université de BORDEAUX

Collège Sciences et Technologies
UF Mathématiques et Interactions

Projet Labyrinthe

Ethan Facca
Lucien Glasson
Aloyse Arnaud

CMI Optim L2

2024–2025

Projet de programmation

Table des matières

1 Introduction **4**
- 1.1 Contexte et Motivation . . . 4
- 1.2 Objectifs du Projet . . . 4

2 Algorithmes de génération **5**
- 2.1 Backtracking . . . 5
  - 2.1.1 Fonctionnement de l'algorithme . . . 5
  - 2.1.2 Preuve de la perfection du labyrinthe . . . 6
  - 2.1.3 Backtracking imparfait . . . 7
- 2.2 Fractal . . . 7
  - 2.2.1 Fonctionnement de l'algorithme . . . 7
  - 2.2.2 Preuve de perfection . . . 9
  - 2.2.3 Fractal imparfaite . . . 9
- 2.3 Diagonal . . . 10
  - 2.3.1 Fonctionnement de l'algorithme . . . 10
  - 2.3.2 Preuve de la perfection du labyrinthe . . . 11
  - 2.3.3 Diagonal imparfait . . . 12
- 2.4 Wallmaker . . . 13
  - 2.4.1 Fonctionnement de l'algorithme . . . 13
  - 2.4.2 Preuve de perfection . . . 14
  - 2.4.3 Wallmaker imparfait . . . 14

3 Algorithmes de résolution **15**
- 3.1 Depth First Search . . . 15
- 3.2 Breadth First Search . . . 15

4 Validation de labyrinthes **17**
- 4.1 Fonctionnement . . . 17
- 4.2 Types de parcours . . . 17

5 Jeux et visites de labyrinthe **18**
- 5.1 Visites automatisés . . . 18
  - 5.1.1 Visite dans le brouillard . . . 18
  - 5.1.2 Visite dans le brouillard main droite/gauche . . . 18
  - 5.1.3 Visite des impasses . . . 18
  - 5.1.4 Visite des impasses main droite/gauche . . . 19
  - 5.1.5 Visite du Petit Poucet . . . 19
  - 5.1.6 Visite de Tom Pouce main droite/gauche . . . 19
  - 5.1.7 Visite avec mémorisation des passages . . . 20
  - 5.1.8 Visite avec mémorisation des passages main droite/gauche . . . 20
  - 5.1.9 Visite avec mémorisation des passages et impasses . . . 20
  - 5.1.10 Visite avec mémorisation des passages et impasses main droite/gauche . . . 21
- 5.2 Jeu interactif . . . 21
  - 5.2.1 Jeu de la marche . . . 21
  - 5.2.2 Jeu de la marche fantôme . . . 21

6 Visualisation de labyrinthes **23**

7 Analyse **24**
7.1 Étudier l'impact de la génération du labyrinthe 24
7.2 Comparer les performances des algorithmes de visite 25
7.3 Interactions entre génération et visite 26
7.4 Synthèse et recommandations 26

8 Annexe **28**

1 Introduction

1.1 Contexte et Motivation

Les labyrinthes sont des objets d'étude intéressants en mathématiques, en informatique et dans de nombreux autres domaines. La génération, la résolution et la visualisation de labyrinthes sont des problèmes classiques qui trouvent des applications dans des domaines variés tels que les jeux vidéo, la robotique, la planification de parcours et l'intelligence artificielle.

1.2 Objectifs du Projet

Le projet Maze a été développé dans le cadre de la formation CMI OPTIM pour offrir une plateforme complète permettant de générer, résoudre, vérifier et visualiser des labyrinthes. Les objectifs principaux de ce projet sont les suivants :

1. **Génération de Labyrinthes** : Implémenter plusieurs algorithmes de génération de labyrinthes, chacun ayant ses propres caractéristiques et complexités. Les algorithmes doivent être capables de générer des labyrinthes parfaits (sans cycles) et imparfaits (avec des cycles).
2. **Résolution de Labyrinthes** : Développer des algorithmes efficaces pour résoudre les labyrinthes générés. Ces algorithmes doivent être capables de trouver un chemin de la cellule de départ à la cellule d'arrivée, en explorant les différentes possibilités de manière systématique.
3. **Vérification de Labyrinthes** : Mettre en place des mécanismes pour vérifier la validité des labyrinthes générés. Cela inclut la vérification de la connexité (toutes les cellules sont accessibles) et l'absence de cycles pour les labyrinthes parfaits.
4. **Visualisation Interactive** : Offrir des outils de visualisation permettant de voir le processus de génération et de résolution des labyrinthes en temps réel. La visualisation doit être interactive, permettant à l'utilisateur d'explorer le labyrinthe et de voir les algorithmes en action.
5. **Jeux et Visites** : Intégrer des fonctionnalités de jeu permettant aux utilisateurs de naviguer dans les labyrinthes de manière interactive, ou avec des modes de visite variés.

2 Algorithmes de génération

2.1 Backtracking

2.1.1 Fonctionnement de l'algorithme

Cet algorithme crée un labyrinthe en se déplaçant aléatoirement dans le labyrinthe, en détruisant des murs pour faire un chemin. Le labyrinthe est initialisé avec tous les murs.

Son fonctionnement est le suivant :

- On commence par choisir une cellule dans le labyrinthe, ce sera notre point de départ.
- Ensuite, on marque cette cellule comme visitée et on se déplace aléatoirement dans une cellule voisine non visitée en supprimant le mur entre les deux cellules.
- Si toutes les cellules voisines ont été visitées, on revient en arrière jusqu'à trouver une cellule avec des voisins non visités.
- L'algorithme se termine quand on est revenu à la cellule de départ.

Algorithme 1 : Backtracking

Entrée : Un labyrinthe *maze* de dimensions $width \times height$, un booléen $parfait$ et un floatant *probability*
Sortie : Un labyrinthe généré

```
Créer une pile pour stocker les coordonnées des cellules visitées.
Marquer la cellule de départ (0, 0) comme visitée et l'ajouter à la pile.
tant que la pile n'est pas vide faire
    Récupérer les coordonnées de la cellule courante à partir de la pile.
    Trouver un voisin non visité de la cellule courante.
    Si un voisin non visité est trouvé Alors
        Récupérer tous les voisins non visités de la cellule courante.
        pour chaque voisin non visité sauf celui sélectionné faire
            Ajouter un mur entre la cellule courante et ce voisin.
        fin
        Marquer le voisin sélectionné comme visité.
        Ajouter les coordonnées du voisin sélectionné à la pile.
    Sinon
        Compter le nombre de voisins non visités absolus à la cellule courante.
        Si aucun voisin non visité n'est trouvé Alors
            Retirer les coordonnées de la cellule courante de la pile.
        Sinon
            Récupérer les coordonnées des voisins non visités.
            Retirer un mur entre la cellule courante et un de ses voisins non visités.
            Ajouter les coordonnées de ce voisin à la pile.
        FinSi
    FinSi
fin
Marquer le labyrinthe comme généré.
```

2.1.2 Preuve de la perfection du labyrinthe

Nous devons démontrer que l'algorithme produit des labyrinthes parfaits. Pour cela, notre algorithme doit satisfaire ces deux propriétés pour toutes les configurations possibles de labyrinthe.

Connexité :

1. **Initialisation :**
   - Soit $G = (V, E)$ un graphe représentant le labyrinthe, où V est l'ensemble des cellules et E est l'ensemble des passages entre les cellules.
   - L'algorithme commence à une position de départ $v_0 \in V$.
2. **Parcours :**
   - L'algorithme utilise une approche de backtracking pour parcourir le graphe.
   - À chaque étape, il avance d'une cellule v_i à une cellule v_j non visitée et crée un passage $e_{ij} \in E$ entre v_i et v_j.
3. **Visite de toutes les cellules :**
   - En parcourant le graphe de cette manière, toutes les cellules finissent par être connectées, car chaque cellule est atteinte au moins une fois.
   - Soit $V' \subseteq V$ l'ensemble des cellules visitées. Initialement, $V' = \{v_0\}$.
   - À chaque étape, une cellule $v_j \notin V'$ est ajoutée à V' et un passage e_{ij} est ajouté à E.
   - L'algorithme utilise une pile pour stocker les coordonnées des cellules visitées, garantissant que chaque cellule est visitée et connectée à une autre cellule au moins une fois.
4. **Backtracking :**
   - Lorsqu'une cellule courante v_i n'a plus de voisins non visités, l'algorithme revient en arrière pour explorer d'autres chemins possibles.
   - Cela assure que toutes les cellules seront visitées et connectées.

Absence de cycles :

1. **Avancement systématique :**
   - L'algorithme avance systématiquement d'une cellule v_i à une cellule v_j non visitée à la fois, en créant des passages e_{ij} uniquement entre la cellule actuelle v_i et la cellule suivante v_j.
2. **Évitement des cycles :**
   - En choisissant un voisin non visité de manière aléatoire et en marquant chaque cellule visitée, l'algorithme évite de revisiter les cellules déjà visitées, ce qui empêche la formation de cycles.
   - Formulons cela mathématiquement : si v_j est un voisin de v_i et $v_j \in V'$, alors v_j ne sera pas choisi comme prochain nœud.
3. **Connexion unique :**
   - Si une cellule courante v_i n'a plus de voisins non visités, l'algorithme revient en arrière pour explorer d'autres chemins.
   - Cela garantit que chaque cellule est connectée de manière unique.
   - Soit $P(v_i, v_j)$ le chemin entre v_i et v_j. Pour tout $v_i, v_j \in V$, il existe un unique chemin $P(v_i, v_j)$.
4. **Création de passages :**
   - Lorsqu'un voisin non visité v_j est trouvé, l'algorithme crée un passage e_{ij} entre la cellule courante v_i et ce voisin v_j, et marque ce voisin v_j comme visité.
   - Cela assure que chaque connexion est unique et ne forme pas de cycle.

Conclusion :
L'algorithme génère des labyrinthes parfaits, car il garantit la connexité et l'absence de cycles. Chaque cellule est accessible depuis n'importe quelle autre cellule, et il n'y a qu'un seul chemin entre deux cellules quelconques. En utilisant une approche de backtracking et en marquant chaque cellule visitée, l'algorithme assure que toutes les cellules sont connectées sans former de cycles, produisant ainsi un labyrinthe parfait.

2.1.3 Backtracking imparfait

Dans l'algorithme de backtracking pour générer un labyrinthe, les paramètres `bool perfect` et `double probability` permettent de contrôler si le labyrinthe généré sera parfait ou imparfait.

Paramètre `perfect` :

— Ce paramètre est un booléen qui détermine si le labyrinthe doit être parfait ou non.

— Si `perfect` est `true`, l'algorithme génère un labyrinthe parfait sans cycles.

— Si `perfect` est `false`, l'algorithme peut introduire des imperfections (cycles) dans le labyrinthe en fonction de la probabilité spécifiée.

Paramètre `probability` :

— Ce paramètre est un nombre décimal compris entre 0 et 1 qui détermine la probabilité d'introduire une imperfection à chaque étape où l'algorithme rencontre une cellule sans voisins non visités.

— Une valeur de `probability` proche de 0 signifie que les imperfections seront rares, tandis qu'une valeur proche de 1 signifie que les imperfections seront fréquentes.

Dans l'algorithme de backtracking, lorsqu'une cellule courante n'a plus de voisins non visités, l'algorithme vérifie si le labyrinthe doit être parfait ou non. Si le labyrinthe n'est pas parfait (`perfect` est `false`), l'algorithme utilise la probabilité pour décider s'il doit introduire une imperfection en retirant un mur vers une cellule déjà visitée. Cet algorithme est inséré après la ligne 16 dans l'algorithme de backtracking.

Algorithme 2 : Backtracking imparfait

```
1 Si le labyrinthe n'est pas parfait et qu'une condition aléatoire basée sur la probabilité
    est satisfaite Alors
2 |  Sélectionner un voisin aléatoire déjà visité.
3 |  Retirer un mur entre la cellule courante et ce voisin.
4 FinSi
```

2.2 Fractal

2.2.1 Fonctionnement de l'algorithme

Cet algorithme génère un labyrinthe d'une manière récursive qui peut faire penser aux fractales, d'où son nom.
Le labyrinthe est initialisé comme un labyrinthe de taille 1*1 qui ne contient donc aucun mur.
Ensuite, à chaque itération de l'algorithme, le labyrinthe carré de taille $n \times n$ est dupliqué 3 fois afin d'obtenir un labyrinthe de taille $2n \times 2n$.

Son fonctionnement est le suivant :

- On double la taille du labyrinthe en le dupliquant à droite, en bas, et en bas à droite. On se retrouve avec un labyrinthe composé de 4 parties identiques.
- On ajoute des murs sur les lignes et colonnes du milieu du labyrinthe, i.e. on délimite les 4 parties de notre labyrinthe avec des murs. On a donc 4 délimitations, une entre les deux parties du haut, du bas, de gauche et de droite.
- On supprime, aléatoirement, 3 murs dans ces délimitations pour créer un chemin entre les 4 parties. Il faut pour cela que les murs choisis soient sur des délimitations différentes à chaque fois.

- On répète le processus un nombre déterminé de fois.

Notation pour l'algorithme fractal

$\leftarrow$ représente une assignation

$maze[i][j]$ représente la case sur la i-ième ligne et la j-ième colonne.

Pour les murs, nous utilisons la même notation que dans les fichiers de sauvegarde. Ainsi, on appelle mur vertical (resp horizontal) à l'emplacement (i, j) le mur entre les cases $maze[i][j]$ et $maze[i][j+1]$ (resp $maze[i+1][j]$)

Algorithme 3 : Fractal

Entrée : Un entier n et un labyrinthe $maze$ de taille 1×1, un booléen $parfait$ et un floatant $probability$

Sortie : Un labyrinthe généré de taille 2^n

```
pour c allant de 1 à n faire
    Créer un nouveau labyrinthe de taille 2c × 2c stocké dans new_maze
    /* Création du nouveau labyrinthe */
    pour i allant de 1 à c faire
        pour j allant de 1 à c faire
            new_maze[i][j] ← maze[i][j]
            new_maze[i + c][j] ← maze[i][j]
            new_maze[i][j + c] ← maze[i][j]
            new_maze[i + c][j + c] ← maze[i][j]
        fin
    fin
    /* Création des délimitations */
    pour i allant de 1 à c faire
        Placer un mur vertical à l'emplacement (i, c/2)
        Placer un mur horizontal à l'emplacement (c/2, i)
    fin
    /* Suppression des mur pour obtenir un labyrinthe parfait */
    Choisir aléatoirement 3 délimitation sur les 4
    pour chaque délimitation choisie faire
        retirer aléatoirement un mur parmi les murs de cette délimitation
    fin
    maze ← new_maze
fin
Marquer le labyrinthe comme généré.
```

2.2.2 Preuve de perfection

Pour prouver la perfection des labyrinthes générés par cet algorithme, nous allons faire une preuve par récurrence.

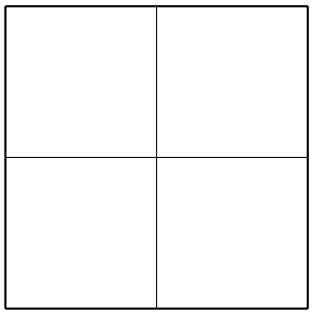
Initialisation
Le labyrinthe de départ est de taille 1×1 ainsi, il est forcément parfait.

Récurrence
Réalisons un tour de boucle en partant d'un labyrinthe parfait.
Montrons que le nouveau labyrinthe est, lui aussi, parfait.

Après avoir ajouté les délimitations au nouveau labyrinthe. Ce dernier est séparé en 4 parties qui sont des labyrinthes parfaits. Ainsi, en retirant 1 mur à une séparation, les deux parties se lient et forment une nouvelle partie, elle aussi parfaite.
Une fois que les 3 murs ont été retirés de cette manière, les 4 sous-parties sont toutes reliées entre elles et elles forment un labyrinthe parfait.
Le nouveau labyrinthe est donc bien parfait.

Remarque
Une manière plus intuitive de comprendre cette preuve est de considérer que les 4 sous-parties sont des cases. En effet, un labyrinthe d'une case seule est parfait, donc une sous-partie est équivalente à un labyrinthe formé d'une unique case en ce qui concerne le fait d'être parfait.

Ainsi, on se retrouve avec le labyrinthe suivant :



Dans cette situation, on voit bien que si l'on retire 3 des 4 murs internes au labyrinthe (les murs qui ne sont pas des bordures) on obtient un labyrinthe parfait.

2.2.3 Fractal imparfaite

Cet algorithme possède plein de manières différentes de générer des labyrinthes imparfait. Comme nous voulions utiliser des probabilités lors de la génération de labyrinthes imparfaits pour pouvoir contrôler le nombre de cycles, nous nous sommes dirigés vers l'approche suivante :
Au moment de creuser les délimitations pour que les 4 parties se rejoignent. Nous creusons les 4 délimitations plutôt que 3 parmi les 4, ainsi une boucle est créée. Enfin, avec une probabilité donnée par l'utilisateur, nous recréons un mur qui vient d'être supprimé

Ainsi, on obtient l'algorithme suivant :

Algorithme 4 : Fractal Imparfait

Entrée : Un entier n et un labyrinthe $maze$ de taille 1×1, un booléen $parfait$ et un floatant $probability$

Sortie : Un labyrinthe généré de taille 2^n

pour c allant de 1 à n **faire**
  Créer un nouveau labyrinthe de taille $2c \times 2c$ stocké dans new_maze
  `/* Création du nouveau labyrinthe */`
  pour i **allant de** 1 **à** c **faire**
    pour j **allant de** 1 **à** c **faire**
      $new\_maze[i][j] \leftarrow maze[i][j]$
      $new\_maze[i + c][j] \leftarrow maze[i][j]$
      $new\_maze[i][j + c] \leftarrow maze[i][j]$
      $new\_maze[i + c][j + c] \leftarrow maze[i][j]$
    fin
  fin
  `/* Création des délimitations */`
  pour i **allant de** 1 **à** c **faire**
    Placer un mur vertical à l'emplacement $(i, \frac{c}{2})$
    Placer un mur horizontal à l'emplacement $(\frac{c}{2}, i)$
  fin
  `/* Suppression des mur pour obtenir un labyrinthe parfait */`
  Supprimer un mur dans chacune des délimitations
  Si *le labyrinthe n'est pas parfait et qu'une condition aléatoire basée sur la probabilité est satisfaite* **Alors**
    Recréer un des mur supprimés à l'étape précédente
  FinSi
  $maze \leftarrow new\_maze$
fin
Marquer le labyrinthe comme généré.

2.3 Diagonal

2.3.1 Fonctionnement de l'algorithme

Cet algorithme génère un labyrinthe en le parcourant en diagonale. Le labyrinthe est initialisé sans aucun mur.

Son fonctionnement est le suivant :

- On commence par choisir un coin du labyrinthe aléatoirement.
- À chaque étape, il place des murs horizontalement ou verticalement, en fonction de la position actuelle.
- Une ouverture est laissée à un endroit choisi aléatoirement pour permettre un passage.

Algorithme 5 : Diagonal

Entrée : Un labyrinthe *maze* de dimensions $width \times height$, un booléen $parfait$ et un floatant *probability*
Sortie : Un labyrinthe généré

```
Initialiser la largeur et la hauteur du labyrinthe.
Déterminer aléatoirement le point de départ (coin supérieur ou coin inférieur).
Initialiser les coordonnées x et y en fonction du point de départ.
tant que les coordonnées x et y sont dans les limites du labyrinthe faire
    Si un nombre aléatoire est pair Alors
        Créer une sortie verticale.
        Créer une sortie horizontale.
    Sinon
        Créer une sortie horizontale.
        Créer une sortie verticale.
    FinSi
fin
```

Algorithme 6 : Création de sortie

```
si la position actuelle est dans les limites du labyrinthe alors
    Déterminer les positions de début et de fin pour la sortie.
    si les positions de début et de fin sont égales alors
        Mettre à jour la position actuelle.
        Retourner.
    fin
    Générer une position aléatoire pour la sortie.
    pour chaque position entre le début et la fin faire
        si la position n'est pas la position de sortie alors
            Ajouter un mur à cette position.
        fin
    fin
    Mettre à jour la position actuelle.
fin
```

2.3.2 Preuve de la perfection du labyrinthe

Nous devons démontrer que l'algorithme produit des labyrinthes parfaits. Pour cela, notre algorithme doit satisfaire ces deux propriétés pour toutes les configurations possibles de labyrinthe.

Soit $G = (V, E)$ le graphe représentant le labyrinthe, où V est l'ensemble des cellules et E est l'ensemble des passages entre les cellules.

— L'algorithme commence à une position de départ $v_0 \in V$.

— À chaque étape, l'algorithme avance d'une cellule v_i à une cellule adjacente v_{i+1} et crée un passage (arête) $e_i = (v_i, v_{i+1})$ entre ces deux cellules.

— L'algorithme parcourt le labyrinthe en alternant entre les directions horizontale et verticale, garantissant que chaque cellule est atteinte au moins une fois.

Connexité :

Pour prouver la connexité, nous devons montrer que pour chaque paire de cellules $u, v \in V$, il existe un chemin dans G reliant u à v.

— La cellule de départ v_0 est connectée à elle-même.

— Supposons que, après k étapes, il existe un chemin reliant v_0 à toute cellule v_i atteinte jusqu'à l'étape k.

— À l'étape $k+1$, l'algorithme avance à une nouvelle cellule v_{k+1} et crée un passage $e_k = (v_k, v_{k+1})$.

— Par l'hypothèse de récurrence, il existe un chemin de v_0 à v_k, et donc un chemin de v_0 à v_{k+1}.

Par récurrence, chaque cellule $v \in V$ est atteinte et connectée à v_0, prouvant que G est connexe.

Absence de cycles :

Pour prouver l'absence de cycles, nous devons montrer que chaque cellule est connectée de manière unique à ses voisines, garantissant qu'il n'y a qu'un seul chemin entre deux cellules quelconques.

— L'algorithme commence à une position de départ v_0.

— À chaque étape, l'algorithme avance d'une cellule v_i à une cellule adjacente v_{i+1} et crée un passage $e_i = (v_i, v_{i+1})$.

Pour prouver l'absence de cycles, nous devons montrer que l'ajout de chaque nouvelle arête e_i ne crée pas de cycle dans G.

— Au départ, G est un graphe avec une seule cellule v_0 et aucune arête, donc il n'y a pas de cycle.

— Supposons que, après k étapes, G est un arbre (un graphe connexe sans cycles).

— À l'étape $k+1$, l'algorithme ajoute une nouvelle arête $e_k = (v_k, v_{k+1})$ reliant une cellule v_k déjà atteinte à une nouvelle cellule v_{k+1}.

— Puisque v_{k+1} n'était pas encore atteinte, l'ajout de e_k ne peut pas créer de cycle.

Par récurrence, G reste un arbre après chaque étape, garantissant l'absence de cycles.

En conclusion, l'algorithme génère des labyrinthes parfaits, car il garantit la connexité et l'absence de cycles. Chaque cellule est accessible depuis n'importe quelle autre cellule, et il n'y a qu'un seul chemin entre deux cellules quelconques.

2.3.3 Diagonal imparfait

Dans l'algorithme Diagonal pour générer un labyrinthe imparfait, les paramètres `bool perfect` et `double probability` permettent de contrôler si le labyrinthe généré sera parfait ou imparfait.

Paramètre `perfect` :

— Ce paramètre est un booléen qui détermine si le labyrinthe doit être parfait ou non.

— Si `perfect` est `true`, l'algorithme génère un labyrinthe parfait sans cycles.

— Si `perfect` est `false`, l'algorithme peut introduire des imperfections (cycles) dans le labyrinthe en fonction de la probabilité spécifiée.

Paramètre `probability` :

— Ce paramètre est un nombre décimal compris entre 0 et 1 qui détermine la probabilité d'introduire une imperfection à chaque étape où l'algorithme rencontre une cellule sans voisins non visités.

