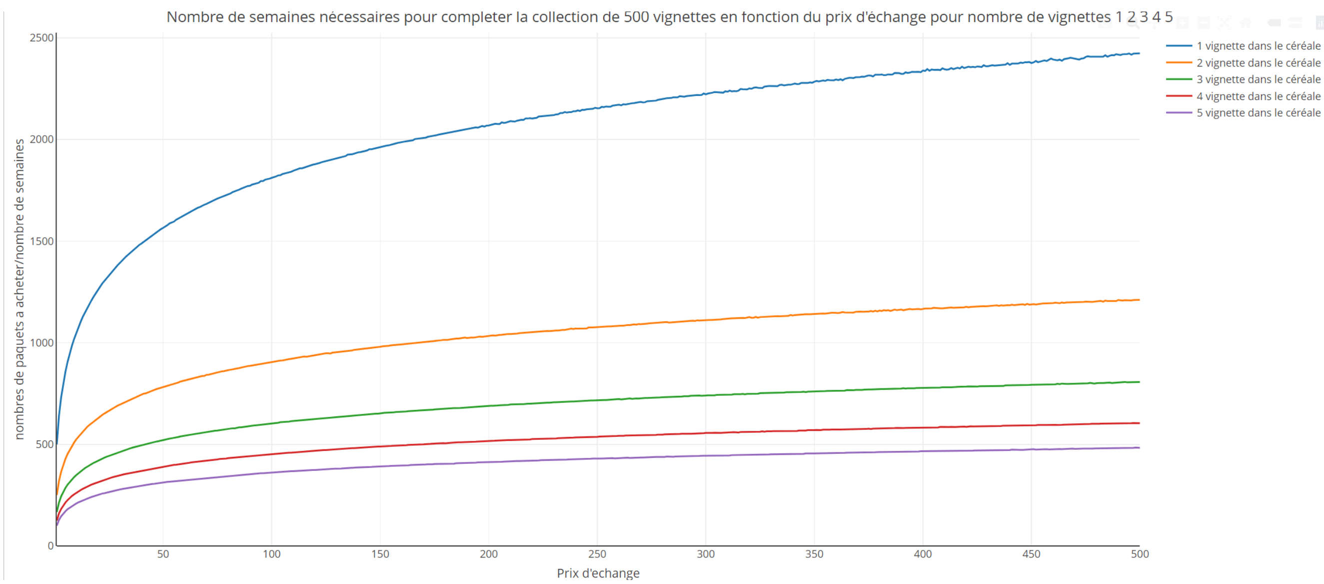

FIGURE 12 – Nombre de semaines en fonction du prix d'échange pour une collection de 500 vignettes

Comparaison et résultat

Dans ce chapitre on va discuter les différentes approche et méthode pour estimer le nombres de céréale à acheter et leur exactitude.

