TRAVAIL DE SESSION

PRESENTÉ
À

ADAM JOLY

COMME EXIGENCE PARTIELLE
DU COURS

ANALYSE D'ALGORITHMES

LABYRINTHES

FAIT PAR

JOSUE LUBAKI
ISMAEL GANSONRE
JONATHAN KANYINDA
JORDAN KUIBIA
EDGARD KOFFI

TRAVAIL PRATIQUE 1

01 AVRIL 2021

Table des matières

Présentation du Projet

Nom du Projet : Labyrinthe
Langage Choisis : C#
Dépendance : .NET Framework 4.0

Voici L'organisation de notre application :

- Data :
  - Map.cs
- Model :
  - Liaison.cs
  - Nœud.cs
- Utils :
  - Utils.cs (contient les fonctions statiques)
- Labyrinthe (Formulaire)
- Accueil (Formulaire)

Participation

Les étudiants Participants :

- ✓ Josue Lubaki
- ✓ Ismaël Gansonre
- ✓ Jonathan Kanyinda
- ✓ Jordan Kuibia
- ✓ Edgard Koffi

Problèmes et difficultés rencontrés

Parmi les difficultés, il a été difficile pour nous de trouver la structure de données parfaites où tout le monde serait à l'aise à travailler, une de simple raison pour laquelle nous avons travaillé avec le tableau.

Il était fastidieux de penser à un code qui lirait le tableau sur les quatre directions au moment de l'affichage.

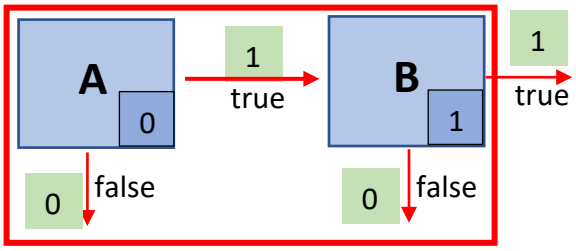
- X+1, Y
- X-1, Y
- X, Y+1
- X, Y-1

D'où on a choisi l'approche bidirectionnelle (DROITE et BAS) au moment de l'affichage tout en commençant par :

- A [0,1] (position 0, lire à Droite)
- A [0,0] (position 0, lire à Bas)
- B [1,1] (position 1, lire à Droite)
- B [1,0] (position 1, lire à Bas)
- Ainsi de suite...

Et pour ce faire on a songé à créer une entité « Map » qui est en soi un tableau à deux dimensions.

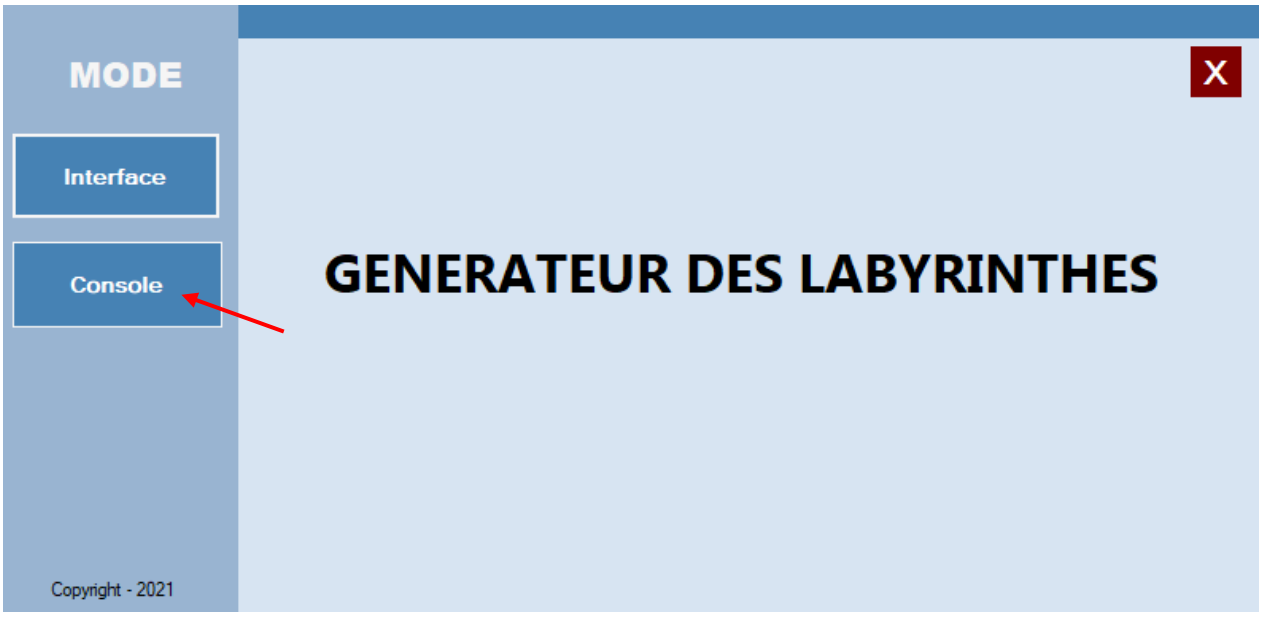
- ✓ La première dimension indique le numéro de la case où on placera la case par rapport à la Map
- ✓ La seconde dimension indique la direction à naviguer, si 1 (Droite) / si 0 (Bas)