— Une valeur de `probability` proche de 0 signifie que les imperfections seront rares, tandis qu'une valeur proche de 1 signifie que les imperfections seront fréquentes.

Dans l'algorithme diagonal, la génération du labyrinthe se fait en créant des sorties diagonales. Si le labyrinthe n'est pas parfait, l'algorithme utilise une probabilité pour décider s'il doit introduire des imperfections en supprimant des murs supplémentaires, créant ainsi des boucles et des chemins multiples entre les cellules. Cet algorithme remplace les lignes de 7 à 9 dans l'algorithme de Création de sortie.

Algorithme 7 : Diagonal imparfait

```
1 si la position n'est pas la position de sortie et que soit le labyrinthe est parfait soit
    la condition aléatoire basée sur la probabilité n'est pas satisfaite alors
2 |  Ajouter un mur à cette position.
3 fin
```

2.4 Wallmaker

2.4.1 Fonctionnement de l'algorithme

Cet algorithme génère un labyrinthe en créant des murs aléatoirement. Le labyrinthe est initialisé sans aucun mur.

Son fonctionnement est le suivant :

- On part d'une grille vide et on va créer les murs.
- On crée un tableau avec tous les murs possibles de la grille.
- On choisit un mur aléatoirement dans la liste et on vérifie :
  - si le mur appartient à une chaîne de murs (côte à côte) :
    - si oui : si la chaîne de murs touche deux fois le bord du labyrinthe ou qu'il crée un cycle, le mur ne peut pas être mis là, donc on l'enlève du tableau.
  - sinon : on pose le mur et on l'enlève du tableau.
- On continue tant que le tableau n'est pas vide.

Algorithme 8 : Wallmaker

```
   Entrée : Un labyrinthe maze de dimension width × height et un tableaux
            contenant l'ensemble des murs possibles
   Sortie  : Un labyrinthe généré
1  tant que le tableaux des mur possible est non vide faire
2  |  Poser un mur de manière aléatoire
3  |  Retirer ce mur du tableaux des murs possibles
4  |  Si le mur est relié à au moins un autre mur Alors
5  |  |  Si la chaîne de mur relie 2 bord du labyrinthe ou crée un cycle Alors
6  |  |  |  On retire le mur que l'on viens de poser
7  |  |  FinSI
8  |  FinSI
9  fin
```

2.4.2 Preuve de perfection

Montrons que les restrictions utilisées par l'algorithme wallmaker permettent de générer un labyrinthe parfait.

Sans aucunes restrictions, l'algorithme va placer tous les murs possibles est donc on obtient un labyrinthe qui ressemble à un quadrillage. Évidemment, un tel labyrinthe n'est pas valide, car il n'est pas connexe, les cellules étant toutes isolées les unes des autres.

Maintenant, ajoutons la restriction qui empêche la création de cycles.
Cette restriction permet de générer des labyrinthes sans cycle, cependant ces labyrinthes ne sont toujours pas connexes. En effet, nous ne considérons pas les bords du labyrinthe comme des murs, donc il se peut qu'une chaîne de mur relie un bord du labyrinthe à un autre. Cela, a pour effet de séparer le labyrinthe en deux et ainsi de retirer la connexité de celui-ci.

Ainsi, on ajoute une propriété permettant d'empêcher une chaîne de labyrinthe de relier deux bords du labyrinthe.
Les labyrinthes générés n'ont toujours pas de cycle, car nous créons tous les murs qu'il est possible d'ajouter et ils sont connexes. Donc les labyrinthes générés ainsi sont parfaits.

2.4.3 Wallmaker imparfait

Pour rendre un labyrinthe généré par wallmaker imparfait, on commence par générer un labyrinthe parfait avec l'algorithme wallmaker.
Une fois que cela est fait, on parcourt tout le labyrinthe en supprimant des murs avec une probabilité voulue.

Algorithme 9 : Wallmaker imparfait

```
1 pour tout les murs faire
2 |   Si une condition aléatoire basée sur la probabilité est satisfaite Alors
3 |   |   Retirer le mur sélectionné
4 |   FinSi
5 fin
```

3 Algorithmes de résolution

3.1 Depth First Search

Le parcours en profondeur (Depth-First Search) est un algorithme de recherche qui explore un chemin en allant aussi loin que possible dans une direction avant de revenir en arrière pour explorer d'autres chemins.

Son fonctionnement est le suivant :

- Commencer à partir de la cellule de départ du labyrinthe. Marquer cette cellule comme visitée.

Exploration des chemins :

- Depuis la cellule courante, sélectionner une cellule voisine qui n'a pas encore été visitée et qui n'est pas bloquée par un mur.
- Aller dans cette cellule et marquer ce nouveau point comme visité.

Retour en arrière :

- Si la cellule courante n'a aucune cellule voisine accessible et non visitée, revenir à la cellule précédente (retour en arrière dans la pile des cellules visitées).
- Continuer ce processus jusqu'à trouver la cellule de sortie ou épuiser toutes les options.
- L'algorithme se termine une fois que la sortie du labyrinthe est atteinte, ou qu'il n'existe plus de cellules non visitées accessibles, signalant qu'il n'y a pas de solution.

Entrée : Un labyrinthe *maze* de dimensions $width \times height$ et un booléen $parfait$
Sortie : Un labyrinthe généré

Algorithme 10 : Depth First Search

```
1  On crée une pile vide et on y empile la cellule de départ.
2  tant que la pile n'est pas vide faire
3  |  On dépile une cellule qu'on note cell de la pile.
4  |  Si cell a déjà été visitée Alors
5  |  |  On passe au tour de boucle suivant
6  |  SinonSi cell est la sortie Alors
7  |  |  On reconstitue le chemin menant à cell et on le retourne.
8  |  Sinon
9  |  |  On ajoute cell à l'ensemble des cellules visitées.
10 |  |  On empile toutes les cellules accessibles depuis cell dans la pile.
11 |  FinSi
12 fin
```

3.2 Breadth First Search

Le parcours en largeur (Breadth-First Search) est un algorithme de recherche qui visite d'abord tous les voisins directs d'un point avant de passer aux voisins de ces voisins, garantissant ainsi de trouver le chemin le plus court vers une cible si elle existe.

Son fonctionnement est le suivant :

- Commencer à partir de la cellule de départ. Placer cette cellule dans une file d'attente et marquer comme visitée.

Exploration par niveaux :

- Tant que la file d'attente n'est pas vide, retirer la cellule en tête de la file (cellule actuelle).
- Examiner toutes les cellules voisines accessibles (sans murs).
- Pour chaque voisine non encore visitée, l'ajouter à la file d'attente, marquer comme visitée.

Trouver la sortie :

- Ce processus se poursuit jusqu'à atteindre la cellule de sortie (si elle est accessible) ou jusqu'à ce que la file soit vide.
- À chaque étape, l'algorithme explore d'abord les cellules les plus proches, garantissant ainsi de trouver le chemin le plus court.
- L'algorithme se termine lorsqu'il atteint la sortie, ou qu'il a exploré toutes les cellules accessibles sans trouver de solution.

Algorithme 11 : Breadth First Search

```
1  On crée une file vide et on y enfile la cellule de départ.
2  tant que la file n'est pas vide faire
3  |  On défile une cellule qu'on note cell de la file.
4  |  Si cell a déjà été visitée Alors
5  |  |  On passe au tour de boucle suivant.
6  |  SinonSi cell est la sortie Alors
7  |  |  On reconstitue le chemin menant à cell et on le retourne.
8  |  Sinon
9  |  |  On ajoute cell à l'ensemble des cellules visitées.
10 |  |  On enfile toutes les cellules accessibles depuis cell dans la file.
11 |  FinSi
12 fin
```

4 Validation de labyrinthes

Ces algorithmes utilisent une approche itérative pour vérifier la validité d'un labyrinthe et sa perfection (absence de cycles). En fonction du type de vérification souhaité (simple ou parfaite), ils utilisent différentes stratégies de parcours pour déterminer si toutes les cellules du labyrinthe sont accessibles et correctement connectées.

4.1 Fonctionnement

- L'algorithme commence par initialiser les structures de données nécessaires pour le parcours du labyrinthe, comme une file d'attente pour le parcours en largeur (BFS) ou une pile pour le parcours en profondeur (DFS).
- La cellule de départ est marquée comme visitée et ajoutée à la structure de données appropriée.
- Chaque cellule est marquée comme visitée et ses voisins non visités sont explorés.
- Si une cellule n'a plus de voisins accessibles, elle est marquée comme un cul-de-sac.
- Pour un labyrinthe parfait, l'algorithme vérifie également qu'il n'y a pas de cycles (plus de 2 voisins déjà visités).
- L'algorithme parcourt toutes les cellules et s'assure qu'elles sont toutes accessibles.
- Si un cycle est détecté dans un labyrinthe parfait, il est marqué comme non parfait.
- Lorsque toutes les cellules sont explorées, l'algorithme affiche si le labyrinthe est valide.
- Si la vérification de la perfection est activée, l'algorithme indique également si le labyrinthe est parfait ou non.

4.2 Types de parcours

Parcours en largeur (breadth-first search, BFS)

- Utilise une file d'attente pour gérer les cellules à explorer.
- S'arrête dès que la file est vide.

Parcours en profondeur (depth-first search, DFS)

- Utilise une pile pour gérer les cellules à explorer.
- S'arrête dès que la pile est vide.

En résumé, ces algorithmes permettent de vérifier de manière itérative la validité et la perfection d'un labyrinthe en explorant toutes ses cellules. Ils utilisent des structures de données adaptées pour gérer l'exploration et fournissent des résultats clairs sur la validité et la perfection du labyrinthe.

5 Jeux et visites de labyrinthe

5.1 Visites automatisés

5.1.1 Visite dans le brouillard

Description :

`game_fog` est une visite de labyrinthe où le joueur se déplace de manière aléatoire sans avoir une visibilité complète du labyrinthe. Il ne connaît ni sa position ni par où il est passé. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

— Le joueur commence à la position de départ du labyrinthe.
— À chaque étape, le joueur choisit un voisin aléatoire parmi les voisins accessibles.
— La visite continue jusqu'à ce que le joueur atteigne la position de fin.
— La fonction retourne le nombre de pas nécessaires pour sortir du labyrinthe.

5.1.2 Visite dans le brouillard main droite/gauche

Description :

`game_fog_hand` est une visite de labyrinthe où le joueur se déplace en suivant la règle de la main gauche ou droite sans avoir une visibilité complète du labyrinthe. Il ne connaît ni sa position ni par où il est passé. En revanche, il connaît le nombre de cases du labyrinthe. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

— Le joueur commence à la position de départ du labyrinthe.
— Le joueur suit la règle de la main gauche ou droite pour choisir la direction de déplacement en comptant son nombre de pas.
— Si le joueur se rend compte qu'il a parcouru plus de cases qu'il n'y a dans le labyrinthe, c'est-à-dire qu'il est dans une boucle d'un labyrinthe imparfait, alors le joueur tournera à gauche dès qu'il en aura la possibilité et continuera sa visite.
— La visite continue jusqu'à ce que le joueur atteigne la position de fin.
— La fonction retourne le nombre de pas nécessaires pour sortir du labyrinthe.

5.1.3 Visite des impasses

Description :

`game_dead_end` est une visite de labyrinthe où le joueur se déplace de manière aléatoire en évitant les impasses. À chaque impasse visitée, le joueur la marque pour ne plus repasser sur cette case. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

— Le joueur commence à la position de départ du labyrinthe.
— Si la case sur laquelle le joueur est situé a un seul voisin non marqué, on marque la cellule et elle est considérée comme une impasse.
— À chaque étape, le joueur choisit un voisin aléatoire parmi les voisins non marqués.
— Le joueur continue de se déplacer jusqu'à ce qu'il atteigne la position de fin.
— La fonction retourne le nombre de pas nécessaires pour sortir du labyrinthe.

5.1.4 Visite des impasses main droite/gauche

Description :

`game_dead_end_hand` est une visite de labyrinthe où le joueur se déplace en suivant la règle de la main gauche ou droite en évitant les impasses. Il les évite en marquant les cellules avec un ou aucun voisin non visité comme des impasses, ce qui permet d'éviter de revenir sur ces cellules à l'avenir. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

— Le joueur commence à la position de départ du labyrinthe.

— Si la case sur laquelle le joueur est situé à un seul voisin non marqué, on marque la cellule et elle est considérée comme une impasse.

— À chaque étape, le joueur choisit son voisin de droite ou de gauche parmi les voisins non marqués.

— Si le joueur se rend compte qu'il a parcouru plus de cases qu'il n'y a dans le labyrinthe, c'est-à-dire qu'il est dans une boucle d'un labyrinthe imparfait, alors le joueur tournera à gauche dès qu'il en aura la possibilité et continuera sa visite.

— Le joueur continue de se déplacer jusqu'à ce qu'il atteigne la position de fin.

— La fonction retourne le nombre de pas nécessaires pour sortir du labyrinthe.

5.1.5 Visite du Petit Poucet

Description :

`game_tom_thumb` est une visite de labyrinthe où le joueur se déplace de manière aléatoire en se souvenant des cellules déjà visitées. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

— Le joueur commence à la position de départ du labyrinthe.

— À chaque étape, le joueur choisit un voisin aléatoire parmi les voisins non visités et s'il a déjà visité tous ses voisins, il en choisit un parmi ceux-là de manière aléatoire.

— La visite continue jusqu'à ce que le joueur atteigne la position de fin.

— La fonction retourne le nombre de pas nécessaires pour sortir du labyrinthe.

5.1.6 Visite de Tom Pouce main droite/gauche

Description :

`game_tom_thumb_hand` est une visite de labyrinthe où le joueur se déplace en suivant la règle de la main gauche ou droite en se souvenant des cellules déjà visitées. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

— Le joueur commence à la position de départ du labyrinthe.

— À chaque pas, il regarde d'abord du côté de la main choisie (droite ou gauche) et se dirige vers la première cellule adjacente qui n'a pas encore été visitée.

— Si toutes les cellules voisines ont déjà été visités, il continue de suivre la règle de la main choisie pour décider de sa direction.

— Si le joueur se rend compte qu'il a parcouru plus de cases qu'il n'y a dans le labyrinthe, c'est-à-dire qu'il est dans une boucle d'un labyrinthe imparfait, alors le joueur tournera à gauche dès qu'il en aura la possibilité et continuera sa visite.

— La visite continue jusqu'à ce que le joueur atteigne la position de fin.

— La fonction retourne le nombre de pas nécessaires pour sortir du labyrinthe.

5.1.7 Visite avec mémorisation des passages

Description :

`game_splatoon` est une visite de labyrinthe où le joueur se déplace de manière aléatoire en se souvenant du nombre de fois où il est passé par chaque cellule. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

— Le joueur commence à la position de départ du labyrinthe.

— À chaque étape, le joueur choisit un voisin parmi ceux avec le moins de passages et marque un passage sur le voisin vers lequel il se dirige.

— La visite continue jusqu'à ce que le joueur atteigne la position de fin.

— La fonction retourne le nombre de pas nécessaires pour sortir du labyrinthe.

5.1.8 Visite avec mémorisation des passages main droite/gauche

Description :

`game_splatoon_hand` est une visite de labyrinthe où le joueur se déplace en suivant la règle de la main gauche ou droite en se souvenant du nombre de fois où il est passé par chaque cellule. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

— Le joueur commence à la position de départ du labyrinthe.

— À chaque étape, le joueur choisit un voisin en suivant la règle de la main gauche ou droite.

— Si la case proposée par la règle de la main gauche ou droite à plus de passages que ces autres voisins, le joueur choisit celui avec la pondération la plus faible.

— Si le joueur se rend compte qu'il a parcouru plus de cases qu'il n'y a dans le labyrinthe, c'est-à-dire qu'il est dans une boucle d'un labyrinthe imparfait, alors le joueur tournera à gauche dès qu'il en aura la possibilité et continuera sa visite.

— La visite continue jusqu'à ce que le joueur atteigne la position de fin.

— La fonction retourne le nombre de pas nécessaires pour sortir du labyrinthe.

5.1.9 Visite avec mémorisation des passages et impasses

Description :

`game_splatoon_dead_end` est une visite de labyrinthe où le joueur se déplace de manière aléatoire en se souvenant du nombre de fois où il est passé par chaque cellule et en évitant les impasses. À chaque impasse visitée, le joueur la marque pour ne plus repasser sur cette case. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

- Le joueur commence à la position de départ du labyrinthe.
- Si la case sur laquelle le joueur est situé a un seul voisin non marqué, on marque la cellule et elle est considérée comme une impasse.
- À chaque étape, le joueur choisit un voisin parmi ceux avec le moins de passages et qui n'est pas une impasse et ajoute une pondération à la case choisie.
- Le joueur continue de se déplacer jusqu'à ce qu'il atteigne la position de fin.
- La fonction retourne le nombre de pas nécessaires pour sortir du labyrinthe.

5.1.10 Visite avec mémorisation des passages et impasses main droite/gauche

Description :

`game_splatoon_dead_end_hand` est une visite de labyrinthe où le joueur se déplace en suivant la règle de la main gauche ou droite en se souvenant du nombre de fois où il est passé par chaque cellule et en évitant les impasses. Il les évite en marquant les cellules avec un ou aucun voisin non visité comme des impasses, ce qui permet d'éviter de revenir sur ces cellules à l'avenir. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

- Le joueur commence à la position de départ du labyrinthe.
- Si la case sur laquelle le joueur est situé à un seul voisin non marqué, on marque la cellule et elle est considérée comme une impasse.
- À chaque étape, le joueur choisit un voisin en suivant la règle de la main gauche ou droite.
- Si la case proposée par la règle de la main gauche ou droite à est une impasse ou plus de passages que ces autres voisins, le joueur choisit celui avec la pondération la plus faible qui n'est pas une impasse.
- Le joueur continue de se déplacer jusqu'à ce qu'il atteigne la position de fin.
- La fonction retourne le nombre de pas nécessaires pour sortir du labyrinthe.

5.2 Jeu interactif

5.2.1 Jeu de la marche

Description :

`game_walk` est un jeu de labyrinthe où le joueur se déplace manuellement en utilisant les flèches du clavier. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin.

Fonctionnement :

- Le joueur commence à la position de départ du labyrinthe.
- Le joueur utilise les flèches du clavier pour se déplacer dans le labyrinthe.
- Le jeu continue jusqu'à ce que le joueur atteigne la position de fin.
- La fonction retourne le nombre de pas utilisés pour sortir du labyrinthe.

5.2.2 Jeu de la marche fantôme

Description :

`game_walk` avec le booléen ghost est un jeu de labyrinthe où le joueur se déplace manuellement en utilisant les flèches du clavier. Le joueur commence à la position de départ et se déplace jusqu'à atteindre la position de fin. Mais le joueur se déplace à l'aveugle, c'est-à-dire sans voir l'ensemble du labyrinthe.

Fonctionnement :

- Le joueur commence à la position de départ du labyrinthe.
- Le joueur utilise les flèches du clavier pour se déplacer dans le labyrinthe.
- Le joueur se déplace à l'aveugle, c'est-à-dire sans voir l'ensemble du labyrinthe, et doit trouver son chemin en explorant.
- Le jeu continue jusqu'à ce que le joueur atteigne la position de fin.
- La fonction retourne le nombre de pas utilisés pour sortir du labyrinthe.

6 Visualisation de labyrinthes

La visualisation des labyrinthes combine des algorithmes de génération de labyrinthes avec des techniques de rendu graphique pour créer des représentations visuelles dynamiques et interactives. Voici une explication générale de la manière dont la visualisation des labyrinthes fonctionne :

Représentation des Cellules et des Murs :

Un labyrinthe est généralement représenté par une grille de cellules, où chaque cellule peut être connectée à ses cellules voisines par des murs. Les cellules et les murs sont des éléments fondamentaux dans la structure du labyrinthe.
- **Cellules** : Chaque cellule peut avoir différents états, tels que non visitée, visitée, ou faisant partie du chemin actuel.
- **Murs** : Les murs séparent les cellules et définissent les chemins possibles dans le labyrinthe.

Visualisation en temps réel :

Pour visualiser le processus de génération du labyrinthe en temps réel, des bibliothèques graphiques comme SFML (Simple and Fast Multimedia Library) sont souvent utilisées. Voici comment cela fonctionne généralement :
- **Création de la Fenêtre de Rendu** : Une fenêtre de rendu est créée pour afficher le labyrinthe. Cette fenêtre est mise à jour régulièrement pour refléter les changements dans le labyrinthe.
- **Dessin des Cellules et des Murs** : Les cellules et les murs sont dessinés dans la fenêtre de rendu. Les cellules peuvent être colorées différemment en fonction de leur état (par exemple, non visitée, visitée, ou faisant partie du chemin actuel).
- **Mise à Jour Dynamique** : À chaque étape de l'algorithme de génération, la fenêtre de rendu est mise à jour pour montrer les nouvelles cellules visitées et les murs ajoutés ou supprimés. Cela permet de visualiser le processus de génération du labyrinthe en temps réel.

Interaction Utilisateur :

Certaines visualisations de labyrinthes permettent également une interaction utilisateur, où l'utilisateur peut contrôler certains aspects de la génération du labyrinthe ou explorer le labyrinthe généré. Les événements utilisateur, tels que les pressions de touches ou les clics de souris, peuvent être capturés et traités pour offrir une expérience interactive.

Affichage Final :

Une fois le labyrinthe généré, la visualisation finale montre le labyrinthe complet avec tous les chemins et les murs. Des fonctionnalités supplémentaires, comme la recherche de chemin ou la résolution du labyrinthe, peuvent également être visualisées.

7 Analyse

Dans cette section, nous allons nous intéresser à des statistiques réalisées sur près de 45000 labyrinthes générés chacun visités 300 fois par nos algorithmes de visite, ce qui représente près de 13 500 000 visites réalisées sur l'ensemble des générateurs de labyrinthes avec 3 tailles différentes (8×8, 64×64, 128×128) et 5 seuils d'imperfections (0 (parfait), 0.25 (25%), 0.5, 0.75, 1).

Afin de réaliser nos tests sur des temps raisonnables de calculs s'élevant néanmoins à 24h de calculs, nous avons pris la décision de brider les algorithmes de visites à un certain nombre de pas qui au-delà de cette limite serait considéré comme non résolu. Pour déterminer cette limite, nous avons réalisé plusieurs tests moins conséquents que ceux utilisés pour les statistiques suivantes qui nous ont permis de trouver un compromis entre rapidité d'exécution et quantité de données. Ainsi, nous avons défini qu'au-delà de $(largeur \times hauteur)^{1.35}$ pas, les algorithmes de visites seraient stoppés et retourneraient une visite sans solution. Il est important de noter que cette limite n'est utilisée seulement pour les statistiques, en pratique les algorithmes de visites ne disposent pas de limite et peuvent donc tourner sans ne jamais s'arrêter.

Pour la suite de cette partie, nous utiliserons les termes suivants :
- fog = Visite dans le brouillard
- fog hand (left, right) = Visite dans le brouillard main droite/gauche
- dead end = Visite des impasses
- dead end hand (left, right) = Visite des impasses main droite/gauche
- tom thumb = Visite du Petit Poucet
- tom thumb hand (left, right) = Visite de Tom Pouce main droite/gauche
- splatoon = Visite avec mémorisation des passages
- splatoon hand (left, right) = Visite avec mémorisation des passages main droite/gauche
- splatoon dead end = Visite avec mémorisation des passages et impasses
- splatoon dead end hand (left, right) = Visite avec mémorisation des passages et impasses main droite/gauche
- EAO = écart absolu à l'optimum
- ERO = écart relatif à l'optimum

7.1 Étudier l'impact de la génération du labyrinthe

Comment les performances des algorithmes de visite évoluent avec la taille ?