4.1 Les méthodes

4.2 Tableau de comparaison

Stratégies de Collecte

5.1 Collecte de Vignettes avec possibilité d'échange

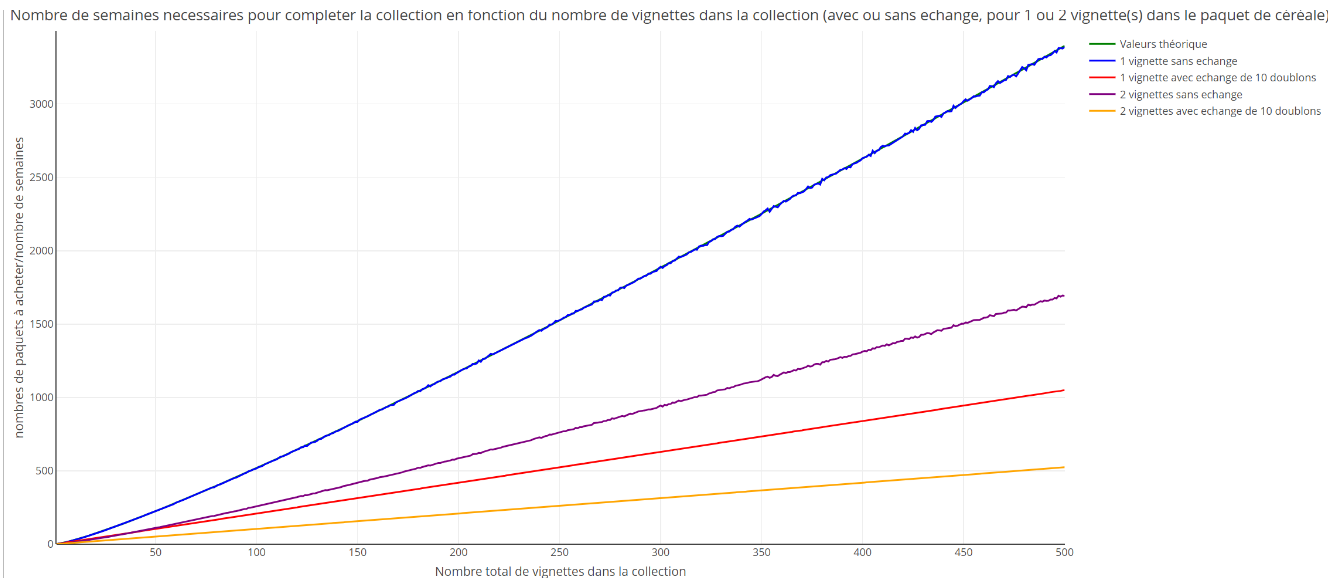

FIGURE 13 – Evolution du temps en fonction du nombre de vignettes dans la collection avec diverses cas

Efficacité des stratégies sans échange : - il est clair que la courbe bleue (1 vignette sans échange) est au-dessus de la violette (2 vignettes sans échange) car nous avons plus de chance de trouver une nouvelle vignette (non obtenues jusqu'à présent) avec 2 vignettes dans le paquet autrement dit, la collection peut être obtenue plus rapidement avec 2 vignettes dans le paquet.

Impact des échanges : -

Point de croisement des courbes rouge et violet : - le point de croisement des coordonnées x=31 et y=81 où la courbe rouge est au-dessus de la courbe violet. - on peut remarquer qu'avant ce point, la courbe rouge est au-dessus de la courbe violette. Ceci vient du fait que les premières vignettes sont plus facilement trouvable et vu que la courbe violette représente 2 vignettes dans le paquet et donc qu'il y a 2 fois plus de chance de trouver une nouvelle vignette, la courbe violet est plus rapide au début en terme de temps de collection. Mais, lorsque la collection augmente (et donc N augmente), on voit tout de suite le changement de rôle entre les 2 courbes. En effet, la violet passe au-dessus et diverge même s'il y a 2 fois plus de vignettes dans le paquet, si on considère qu'il n'est pas possible d'échanger, il est clair qu'on va prendre plus de temps à collectionner l'ensemble des vignettes alors qu'avec 1 vignette dans le paquet avec les doublons que nous obtenons, nous pouvons les échanger et obtenir une nouvelle vignette et donc la collection se termine plus vite.

5.2 Collecte de Vignettes avec ...

5.3 Collecte de Vignettes avec ...

Conclusion

Mes conclusions

Table des figures

Liste des tableaux

Bibliographie

[1] Problème du collectionneur de vignettes. [Wikipédia] : https://fr.wikipedia.org/wiki/ProblÃÍme_du_collectionneur_de_vignettes.

[2] M. A. Diniz, D. Lopes, A. Polpo, and L. E. B. Salasar. The sticker collector's problem. *The College Mathematics Journal*, 47(4) :255–263, 2016. Published by the Mathematical Association of America.

