Mini Projet Algorithmique

Word Maze Game

Préparé Par:

Yassine HACHANI

Mohamed Dhia SELMI

Table of Contents

[Introduction 1](#_Toc191157106)

[Presentation du projet 1](#_Toc191157107)

[Objectifs du projet 1](#_Toc191157108)

[Technologies utilisées 1](#_Toc191157109)

[Description du jeu 1](#_Toc191157110)

[Règles du jeu 1](#_Toc191157111)

[Fonctionnalités principals 2](#_Toc191157112)

[Modélisation du labyrinthe de mots 3](#_Toc191157113)

[Représentation des mots et du labyrinth 3](#_Toc191157114)

[Sélection et placement des mots 3](#_Toc191157115)

[Remplissage des cases vides 4](#_Toc191157116)

[Gestion des obstacles 4](#_Toc191157117)

[Implémentation 5](#_Toc191157118)

[Structure des données utilisée 5](#_Toc191157119)

[Structure de stockage du labyrinth 6](#_Toc191157120)

[Utilisation des Maps 6](#_Toc191157121)

[Dans la vérification des chemins 6](#_Toc191157122)

[Dans la gestion des chemins 6](#_Toc191157123)

[Conclusion et Perspectives 6](#_Toc191157124)

[Conclusion 6](#_Toc191157125)

[Perspectives 7](#_Toc191157126)

[Figure 1: Direction de mot dans la labyrinth de mot 4](#_Toc191157052)

[Figure 2 : Exemple de Labyrinth de Niveau Difficile 5](#_Toc191157053)

[Équation 1 : Formule de Calcule de Score 3](#_Toc191157080)

# Introduction

## Presentation du projet

Le projet est un jeu de labyrinthe de mots basé sur des algorithmes de recherche et d'optimisation. L’objectif principal est d’implémenter diverses stratégies de recherche de chemin dans un labyrinthe de mots généré aléatoirement en utilisant des algorithmes classiques d’exploration et d’optimisation.

## Technologies utilisées

* **Environnement de développement (IDE)** : VS Code
* **Random (java.util.Random)** : Utilisé pour générer aléatoirement le labyrinthe de mots.
* **Java Collections Framework (JCF)** : Utilisé pour manipuler les structures de données comme les listes, les files et les piles nécessaires aux algorithmes DFS, BFS et Dijkstra.
* **Swing (JFrame, JPanel, etc.)** : Utilisé pour concevoir l’interface graphique interactive du jeu.

# Description du jeu

## Règles du jeu

Le jeu commence par un **menu principal**, où le joueur doit choisir le niveau de difficulté parmi les options suivantes :

* **Débutant**
* **Intermédiaire**
* **Difficile**

Une fois le labyrinthe de mots généré aléatoirement en fonction du niveau choisi, le joueur a deux options :

1. **Jouer manuellement** : Le joueur tente de trouver un chemin entre le mot de départ et le mot cible en explorant le labyrinthe.
2. **Lancer un algorithme de recherche** : Le joueur peut observer comment l’un des trois algorithmes (**DFS, BFS ou Dijkstra**) résout automatiquement le labyrinthe.

## Fonctionnalités principals

 **Sélection du niveau de difficulté**

* Le joueur peut choisir entre trois niveaux : **Débutant, Intermédiaire et Difficile**, influençant la taille et la complexité du labyrinthe généré.

 **Génération aléatoire du labyrinthe de mots**

* Chaque partie est unique grâce à un algorithme qui génère aléatoirement la disposition des mots et les connexions entre eux.

 **Sélection de mots à partir d’un fichier texte**

* Les mots du labyrinthe sont extraits d’un fichier texte préalablement rempli, permettant une flexibilité dans le choix des mots utilisés dans le jeu.

 **Mode manuel (jeu interactif)**

* Le joueur peut explorer le labyrinthe en sélectionnant des mots pour essayer de rejoindre la destination.

 **Mode automatique (algorithmes de recherche)**

* Le joueur peut choisir d'exécuter un algorithme pour observer comment il trouve le chemin optimal :
  + **DFS (Depth-First Search)** : Exploration en profondeur.
  + **BFS (Breadth-First Search)** : Exploration en largeur.
  + **Dijkstra** : Recherche du chemin le plus court en prenant en compte les coûts.