Avec l'augmentation de la taille des labyrinthes, les performances des algorithmes de visite se détériorent fortement. Le nombre moyen de cases visitées avant de trouver la sortie est exponentielle : pour Fog, il passe de 90 en 8×8 à plus de 180 000 en 128×128, soit une multiplication par 2000.
Le taux de solutions optimales chute drastiquement : inférieur à 1% sur de petits labyrinthes, il devient quasi nul au-delà de 64×64.
Pour les petits labyrinthes, les algorithmes trouvent des solutions plus proches de l'optimum. Cela est dû au fait que, dans un petit espace, l'exploration est plus efficace et qu'il y a moins de chemins possibles à considérer, même si l'ERO ne dépasse jamais 50 % pour aucune visite. Pour les grandes instances, l'ERO est inférieur à 1 %.

Y a-t-il des générateurs favorisant un taux de réussite plus élevé pour des visites en prenant compte l'imperfection ?

L'efficacité des générateurs dépend fortement du niveau d'imperfection du labyrinthe. Wall Maker est le plus stable en 128×128, avec un taux de réussite de 100 % pour tous les algorithmes jusqu'à une probabilité de 0.5, même si le taux de solutions optimales chute en dessous de 1 %. À proba 1, Fog et Dead End échouent totalement.
Fractal devient plus difficile avec l'imperfection. Malgré une réussite totale à proba 0, dès 0.5, Fog, Dead End et Tom Thumb tombent entre 50 et 70 % de succès. À proba 1, ils échouent complètement. On peut expliquer cette variation du fait que les boucles sont beaucoup fréquentes et qu'ils n'ont pas consciences de ces cycles. D'autant plus que le Dead End est un algorithme se basant sur les impasses mais dans un

labyrinthe imparfait n'en possédant que très peu, Dead End se comporte que Fog
Backtracking est fortement impacté. Bien que parfait à proba 0, dès 0.5, Fog, Dead End et Tom Thumb passent sous 60 %. À proba 1, tous échouent totalement.
Diagonal suit une tendance similaire, avec une réussite qui chute dès 0.5 pour Fog, Dead End et Tom Thumb, puis une invalidité totale à proba 1.
Wall Maker est le générateur le plus fiable jusqu'à 0.5, tandis que Fractal, Backtracking et Diagonal deviennent extrêmement complexes dès que l'imperfection augmente.

7.2 Comparer les performances des algorithmes de visite

Quel est l'algo de visite le plus rapide en moyenne, varie-t-elle selon la génération du labyrinthe ?

L'algorithme de visite le plus rapide en moyenne est Splatoon Dead End, suivi de Tom Thumb. En 8×8, il trouve la sortie en environ 40 à 50 cases. En 64×64, il utilise entre 3 500 et 4 500 cases, et en 128×128, il reste performant avec 14 000 à 18 000 cases. Ces performances sont dues au fait que l'algorithme cherche constamment à découvrir les cellules qu'il connait le moins et donc étendre sa zone de recherche. D'un autre côté, Fog dépasse 180 000 cases dans les pires cas.
L'efficacité des visiteurs dépend aussi du générateur de labyrinthe. Wall Maker et Diagonal sont les plus favorables à une exploration rapide. Splatoon Dead End et Tom Thumb y maintiennent un faible nombre de cases explorées, en 128×128, avec environ 14 000 à 18 000 cases, bien en dessous de Fog, qui dépasse les 180 000 cases.
À l'inverse, Backtracking et Fractal compliquent la visite en forçant les algorithmes à explorer plus de cases. En 128×128, même les meilleurs visiteurs dépassent les 30 000 cases.
Conclusion, Splatoon Dead End est le plus rapide et fonctionne mieux avec Wall Maker et Diagonal, tandis que Backtracking et Fractal ralentissent considérablement la visite.

Quels algorithmes sont les plus stables (faible écart-type) et lesquels sont plus aléatoires ?

Certains algorithmes de visite sont plus stables que d'autres, leur performance variant peu d'un labyrinthe à l'autre. Splatoon Dead End et Tom Thumb font partie des plus réguliers, avec un écart-type modéré d'environ 9 000 à 10 000 cases. Cela signifie qu'ils suivent des stratégies d'exploration bien définies et sont moins sensibles aux variations de structure des labyrinthes. Leur efficacité reste constante, même sur des instances complexes, ce qui en fait des solutions fiables.

À l'inverse, Fog et Dead End sont des algorithmes bien plus aléatoires. Leur écart-type dépasse souvent 50 000, voire 100 000 cases, traduisant une grande variabilité dans leur capacité à trouver la sortie rapidement. Selon la génération du labyrinthe, ils peuvent parfois explorer un nombre excessif de cases, augmentant considérablement leur temps de résolution.

Quel algorithme trouve le plus souvent la sortie et parvient-il à trouver le chemin optimal ?

L'algorithme qui trouve le plus souvent la sortie tout en atteignant un chemin optimal dépend du type de labyrinthe et du niveau d'imperfection.

Splatoon Dead End est le plus fiable, avec un taux de réussite de 100 % en 128×128 pour proba = 0 et 0.5, quel que soit le générateur utilisé. Tom Thumb suit de près avec une réussite de 99 à 100 % en proba = 0, mais chute à 85-90 % lorsque proba $\geqslant$ 0.5. En revanche, Fog et Dead End deviennent beaucoup moins fiables : en proba = 1, leur taux de réussite tombe à 0 %, et même en proba = 0.5, ils ne dépassent pas 50 à 70 %.

Cependant, l'optimalité du chemin trouvé est différente. Splatoon Dead End et Tom Thumb explorent trop de cases inutilement, avec un ERO $<$ 0,5 %, les éloignant fortement de l'optimal. À l'inverse, Fog et Dead End, bien que moins efficaces pour trouver la sortie de manière régulière, trouvent des trajets plus proches de l'optimal, même s'ils échouent très souvent.

En conclusion, Splatoon Dead End est le plus fiable pour atteindre la sortie, mais il génère des chemins plus stables, donc moins efficaces et rarement optimaux. Fog et Dead End, bien qu'aléatoires et moins performants en termes de réussite, trouvent des trajets plus courts lorsqu'ils réussissent. L'imperfection du labyrinthe renforce ces tendances en rendant les solutions de plus en plus éloignées de l'optimal.

7.3 Interactions entre génération et visite

Y a-t-il un couple génération + visite qui fonctionne particulièrement bien ?

L'efficacité d'un couple génération + visite dépend de la taille du labyrinthe et du niveau d'imperfection, certaines combinaisons permettant d'optimiser la réussite et de minimiser l'exploration inutile.

En 8×8, le couple Wall Maker avec Splatoon Dead End Right est le plus performant avec un taux de réussite de 100 %, un écart relatif à l'optimal de 18,838 % et seulement 48 cases visitées, garantissant une exploration minimale et efficace.

En 64×64, la meilleure combinaison est Diagonal avec Splatoon Dead End Hand, qui offre un taux de réussite de 100% avec un ERO de 3,728 % et 3 755 cases explorées, ce qui permet de se rapprocher au mieux du chemin optimal. En effet, cet algorithme est basé sur le principe d'impasse qui est un paterne très important de la génération Diagonale, ce qui en fait une très bonne combinaison.

En 128×128, Wall Maker avec Splatoon Dead End Right est à nouveau la solution la plus efficace avec un taux de réussite de 100 %, un ERO de 0,973 % et 8 892 cases visitées, ce qui assure un bon équilibre entre exploration et efficacité.

À l'inverse, les pires combinaisons sont celles associant Fractal ou Backtracking avec Fog ou Dead End, notamment en 128×128 avec une probabilité d'imperfection de 1, où le taux de réussite tombe à presque 0 %, l'ERO dépasse 0,5 % et le nombre de cases explorées peut dépasser 40 000, rendant la navigation extrêmement inefficace. En effet, les explorations créées à l'aide de ces deux générations favorisent des sous graphes profonds, ce qui rend très difficile le parcours si le visiteur est allé dans le mauvais sous graphe.

Ainsi, la meilleure combinaison globale reste Wall Maker avec Splatoon Dead End Right en 8×8 et en 128×128, tandis qu'en 64×64, la solution la plus performante est Diagonal avec Splatoon Dead End Left, évitant ainsi les labyrinthes issus de Fractal et Backtracking qui compliquent considérablement l'exploration, surtout pour les visiteurs comme Fog et Dead End.

Pour garantir une exploration rapide et efficace, il est donc recommandé d'utiliser Splatoon Dead End Right en 8×8 et en 128×128, et Splatoon Dead End Left en 64×64, tout en privilégiant Wall Maker ou Diagonal comme générateurs.

7.4 Synthèse et recommandations

Quels sont les meilleurs algorithmes de visite selon différents critères ?

Les performances des algorithmes de visite varient en fonction de plusieurs critères, notamment la rapidité, la stabilité et l'efficacité. Certains algorithmes se démarquent par leur capacité à explorer un nombre réduit de cases tout en trouvant rapidement la sortie, tandis que d'autres garantissent une plus grande régularité dans leurs performances.

Parmi les plus rapides, Splatoon Dead End Right s'impose en 8×8 avec 48 cases visitées en moyenne et en 128×128 avec 8 892 cases visitées. En 64×64, c'est Splatoon Dead End Left qui se révèle légèrement plus performant avec 3 755 cases explorées. Ces algorithmes suivent une stratégie d'exploration optimisée

qui leur permet de parcourir efficacement les labyrinthes sans trop s'égarer.

La stabilité est également un critère clé, et là encore, les variantes de Splatoon Dead End dominent. En 8×8, Splatoon Dead End Hand affiche un écart-type de 33 cases, garantissant une performance prévisible. En 64×64, il présente une régularité similaire avec un écart-type de 2 341 cases. En 128×128, Splatoon Dead End Hand maintient également une faible variation avec un écart-type de 5 920 cases. Cette constance les rend particulièrement fiables, quel que soit le type de labyrinthe rencontré.

En termes d'efficacité (proximité avec le chemin optimal), Splatoon Dead End Hand est le plus performant en 8×8 avec un ERO de 18.838 %, en 64×64 avec un ERO de 3.728 % et en 128×128 avec 0.973 %. Cet algorithme parvient à équilibrer vitesse et précision, limitant les détours inutiles tout en assurant une exploration complète du labyrinthe.

Ainsi, Splatoon Dead End Hand est le meilleur choix pour toutes tailles de labyrinthes Cet algorithme surpassent largement Fog et Dead End, qui, bien que parfois proches du chemin optimal, échouent fréquemment à trouver la sortie dans des temps raisonnables. Pour une exploration efficace et stable, Splatoon Dead End Hand reste le meilleur.

Quels paramètres (taille, génération, imperfection) ont le plus d'influence ?

Les performances des algorithmes de visite sont influencées par plusieurs paramètres, dont la taille du labyrinthe, le type de génération et le niveau d'imperfection.

La taille du labyrinthe a un impact majeur sur la difficulté. Plus la taille augmente, plus le nombre de cases explorées avant de trouver la sortie croît de manière exponentielle. Par exemple, en 8×8, un algorithme rapide comme Splatoon Dead End Right explore environ 48 cases en moyenne. En 128×128, ce même algorithme en explore environ 8 892, soit une augmentation drastique. À l'inverse, des algorithmes inefficaces comme Fog ou Dead End deviennent pratiquement inutilisables dans les grands labyrinthes, dépassant parfois 180 000 cases explorées.

Le type de génération influence la complexité du labyrinthe. Des générateurs comme Wall Maker et Diagonal produisent des labyrinthes plus simples à parcourir, favorisant les visiteurs rapides. En revanche, Backtracking et Fractal génèrent des structures plus complexes qui obligent les algorithmes à explorer davantage. Par exemple, dans un 128×128 générés par Fractal, Splatoon Dead End peut explorer jusqu'à 44 566 cases en cas de forte imperfection, contre seulement 12 716 dans un labyrinthe Wall Maker.

L'imperfection rend également l'exploration plus difficile. Dans un labyrinthe parfait (proba = 0), tous les visiteurs réussissent et trouvent des chemins relativement courts. À proba = 0.5, l'ERO diminue, ce qui signifie que les algorithmes s'éloignent du chemin optimal et explorent davantage de cases inutiles et rentrent dans des cycles sans forcément en avoir conscience. À proba = 1, certains visiteurs comme Fog et Dead End échouent totalement dans les grands labyrinthes, avec 0 % de réussite en 128×128.

En conclusion, tous ces facteurs sont déterminants et interagissent entre eux pour influencer les performances des algorithmes de visite. La taille du labyrinthe impacte directement le nombre de cases explorées, rendant l'exploration exponentiellement plus complexe à mesure qu'elle augmente. Le type de génération modifie la structure du labyrinthe, rendant certains plus faciles à parcourir tandis que d'autres obligent les visiteurs à explorer davantage. Enfin, l'imperfection accentue encore ces effets, augmentant la difficulté et réduisant les chances d'atteindre un chemin optimal. Aucun de ces paramètres ne peut être négligé, car c'est leur combinaison qui détermine réellement l'efficacité d'un algorithme de visite.

8 Annexe

Abréviation	Signification
f	Fog
fl	Fog Left
fr	Fog Right
de	Dead End
del	Dead End Left
der	Dead End Right
tt	Tom Thumb
ttl	Tom Thumb Left
ttr	Tom Thumb Right
s	Splatoon
sl	Splatoon Left
sr	Splatoon Right
sde	Splatoon Dead End
sdel	Splatoon Dead End Left
sder	Splatoon Dead End Right

TABLE 1 – Algorithme back tracking 8×8 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	90	51	80	10.993%	6138	0.076%
fog left	48	33	31	35.22%	15000	0.044%
fog right	48	33	31	35.171%	15000	0.046%
dead end	101	66	91	9.584%	9727	0.045%
dead end left	48	33	38	19.525%	15000	0.031%
dead end right	48	33	39	18.334%	15000	0.028%
tom thumb	65	59	61	6.044%	10626	0.034%
tom thumb left	48	33	45	7.294%	15000	0.02%
tom thumb right	48	33	44	7.599%	15000	0.023%
splatoon	55	44	38	30.255%	14970	0.043%
splatoon left	49	37	32	34.295%	14997	0.044%
splatoon right	49	35	32	34.2%	15000	0.044%
splatoon dead end	48	33	39	19.002%	15000	0.033%
splatoon dead end left	48	33	39	19.007%	15000	0.031%
splatoon dead end right	48	33	39	18.838%	15000	0.032%

TABLE 2 – Algorithme diagonal 8×8 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	92	53	83	9.784%	6602	0.066%
fog left	51	35	38	25.438%	15000	0.039%
fog right	51	35	38	25.932%	15000	0.039%
dead end	107	70	99	7.202%	10998	0.037%
dead end left	51	35	44	13.815%	15000	0.028%
dead end right	51	35	43	14.458%	15000	0.029%
tom thumb	73	61	70	4.259%	10574	0.03%
tom thumb left	51	35	48	5.645%	15000	0.018%
tom thumb right	52	35	49	5.508%	15000	0.019%
splatoon	66	53	53	20.111%	14890	0.042%
splatoon left	56	42	43	23.268%	14979	0.041%
splatoon right	55	41	42	24.016%	14985	0.036%
splatoon dead end	51	35	44	14.129%	15000	0.032%
splatoon dead end left	51	35	44	13.671%	15000	0.029%
splatoon dead end right	51	35	44	13.816%	15000	0.031%

TABLE 3 – Algorithme fractal 8×8 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	91	54	83	9.518%	6854	0.06%
fog left	52	35	40	23.787%	15000	0.041%
fog right	52	36	40	23.147%	15000	0.037%
dead end	104	70	97	6.775%	11036	0.032%
dead end left	52	35	45	12.446%	15000	0.031%
dead end right	52	35	45	12.673%	15000	0.031%
tom thumb	77	62	74	3.769%	10665	0.028%
tom thumb left	52	35	50	5.248%	15000	0.018%
tom thumb right	52	35	49	5.432%	15000	0.017%
splatoon	67	53	55	18.383%	14904	0.045%
splatoon left	62	47	49	19.647%	14970	0.043%
splatoon right	62	48	49	20.049%	14954	0.039%
splatoon dead end	51	35	45	12.504%	15000	0.031%
splatoon dead end left	52	35	46	12.166%	15000	0.027%
splatoon dead end right	52	35	45	12.555%	15000	0.027%

TABLE 4 – Algorithme wall maker 8×8 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	95	55	86	8.778%	7012	0.058%
fog left	53	36	42	21.198%	15000	0.04%
fog right	53	35	42	21.064%	15000	0.039%
dead end	105	72	99	5.819%	11812	0.031%
dead end left	53	35	47	10.625%	15000	0.027%
dead end right	53	36	48	10.425%	15000	0.027%
tom thumb	84	65	81	3.172%	10564	0.026%
tom thumb left	53	36	51	4.886%	15000	0.02%
tom thumb right	54	36	51	4.857%	15000	0.02%
splatoon	70	53	59	15.529%	14957	0.044%
splatoon left	60	44	48	19.047%	14974	0.042%
splatoon right	60	44	49	18.547%	14986	0.039%
splatoon dead end	54	36	48	10.363%	15000	0.026%
splatoon dead end left	54	36	48	10.394%	15000	0.028%
splatoon dead end right	53	35	47	11.292%	15000	0.028%

TABLE 5 – Statistiques pour les labyrinthes de taille 8 × 8 imparfait avec une probabilité de 0 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	92	81	83	9.717%	26606	**0.064**%
fog left	**51**	**35**	**38**	**26.197**%	**60000**	0.041%
fog right	**51**	**35**	**38**	26.106%	**60000**	0.04%
dead end	104	81	97	7.228%	43573	0.036%
dead end left	**51**	**35**	44	13.978%	**60000**	0.029%
dead end right	**51**	**35**	44	13.863%	**60000**	0.029%
tom thumb	75	74	71	4.218%	42429	0.03%
tom thumb left	**51**	**35**	48	5.734%	**60000**	0.019%
tom thumb right	**51**	**35**	48	5.804%	**60000**	0.02%
splatoon	65	51	51	20.593%	59721	0.043%
splatoon left	57	43	43	23.555%	59920	0.042%
splatoon right	57	43	43	23.684%	59925	0.04%
splatoon dead end	**51**	**35**	44	13.871%	**60000**	0.031%
splatoon dead end left	**51**	**35**	44	13.674%	**60000**	0.029%
splatoon dead end right	**51**	**35**	44	14.019%	**60000**	0.029%

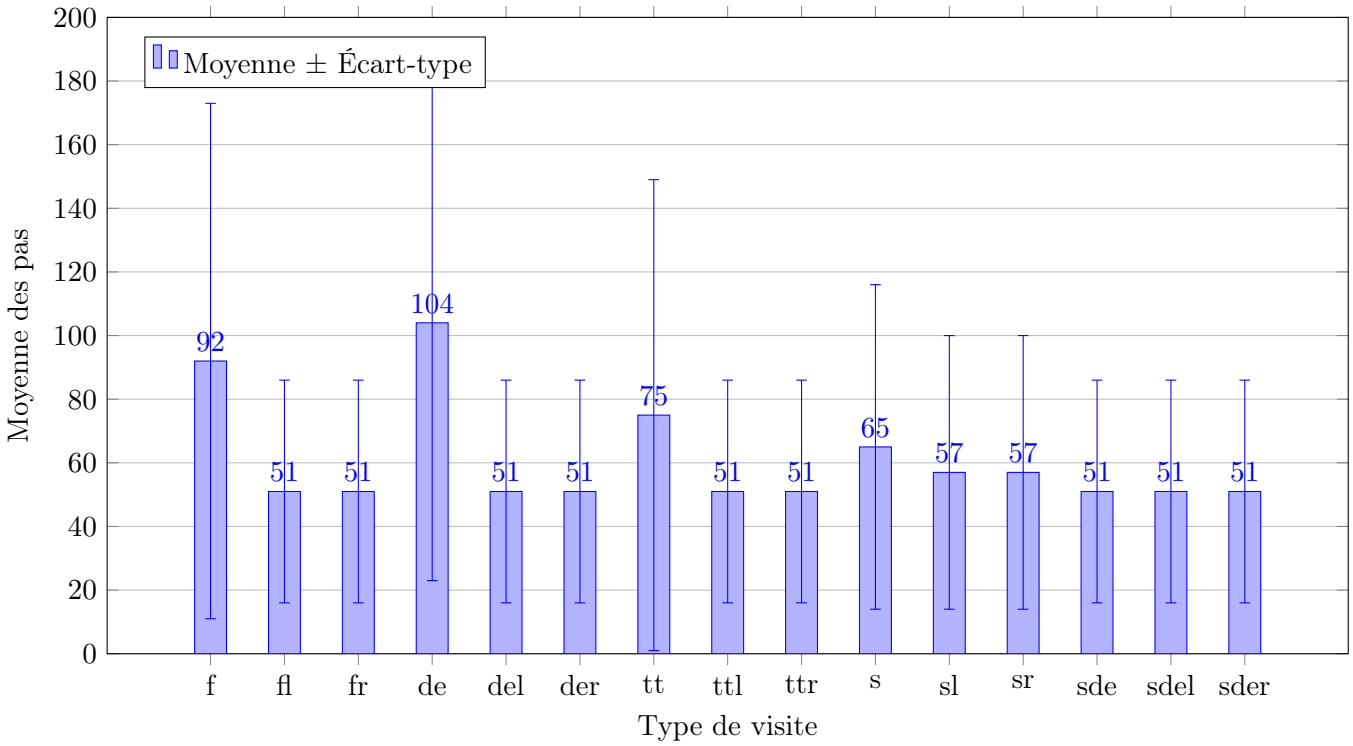


FIGURE 1 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (8 × 8)

TABLE 6 – Algorithme back tracking 64 × 64 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	25846	8822	25731	0.443%	2370	0.002%
fog left	3745	2351	3385	9.62%	15000	0.002%
fog right	3723	2332	3375	9.337%	15000	0.001%
dead end	29355	12408	29225	0.445%	4436	0.001%
dead end left	3732	2328	3587	3.88%	15000	0.001%
dead end right	3733	2334	3586	3.926%	15000	0.001%
tom thumb	23785	12426	23690	0.4%	4636	0.004%
tom thumb left	3735	2332	3686	1.318%	15000	0.002%
tom thumb right	3725	2328	3676	1.313%	15000	0.001%
splatoon	10414	11535	10049	3.503%	14883	0.001%
splatoon left	6493	6910	6136	5.501%	14993	0.002%
splatoon right	6921	8295	6548	5.384%	14946	0.001%
splatoon dead end	3758	2339	3612	3.879%	15000	0.001%
splatoon dead end left	3755	2341	3615	3.728%	15000	0.001%
splatoon dead end right	3748	2337	3607	3.756%	15000	0.001%

TABLE 7 – Algorithme diagonal 64 × 64 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	25632	8936	25524	0.42%	2430	0.004%
fog left	3671	2330	3255	11.337%	15000	0.001%
fog right	3669	2350	3265	11.016%	15000	0%
dead end	29454	12399	29326	0.435%	4477	0.002%
dead end left	3675	2310	3493	4.946%	15000	0.001%
dead end right	3650	2328	3466	5.045%	15000	0%
tom thumb	17478	11451	17378	0.569%	5716	0%
tom thumb left	3675	2341	3621	1.472%	15000	0%
tom thumb right	3674	2322	3618	1.522%	15000	0%
splatoon	10220	11841	9808	4.034%	14922	0.001%
splatoon left	4276	4433	3836	10.307%	14999	0.001%
splatoon right	4246	3818	3822	9.988%	15000	0%
splatoon dead end	3697	2335	3525	4.659%	15000	0%
splatoon dead end left	3684	2323	3506	4.828%	15000	0%
splatoon dead end right	3680	2331	3505	4.777%	15000	0.001%

TABLE 8 – Algorithme fractal 64×64 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	27993	10897	27917	0.272%	3440	0.003%
fog left	3907	2356	3738	4.319%	15000	0.001%
fog right	3919	2352	3742	4.497%	15000	0.001%
dead end	30241	17821	30181	0.197%	9715	0%
dead end left	3939	2350	3882	1.447%	15000	0%
dead end right	3925	2355	3865	1.517%	15000	0%
tom thumb	25166	13450	25129	0.146%	5653	0.001%
tom thumb left	3960	2364	3938	0.568%	15000	0%
tom thumb right	3905	2355	3882	0.593%	15000	0%
splatoon	12483	12654	12318	1.321%	14785	0.001%
splatoon left	8859	10405	8683	1.98%	14816	0.001%
splatoon right	10194	12064	10020	1.711%	14715	0.001%
splatoon dead end	3917	2352	3854	1.6%	15000	0.001%
splatoon dead end left	3925	2359	3862	1.598%	15000	0.001%
splatoon dead end right	3935	2349	3878	1.443%	15000	0%