Abstract

In English

Résumé

En Français

Annexe : Fonctions de simulation

```cpp
#ifndef COUPON_COLLECTOR_S_SIMULATION_FUNCTIONS_H
#define COUPON_COLLECTOR_S_SIMULATION_FUNCTIONS_H

#include <iostream>
#include <cmath>

int simulateCollection(int collectionNumber);

int simulateCollectionWithExchange(int collectionNumber);

int simulateWithMultipleCollections(int collectionNumber, int vignetteNumber);

int simulateWithMultipleCollectionsWithExchange(int collectionNumber, int
    vignetteNumber);

double theoricalValueUsingHarmonicSerie(int collectionNumber);

double theoricalValueUsingAsymptoticDevelopment(int collectionNumber);

int simulateCollectionWithExchange(int collectionNumber, int Exchange);

int simulateWithMultipleCollectionsWithExchange(int collectionNumber, int
    vignetteNumber, int Exchange);

#endif //COUPON_COLLECTOR_S_SIMULATION_FUNCTIONS_H
```

```cpp
#include "Functions.h"

int simulateCollection(int collectionNumber) {

    bool *collected = new bool[collectionNumber]; // Allocation dynamique du tableau

    for (int i = 0; i < collectionNumber; i++) //initialiser tableau de vignettes
        collected[i] = false;

    int weeks = 0;
    bool allCollected = false;

    while (!allCollected) {
        weeks++;
        int newVignette = rand() % collectionNumber; //choisir un valeur alétoire qui
            represente une vignette
        if (!collected[newVignette])
            collected[newVignette] = true;

        //verifier si tout les vignettes sont collectées pour finir, si la nouvelle
            vignette est déja présente je verifie pas
        allCollected = true;
        for (int i = 0; i < collectionNumber; i++) {
            if (!collected[i]) {
                allCollected = false;
                break;
            }
        }
    }
    delete[] collected; // Libération de la mémoire allouée dynamiquement
    return weeks;
}
```

```
int simulateCollectionWithExchange(int collectionNumber) {

    bool *collected = new bool[collectionNumber]; // Allocation dynamique du tableau

    for (int i = 0; i < collectionNumber; i++)
        collected[i] = false;

    int weeks = 0;
    int duplicatesVignette = 0;
    bool allCollected = false;
    int notCollectedCounter = collectionNumber;
    while (!allCollected) {
        weeks++;

        int newVignette = rand() % collectionNumber;
        if (!collected[newVignette]) {
            collected[newVignette] = true;
            //compter vignettes restantes
            notCollectedCounter--;
        } else
            duplicatesVignette++;


        //faire echange a la fin (si nombre de vignettes manquantes = doublons/10 on
           echange) et mettre fin
        if (notCollectedCounter <= duplicatesVignette / 10) {
            break;
        }


        //verifier si tout les vignettes collecter pour finir
        allCollected = true;
        for (int i = 0; i < collectionNumber; i++) {
            if (!collected[i]) {
                allCollected = false;
                break;
            }
        }
    }
    delete[] collected; // Libération de la mémoire allouée dynamiquement
    return weeks;
}

int simulateWithMultipleCollections(int collectionNumber, int vignetteNumber) {

    bool *collected = new bool[collectionNumber]; // Allocation dynamique du tableau

    for (int i = 0; i < collectionNumber; i++)
        collected[i] = false;

    int weeks = 0;
    bool allCollected = false;

    //exception si collectionNumber<vignetteNumber alors weeks = 1
    if (collectionNumber < vignetteNumber) {
        return 1;
    }
    int *newVignetteTable = new int[vignetteNumber]; // si on a plus d'une vignette
```

```
        dans la boite de céréales

    while (!allCollected) {
        weeks++;

        for (int i = 0; i < vignetteNumber; i++) {
            int randomVignette;
            bool isUnique; //Assurer que les vignettes trouvées dans le céréale soient
                différentes

            do {
                // Générer un nombre aléatoire entre 0 et collectionNumber - 1
                randomVignette = rand() % collectionNumber;

                // Vérifier si le nombre généré est déjà présent dans le tableau
                isUnique = true;
                for (int j = 0; j < i; j++) {
                    if (newVignetteTable[j] == randomVignette) {
                        isUnique = false;
                        break;
                    }
                }
            } while (!isUnique);

            // Assigner le nombre aléatoire unique au tableau
            newVignetteTable[i] = randomVignette;

            // Marquer la vignette comme collectée
            if (!collected[newVignetteTable[i]])
                collected[newVignetteTable[i]] = true;
        }

        //verifier si toutes les vignettes sont collectées
        allCollected = true;
        for (int i = 0; i < collectionNumber; i++) {
            if (!collected[i]) {
                allCollected = false;
                break;
            }
        }
    }
    delete[] newVignetteTable;
    delete[] collected; // Libération de la mémoire allouée dynamiquement
    return weeks;
}

int simulateWithMultipleCollectionsWithExchange(int collectionNumber, int
    vignetteNumber) {

    bool *collected = new bool[collectionNumber]; // Allocation dynamique du tableau

    for (int i = 0; i < collectionNumber; i++)
        collected[i] = false;

    int weeks = 0;
    int duplicatesVignette = 0;
    bool allCollected = false;
    int notCollectedCounter = collectionNumber;