 **Calcul du score**

* Un système de score est mis en place, basé sur la formule de l’equation 1:

Équation 1 : Formule de Calcule de Score

Cela permet d’évaluer l’efficacité du chemin naviguée par le joueur pour atteindre un mot cible.

 **Interface graphique intuitive**

* Utilisation de **JFrame** et **JPanel** pour afficher le labyrinthe et permettre une interaction fluide avec le joueur.

 **Affichage des résultats**

Après l'exécution d'un algorithme, le chemin trouvé est visualisé, et des informations sur ses performances sont affichées, notamment :

* Le nombre de nœuds explorés
* Le temps d'exécution
* L'efficacité du parcours

Si le joueur atteint la case rouge (la sortie), la partie se termine. L'annexe affiche alors :

* Le score du joueur
* Le temps total écoulé
* Une information indiquant si un nouveau record a été atteint pour ce niveau

Cette présentation permet une meilleure compréhension des performances et des progrès réalisés par le joueur.

# Modélisation du labyrinthe de mots

Le **Word Maze Game** est construit à partir d’un ensemble de mots sélectionnés aléatoirement et positionnés stratégiquement dans une grille.

### Sélection et placement des mots

 Le jeu dispose d’un **fichier texte contenant 100 mots**.

 **x mots sont tirés aléatoirement** à chaque partie pour constituer le labyrinthe.La valeur x depend de la niveau de jeu. En effet, 3 pour le niveau facile, 8 pour celui intermediaire et 12 pour le dernier niveau.

 Le labyrinthe est **rempli de manière stratégique** en suivant ces règles, comme illustrer dans la figure 1 :

* Un **point de départ** est choisi au hasard dans la grille.
* Chaque mot est placé dans une direction déterminée selon **les directions de la boussole** (Nord, Sud, Est, Ouest, Nord-Est, Nord-Ouest, Sud-Est, Sud-Ouest).

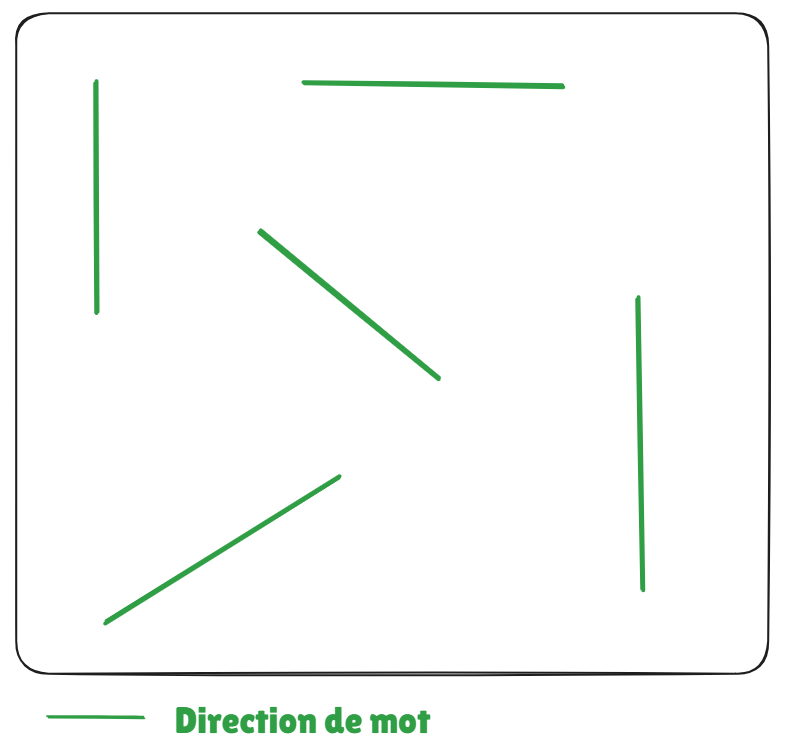


Figure 1: Direction de mot dans la labyrinth de mot

### Remplissage des cases vides

Une fois tous les mots placés, les cases vides de la grille sont **remplies avec des lettres aléatoires**, rendant le labyrinthe plus complexe et rendant difficile la reconnaissance immédiate des mots cachés.

### Gestion des obstacles

Pour finaliser la génération du labyrinthe et renforcer le défi du jeu, des **obstacles** (cases bloquées) sont ajoutés comme illustré dans la **Figure 2**.