TABLE 9 – Algorithme wall maker 64×64 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	26706	10534	26627	0.296%	3270	0.002%
fog left	3902	2351	3695	5.297%	15000	0.001%
fog right	3885	2348	3676	5.396%	15000	0.001%
dead end	30661	17063	30586	0.243%	8583	0.001%
dead end left	3912	2353	3832	2.036%	15000	0%
dead end right	3901	2362	3825	1.945%	15000	0.001%
tom thumb	23583	13592	23538	0.191%	6160	0.001%
tom thumb left	3879	2364	3852	0.719%	15000	0.001%
tom thumb right	3893	2372	3867	0.679%	15000	0.001%
splatoon	13215	14224	13004	1.593%	14457	0.001%
splatoon left	10558	12669	10349	1.984%	14614	0.001%
splatoon right	9945	11774	9730	2.166%	14824	0.001%
splatoon dead end	3853	2351	3777	1.973%	15000	0.001%
splatoon dead end left	3886	2349	3808	2.005%	15000	0.001%
splatoon dead end right	3883	2351	3804	2.021%	15000	0%

TABLE 10 – Statistiques pour les labyrinthes de taille 64 × 64 imparfait avec une probabilité de 0 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	26687	22489	26595	0.343%	11510	**0.003**%
fog left	3806	2349	3518	**7.566**%	**60000**	0.001%
fog right	**3799**	2348	**3515**	7.486%	**60000**	0.001%
dead end	30100	22482	30012	0.29%	27211	0.001%
dead end left	3814	**2338**	3698	3.036%	**60000**	0.001%
dead end right	3802	2347	3685	3.065%	**60000**	0.001%
tom thumb	22454	21205	22387	0.3%	22165	0.001%
tom thumb left	3813	2353	3774	1.008%	**60000**	0.001%
tom thumb right	**3799**	2346	3761	1.016%	**60000**	0.001%
splatoon	11569	12773	11280	2.498%	59047	0.001%
splatoon left	7523	9513	7227	3.943%	59422	0.001%
splatoon right	7810	9931	7512	3.808%	59485	0.001%
splatoon dead end	3806	2346	3692	3%	**60000**	0.001%
splatoon dead end left	3813	2345	3698	3.006%	**60000**	0.001%
splatoon dead end right	3811	2344	3699	2.964%	**60000**	0.001%

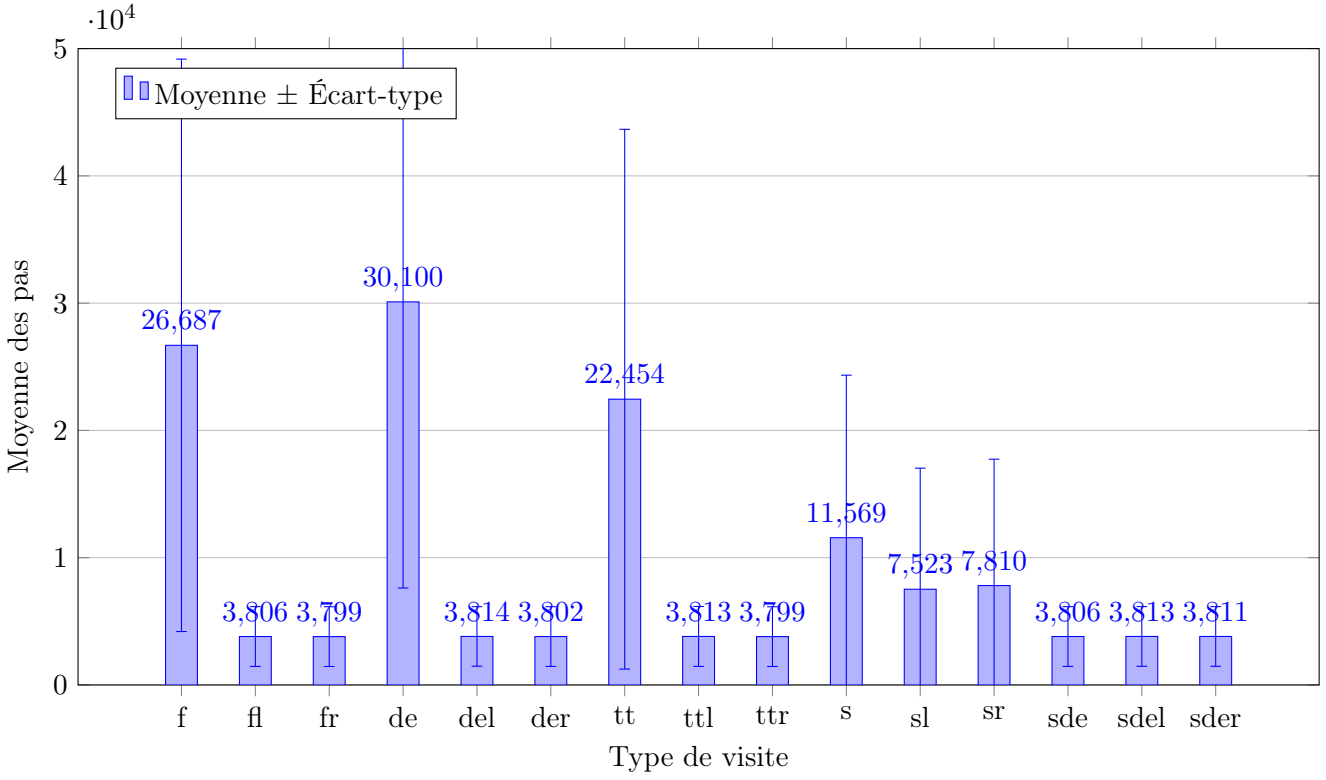


FIGURE 2 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (64 × 64)

TABLE 11 – Algorithme back tracking 128 × 128 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	180300	50025	180041	0.143%	1702	0.001%
fog left	15557	9439	14643	5.876%	15000	0%
fog right	15431	9447	14473	6.213%	15000	0.001%
dead end	190990	75486	190687	0.159%	3963	0.001%
dead end left	15539	9448	15213	2.099%	15000	0%
dead end right	15365	9407	15026	2.212%	15000	0%
tom thumb	163905	69033	163618	0.175%	3303	0.002%
tom thumb left	15389	9375	15269	0.778%	15000	0%
tom thumb right	15334	9445	15216	0.77%	15000	0%
splatoon	73836	84233	72875	1.301%	14733	0%
splatoon left	44293	61328	43358	2.112%	14872	0%
splatoon right	51038	68695	50077	1.883%	14938	0%
splatoon dead end	15516	9428	15175	2.192%	15000	0%
splatoon dead end left	15543	9492	15214	2.12%	15000	0%
splatoon dead end right	15416	9346	15069	2.256%	15000	0%

TABLE 12 – Algorithme diagonal 128 × 128 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	169788	47505	169517	0.16%	1582	0.001%
fog left	14833	9306	13219	10.881%	15000	0%
fog right	14811	9317	13184	10.982%	15000	0%
dead end	185740	64753	185373	0.198%	2937	0.001%
dead end left	14732	9333	14074	4.47%	15000	0%
dead end right	14860	9386	14223	4.285%	15000	0%
tom thumb	109543	64850	109130	0.377%	4683	0%
tom thumb left	14829	9331	14630	1.338%	15000	0%
tom thumb right	14786	9331	14584	1.368%	15000	0%
splatoon	58223	71202	56701	2.614%	14945	0%
splatoon left	17398	18660	15772	9.35%	15000	0%
splatoon right	16238	15549	14641	9.837%	15000	0%
splatoon dead end	14804	9343	14162	4.333%	15000	0%
splatoon dead end left	14776	9354	14119	4.442%	15000	0%
splatoon dead end right	14650	9295	13989	4.509%	15000	0%

TABLE 13 – Algorithme fractal 128 × 128 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	187795	62052	187631	0.087%	2684	0.001%
fog left	15880	9441	15460	2.651%	15000	0%
fog right	16107	9420	15664	2.75%	15000	0.001%
dead end	200040	112790	199895	0.072%	9101	0%
dead end left	16082	9452	15937	0.9%	15000	0%
dead end right	16091	9482	15962	0.803%	15000	0%
tom thumb	168645	78232	168537	0.064%	4391	0%
tom thumb left	16062	9439	16009	0.328%	15000	0%
tom thumb right	16094	9498	16041	0.332%	15000	0%
splatoon	88852	94497	88441	0.463%	14516	0%
splatoon left	65360	87953	64928	0.661%	14566	0%
splatoon right	70754	96034	70327	0.604%	14551	0%
splatoon dead end	16015	9533	15878	0.856%	15000	0%
splatoon dead end left	16013	9417	15876	0.852%	15000	0%
splatoon dead end right	15886	9481	15734	0.956%	15000	0%

TABLE 14 – Algorithme wall maker 128 × 128 imparfait avec une probabilité de 0 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	168865	56197	168676	0.111%	2315	0%
fog left	15668	9450	14913	4.822%	15000	0%
fog right	15685	9465	14944	4.725%	15000	0%
dead end	193050	98562	192808	0.126%	6742	0%
dead end left	15555	9426	15270	1.831%	15000	0%
dead end right	15669	9437	15393	1.759%	15000	0%
tom thumb	143773	78124	143600	0.12%	5165	0%
tom thumb left	15750	9445	15664	0.545%	15000	0%
tom thumb right	15577	9497	15485	0.587%	15000	0%
splatoon	96864	105796	96067	0.823%	13179	0%
splatoon left	76077	95312	75260	1.074%	13427	0%
splatoon right	69301	90142	68495	1.163%	13387	0%
splatoon dead end	15515	9443	15246	1.733%	15000	0%
splatoon dead end left	15755	9427	15493	1.666%	15000	0%
splatoon dead end right	15607	9390	15330	1.775%	15000	0%

TABLE 15 – Statistiques pour les labyrinthes de taille 128×128 imparfait avec une probabilité de 0 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	177525	146215	177314	0.119%	8283	0.001%
fog left	15485	9418	14559	5.981%	**60000**	0%
fog right	15509	9424	**14566**	**6.076**%	**60000**	0%
dead end	194544	146103	194314	0.118%	22743	0.001%
dead end left	15477	9427	15124	2.284%	**60000**	0%
dead end right	15496	9438	15151	2.229%	**60000**	0%
tom thumb	144651	136625	144409	0.167%	17542	0.001%
tom thumb left	15507	9409	15393	0.736%	**60000**	0%
tom thumb right	15448	9455	15332	0.753%	**60000**	0%
splatoon	78858	93051	77928	1.18%	57373	0%
splatoon left	50000	76984	49039	1.921%	57865	0%
splatoon right	51200	79187	50244	1.867%	57876	0%
splatoon dead end	15462	9447	15115	2.244%	**60000**	0%
splatoon dead end left	15522	9434	15175	2.231%	**60000**	0%
splatoon dead end right	**15390**	**9389**	15031	2.335%	**60000**	0%

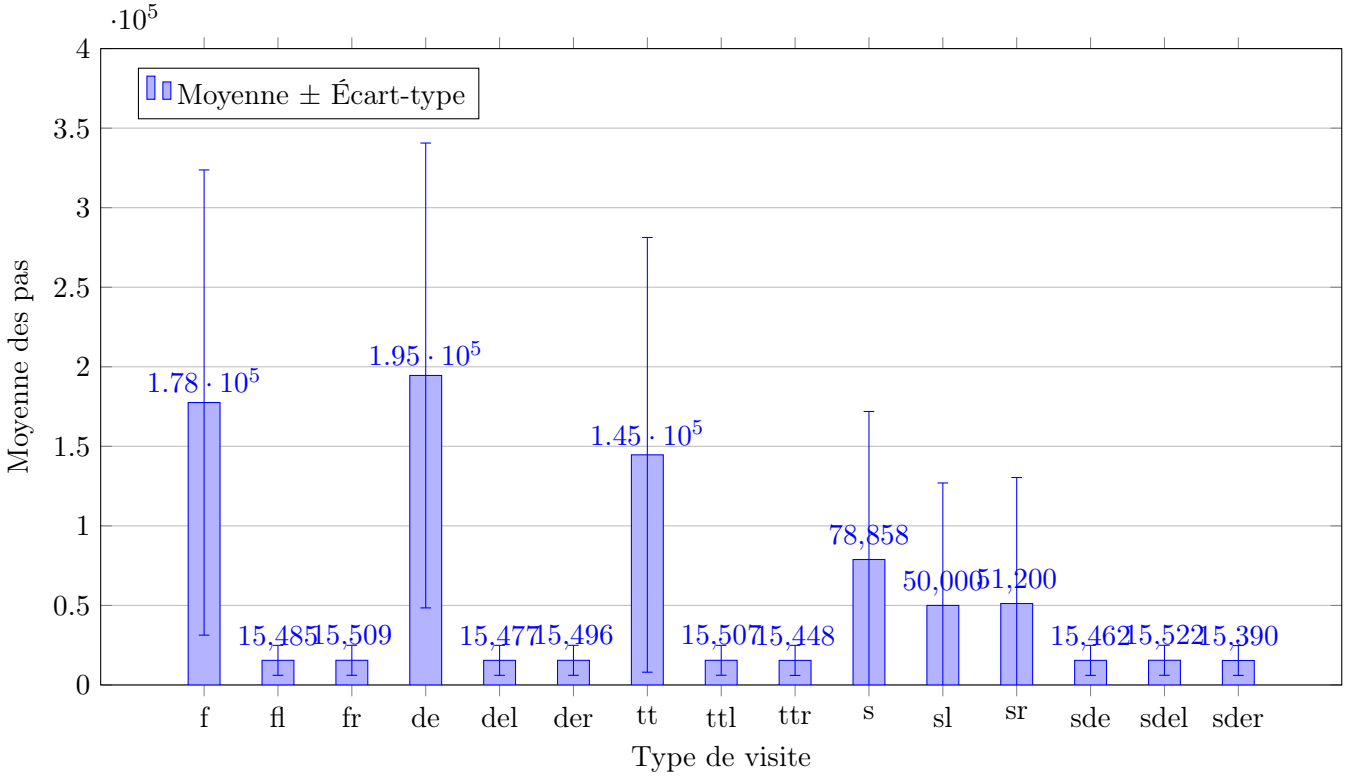


FIGURE 3 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (128×128)

TABLE 16 – Algorithme back tracking 8×8 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	95	60	86	9.2%	8413	0.055%
fog left	70	64	60	15.063%	12152	0.04%
fog right	71	64	61	14.379%	11597	0.039%
dead end	96	63	88	8.47%	9606	0.051%
dead end left	71	64	62	12.624%	12317	0.035%
dead end right	72	65	62	12.788%	12432	0.037%
tom thumb	64	60	61	5.462%	12891	0.024%
tom thumb left	47	43	44	7.404%	14642	0.022%
tom thumb right	47	41	44	7.361%	14666	0.021%
splatoon	51	43	40	21.358%	14973	0.039%
splatoon left	47	36	36	23.594%	15000	0.036%
splatoon right	45	34	34	23.978%	14999	0.036%
splatoon dead end	47	37	39	18.559%	14997	0.035%
splatoon dead end left	44	32	35	21.232%	15000	0.035%
splatoon dead end right	45	33	36	19.579%	15000	0.031%

TABLE 17 – Algorithme diagonal 8×8 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	95	63	87	8.171%	9589	0.05%
fog left	73	63	64	12.137%	10814	0.042%
fog right	72	63	63	12.322%	10771	0.036%
dead end	94	64	86	7.795%	10069	0.044%
dead end left	75	64	68	9.795%	11089	0.038%
dead end right	73	63	66	10.269%	11185	0.037%
tom thumb	66	61	63	4.6%	13459	0.022%
tom thumb left	46	40	43	6.821%	14517	0.02%
tom thumb right	47	41	44	6.748%	14532	0.02%
splatoon	54	45	45	16.346%	14972	0.039%
splatoon left	46	34	37	19.943%	14998	0.04%
splatoon right	47	36	38	19.326%	15000	0.037%
splatoon dead end	49	39	41	15.967%	14999	0.037%
splatoon dead end left	44	33	37	17.251%	14999	0.034%
splatoon dead end right	44	33	37	16.544%	15000	0.034%

TABLE 18 – Algorithme fractal 8×8 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	93	56	85	9.134%	7250	0.064%
fog left	59	49	48	18.892%	14420	0.042%
fog right	61	51	50	17.888%	14333	0.039%
dead end	100	66	93	7.192%	10095	0.042%
dead end left	63	54	55	12.398%	14267	0.035%
dead end right	59	49	52	12.131%	14565	0.032%
tom thumb	74	64	72	3.88%	11535	0.025%
tom thumb left	49	39	46	5.95%	14769	0.02%
tom thumb right	49	38	47	5.74%	14723	0.021%
splatoon	65	53	55	16.489%	14900	0.042%
splatoon left	54	43	43	20.681%	14966	0.044%
splatoon right	56	45	44	20.464%	14968	0.037%
splatoon dead end	55	41	47	13.925%	14978	0.035%
splatoon dead end left	51	36	43	15.045%	14999	0.034%
splatoon dead end right	49	36	41	16.22%	14997	0.036%

TABLE 19 – Algorithme wall maker 8×8 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	97	64	89	7.798%	9598	0.045%
fog left	71	62	63	11.617%	10694	0.045%
fog right	71	61	63	11.621%	10853	0.041%
dead end	94	65	87	7.562%	10260	0.044%
dead end left	72	63	66	9.368%	11277	0.034%
dead end right	74	63	68	8.934%	11295	0.036%
tom thumb	68	61	65	4.15%	13391	0.022%
tom thumb left	48	41	44	6.388%	14500	0.02%
tom thumb right	46	40	43	6.615%	14568	0.021%
splatoon	55	46	47	15.456%	14981	0.042%
splatoon left	47	35	38	17.972%	14994	0.039%
splatoon right	46	33	37	18.485%	14999	0.035%
splatoon dead end	50	40	43	14.12%	14999	0.038%
splatoon dead end left	45	31	38	15.003%	15000	0.03%
splatoon dead end right	45	32	39	14.609%	15000	0.034%

TABLE 20 – Statistiques pour les labyrinthes de taille 8×8 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	95	80	87	8.51%	34850	**0.053**%
fog left	68	67	58	14.558%	48080	0.042%
fog right	68	67	58	14.173%	47554	0.039%
dead end	96	79	89	7.74%	40030	0.045%
dead end left	70	68	62	11.099%	48950	0.035%
dead end right	69	67	61	11.069%	49477	0.035%
tom thumb	68	67	65	4.509%	51276	0.023%
tom thumb left	47	41	44	6.633%	58428	0.021%
tom thumb right	47	41	44	6.604%	58489	0.021%
splatoon	56	47	47	17.306%	59826	0.041%
splatoon left	48	37	**38**	**20.552**%	59958	0.04%
splatoon right	48	37	**38**	20.541%	59966	0.037%
splatoon dead end	50	39	43	15.562%	59973	0.036%
splatoon dead end left	**46**	**33**	**38**	17.043%	**59998**	0.033%
splatoon dead end right	**46**	34	**38**	16.72%	59997	0.034%

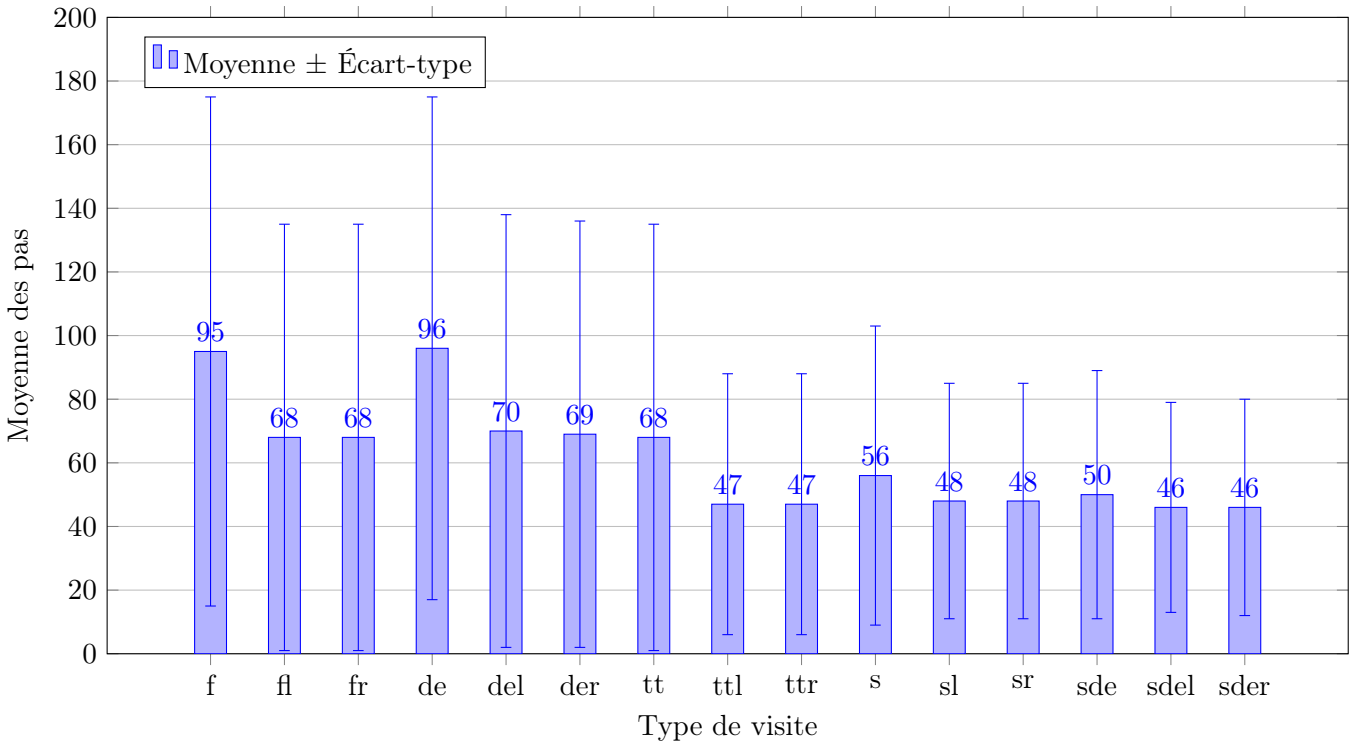


FIGURE 4 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (8×8, probabilité 0.25)

TABLE 21 – Algorithme back tracking 64×64 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	27666	17987	27606	0.218%	10912	0.001%
fog left	27688	7241	27631	0.204%	1325	0.001%
fog right	27486	6763	27430	0.206%	1155	0.003%
dead end	26907	18150	26851	0.209%	11487	0.001%
dead end left	26159	6692	26105	0.208%	1221	0.003%
dead end right	26166	7023	26111	0.207%	1300	0.003%
tom thumb	15649	16277	15633	0.099%	14159	0%
tom thumb left	22030	12153	22008	0.103%	4222	0.001%
tom thumb right	21634	12605	21611	0.105%	4417	0.002%
splatoon	4717	4027	4653	1.352%	15000	0.001%
splatoon left	4410	3502	4346	1.442%	15000	0.001%
splatoon right	4428	3563	4364	1.432%	15000	0%
splatoon dead end	4328	3617	4270	1.341%	15000	0.001%
splatoon dead end left	4123	3201	4066	1.383%	15000	0.001%
splatoon dead end right	4151	3225	4095	1.36%	15000	0.001%