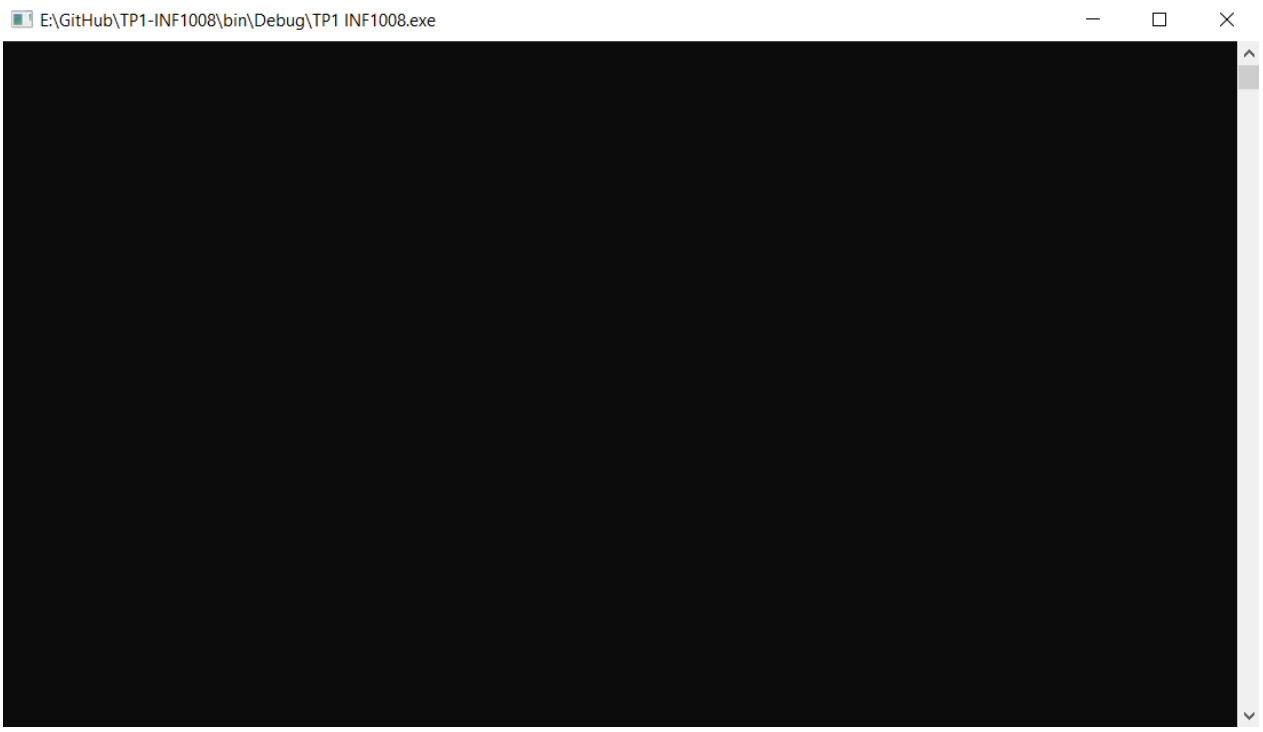


Nous avons également éprouvé de la difficulté par rapport la réinitialisation du compteur d'opération lorsqu'on utilise le Mode Console. De base, on voulait isoler chaque calcul et non incrémenter au fur et à mesure le compteur pour des différentes Labyrinthes. Pour ce, on avait le choix soit de le réinitialiser juste après avoir générer la Map, mais en faisant de la sorte, on perdait la valeur du nombre d'opération de l'initialisation de la Map, une information assez précieuse. Pour résoudre ce problème, on ne sauvegarde les informations dans le fichier .txt que si l'Utilisateur effectue ces trois opérations : générer la Map, lancer l'algorithme de Prim et Afficher le Labyrinthe sur la Console. Tant les trois opérations n'ont pas été complété, les informations sur les calculs ne seront pas écrites dans le fichier .txt correspondant.