Chaque case a 20 % de probabilité d’être bloquée, sauf celles contenant un mot à trouver.



Figure 2 : Exemple de Labyrinth de Niveau Difficile

# Implémentation

## Structure des données utilisée

L’implémentation du **Word Maze Game** repose sur une conception orientée objet et l’utilisation de structures de données adaptées pour gérer efficacement le labyrinthe et les chemins possibles.

#### **Utilisation des classes et objets**

Le jeu est structuré en plusieurs classes principales :

* **Labyrinth**: Représente le labyrinthe et gère la génération des mots, des obstacles et des chemins possibles.
* **LabyrinthNode**: Modélise une case du labyrinthe, qui peut contenir un alphabet ou être bloquée.
* **WordTracker**: Implémente les algorithmes de recherche de chemin (**DFS, BFS, Dijkstra**).
* **GamePlay**: Gère l’interaction entre l’interface graphique et la logique du jeu.

### Structure de stockage du labyrinth

 Le **labyrinthe est représenté sous forme d’une liste** de listes (**List<List<LabyrinthNode>>**), ce qui permet une organisation efficace de la grille en 2D.

 Chaque **LabyrinthNode** contient des informations sur son état :

* Si elle est la **case de debut**
* Si elle est la **case de fin**
* Si elle est une **case colorée**
* Si elle est une **case bloquée**
* Si elle est une **case indispensable**

### **Utilisation des Maps**

### **Dans la** vérification des chemins

Dans le **Word Maze Game**, une structure de **Map** est utilisée pour **vérifier les chemins trouvés par le joueur** et valider si le mot sélectionné suit un chemin valide dans le labyrinthe. En effet, Une **Map<String, String>** est mise en place pour stocker la signature des mots sous forme de coordonnées.

* Chaque **clé** représente un mot (une Liste de LabyrinthNode dans le graphe).
* Chaque **valeur** contient la signature de la mot sous la forme (x1,y1;x2,y2…xn,yn).

### Dans la gestion des chemins

Pour mémoriser les chemins identifiés par les algorithmes de recherche, une **Map<String, String>** est employée :

* Chaque **clé** représente un mot, correspondant à une Liste de LabyrinthNode du graphe.
* Chaque **valeur** stocke le mot précédent dans le chemin optimal, permettant ainsi de reconstruire le trajet suivi jusqu’à ce mot.

# Conclusion et Perspectives

## **Conclusion**

Ce projet de **Word Maze Game** a permis d’explorer et d’implémenter différentes stratégies de recherche de chemin dans un labyrinthe généré aléatoirement. En utilisant les algorithmes **DFS, BFS et Dijkstra**, nous avons pu comparer leur efficacité et observer leurs comportements dans un environnement dynamique. L’intégration d’une interface graphique avec **JFrame et JPanel** a offert une expérience utilisateur interactive, rendant le jeu accessible et intuitif.

Les résultats obtenus montrent que **chaque algorithme a ses propres avantages et inconvénients** :

* **DFS** est rapide mais ne garantit pas toujours le chemin optimal.
* **BFS** trouve le chemin le plus court en termes de nombre d’étapes, mais peut être plus lent dans de grandes grilles.
* **Dijkstra** offre une solution optimisée en prenant en compte les obstacles, mais avec un coût computationnel plus élevé.

Ce projet a donc permis d’allier **modélisation algorithmique, conception d’interfaces graphiques et gestion des structures de données**, illustrant ainsi l’application concrète des concepts appris en informatique.

## **Perspectives**

Plusieurs améliorations et extensions peuvent être envisagées pour enrichir le projet :

1. **Ajout d’un mode multijoueur** : Permettre à plusieurs joueurs de s’affronter en trouvant les mots le plus rapidement possible.
2. **Intégration d’un algorithme A**\* : Optimiser davantage la recherche du chemin optimal en combinant heuristique et coût réel.
3. **Génération de labyrinthes plus complexes** : Introduire des **niveaux dynamiques** avec des obstacles évolutifs.
4. **Amélioration de l’interface graphique** : Ajouter des animations et un meilleur affichage des parcours empruntés.
5. **Optimisation des performances** : Réduire le temps de calcul des algorithmes en optimisant la gestion de la mémoire et des structures de données.