TABLE 22 – Algorithme diagonal 64×64 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	24511	18474	24455	0.228%	12492	0.001%
fog left	23962	6536	23900	0.261%	1208	0.004%
fog right	24192	6634	24135	0.236%	1202	0.009%
dead end	23956	18263	23901	0.228%	12681	0.001%
dead end left	23643	6419	23586	0.243%	1173	0.006%
dead end right	23222	6748	23166	0.24%	1223	0.003%
tom thumb	12260	14158	12247	0.109%	14576	0%
tom thumb left	17885	12256	17865	0.116%	4467	0.001%
tom thumb right	17913	12291	17892	0.115%	4462	0.002%
splatoon	4308	3756	4250	1.341%	15000	0.001%
splatoon left	3988	3239	3930	1.447%	15000	0.001%
splatoon right	3996	3298	3938	1.46%	15000	0.001%
splatoon dead end	4155	3523	4098	1.36%	15000	0.001%
splatoon dead end left	3845	3098	3791	1.419%	15000	0.001%
splatoon dead end right	3914	3142	3859	1.389%	15000	0.001%

TABLE 23 – Algorithme fractal 64×64 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	27597	11899	27524	0.265%	4249	0.001%
fog left	8334	11079	8192	1.696%	12088	0.001%
fog right	8572	11764	8432	1.642%	12314	0.001%
dead end	28307	14828	28241	0.235%	6740	0.001%
dead end left	8513	11342	8419	1.106%	12523	0%
dead end right	8430	11478	8339	1.083%	12853	0.001%
tom thumb	24689	14714	24662	0.112%	6663	0.001%
tom thumb left	7838	12263	7817	0.273%	11687	0.001%
tom thumb right	8353	12464	8331	0.256%	10626	0%
splatoon	13434	13813	13305	0.96%	14772	0.001%
splatoon left	8396	10961	8264	1.582%	14879	0.001%
splatoon right	8169	9533	8032	1.683%	14867	0.002%
splatoon dead end	7804	9243	7717	1.108%	14967	0.001%
splatoon dead end left	5197	6014	5113	1.619%	14982	0%
splatoon dead end right	4999	5143	4908	1.82%	14998	0.001%

TABLE 24 – Algorithme wall maker 64×64 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	24234	18434	24184	0.207%	12991	0.001%
fog left	24237	6054	24186	0.208%	970	0.004%
fog right	23902	5883	23850	0.217%	924	0.002%
dead end	22758	18091	22713	0.201%	13495	0.001%
dead end left	24904	6326	24858	0.185%	1033	0.006%
dead end right	23273	5924	23224	0.209%	974	0.002%
tom thumb	12308	14077	12296	0.097%	14665	0%
tom thumb left	19138	11258	19119	0.103%	3775	0.001%
tom thumb right	20143	11509	20124	0.095%	3796	0.001%
splatoon	4546	3887	4496	1.113%	15000	0.001%
splatoon left	4296	3347	4245	1.181%	15000	0.001%
splatoon right	4262	3328	4211	1.19%	15000	0.001%
splatoon dead end	4165	3394	4119	1.107%	15000	0.001%
splatoon dead end left	3987	2962	3942	1.127%	15000	0.001%
splatoon dead end right	3976	2961	3932	1.116%	15000	0.001%

TABLE 25 – Statistiques pour les labyrinthes de taille 64×64 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	25592	20628	25535	0.223%	40644	0.001%
fog left	12179	17222	12056	1.005%	15591	0.001%
fog right	12085	17309	11963	1.016%	15595	**0.002**%
dead end	25016	20328	24962	0.216%	44403	0.001%
dead end left	12039	16886	11953	0.708%	15950	0.001%
dead end right	11831	16823	11748	0.703%	16350	0.001%
tom thumb	14887	16751	14871	0.104%	50063	0%
tom thumb left	13944	19842	13923	0.152%	24151	0.001%
tom thumb right	14622	20472	14601	0.144%	23301	0.001%
splatoon	6726	8607	6651	1.116%	59772	0.001%
splatoon left	5266	6471	5190	1.445%	59879	0.001%
splatoon right	5207	5863	5130	**1.486**%	59867	0.001%
splatoon dead end	5111	5749	5050	1.208%	59967	0.001%
splatoon dead end left	4288	4060	4228	1.403%	59982	0.001%
splatoon dead end right	**4260**	**3750**	**4198**	1.445%	**59998**	0.001%

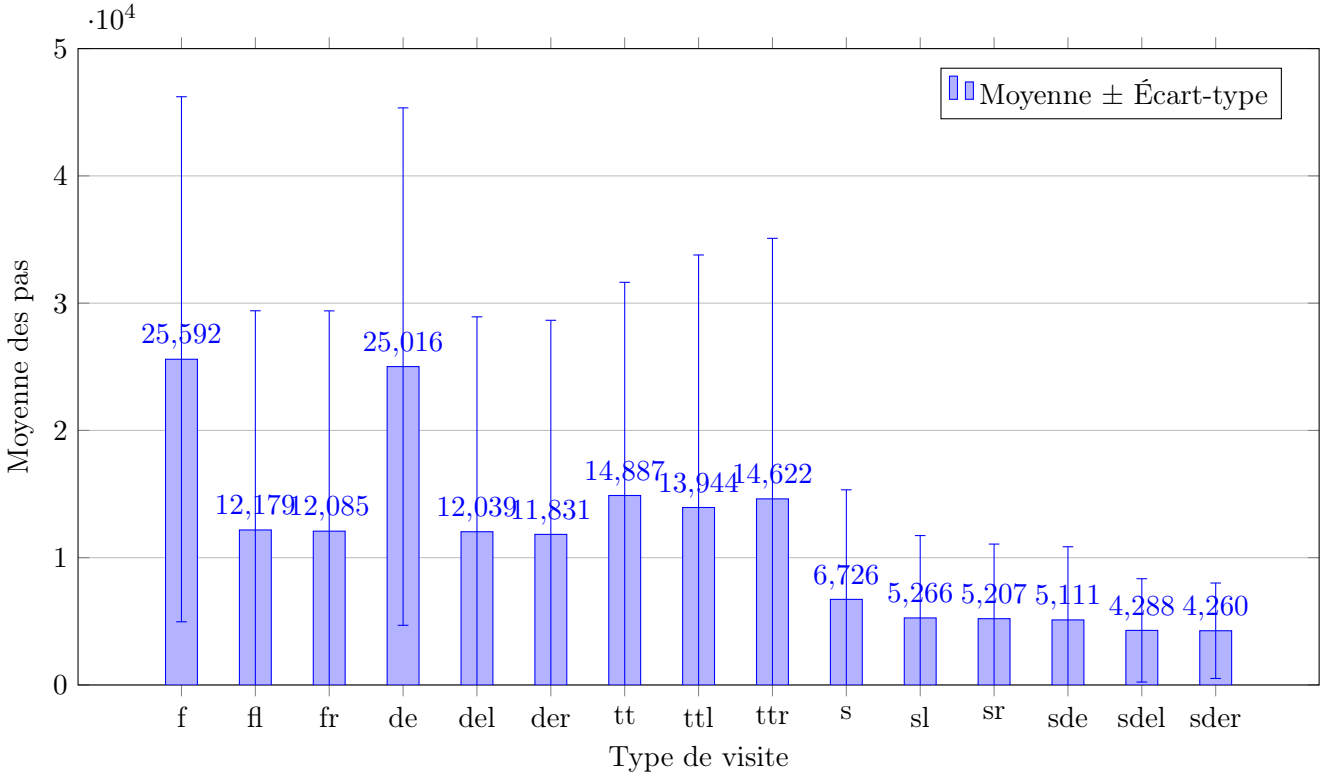


FIGURE 5 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (64×64, probabilité 0.25)

TABLE 26 – Algorithme back tracking 128×128 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	166139	120352	166023	0.07%	12457	0%
fog left	168421	24797	168319	0.061%	440	0.002%
fog right	164505	24556	164406	0.06%	427	0.002%
dead end	159257	118556	159152	0.066%	12852	0%
dead end left	157789	25308	157692	0.062%	444	0.002%
dead end right	169384	25261	169285	0.058%	481	0.002%
tom thumb	84761	96011	84733	0.032%	14563	0%
tom thumb left	152836	47347	152786	0.033%	1647	0.001%
tom thumb right	150987	47316	150936	0.034%	1636	0.001%
splatoon	20217	17454	20098	0.588%	15000	0%
splatoon left	19826	15561	19708	0.595%	15000	0%
splatoon right	19635	15367	19517	0.6%	15000	0%
splatoon dead end	18805	15399	18698	0.571%	15000	0%
splatoon dead end left	18507	14106	18400	0.576%	15000	0%
splatoon dead end right	18317	13930	18209	0.586%	15000	0%

TABLE 27 – Algorithme diagonal 128×128 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	135212	114740	135103	0.081%	13818	0%
fog left	153007	23951	152890	0.077%	429	0.002%
fog right	168413	24706	168299	0.068%	403	0%
dead end	136384	115479	136275	0.08%	13884	0%
dead end left	151895	24744	151781	0.075%	432	0.007%
dead end right	149916	22960	149800	0.077%	362	0.003%
tom thumb	60068	77170	60043	0.041%	14855	0%
tom thumb left	122307	48585	122260	0.039%	1789	0%
tom thumb right	126880	50493	126835	0.036%	1875	0%
splatoon	17826	15550	17714	0.628%	15000	0%
splatoon left	18786	15134	18673	0.603%	15000	0%
splatoon right	18626	14993	18515	0.599%	15000	0%
splatoon dead end	17360	14800	17249	0.639%	15000	0%
splatoon dead end left	18242	14770	18134	0.592%	15000	0%
splatoon dead end right	17859	14551	17751	0.606%	15000	0%

TABLE 28 – Algorithme fractal 128 × 128 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	179961	69364	179806	0.086%	3511	0.001%
fog left	44716	70466	44369	0.776%	11939	0%
fog right	39985	63482	39640	0.863%	11951	0%
dead end	182363	90299	182224	0.076%	5906	0%
dead end left	39001	61498	38765	0.606%	12580	0%
dead end right	46970	71254	46772	0.42%	11709	0%
tom thumb	169671	87189	169600	0.042%	5491	0%
tom thumb left	52933	75298	52883	0.095%	8625	0%
tom thumb right	55586	78566	55537	0.089%	8543	0%
splatoon	98151	101085	97838	0.32%	14375	0%
splatoon left	52543	70435	52237	0.584%	14805	0%
splatoon right	60747	79676	60442	0.501%	14837	0%
splatoon dead end	51147	67081	50948	0.39%	14897	0%
splatoon dead end left	29014	40739	28806	0.718%	14988	0%
splatoon dead end right	28464	42310	28258	0.723%	14990	0%

TABLE 29 – Algorithme wall maker 128 × 128 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	130765	111750	130668	0.074%	14084	0%
fog left	146344	20615	146248	0.066%	316	0%
fog right	151208	20908	151122	0.057%	313	0.003%
dead end	123078	109418	122991	0.071%	14347	0%
dead end left	152570	20984	152477	0.061%	317	0.003%
dead end right	156723	21609	156625	0.062%	332	0%
tom thumb	60845	76308	60824	0.035%	14889	0%
tom thumb left	145201	45612	145155	0.032%	1462	0%
tom thumb right	133756	43101	133711	0.033%	1353	0.001%
splatoon	19292	16242	19195	0.505%	15000	0%
splatoon left	19573	14915	19476	0.496%	15000	0%
splatoon right	19539	14972	19442	0.498%	15000	0%
splatoon dead end	17635	14254	17546	0.502%	15000	0%
splatoon dead end left	17964	13293	17877	0.486%	15000	0%
splatoon dead end right	17962	13215	17877	0.474%	15000	0%

TABLE 30 – Statistiques pour les labyrinthes de taille 128 × 128 imparfait avec une probabilité de 0.25 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	146147	125250	146037	0.076%	43870	0%
fog left	54851	92530	54525	0.593%	13124	**0.001**%
fog right	50657	87763	50333	**0.639**%	13094	0%
dead end	144357	124771	144251	0.073%	46989	0%
dead end left	48985	83753	48761	0.459%	13773	**0.001**%
dead end right	57260	94197	57072	0.33%	12884	0%
tom thumb	79607	98669	79577	0.037%	49798	0%
tom thumb left	84254	124472	84204	0.059%	13523	0%
tom thumb right	85087	126133	85039	0.057%	13407	0%
splatoon	38248	62715	38089	0.415%	59375	0%
splatoon left	27601	40288	27443	0.573%	59805	0%
splatoon right	29552	45652	29395	0.532%	59837	0%
splatoon dead end	26194	38710	26067	0.483%	59897	0%
splatoon dead end left	20930	**24193**	20803	0.61%	59988	0%
splatoon dead end right	**20649**	24761	**20522**	0.613%	**59990**	0%

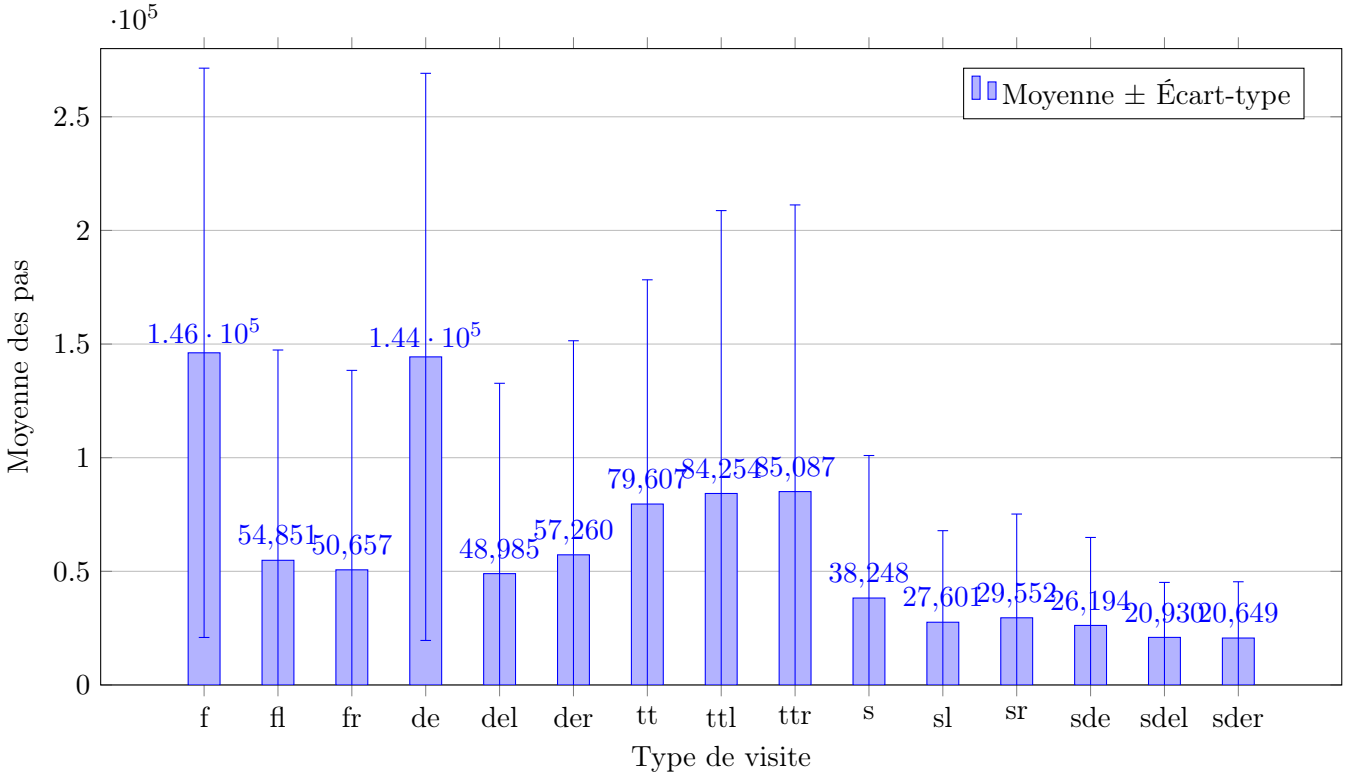


FIGURE 6 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (128 × 128, probabilité 0.25)

TABLE 31 – Algorithme back tracking 8 × 8 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	95	63	87	8.283%	9845	0.043%
fog left	80	66	72	10.558%	9235	0.044%
fog right	82	66	74	10.036%	8726	0.042%
dead end	95	65	87	8.113%	10216	0.043%
dead end left	82	66	75	9.376%	9330	0.037%
dead end right	81	66	73	9.665%	9384	0.044%
tom thumb	58	56	55	5.742%	13897	0.022%
tom thumb left	48	48	45	7.182%	14223	0.02%
tom thumb right	48	48	45	7.282%	14271	0.023%
splatoon	49	40	40	18.666%	14995	0.038%
splatoon left	43	34	34	20.542%	15000	0.035%
splatoon right	44	34	35	20.555%	15000	0.039%
splatoon dead end	46	37	38	18.056%	15000	0.038%
splatoon dead end left	43	32	35	18.542%	15000	0.034%
splatoon dead end right	43	33	34	19.132%	15000	0.036%

TABLE 32 – Algorithme diagonal 8 × 8 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	93	66	86	7.609%	11059	0.045%
fog left	82	61	75	8.39%	6937	0.051%
fog right	81	61	74	8.556%	7154	0.052%
dead end	92	66	86	7.503%	11298	0.042%
dead end left	82	62	76	8.062%	7400	0.052%
dead end right	84	61	77	7.805%	6979	0.053%
tom thumb	56	54	52	5.548%	14447	0.024%
tom thumb left	47	48	43	7.07%	14049	0.024%
tom thumb right	48	50	44	6.874%	13979	0.02%
splatoon	46	37	39	16.034%	15000	0.044%
splatoon left	41	30	34	18.38%	15000	0.036%
splatoon right	41	30	33	18.412%	15000	0.038%
splatoon dead end	45	35	38	16.156%	14999	0.039%
splatoon dead end left	40	29	33	17.419%	15000	0.039%
splatoon dead end right	41	30	34	17.551%	15000	0.039%

TABLE 33 – Algorithme fractal 8×8 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	95	60	87	8.403%	8398	0.056%
fog left	67	59	58	14.136%	12971	0.04%
fog right	67	59	57	14.626%	13018	0.042%
dead end	96	63	89	7.473%	9451	0.045%
dead end left	67	60	59	11.832%	12975	0.04%
dead end right	67	60	59	11.866%	12965	0.037%
tom thumb	73	63	70	3.962%	12263	0.028%
tom thumb left	46	39	43	6.195%	14394	0.022%
tom thumb right	48	41	45	5.937%	14338	0.023%
splatoon	63	53	53	15.968%	14939	0.045%
splatoon left	49	39	39	21.021%	14981	0.038%
splatoon right	50	40	40	19.831%	14990	0.04%
splatoon dead end	57	46	48	15.477%	14974	0.042%
splatoon dead end left	46	34	38	18.225%	14997	0.035%
splatoon dead end right	46	36	38	18.757%	14996	0.043%

TABLE 34 – Algorithme wall maker 8×8 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	93	67	86	7.452%	11313	0.046%
fog left	79	61	72	8.947%	7616	0.05%
fog right	83	62	76	8.497%	7606	0.051%
dead end	92	66	85	7.424%	11249	0.039%
dead end left	81	61	75	8.088%	7298	0.049%
dead end right	81	61	74	8.082%	7413	0.046%
tom thumb	56	54	53	5.277%	14385	0.024%
tom thumb left	47	49	44	6.822%	14116	0.023%
tom thumb right	47	48	44	6.829%	14103	0.023%
splatoon	47	37	39	15.703%	14996	0.042%
splatoon left	40	30	33	18.223%	15000	0.038%
splatoon right	41	30	34	18.011%	15000	0.038%
splatoon dead end	45	36	38	15.462%	15000	0.039%
splatoon dead end left	40	29	34	16.997%	15000	0.037%
splatoon dead end right	41	29	34	16.524%	15000	0.037%

TABLE 35 – Statistiques pour les labyrinthes de taille 8×8 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	94	78	87	7.897%	40615	**0.047**%
fog left	76	79	67	10.885%	36759	0.045%
fog right	76	80	68	10.819%	36504	0.046%
dead end	94	77	87	7.625%	42214	0.042%
dead end left	77	80	69	9.577%	37003	0.044%
dead end right	77	80	69	9.625%	36741	0.044%
tom thumb	60	60	57	5.103%	54992	0.024%
tom thumb left	47	47	44	6.821%	56782	0.022%
tom thumb right	48	48	44	6.731%	56691	0.022%
splatoon	51	43	43	16.57%	59930	0.042%
splatoon left	**43**	33	35	**19.629**%	59981	0.037%
splatoon right	44	34	36	19.257%	59990	0.039%
splatoon dead end	48	39	40	16.249%	59973	0.04%
splatoon dead end left	**43**	**31**	**35**	17.823%	**59997**	0.036%
splatoon dead end right	**43**	32	**35**	18.023%	59996	0.039%

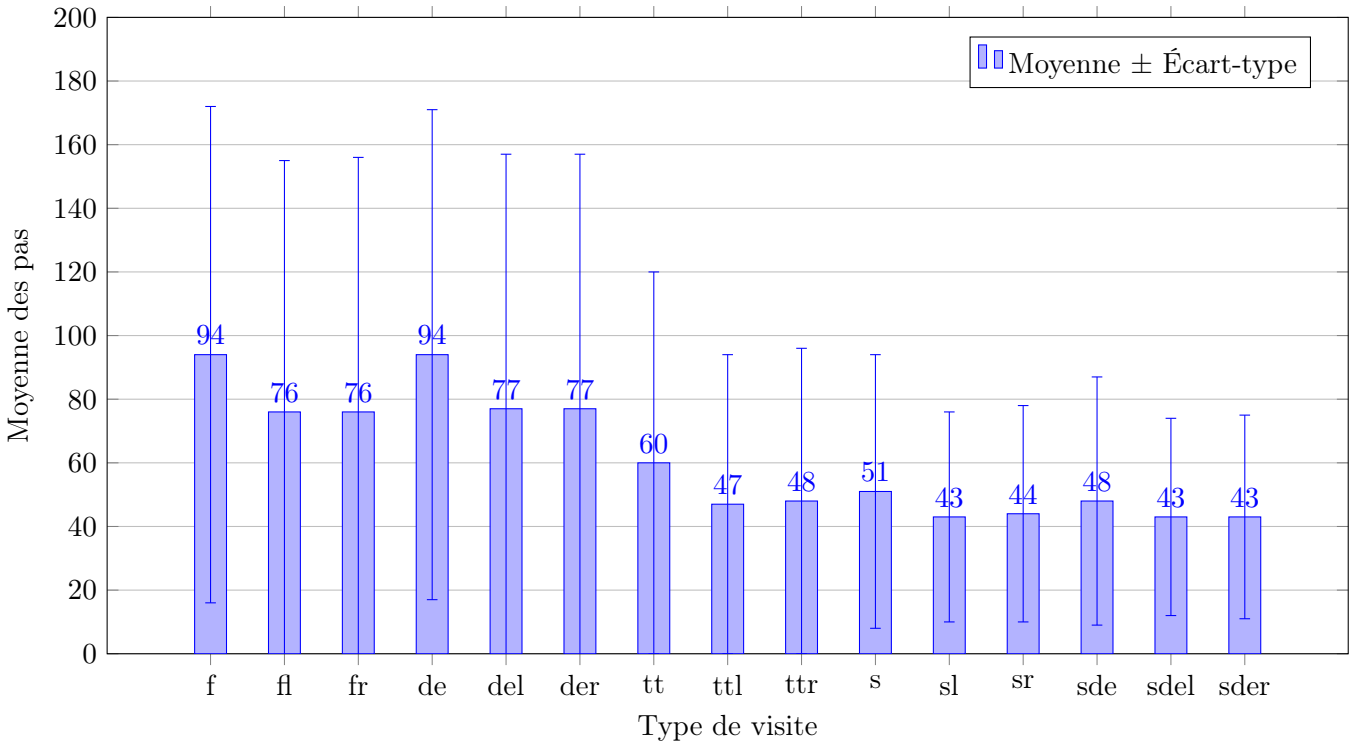


FIGURE 7 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (8×8, probabilité 0.5)