Guide Utilisateur

Au démarrage de l'Application, nous avons deux fenêtres qui s'ouvre :



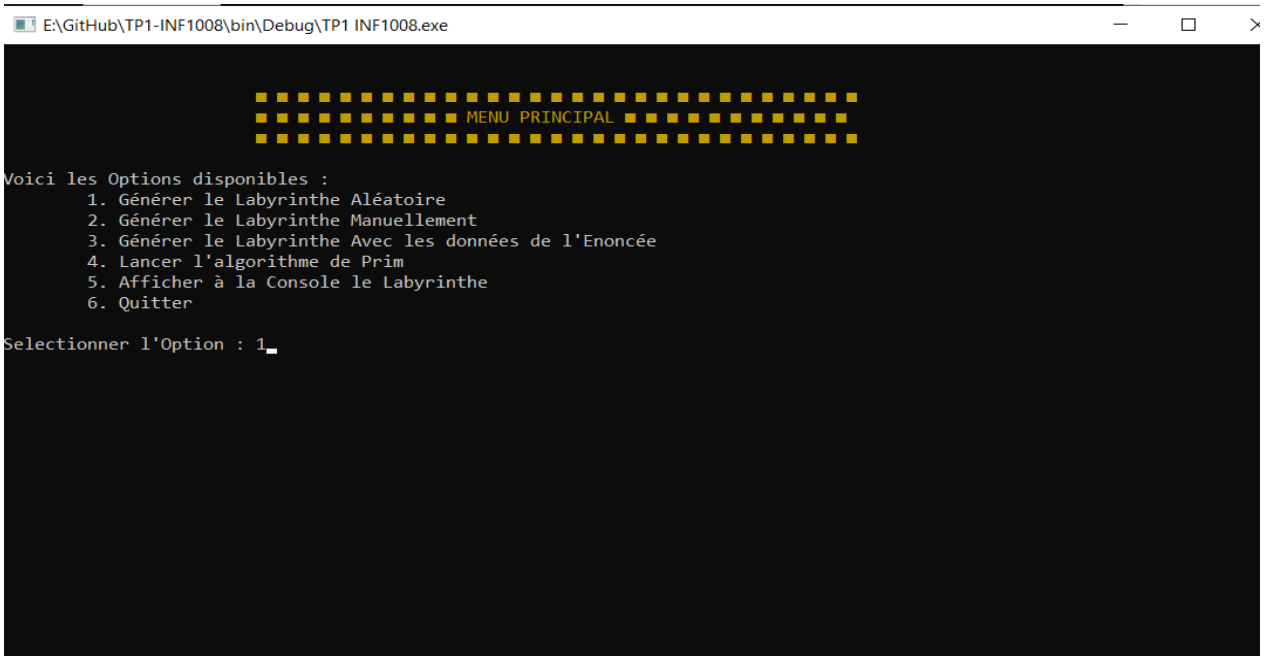


Sur la première fenêtre, vous avez le choix de travailler soit avec le UI ou avec la Console,

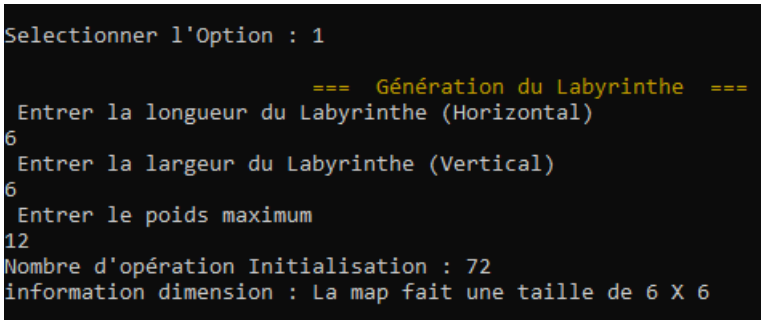
Sélectionnons la Console (Par exemple)

Voici le Menu principal, les trois premières options sont des options qui permettent de générer un Labyrinthe (Aléatoirement, Manuellement ou Avec Les données du travail).