```

```cpp
    if (collectionNumber < vignetteNumber) {
        return 1;
    }
    int *newVignetteTable = new int[vignetteNumber]; //si on a plus d'une vignette dans la boite de céréales

    while (!allCollected) {
        weeks++;

        for (int i = 0; i < vignetteNumber; i++) {
            int randomVignette;
            bool isUnique; //Assurer que les vignettes trouvées dans le céréale soient différentes

            do {
                // Générer un nombre aléatoire entre 0 et collectionNumber - 1
                randomVignette = rand() % collectionNumber;

                // Vérifier si le nombre généré est déjà présent dans le tableau
                isUnique = true;
                for (int j = 0; j < i; j++) {
                    if (newVignetteTable[j] == randomVignette) {
                        isUnique = false;
                        break;
                    }
                }
            } while (!isUnique);

            // Assigner le nombre aléatoire unique au tableau
            newVignetteTable[i] = randomVignette;

            // Marquer la vignette comme collectée
            if (!collected[newVignetteTable[i]]) {
                collected[newVignetteTable[i]] = true;
                notCollectedCounter--;
            } else
                duplicatesVignette++;

        }
        if (notCollectedCounter <= duplicatesVignette / 10) {
            break;
        }

        //verifier si toutes les vignettes sont collectées
        allCollected = true;
        for (int i = 0; i < collectionNumber; i++) {
            if (!collected[i]) {
                allCollected = false;
                break;
            }
        }
    }
    delete[] newVignetteTable;
    delete[] collected; // Libération de la mémoire allouée dynamiquement

    return weeks;
}

//on utilisant série harmonique: N*sum(1/k) de 1 a N
```

```
double theoricalValueUsingHarmonicSerie(int collectionNumber) {
    double somme = 0;
    for (int i = 1; i <= collectionNumber; i++)
        somme += 1.0 / i;
    return collectionNumber * somme;
}

//En utilisant le développement asymptotique E(Tn)= n Hn =nln(n)+   n   +1/2+o(1) où
      0   .5772156649 est la constante d'Euler-Mascheroni.
double theoricalValueUsingAsymptoticDevelopment(int collectionNumber) {
    return collectionNumber * (std::log(collectionNumber) + 0.57721) + 1. / 2;
}

int simulateCollectionWithExchange(int collectionNumber, int Exchange) {

    bool *collected = new bool[collectionNumber]; // Allocation dynamique du tableau

    for (int i = 0; i < collectionNumber; i++)
        collected[i] = false;

    int weeks = 0;
    int duplicatesVignette = 0;
    bool allCollected = false;
    int notCollectedCounter = collectionNumber;
    while (!allCollected) {
        weeks++;

        int newVignette = rand() % collectionNumber;
        if (!collected[newVignette]) {
            collected[newVignette] = true;
            //compter vignettes restantes
            notCollectedCounter--;
        } else
            duplicatesVignette++;


        //faire echange a la fin (si nombre de vignettes manquantes = doublons/10 on
           echange) et mettre fin
        if (notCollectedCounter <= duplicatesVignette / Exchange) {
            break;
        }


        //verifier si tout les vignettes collecter pour finir
        allCollected = true;
        for (int i = 0; i < collectionNumber; i++) {
            if (!collected[i]) {
                allCollected = false;
                break;
            }
        }
    }
    delete[] collected; // Libération de la mémoire allouée dynamiquement
    return weeks;
}

int simulateWithMultipleCollectionsWithExchange(int collectionNumber, int
    vignetteNumber, int exchange) {

```

```
    bool *collected = new bool[collectionNumber]; // Allocation dynamique du tableau

    for (int i = 0; i < collectionNumber; i++)
        collected[i] = false;

    int weeks = 0;
    int duplicatesVignette = 0;
    bool allCollected = false;
    int notCollectedCounter = collectionNumber;

    if (collectionNumber < vignetteNumber) {
        return 1;
    }
    int *newVignetteTable = new int[vignetteNumber]; //si on a plus d'une vignette dans la boite de céréales

    while (!allCollected) {
        weeks++;

        for (int i = 0; i < vignetteNumber; i++) {
            int randomVignette;
            bool isUnique; //Assurer que les vignettes trouvées dans le céréal soient différentes

            do {
                // Générer un nombre aléatoire entre 0 et collectionNumber - 1
                randomVignette = rand() % collectionNumber;

                // Vérifier si le nombre généré est déjà présent dans le tableau
                isUnique = true;
                for (int j = 0; j < i; j++) {
                    if (newVignetteTable[j] == randomVignette) {
                        isUnique = false;
                        break;
                    }
                }
            } while (!isUnique);

            // Assigner le nombre aléatoire unique au tableau
            newVignetteTable[i] = randomVignette;

            // Marquer la vignette comme collectée
            if (!collected[newVignetteTable[i]]) {
                collected[newVignetteTable[i]] = true;
                notCollectedCounter--;
            } else
                duplicatesVignette++;

        }
        if (notCollectedCounter <= duplicatesVignette / exchange) {
            break;
        }

        //verifier si toutes les vignettes sont collectées
        allCollected = true;
        for (int i = 0; i < collectionNumber; i++) {
            if (!collected[i]) {
                allCollected = false;
                break;
```

```
            }
        }
    }
    delete[] newVignetteTable;
    delete[] collected; // Libération de la mémoire allouée dynamiquement

    return weeks;
}
```