TABLE 36 – Algorithme back tracking 64×64 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	23802	18316	23751	0.215%	13137	0.001%
fog left	27915	4980	27869	0.163%	620	0.008%
fog right	26049	4685	26006	0.162%	582	0.002%
dead end	23207	18185	23157	0.213%	13303	0.001%
dead end left	25739	4965	25694	0.176%	653	0.008%
dead end right	26493	4802	26449	0.164%	592	0.008%
tom thumb	10914	12998	10901	0.124%	14766	0.001%
tom thumb left	21826	10105	21803	0.104%	2832	0.001%
tom thumb right	21720	10283	21699	0.101%	2870	0.001%
splatoon	4130	3418	4078	1.253%	15000	0.001%
splatoon left	4162	3212	4110	1.24%	15000	0.001%
splatoon right	4146	3201	4094	1.255%	15000	0.001%
splatoon dead end	3996	3237	3946	1.265%	15000	0.001%
splatoon dead end left	4070	3105	4020	1.229%	15000	0.001%
splatoon dead end right	4066	3070	4016	1.228%	15000	0.001%

TABLE 37 – Algorithme diagonal 64×64 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	19314	17005	19267	0.243%	14208	0.001%
fog left	27915	4980	27869	0.163%	620	0.008%
fog right	24977	4192	24930	0.186%	446	0.011%
dead end	19331	16939	19284	0.241%	14193	0.001%
dead end left	24053	3940	24009	0.18%	441	0.02%
dead end right	24371	3914	24327	0.183%	443	0.005%
tom thumb	7673	9744	7661	0.159%	14953	0.001%
tom thumb left	21463	10899	21441	0.102%	3123	0.001%
tom thumb right	21127	11150	21105	0.105%	3253	0.002%
splatoon	3575	2894	3528	1.316%	15000	0.001%
splatoon left	3921	3016	3873	1.213%	15000	0.001%
splatoon right	3921	3042	3873	1.205%	15000	0.001%
splatoon dead end	3566	2863	3519	1.321%	15000	0.001%
splatoon dead end left	3881	2990	3834	1.209%	15000	0.001%
splatoon dead end right	3854	2960	3808	1.209%	15000	0.001%

TABLE 38 – Algorithme fractal 64 × 64 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	27867	13109	27797	0.25%	5171	0.001%
fog left	13772	14059	13663	0.797%	8946	0.001%
fog right	12873	13429	12755	0.912%	9356	0.001%
dead end	28131	14308	28066	0.23%	6389	0.001%
dead end left	11707	13256	11616	0.78%	10132	0.001%
dead end right	11921	13239	11831	0.751%	10661	0.001%
tom thumb	24691	15944	24667	0.095%	8173	0.001%
tom thumb left	11232	13694	11212	0.18%	8644	0.001%
tom thumb right	11755	12758	11735	0.171%	6966	0%
splatoon	12747	13035	12632	0.905%	14865	0.001%
splatoon left	7857	9587	7734	1.563%	14963	0.001%
splatoon right	7370	9645	7255	1.562%	14954	0.001%
splatoon dead end	9225	9587	9134	0.986%	14964	0.001%
splatoon dead end left	5183	5644	5093	1.729%	14998	0.001%
splatoon dead end right	5791	6779	5700	1.569%	14996	0.001%

TABLE 39 – Algorithme wall maker 64 × 64 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	19425	17044	19380	0.235%	14280	0%
fog left	24373	3812	24329	0.18%	428	0.005%
fog right	25355	4091	25310	0.174%	441	0.007%
dead end	19111	16801	19066	0.237%	14321	0.001%
dead end left	24048	4005	24002	0.188%	422	0.021%
dead end right	22926	3681	22884	0.183%	427	0.007%
tom thumb	8010	10116	7999	0.145%	14939	0%
tom thumb left	19958	9551	19936	0.113%	2567	0.003%
tom thumb right	20858	9881	20835	0.106%	2513	0.002%
splatoon	3633	2933	3587	1.259%	15000	0.001%
splatoon left	3872	2871	3825	1.193%	15000	0.001%
splatoon right	3815	2850	3769	1.207%	15000	0.001%
splatoon dead end	3621	2878	3576	1.243%	15000	0.001%
splatoon dead end left	3815	2809	3770	1.168%	15000	0.001%
splatoon dead end right	3790	2792	3745	1.184%	15000	0.001%

TABLE 40 – Statistiques pour les labyrinthes de taille 64 × 64 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	21553	18899	21503	0.233%	46796	0.001%
fog left	15510	19480	15410	0.648%	10463	0.002%
fog right	14588	18618	14481	0.737%	10825	0.002%
dead end	21502	18800	21452	0.23%	48206	0.001%
dead end left	13408	17835	13323	0.636%	11648	**0.003**%
dead end right	13475	17292	13391	0.623%	12123	0.002%
tom thumb	11307	14516	11293	0.125%	52831	0.001%
tom thumb left	16146	21480	16125	0.132%	17166	0.001%
tom thumb right	17008	22224	16987	0.124%	15602	0.001%
splatoon	6006	8051	5941	1.08%	59865	0.001%
splatoon left	4951	5721	4884	1.353%	59963	0.001%
splatoon right	4811	5690	4746	1.352%	59954	0.001%
splatoon dead end	5099	5952	5041	1.145%	59964	0.001%
splatoon dead end left	**4237**	**3859**	**4179**	**1.364**%	**59998**	0.001%
splatoon dead end right	4375	4320	4317	1.327%	59996	0.001%

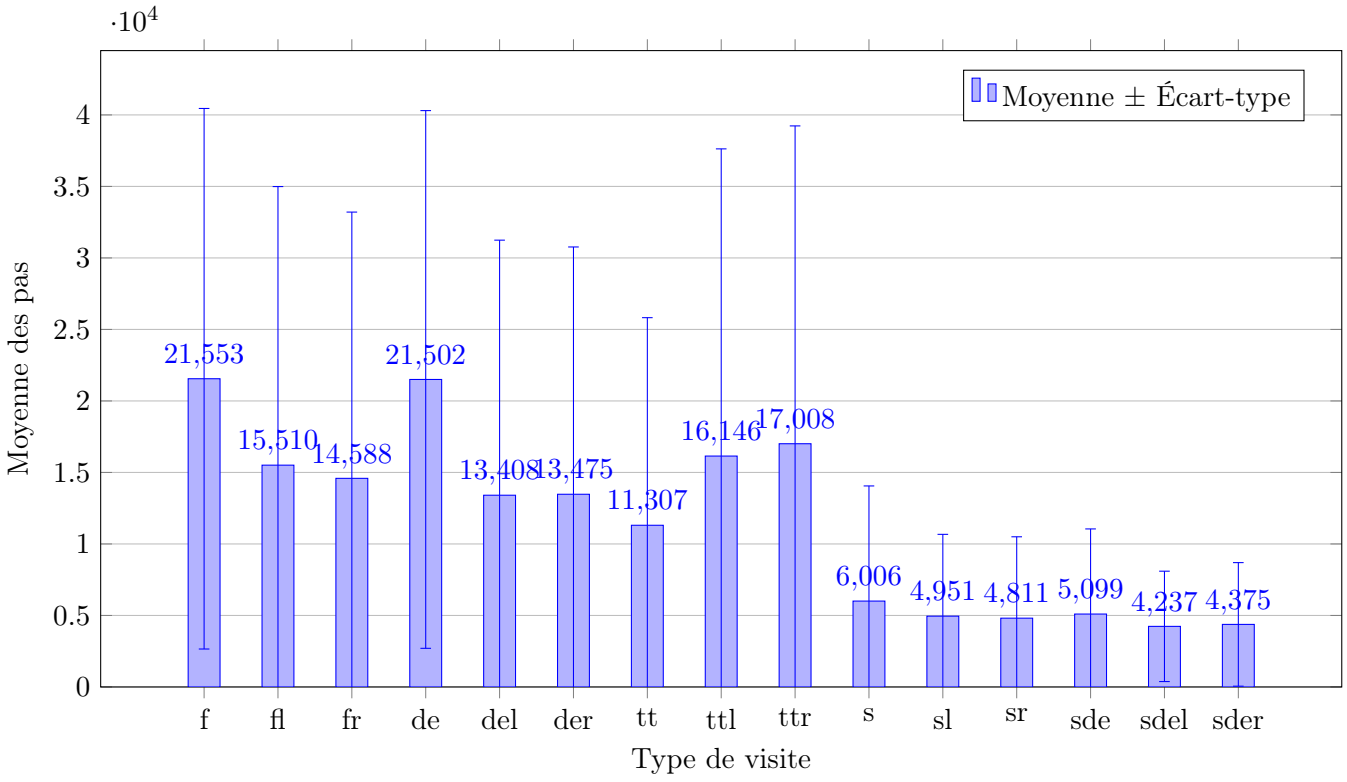


FIGURE 8 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (64 × 64, probabilité 0.5)

TABLE 41 – Algorithme back tracking 128×128 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	130021	111609	129923	0.076%	14175	0%
fog left	174623	17009	174538	0.048%	203	0.005%
fog right	178409	17310	178326	0.047%	209	0.01%
dead end	128227	112496	128132	0.074%	14287	0%
dead end left	161693	16681	161618	0.047%	190	0%
dead end right	181463	18333	181383	0.044%	214	0.005%
tom thumb	55854	70681	55832	0.04%	14915	0%
tom thumb left	145765	38802	145720	0.031%	1080	0.001%
tom thumb right	139275	35967	139231	0.031%	957	0.001%
splatoon	17581	14626	17481	0.565%	15000	0%
splatoon left	19239	14626	19140	0.516%	15000	0%
splatoon right	19414	14594	19315	0.512%	15000	0%
splatoon dead end	16905	13791	16809	0.57%	15000	0%
splatoon dead end left	18561	13907	18466	0.514%	15000	0%
splatoon dead end right	18735	13860	18639	0.512%	15000	0%

TABLE 42 – Algorithme diagonal 128×128 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	98030	95017	97938	0.094%	14784	0%
fog left	153645	14800	153554	0.06%	149	0%
fog right	159732	14128	159648	0.053%	154	0.013%
dead end	98281	94271	98189	0.093%	14767	0%
dead end left	142169	13702	142073	0.067%	156	0.006%
dead end right	147311	13915	147226	0.058%	138	0.014%
tom thumb	36362	47702	36342	0.057%	14987	0%
tom thumb left	143900	40784	143848	0.036%	1073	0.001%
tom thumb right	143096	40984	143047	0.035%	1106	0%
splatoon	14826	11985	14733	0.622%	15000	0%
splatoon left	19067	14759	18975	0.482%	15000	0%
splatoon right	19068	14706	18975	0.486%	15000	0%
splatoon dead end	14871	11912	14779	0.617%	15000	0%
splatoon dead end left	19065	14785	18974	0.482%	15000	0%
splatoon dead end right	19111	14759	19019	0.479%	15000	0%

TABLE 43 – Algorithme fractal 128×128 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	184724	81338	184577	0.08%	4647	0%
fog left	73265	79940	72993	0.371%	8237	0%
fog right	67797	79735	67511	0.421%	8846	0%
dead end	185376	89972	185235	0.076%	5778	0.001%
dead end left	76260	78896	76048	0.277%	8350	0%
dead end right	76657	77819	76461	0.256%	7829	0%
tom thumb	162883	99090	162829	0.033%	7342	0%
tom thumb left	68778	72079	68730	0.069%	6082	0%
tom thumb right	71646	73978	71598	0.067%	5807	0%
splatoon	81758	86676	81503	0.312%	14793	0%
splatoon left	53586	75393	53319	0.499%	14879	0%
splatoon right	47799	65220	47546	0.531%	14887	0%
splatoon dead end	54287	61870	54091	0.361%	14983	0%
splatoon dead end left	27286	34678	27092	0.711%	14994	0%
splatoon dead end right	28192	37019	27988	0.724%	14997	0%

TABLE 44 – Algorithme wall maker 128×128 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	98205	94599	98116	0.091%	14779	0%
fog left	146032	11999	145939	0.063%	119	0%
fog right	141235	12260	141142	0.066%	117	0%
dead end	95214	92514	95126	0.093%	14797	0%
dead end left	155663	14636	155578	0.054%	151	0.02%
dead end right	141712	12367	141615	0.068%	118	0%
tom thumb	37448	48995	37428	0.054%	14987	0%
tom thumb left	131254	34705	131206	0.036%	823	0.002%
tom thumb right	133487	35740	133438	0.037%	868	0.001%
splatoon	15414	12334	15323	0.585%	15000	0%
splatoon left	18184	13469	18093	0.496%	15000	0%
splatoon right	18235	13504	18145	0.491%	15000	0%
splatoon dead end	14963	11977	14875	0.588%	15000	0%
splatoon dead end left	17940	13181	17853	0.486%	15000	0%
splatoon dead end right	17664	13185	17577	0.492%	15000	0%

TABLE 45 – Statistiques pour les labyrinthes de taille 128×128 imparfait avec une probabilité de 0.5 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	115782	110419	115683	0.085%	48385	0%
fog left	77997	111944	77735	0.336%	8708	**0.001**%
fog right	72715	108294	72440	0.378%	9326	**0.001**%
dead end	116127	111221	116030	0.084%	49629	0%
dead end left	80612	109677	80407	0.254%	8847	**0.001**%
dead end right	81460	112173	81270	0.233%	8299	0%
tom thumb	60025	86003	59999	0.043%	52231	0%
tom thumb left	92533	130427	92485	0.052%	9058	0%
tom thumb right	94240	133071	94192	0.051%	8738	0%
splatoon	32224	53106	32090	0.415%	59793	0%
splatoon left	27466	42465	27329	0.499%	59879	0%
splatoon right	26088	37077	25955	0.512%	59887	0%
splatoon dead end	25248	36845	25130	0.467%	59983	0%
splatoon dead end left	**20713**	**21487**	**20595**	0.565%	59994	0%
splatoon dead end right	20925	22505	20806	**0.572**%	**59997**	0%

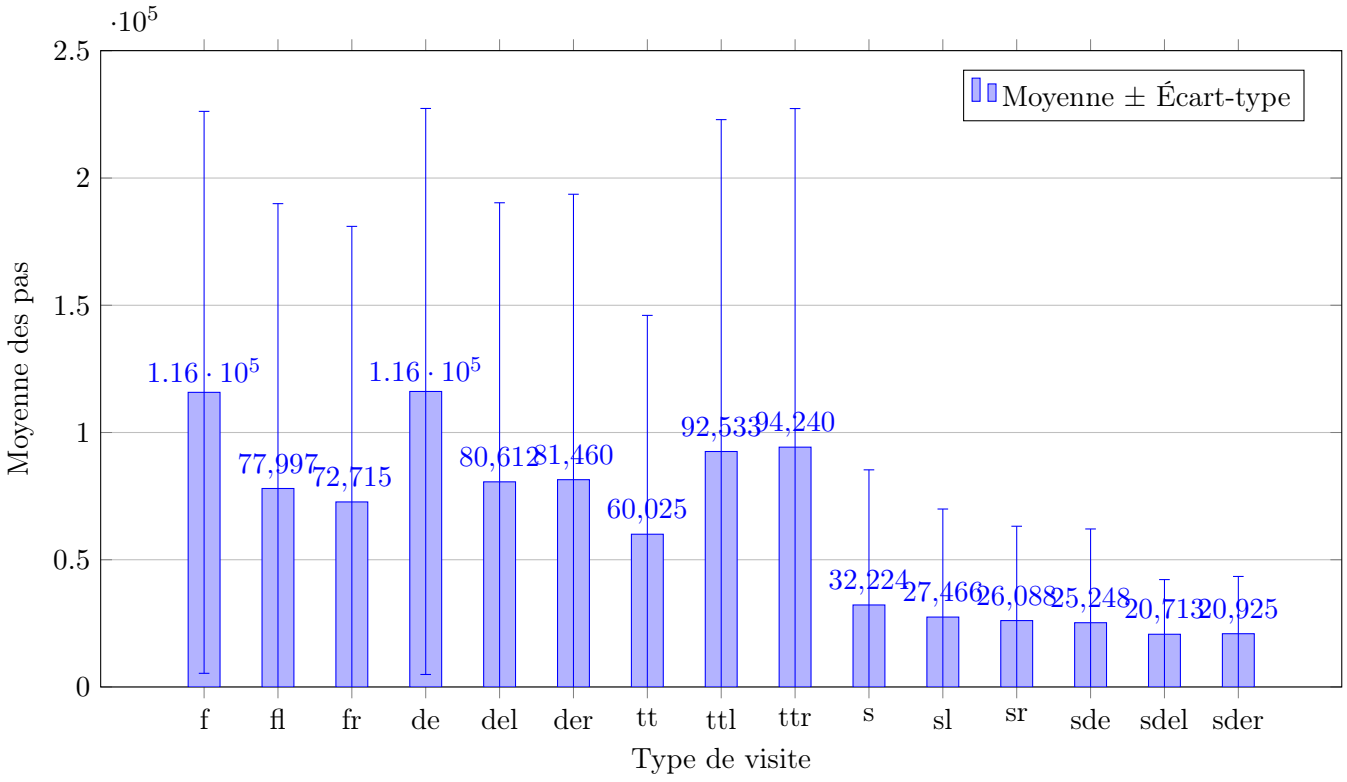


FIGURE 9 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (128×128, probabilité 0.5)

TABLE 46 – Algorithme back tracking 8×8 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	95	66	88	7.747%	10730	0.048%
fog left	84	63	76	8.874%	7489	0.046%
fog right	86	64	79	8.268%	7340	0.048%
dead end	93	66	86	7.814%	11009	0.046%
dead end left	87	65	80	8.275%	7699	0.053%
dead end right	84	64	77	8.285%	7574	0.048%
tom thumb	56	54	52	5.894%	14310	0.022%
tom thumb left	50	54	47	6.703%	13863	0.022%
tom thumb right	50	53	47	6.93%	13768	0.024%
splatoon	45	36	37	17.593%	15000	0.04%
splatoon left	42	32	34	18.943%	15000	0.034%
splatoon right	42	31	34	19.018%	15000	0.035%
splatoon dead end	45	35	37	17.458%	14998	0.04%
splatoon dead end left	42	31	34	18.386%	15000	0.035%
splatoon dead end right	42	31	34	18.178%	15000	0.033%

TABLE 47 – Algorithme diagonal 8×8 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	91	68	84	7.391%	12135	0.041%
fog left	90	57	84	7.053%	5383	0.054%
fog right	88	58	81	7.275%	5495	0.056%
dead end	89	66	83	7.487%	12140	0.04%
dead end left	88	57	82	6.997%	5297	0.056%
dead end right	88	57	82	7.02%	5389	0.06%
tom thumb	50	48	47	6.253%	14735	0.025%
tom thumb left	44	47	41	7.75%	14047	0.025%
tom thumb right	45	49	42	7.615%	14105	0.025%
splatoon	42	32	35	16.501%	15000	0.044%
splatoon left	37	26	30	18.753%	15000	0.04%
splatoon right	37	26	30	18.727%	15000	0.044%
splatoon dead end	41	31	34	16.652%	15000	0.041%
splatoon dead end left	37	26	30	18.609%	15000	0.041%
splatoon dead end right	37	26	31	18.281%	15000	0.039%

TABLE 48 – Algorithme fractal 8×8 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	94	62	87	8.162%	8999	0.055%
fog left	72	63	63	12.391%	11585	0.044%
fog right	69	62	61	12.674%	11709	0.043%
dead end	95	62	88	7.844%	9342	0.051%
dead end left	74	64	66	11.015%	11477	0.042%
dead end right	73	64	65	11.375%	11494	0.038%
tom thumb	70	62	67	3.996%	12810	0.025%
tom thumb left	46	42	43	6.166%	14256	0.022%
tom thumb right	45	43	43	6.232%	14059	0.024%
splatoon	57	48	48	15.763%	14974	0.05%
splatoon left	45	36	36	20.034%	14992	0.045%
splatoon right	45	35	36	20.162%	14989	0.042%
splatoon dead end	54	45	46	15.448%	14980	0.048%
splatoon dead end left	44	35	35	19.541%	14990	0.041%
splatoon dead end right	44	35	36	19.262%	14996	0.043%

TABLE 49 – Algorithme wall maker 8×8 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	91	67	85	7.321%	12093	0.039%
fog left	86	58	79	7.373%	5487	0.061%
fog right	87	57	81	7.206%	5367	0.062%
dead end	91	67	84	7.365%	12146	0.044%
dead end left	86	56	80	7.286%	5392	0.057%
dead end right	88	57	82	6.91%	5324	0.062%
tom thumb	50	47	47	6.224%	14710	0.024%
tom thumb left	46	49	42	7.349%	14102	0.025%
tom thumb right	45	49	42	7.496%	13958	0.027%
splatoon	42	32	35	16.326%	15000	0.042%
splatoon left	37	26	31	18.435%	15000	0.041%
splatoon right	38	27	31	18.384%	15000	0.041%
splatoon dead end	42	32	35	16.397%	15000	0.045%
splatoon dead end left	38	26	31	17.889%	15000	0.04%
splatoon dead end right	37	26	31	18.063%	15000	0.043%

TABLE 50 – Statistiques pour les labyrinthes de taille 8 × 8 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	93	77	86	7.622%	43957	0.045%
fog left	81	86	73	9.431%	29944	0.049%
fog right	80	86	73	9.357%	29911	**0.05**%
dead end	92	76	85	7.613%	44637	0.045%
dead end left	82	87	75	8.791%	29865	**0.05**%
dead end right	81	86	74	8.838%	29781	0.049%
tom thumb	56	55	53	5.513%	56565	0.024%
tom thumb left	47	50	43	6.979%	56268	0.024%
tom thumb right	46	51	43	7.065%	55890	0.025%
splatoon	47	38	39	16.502%	59974	0.044%
splatoon left	**40**	31	**33**	19.085%	59992	0.04%
splatoon right	**40**	**30**	**33**	**19.119**%	59989	0.041%
splatoon dead end	46	37	38	16.435%	59978	0.044%
splatoon dead end left	**40**	**30**	**33**	18.634%	59990	0.04%
splatoon dead end right	**40**	**30**	**33**	18.472%	**59996**	0.04%

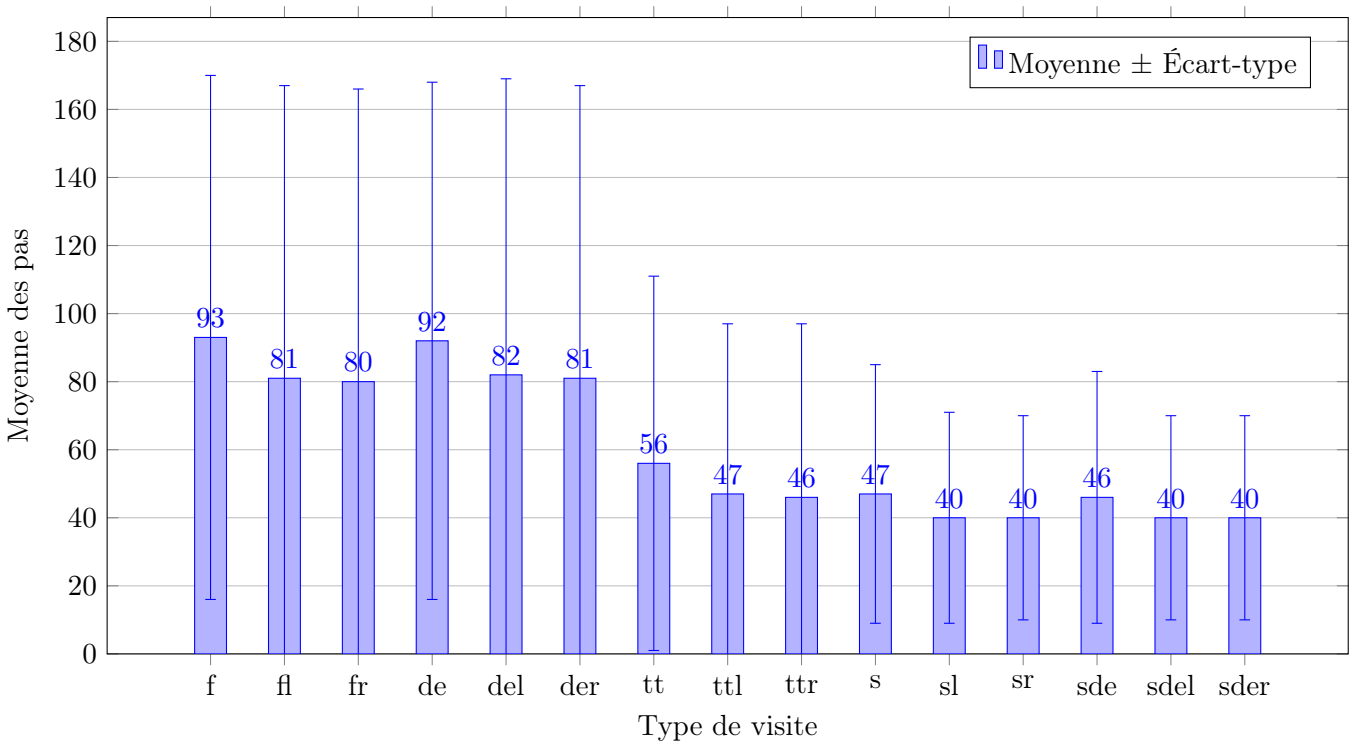


FIGURE 10 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (8 × 8, probabilité 0.75)