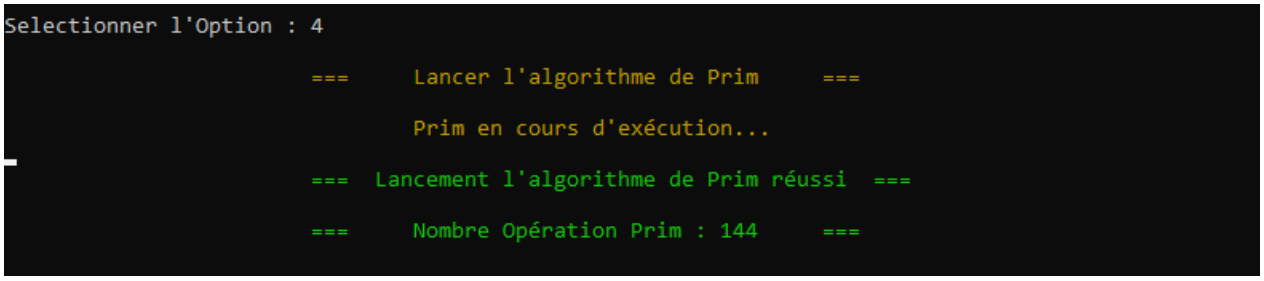
Option 1 : Le Labyrinthe Aléatoire



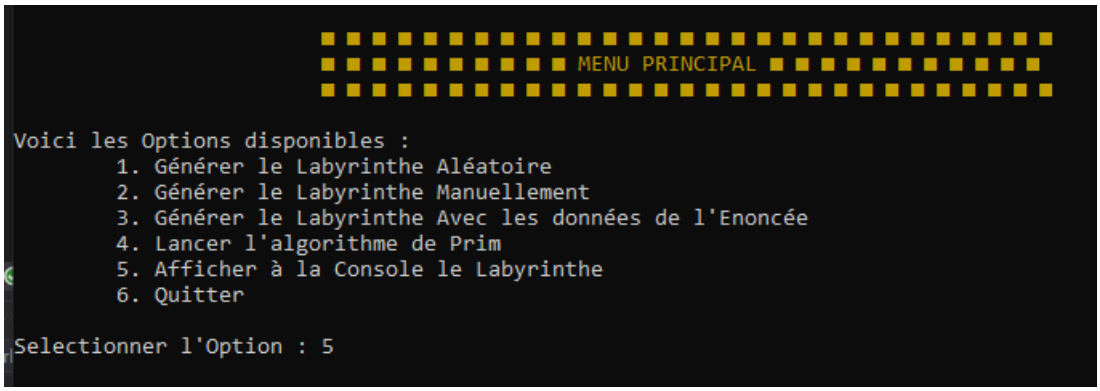
Résultat Option 1 :



Ensuite, faisons l'option 4 pour lancer l'algorithme de Prim sans quoi, on ne pourra pas afficher le labyrinthe généré.



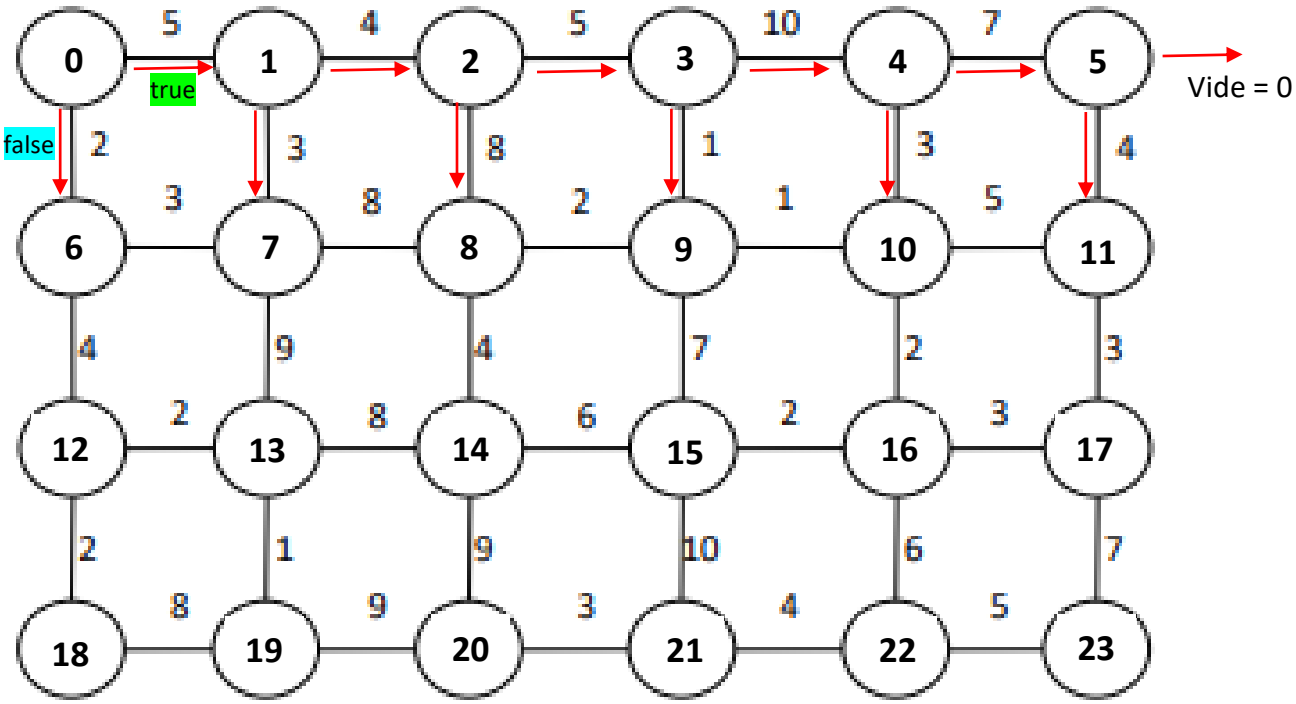
À présent, on peut afficher sur la console notre Labyrinthe, en faisant l'option 5



Et voici le Résultat :

- E (Entrée) et
- S (Sortie) /
- le symbole ■ représente les murs et
- (╗, ╢, ╦, ╔, ═, ╬, ╠, ╚, ╩) sont des chemins navigables.

Option 3 : Le Labyrinthe avec les données fournies dans le Travail




```
1 référence
public Map PreRemplissage() {

    // initialisation de la map
    map = new Map(6, 4);

    // Remplissage des poids
    map.AffectationPoids(0, 5, true);
    map.AffectationPoids(0, 2, false);
    map.AffectationPoids(1, 4, true);
    map.AffectationPoids(1, 3, false);
    map.AffectationPoids(2, 5, true);
    map.AffectationPoids(2, 8, false);
    map.AffectationPoids(3, 10, true);
    map.AffectationPoids(3, 1, false);
    map.AffectationPoids(4, 7, true);
    map.AffectationPoids(4, 3, false);
    map.AffectationPoids(5, 0, true);
    map.AffectationPoids(5, 4, false);
```

```
/**
 * Methode qui permet d'affecter les poids Manuellement (non aléatoire)
 * en deux directions (toRight = DROITE) et (!toRight = BAS)
 */
50 références
public void AffectationPoids(int x, int poids, bool toRight) {

    if (toRight)
        map[x, 1] = poids;
    else
        map[x, 0] = poids;

    nbOperationMap += 1;
}
```

Figure 1 Remplissage de la Première ligne de la Map

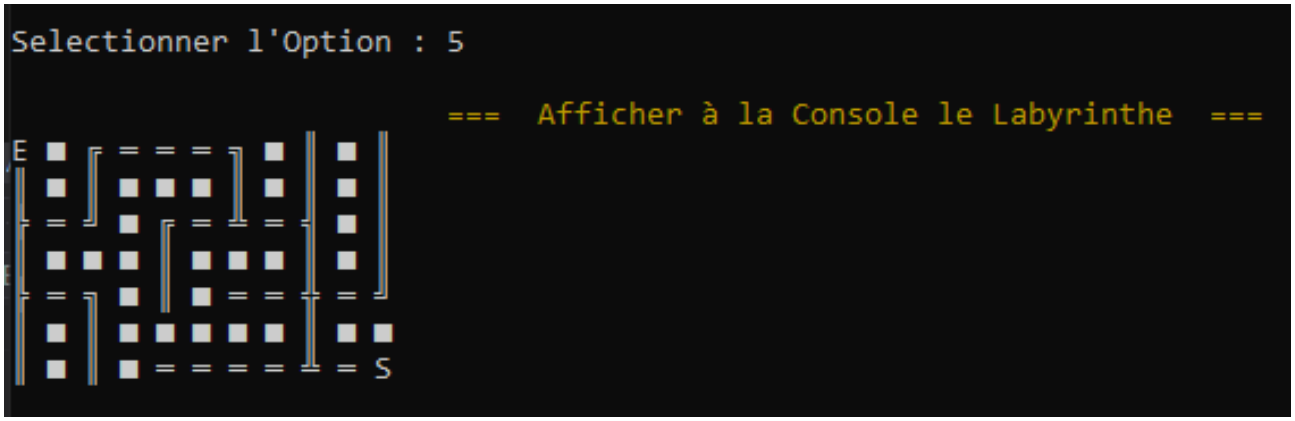
Sélectionnons l'Option 3 :

```
Selectionner l'Option : 3
Nombre d'opération : 48
information dimension : La map fait une taille de 6 X 4
```