TABLE 51 – Algorithme back tracking 64×64 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	21290	17762	21242	0.224%	13871	0.001%
fog left	27657	3867	27616	0.149%	401	0.012%
fog right	27470	4069	27428	0.152%	413	0.015%
dead end	21264	17727	21217	0.221%	13937	0.001%
dead end left	24727	3750	24686	0.165%	417	0.014%
dead end right	28396	4089	28355	0.143%	433	0.007%
tom thumb	9156	11345	9144	0.138%	14900	0.001%
tom thumb left	22695	9557	22671	0.106%	2319	0.002%
tom thumb right	22076	9282	22053	0.104%	2254	0.002%
splatoon	3767	3076	3719	1.278%	15000	0.001%
splatoon left	4020	3036	3972	1.197%	15000	0.001%
splatoon right	4013	3025	3964	1.207%	15000	0.001%
splatoon dead end	3771	3049	3723	1.26%	15000	0.001%
splatoon dead end left	3983	2980	3936	1.191%	15000	0.001%
splatoon dead end right	3965	2964	3917	1.195%	15000	0.001%

TABLE 52 – Algorithme diagonal 64×64 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	16464	15397	16419	0.27%	14664	0.001%
fog left	24199	2762	24157	0.172%	236	0.013%
fog right	23234	2885	23191	0.183%	276	0.018%
dead end	16610	15503	16566	0.268%	14619	0.001%
dead end left	23280	2963	23236	0.186%	242	0.021%
dead end right	23560	2729	23518	0.178%	232	0.022%
tom thumb	6113	7836	6101	0.192%	14993	0.001%
tom thumb left	18287	10968	18268	0.106%	3615	0.002%
tom thumb right	19251	11450	19232	0.099%	3669	0.001%
splatoon	3248	2525	3203	1.378%	15000	0.001%
splatoon left	3159	2326	3115	1.416%	15000	0.001%
splatoon right	3216	2372	3171	1.392%	15000	0.001%
splatoon dead end	3260	2542	3215	1.368%	15000	0.001%
splatoon dead end left	3166	2312	3121	1.417%	15000	0.001%
splatoon dead end right	3223	2340	3178	1.386%	15000	0.001%

TABLE 53 – Algorithme fractal 64×64 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	27966	14610	27899	0.237%	6564	0.003%
fog left	24199	2762	24157	0.172%	236	0.013%
fog right	18765	12912	18676	0.475%	5814	0.002%
dead end	28368	15148	28302	0.231%	6974	0.002%
dead end left	17110	13245	17030	0.468%	6865	0.002%
dead end right	16827	13101	16743	0.495%	6714	0%
tom thumb	23737	16955	23717	0.085%	9733	0.001%
tom thumb left	13463	12705	13444	0.145%	5717	0%
tom thumb right	13305	12380	13285	0.15%	5808	0.001%
splatoon	9970	9840	9882	0.883%	14983	0.001%
splatoon left	6548	8133	6459	1.353%	14991	0.001%
splatoon right	6319	7976	6226	1.474%	14990	0.001%
splatoon dead end	8817	8861	8735	0.937%	14980	0.001%
splatoon dead end left	5844	6617	5762	1.395%	14992	0.001%
splatoon dead end right	5667	6350	5585	1.457%	14999	0.001%

TABLE 54 – Algorithme wall maker 64×64 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	16582	15667	16538	0.267%	14642	0%
fog left	22948	2891	22902	0.201%	257	0.012%
fog right	20987	2760	20946	0.194%	255	0.035%
dead end	16363	15294	16319	0.27%	14638	0.001%
dead end left	24729	2935	24685	0.178%	254	0.004%
dead end right	25341	3123	25302	0.151%	255	0.016%
tom thumb	6216	7877	6204	0.193%	14982	0.001%
tom thumb left	19549	10571	19528	0.107%	3163	0.001%
tom thumb right	19874	10651	19853	0.104%	3151	0.002%
splatoon	3260	2506	3215	1.379%	15000	0.001%
splatoon left	3114	2232	3069	1.437%	15000	0.001%
splatoon right	3139	2259	3095	1.422%	15000	0%
splatoon dead end	3251	2545	3207	1.368%	15000	0.001%
splatoon dead end left	3104	2203	3060	1.427%	15000	0.001%
splatoon dead end right	3074	2202	3029	1.442%	15000	0.001%

TABLE 55 – Statistiques pour les labyrinthes de taille 64×64 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	19362	17905	19314	0.249%	49741	0.001%
fog left	19477	21011	19394	0.426%	6480	0.003%
fog right	19563	21147	19481	0.422%	6758	**0.004**%
dead end	19466	17929	19418	0.247%	50168	0.001%
dead end left	17959	20111	17884	0.421%	7778	0.003%
dead end right	17972	20346	17894	0.434%	7634	0.002%
tom thumb	10113	13788	10099	0.134%	54608	0.001%
tom thumb left	17385	22415	17365	0.118%	14814	0.001%
tom thumb right	17490	22356	17470	0.116%	14882	0.001%
splatoon	5060	6149	5003	1.116%	59983	0.001%
splatoon left	4210	4837	4153	1.343%	59991	0.001%
splatoon right	4172	4747	4114	**1.384**%	59990	0.001%
splatoon dead end	4773	5539	4719	1.148%	59980	0.001%
splatoon dead end left	4024	4116	3969	1.355%	59992	0.001%
splatoon dead end right	**3982**	**3990**	**3927**	1.375%	**59999**	0.001%

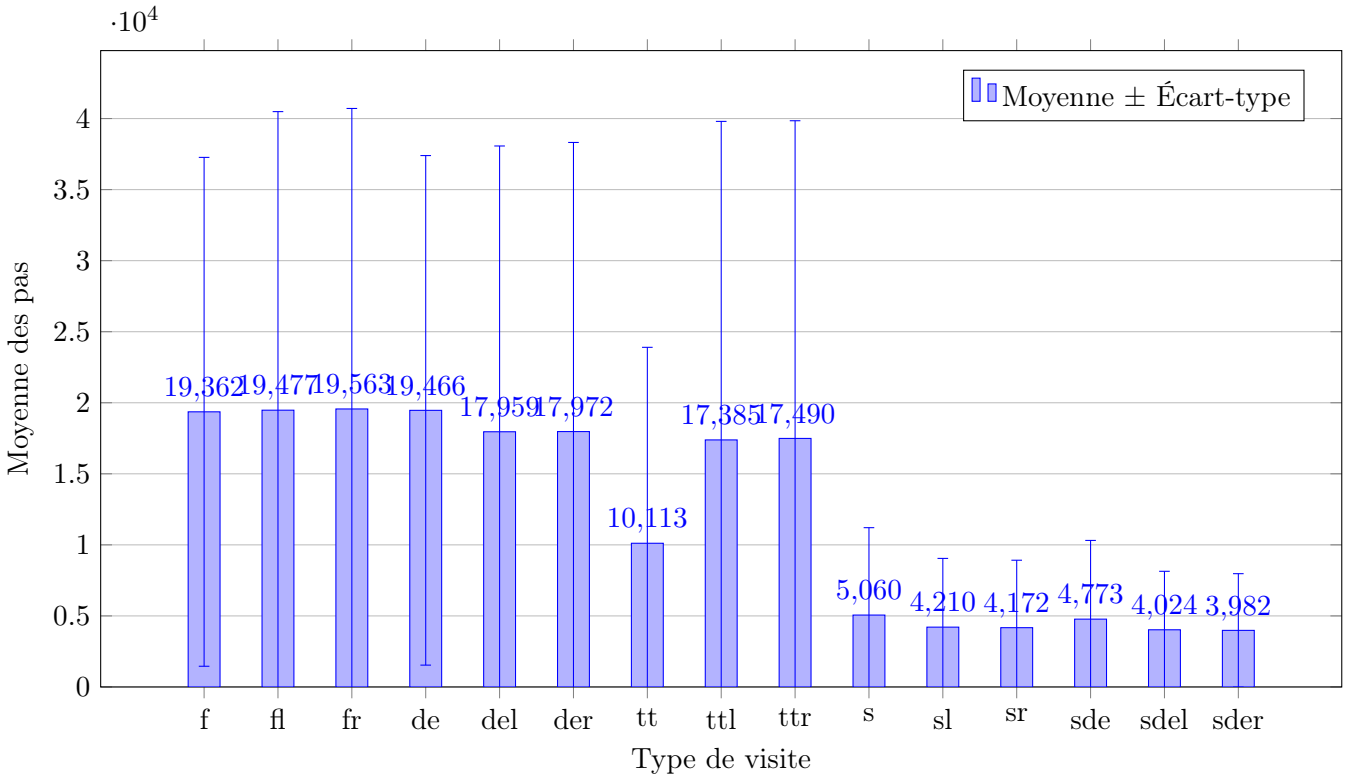


FIGURE 11 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (64×64, probabilité 0.75)

TABLE 56 – Algorithme back tracking 128×128 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	112005	103552	111911	0.083%	14599	0%
fog left	167052	14251	166971	0.049%	132	0%
fog right	193699	14546	193625	0.038%	137	0%
dead end	110610	102380	110518	0.083%	14627	0%
dead end left	164054	14766	163982	0.044%	150	0.013%
dead end right	174363	13921	174292	0.041%	118	0%
tom thumb	43804	57472	43783	0.048%	14973	0%
tom thumb left	146519	33800	146477	0.029%	778	0%
tom thumb right	149624	35280	149580	0.029%	796	0%
splatoon	15971	13050	15878	0.58%	15000	0%
splatoon left	18860	14242	18767	0.493%	15000	0%
splatoon right	19009	14209	18915	0.493%	15000	0%
splatoon dead end	15746	12751	15654	0.586%	15000	0%
splatoon dead end left	18739	14088	18648	0.488%	15000	0%
splatoon dead end right	18902	14062	18810	0.485%	15000	0%

TABLE 57 – Algorithme diagonal 128×128 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	80835	81699	80747	0.108%	14922	0%
fog left	165332	11465	165246	0.052%	79	0%
fog right	132632	9388	132560	0.054%	73	0.014%
dead end	80867	81154	80779	0.108%	14921	0%
dead end left	149858	10217	149770	0.058%	83	0%
dead end right	163162	9485	163081	0.05%	73	0%
tom thumb	27846	37385	27826	0.072%	14999	0%
tom thumb left	132733	41651	132689	0.033%	1338	0.001%
tom thumb right	131858	42814	131814	0.033%	1315	0%
splatoon	13631	10560	13544	0.642%	15000	0%
splatoon left	14925	11088	14837	0.591%	15000	0%
splatoon right	14913	11176	14826	0.587%	15000	0%
splatoon dead end	13592	10568	13504	0.651%	15000	0%
splatoon dead end left	14988	11186	14900	0.589%	15000	0%
splatoon dead end right	14778	11071	14690	0.593%	15000	0%

Table 58 – Algorithme fractal 128 × 128 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	185901	90394	185761	0.075%	5760	0%
fog left	95704	74536	95482	0.232%	5370	0%
fog right	99956	72666	99732	0.224%	5175	0%
dead end	187715	93822	187577	0.074%	6185	0%
dead end left	106123	74845	105928	0.183%	5260	0%
dead end right	109035	74471	108845	0.174%	4936	0.001%
tom thumb	157440	106611	157396	0.027%	8882	0%
tom thumb left	106432	70130	106388	0.042%	3691	0%
tom thumb right	79751	58998	79711	0.05%	3099	0%
splatoon	60555	64235	60356	0.327%	14969	0%
splatoon left	37935	51061	37725	0.553%	14980	0%
splatoon right	42469	59825	42271	0.468%	14981	0%
splatoon dead end	51495	52982	51308	0.364%	14991	0%
splatoon dead end left	38393	50893	38212	0.473%	14997	0%
splatoon dead end right	35482	45107	35302	0.509%	14998	0%

Table 59 – Algorithme wall maker 128 × 128 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	80643	80824	80555	0.109%	14928	0%
fog left	146416	9292	146334	0.056%	78	0.013%
fog right	162869	11366	162800	0.042%	81	0%
dead end	79852	80044	79765	0.109%	14940	0%
dead end left	167785	11021	167713	0.043%	88	0.011%
dead end right	150660	11175	150586	0.049%	71	0%
tom thumb	28430	36835	28410	0.07%	14996	0%
tom thumb left	147991	41777	147945	0.031%	1227	0%
tom thumb right	138611	39391	138566	0.032%	1152	0.001%
splatoon	13632	10571	13544	0.644%	15000	0.001%
splatoon left	13909	10053	13822	0.628%	15000	0%
splatoon right	14089	10156	14002	0.619%	15000	0%
splatoon dead end	13595	10615	13507	0.646%	15000	0%
splatoon dead end left	14098	10169	14010	0.62%	15000	0%
splatoon dead end right	13926	9992	13839	0.627%	15000	0%

TABLE 60 – Statistiques pour les labyrinthes de taille 128 × 128 imparfait avec une probabilité de 0.75 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	101894	103420	101799	0.093%	50209	0%
fog left	99039	126698	98825	0.217%	5659	0%
fog right	103674	126280	103458	0.208%	5466	**0.001**%
dead end	102195	103676	102100	0.093%	50673	0%
dead end left	109303	128130	109115	0.172%	5581	**0.001**%
dead end right	111847	131671	111663	0.165%	5198	**0.001**%
tom thumb	53821	83783	53797	0.045%	53850	0%
tom thumb left	123118	143799	123074	0.036%	7034	0%
tom thumb right	109922	141460	109880	0.038%	6362	0%
splatoon	25929	39118	25813	0.449%	59969	0%
splatoon left	21402	29209	21282	**0.559**%	59980	0%
splatoon right	22614	33724	22497	0.517%	59981	0%
splatoon dead end	23603	32536	23489	0.483%	59991	0%
splatoon dead end left	21554	29187	21442	0.521%	59997	0%
splatoon dead end right	**20772**	**26256**	**20660**	0.538%	**59998**	0%

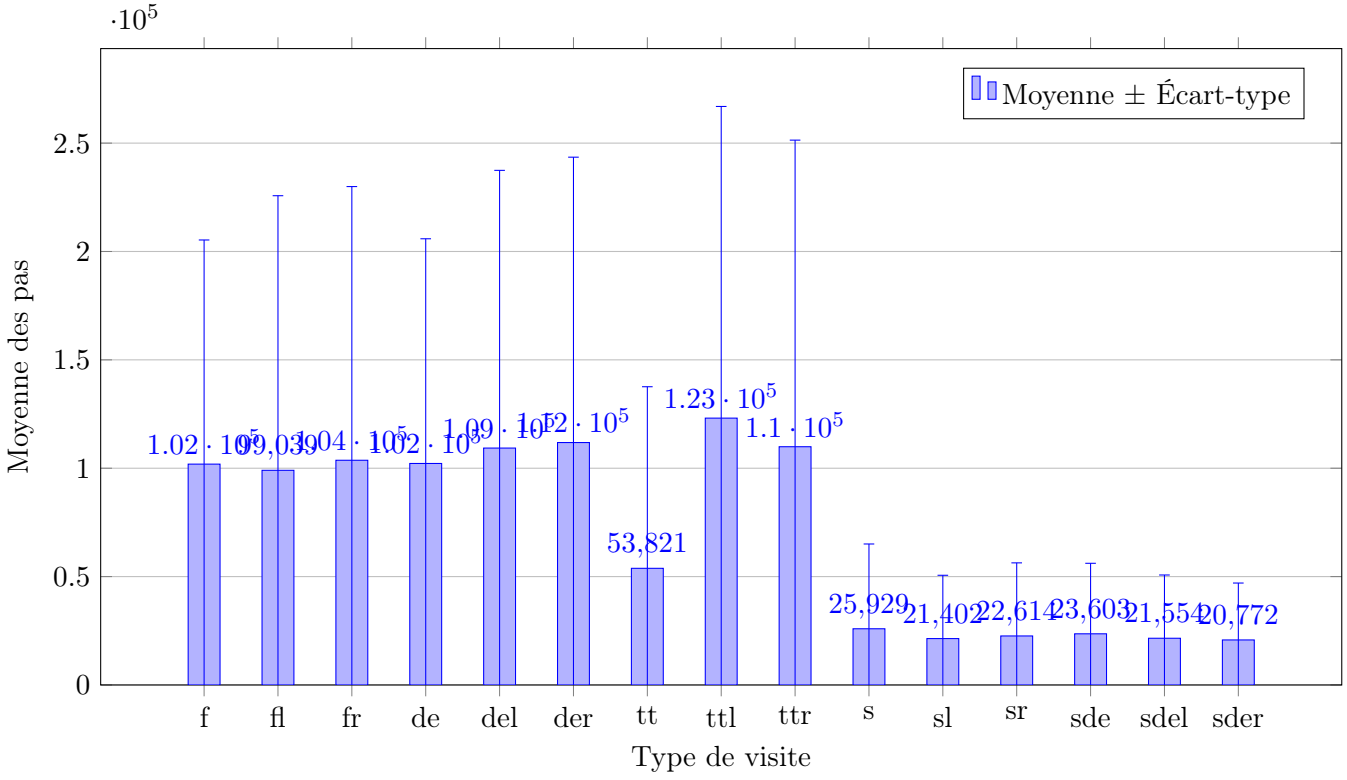


FIGURE 12 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (128 × 128, probabilité 0.75)

TABLE 61 – Algorithme back tracking 8×8 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	93	67	86	7.648%	11329	0.046%
fog left	91	62	85	7.415%	6342	0.052%
fog right	88	61	81	7.617%	6437	0.048%
dead end	94	68	87	7.55%	11373	0.043%
dead end left	89	62	82	7.502%	6467	0.047%
dead end right	89	62	82	7.455%	6510	0.051%
tom thumb	54	52	50	6.006%	14581	0.024%
tom thumb left	51	55	47	6.736%	13707	0.023%
tom thumb right	51	56	48	6.556%	13804	0.024%
splatoon	43	33	36	17.505%	15000	0.041%
splatoon left	40	30	33	18.6%	15000	0.033%
splatoon right	41	31	34	18.407%	15000	0.036%
splatoon dead end	43	33	36	17.34%	15000	0.041%
splatoon dead end left	41	30	33	17.946%	15000	0.037%
splatoon dead end right	41	30	34	18.005%	15000	0.033%

TABLE 62 – Algorithme diagonal 8×8 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	87	67	81	7.509%	12747	0.037%
fog left	96	54	90	6.2%	4306	0.06%
fog right	96	54	91	6.127%	4233	0.069%
dead end	88	67	81	7.444%	12702	0.039%
dead end left	94	54	88	6.304%	4291	0.068%
dead end right	92	53	86	6.469%	4211	0.065%
tom thumb	45	42	42	7.005%	14888	0.027%
tom thumb left	41	45	37	8.563%	14278	0.027%
tom thumb right	40	44	37	8.726%	14276	0.026%
splatoon	39	29	32	17.11%	15000	0.042%
splatoon left	33	21	27	20.108%	15000	0.042%
splatoon right	33	21	27	19.961%	15000	0.044%
splatoon dead end	39	29	33	17.1%	15000	0.046%
splatoon dead end left	33	21	27	20.169%	15000	0.043%
splatoon dead end right	34	21	27	19.857%	15000	0.044%

TABLE 63 – Algorithme fractal 8×8 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	97	65	89	7.796%	9798	0.052%
fog left	78	67	70	9.857%	9502	0.051%
fog right	77	66	69	9.865%	9241	0.046%
dead end	97	64	89	7.795%	9711	0.049%
dead end left	75	66	67	10.107%	9463	0.048%
dead end right	73	65	66	10.372%	9513	0.048%
tom thumb	65	60	62	4.571%	13727	0.027%
tom thumb left	45	48	42	6.5%	13625	0.025%
tom thumb right	46	50	44	5.902%	13414	0.024%
splatoon	53	43	44	15.487%	14990	0.051%
splatoon left	42	35	34	19.486%	15000	0.048%
splatoon right	42	35	34	19.514%	15000	0.043%
splatoon dead end	52	43	44	15.669%	14997	0.049%
splatoon dead end left	41	33	33	20.014%	15000	0.044%
splatoon dead end right	41	34	33	19.535%	15000	0.043%

TABLE 64 – Algorithme wall maker 8×8 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	89	67	82	7.372%	12770	0.04%
fog left	95	54	89	6.233%	4211	0.068%
fog right	92	54	86	6.353%	4279	0.061%
dead end	87	67	80	7.443%	12761	0.038%
dead end left	94	54	88	6.334%	4272	0.072%
dead end right	96	54	90	6.235%	4237	0.061%
tom thumb	45	41	42	6.954%	14894	0.024%
tom thumb left	40	44	36	8.828%	14284	0.028%
tom thumb right	40	44	36	8.881%	14232	0.03%
splatoon	39	28	32	17.302%	15000	0.046%
splatoon left	33	21	27	19.93%	15000	0.044%
splatoon right	33	21	26	20.145%	15000	0.046%
splatoon dead end	39	29	32	17.208%	15000	0.047%
splatoon dead end left	33	21	26	20.178%	15000	0.044%
splatoon dead end right	34	21	27	20.056%	15000	0.041%

TABLE 65 – Statistiques pour les labyrinthes de taille 8 × 8 imparfait avec une probabilité de 1 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	91	76	84	7.571%	46644	0.043%
fog left	87	94	81	7.801%	24361	**0.056**%
fog right	86	93	79	7.854%	24190	0.053%
dead end	91	76	84	7.548%	46547	0.042%
dead end left	85	93	79	7.928%	24493	0.055%
dead end right	85	93	78	8.016%	24471	0.054%
tom thumb	52	51	49	6.019%	58090	0.025%
tom thumb left	44	50	41	7.596%	55894	0.026%
tom thumb right	44	51	41	7.428%	55726	0.026%
splatoon	43	34	36	16.758%	59990	0.045%
splatoon left	**37**	28	**30**	19.485%	**60000**	0.042%
splatoon right	**37**	28	**30**	19.449%	**60000**	0.042%
splatoon dead end	43	35	36	16.753%	59997	0.046%
splatoon dead end left	**37**	**27**	**30**	**19.517**%	**60000**	0.042%
splatoon dead end right	**37**	**27**	**30**	19.304%	**60000**	0.04%