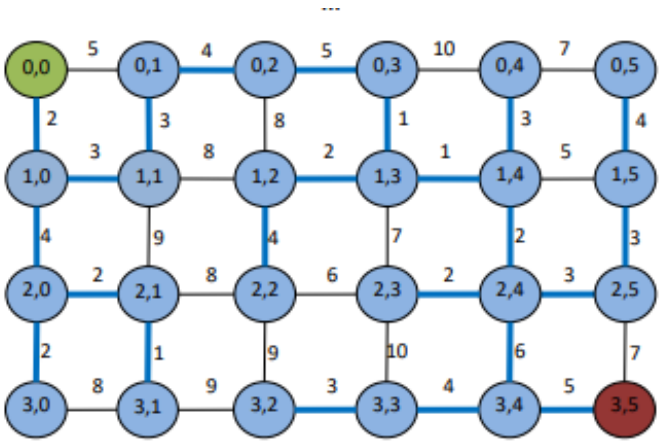
À présent, lançons l'algorithme de Prim :

```
Selectionner l'Option : 4

                    ===     Lancer l'algorithme de Prim     ===

                            Prim en cours d'exécution...

                    ===  Lancement l'algorithme de Prim réussi  ===

                    ===     Nombre Opération Prim : 96      ===
```

Et Affichons le Labyrinthe généré avec l'option 5 :



Ceci équivaut aux chemins du Travail demandé :



Constat

L'initialisation de la matrice

```
2 références
public Map PoidsAleatoires(int min, int max)
{
    nbOperationMap = 0;
    // x et y sont des coordonées au niveau du tableau map
    int x, y;
    Random rand = new Random();
    for (x = 0; x < longueur * largeur; x++)
    {
        for (y = 0; y < 2; y++)
        {
            map[x, y] = rand.Next(min, max) + 1;
            nbOperationMap += 1;
        }
    }
    return this;
}
```

L'opération d'initialisation se fait sur n*m élément deux fois, soit pour y = 0 et y =1, la deuxième boucle est minime, ce qui représente une complexité de $O(2n^2)$, on peut le simplifier à **$O(n^2)$**

Algorithme de Prim

```
160
161             // Tant que le nombre cases possibles est inferieur à la surface du map
162             while (noeudFinale.Count < GetMap().GetLargeur * GetMap().GetLongueur)
163             {
164                 noeudFinale.Add(noeudToAdd);
165
166                 var values = EnumDirection.GetValues<Direction>();
167
168                 Liaison liaisonTempo;
169                 foreach (Direction direction in values)
170                 {
171                     // try si le noeud n'a pas de voisin dans la direction donnée
172                     try
173                     {
```

La boucle while fonctionne en $O(n^2)$, tandis que le foreach en O(4) pour chacune de quatre directions. Donc $O(4n^2)$ qu'on peut simplifier à **$O(n^2)$**

Affichage Labyrinthe

```
250
251         for (int y = 0; y < GetMap().GetLargeur * 2 - 1; y++)
252         {
253             for (int x = 0; x < GetMap().GetLongueur * 2 - 1; x++)
254             {
255
256                 // Si c'est le Premier élément
257                 if (x == 0 && y == 0)
258                     str.Append("E");
259                 else if (x == GetMap().GetLongueur * 2 - 2 && y == GetMap().GetLargeur * 2 - 2)
260                     str.Append("S");
```

Le deux boucle « for » créent une complexité de **O(n^2),** puisqu'on va chercher pour chaque Largeur une valeur de la longueur.

Dimension	Poids	opération Initialisation	opération Prim	Opération Affichage	Total
5X5	10	50	100	81	231
10X10	20	200	400	361	961
15X15	30	450	900	841	2191
20X20	40	800	1600	1521	3921
25X25	50	1250	2500	2401	6151
30X30	60	1800	3600	3481	8881