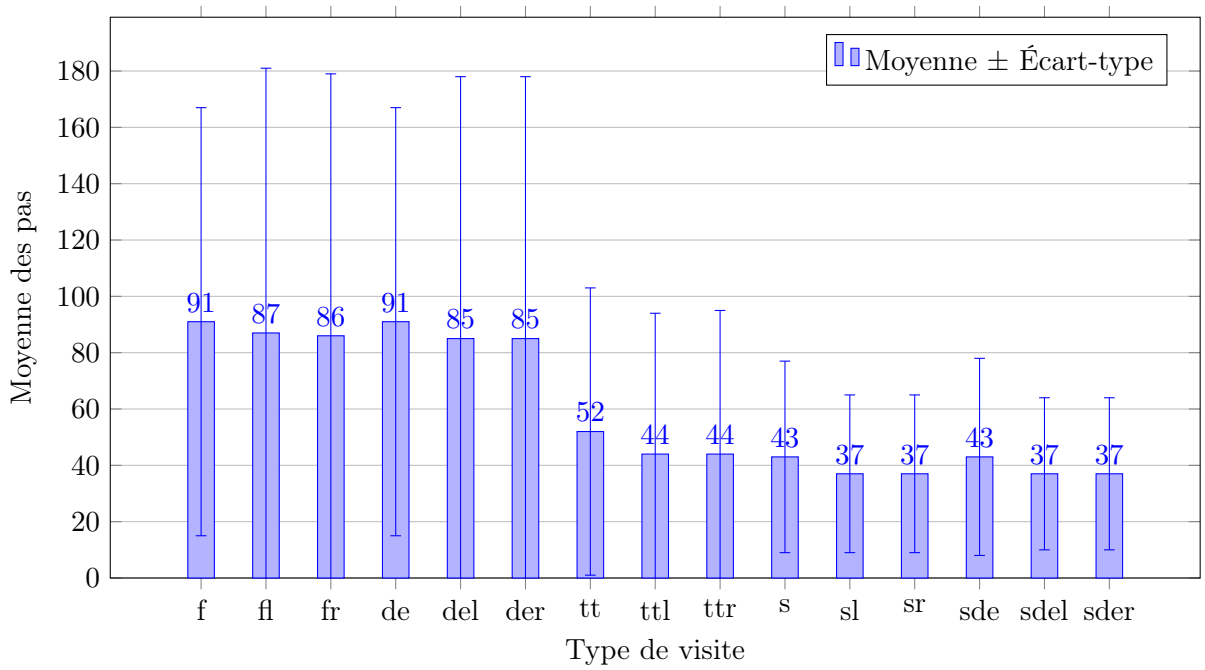


FIGURE 13 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (8 × 8, probabilité 1)

TABLE 66 – Algorithme back tracking 64 × 64 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	19397	17065	19350	0.238%	14307	0.001%
fog left	26045	3307	26007	0.148%	330	0.015%
fog right	25986	3266	25947	0.148%	305	0.003%
dead end	19418	16884	19372	0.237%	14268	0%
dead end left	25424	3342	25388	0.141%	326	0.015%
dead end right	26815	3475	26777	0.142%	324	0.012%
tom thumb	7893	9889	7881	0.152%	14945	0%
tom thumb left	22027	8918	22005	0.101%	2129	0.001%
tom thumb right	21907	8937	21883	0.109%	2065	0%
splatoon	3598	2862	3551	1.301%	15000	0.001%
splatoon left	3897	2893	3851	1.189%	15000	0.001%
splatoon right	3899	2913	3852	1.195%	15000	0.001%
splatoon dead end	3513	2782	3467	1.323%	15000	0.001%
splatoon dead end left	3850	2877	3803	1.199%	15000	0.001%
splatoon dead end right	3881	2875	3834	1.191%	15000	0.001%

TABLE 67 – Algorithme diagonal 64 × 64 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	14891	14432	14847	0.295%	14811	0.001%
fog left	22144	2102	22104	0.179%	169	0.036%
fog right	19911	1967	19872	0.199%	171	0.018%
dead end	14866	14456	14823	0.295%	14809	0.001%
dead end left	22697	2156	22653	0.195%	200	0.005%
dead end right	20183	2036	20147	0.178%	165	0.055%
tom thumb	5258	6509	5246	0.228%	14993	0.001%
tom thumb left	4628	10212	4613	0.341%	12501	0%
tom thumb right	4722	10515	4706	0.34%	12565	0%
splatoon	3022	2289	2978	1.464%	15000	0.001%
splatoon left	2223	1446	2179	1.98%	15000	0.001%
splatoon right	2209	1442	2165	2.007%	15000	0%
splatoon dead end	3050	2317	3005	1.444%	15000	0.001%
splatoon dead end left	2247	1475	2202	1.973%	15000	0.001%
splatoon dead end right	2241	1469	2196	1.979%	15000	0.001%

TABLE 68 – Algorithme fractal 64×64 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	28527	16032	28465	0.219%	7820	0.001%
fog left	21485	10594	21413	0.337%	3484	0.001%
fog right	21295	11023	21225	0.329%	3699	0.002%
dead end	28671	15963	28606	0.226%	7896	0.002%
dead end left	21617	10859	21546	0.324%	3583	0.002%
dead end right	21419	10802	21348	0.33%	3595	0.001%
tom thumb	21770	17898	21755	0.073%	11867	0.001%
tom thumb left	12266	11716	12251	0.123%	5147	0.002%
tom thumb right	12327	11537	12312	0.125%	5200	0%
splatoon	7991	7238	7919	0.906%	15000	0.001%
splatoon left	5846	6582	5774	1.244%	15000	0.001%
splatoon right	5954	6582	5881	1.218%	15000	0.001%
splatoon dead end	7844	7113	7770	0.939%	15000	0.001%
splatoon dead end left	5742	6347	5670	1.252%	15000	0.001%
splatoon dead end right	5636	6416	5562	1.317%	15000	0.001%

TABLE 69 – Algorithme wall maker 64×64 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	14888	14276	14844	0.296%	14790	0.001%
fog left	20003	1895	19966	0.186%	167	0.012%
fog right	20464	1970	20420	0.218%	156	0.026%
dead end	14705	14247	14661	0.3%	14834	0.001%
dead end left	21251	2061	21205	0.216%	182	0.016%
dead end right	21001	1993	20960	0.193%	163	0.031%
tom thumb	5159	6310	5147	0.23%	14991	0.001%
tom thumb left	4968	10866	4952	0.32%	12557	0%
tom thumb right	4447	9851	4431	0.362%	12539	0.001%
splatoon	3029	2298	2986	1.452%	15000	0.002%
splatoon left	2235	1445	2190	1.983%	15000	0.001%
splatoon right	2228	1437	2184	1.975%	15000	0.001%
splatoon dead end	3011	2285	2967	1.469%	15000	0.001%
splatoon dead end left	2233	1444	2189	1.984%	15000	0.001%
splatoon dead end right	2217	1449	2172	1.989%	15000	0.001%

TABLE 70 – Statistiques pour les labyrinthes de taille 64×64 imparfait avec une probabilité de 1 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	18198	17357	18151	0.261%	51728	0.001%
fog left	21815	21812	21748	0.307%	4150	0.004%
fog right	21541	22049	21475	0.305%	4331	0.003%
dead end	18178	17303	18130	0.262%	51807	0.001%
dead end left	21941	21987	21875	0.298%	4291	0.004%
dead end right	21766	22037	21701	0.301%	4247	**0.005**%
tom thumb	9375	13190	9363	0.136%	56796	0.001%
tom thumb left	7122	15059	7105	0.227%	32334	0.001%
tom thumb right	6933	14771	6917	0.238%	32369	0%
splatoon	4410	4701	4358	1.176%	**60000**	0.001%
splatoon left	3550	4024	3498	1.46%	**60000**	0.001%
splatoon right	3572	4044	3520	1.452%	**60000**	0.001%
splatoon dead end	4354	4618	4302	1.196%	**60000**	0.001%
splatoon dead end left	3518	**3910**	3466	1.469%	**60000**	0.001%
splatoon dead end right	**3494**	3925	**3441**	**1.495**%	**60000**	0.001%

TABLE 71 – Algorithme back tracking 128×128 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	98134	93710	98043	0.092%	14800	0%
fog left	177370	13668	177297	0.041%	108	0%
fog right	183575	12059	183503	0.039%	99	0.01%
dead end	99602	95770	99512	0.09%	14826	0%
dead end left	141341	10259	141275	0.046%	91	0.022%
dead end right	173856	12196	173783	0.042%	100	0.02%
tom thumb	37455	49614	37434	0.055%	14985	0%
tom thumb left	134770	31622	134723	0.035%	726	0.001%
tom thumb right	155900	32668	155851	0.031%	671	0.001%
splatoon	15072	12027	14981	0.601%	15000	0%
splatoon left	18382	13603	18291	0.491%	15000	0%
splatoon right	18369	13640	18278	0.492%	15000	0%
splatoon dead end	14997	11928	14906	0.601%	15000	0%
splatoon dead end left	18284	13571	18194	0.491%	15000	0%
splatoon dead end right	18232	13534	18142	0.494%	15000	0%

TABLE 72 – Algorithme diagonal 128×128 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	69658	71957	69572	0.124%	14967	0%
fog left	102591	4141	102477	0.111%	48	0%
fog right	100424	3589	100347	0.077%	33	0%
dead end	69336	70377	69250	0.125%	14957	0%
dead end left	85851	3939	85752	0.115%	44	0%
dead end right	91731	4126	91646	0.093%	34	0%
tom thumb	23719	29689	23699	0.085%	14999	0%
tom thumb left	14721	35814	14690	0.21%	11837	0%
tom thumb right	15744	38444	15714	0.19%	11818	0%
splatoon	12821	9768	12734	0.68%	15000	0%
splatoon left	8977	5849	8891	0.962%	15000	0%
splatoon right	9032	5938	8945	0.963%	15000	0%
splatoon dead end	12791	9696	12703	0.687%	15000	0%
splatoon dead end left	9036	5971	8950	0.957%	15000	0%
splatoon dead end right	8939	5920	8853	0.97%	15000	0%

TABLE 73 – Algorithme fractal 128×128 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	183261	100784	183130	0.071%	7449	0%
fog left	128871	53981	128716	0.12%	2397	0%
fog right	128328	54002	128174	0.12%	2351	0%
dead end	186435	102853	186304	0.07%	7612	0%
dead end left	130318	55884	130165	0.118%	2466	0%
dead end right	125036	54793	124884	0.122%	2471	0.001%
tom thumb	150702	116387	150667	0.023%	11091	0%
tom thumb left	76871	56360	76834	0.048%	3253	0%
tom thumb right	79529	57335	79490	0.048%	3181	0%
splatoon	42754	39139	42601	0.359%	15000	0%
splatoon left	34077	40490	33920	0.46%	15000	0%
splatoon right	32593	38298	32438	0.476%	15000	0%
splatoon dead end	44566	41015	44410	0.351%	15000	0%
splatoon dead end left	33043	40091	32886	0.476%	14999	0%
splatoon dead end right	33577	39505	33425	0.452%	15000	0.001%

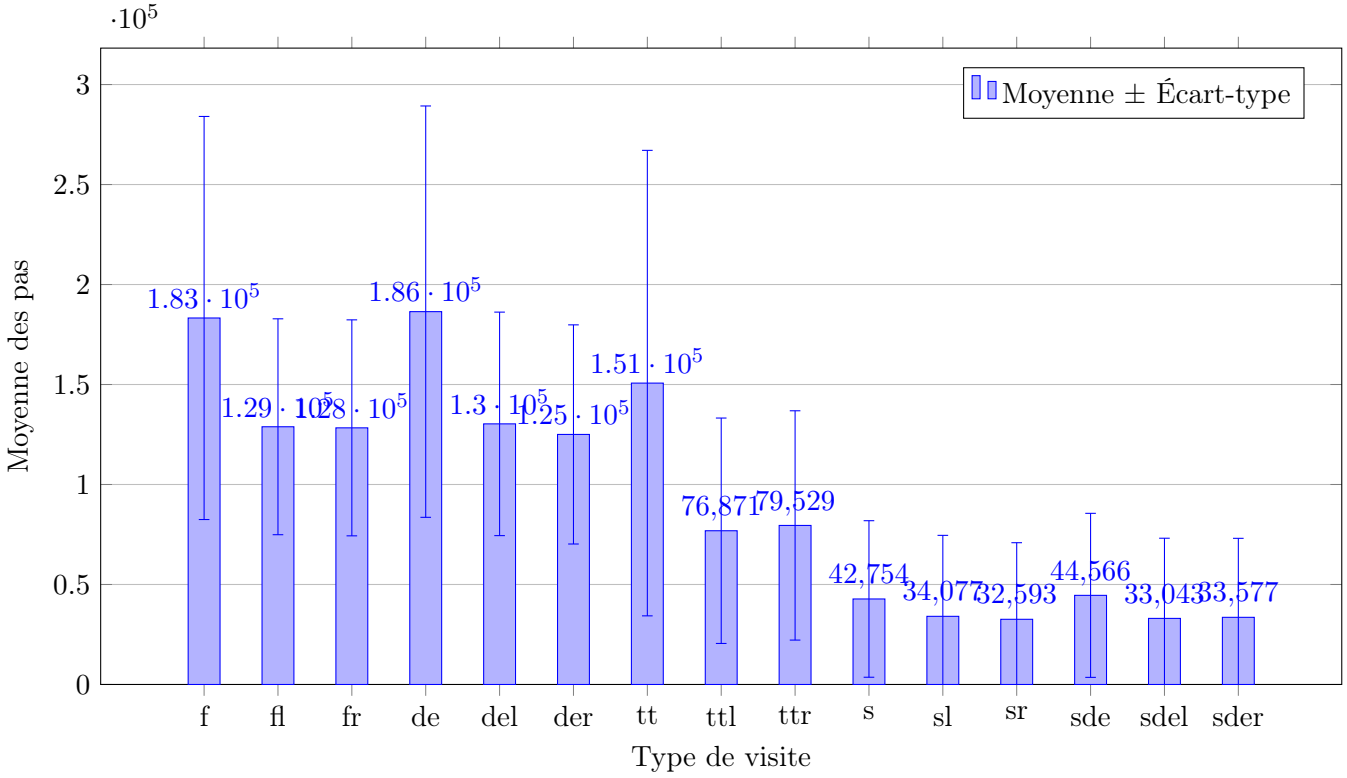


FIGURE 14 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (128 × 128, probabilité 1)

TABLE 74 – Algorithme wall maker 128 × 128 imparfait avec une probabilité de 1 pour 50 labyrinthes générés et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	69481	71752	69395	0.124%	14967	0%
fog left	85602	4499	85535	0.078%	51	0.059%
fog right	124121	4634	124029	0.075%	46	0.043%
dead end	69965	71651	69879	0.124%	14962	0%
dead end left	78818	4783	78742	0.097%	40	0.075%
dead end right	99643	5007	99557	0.087%	56	0%
tom thumb	23606	29684	23585	0.088%	14999	0%
tom thumb left	14834	35259	14803	0.21%	11877	0%
tom thumb right	15868	38447	15836	0.198%	11835	0%
splatoon	12606	9695	12519	0.69%	15000	0%
splatoon left	9054	5898	8968	0.957%	15000	0%
splatoon right	9016	5962	8929	0.964%	15000	0%
splatoon dead end	12716	9680	12629	0.679%	15000	0%
splatoon dead end left	9002	5920	8915	0.964%	15000	0%
splatoon dead end right	8892	5920	8805	0.973%	15000	0%

TABLE 75 – Statistiques pour les labyrinthes de taille 128 × 128 imparfait avec une probabilité de 1 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	93900	99483	93806	0.1%	52183	0%
fog left	129550	134993	129402	0.114%	2604	**0.002**%
fog right	130050	135978	129901	0.114%	2529	**0.002**%
dead end	95111	100683	95017	0.099%	52357	0%
dead end left	129177	136489	129029	0.115%	2641	**0.002**%
dead end right	125911	134595	125764	0.117%	2661	**0.002**%
tom thumb	52476	84654	52452	0.044%	56074	0%
tom thumb left	25217	65968	25185	0.127%	27693	0%
tom thumb right	26593	69419	26561	0.12%	27505	0%
splatoon	20813	25058	20709	0.502%	**60000**	0%
splatoon left	17623	24047	17518	0.596%	**60000**	0%
splatoon right	**17253**	**22889**	**17148**	**0.608**%	**60000**	0%
splatoon dead end	21267	26170	21162	0.494%	**60000**	0%
splatoon dead end left	17341	23707	17236	0.606%	59999	0%
splatoon dead end right	17410	23560	17306	0.596%	**60000**	0%

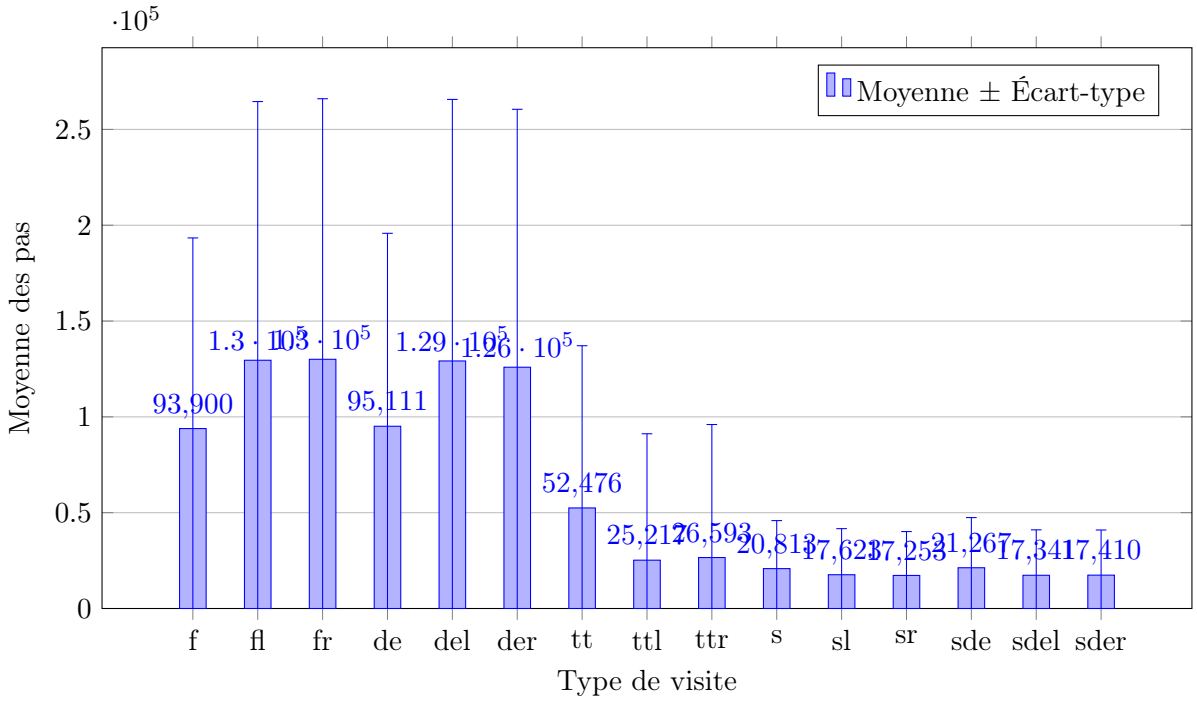


FIGURE 15 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues (128 × 128, probabilité 1)

TABLE 76 – Statistiques pour les labyrinthes de taille [8x8, 64x64, 128x128] imparfait avec des probabilités de 0, 0.25, 0.5, 0.75, 1 pour 50 labyrinthes générés avec 4 algorithmes et visités 300 fois

Type	Moyenne	Ecart-type	EAO	ERO	Valide	Optimale
fog	46606	84003	46551	0.119%	596021	0.016%
fog left	10460	37848	10236	2.149%	385923	**0.024**%
fog right	10311	37094	10084	**2.206**%	386083	0.023%
dead end	48450	87463	48392	0.12%	661187	0.014%
dead end left	10471	37365	10371	0.958%	390820	0.021%
dead end right	10486	37971	10390	0.924%	389866	0.02%
tom thumb	25255	61845	25235	0.081%	729310	0.009%
tom thumb left	10515	41106	10490	0.242%	553145	0.012%
tom thumb right	10300	40694	10274	0.249%	548962	0.012%
splatoon	15151	40009	15018	0.876%	894618	0.015%
splatoon left	11229	30294	11092	1.214%	896635	0.014%
splatoon right	11406	31091	11270	1.189%	896747	0.014%
splatoon dead end	9009	20454	8930	0.88%	899703	0.013%
splatoon dead end left	7744	15646	7665	1.019%	899935	0.012%
splatoon dead end right	**7686**	**15386**	**7606**	1.039%	**899967**	0.012%

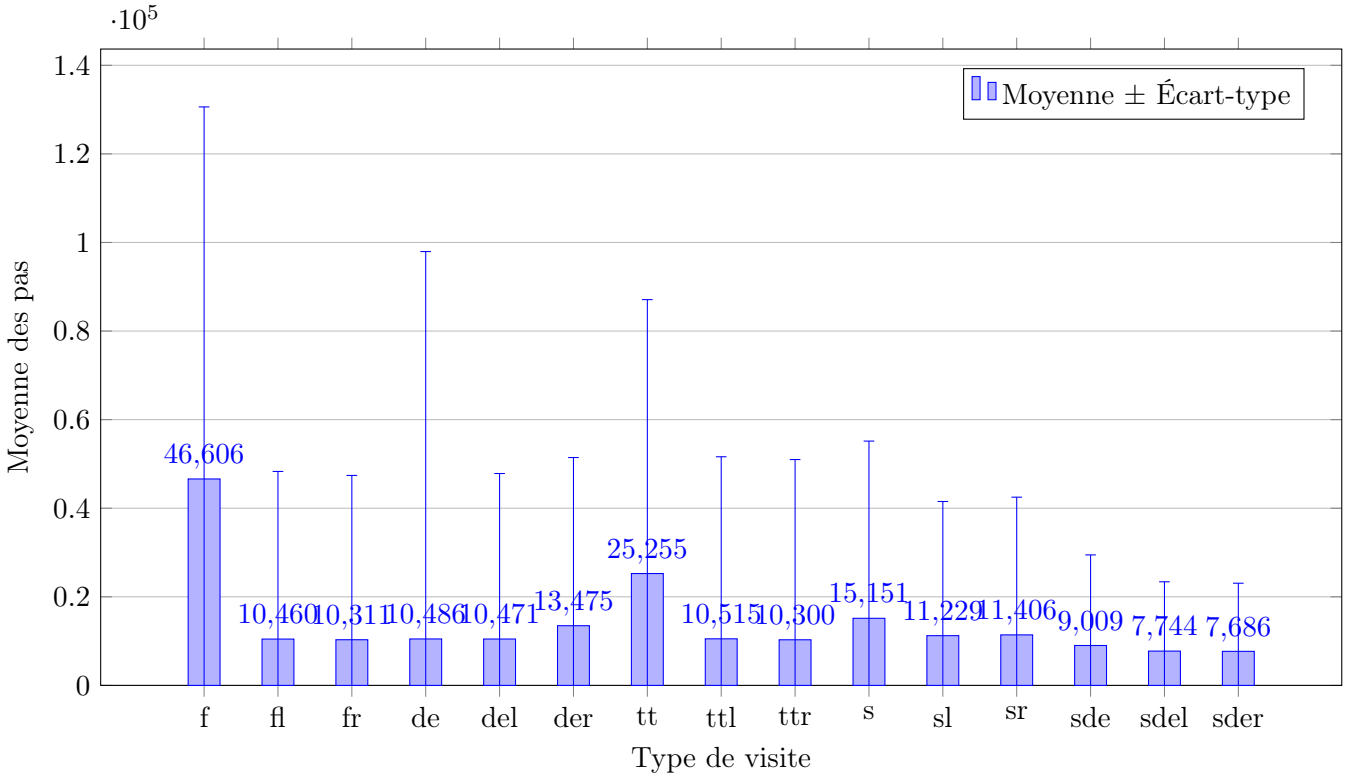


FIGURE 16 – Moyenne du nombre de pas et écart-type pour chaque type de visite toutes générations confondues ([8 × 8, 64 × 64, 128 × 128], probabilités 0, 0.25, 0.5, 0.75, 1)