Année 2018-2019

RAPPORT DE PROJET – FI1

PROJET ALGORITHME D'ARIANE

projet réalisé par :

Fabien Le Bec gr.5
Gaston Lions gr.5

Sommaire :

Présentation du projet :

Le but de ce projet était de réaliser un programme comprenant un algorithme déterministe pour parcourir un labyrinthe en langage Java avec comme aide uniquement la bibliothèque officielle.

Objectif du projet : Ce projet a pour but d'étudier un algorithme de guidage, visant à conduire un objet mobile jusqu'à son but à travers un parcours d'obstacles.

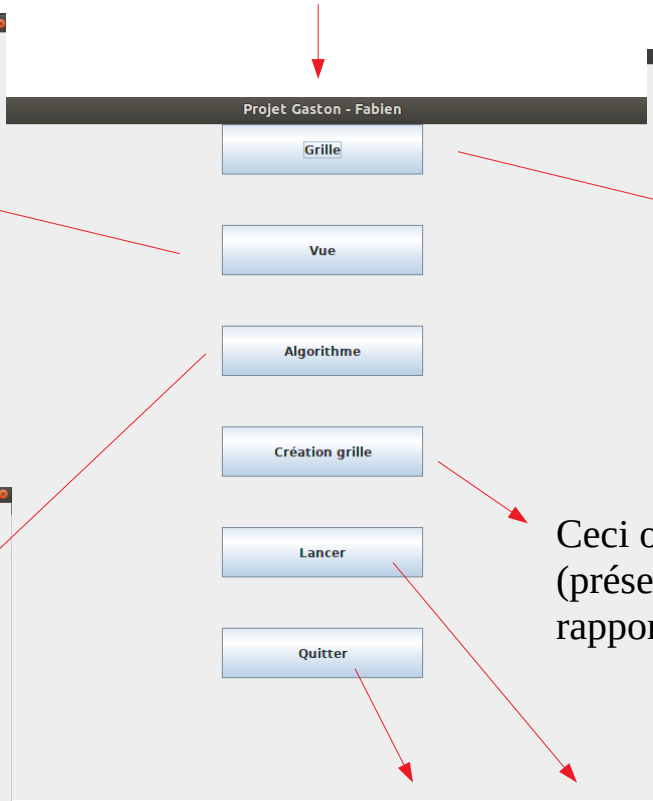
Lors de la conception de celui-ci nous avons du y introduire quelques **fonctionnalités** :

- L'utilisateur peut choisir une grille (avec JFileChooser) ou alors en créer une de façon aléatoire ou manuellement.
- L'utilisateur doit choisir entre deux algorithme, déterministe ou aléatoire. Lors du déterministe Thésée connaît uniquement ses actions précédentes. Il doit bien évidement être plus performant que l'algorithme aléatoire.
- L'utilisateur choisit une vue :
  - Manuel, l'avancement est représenté sur la grille, les étapes sont défiler en appuyant sur une touche.
  - Automatique, la grille n'est pas affichée et il n'y a aucune intervention de l'utilisateur. L'algorithme est lancé 100 fois et on récupère le nombre d'étapes moyen.

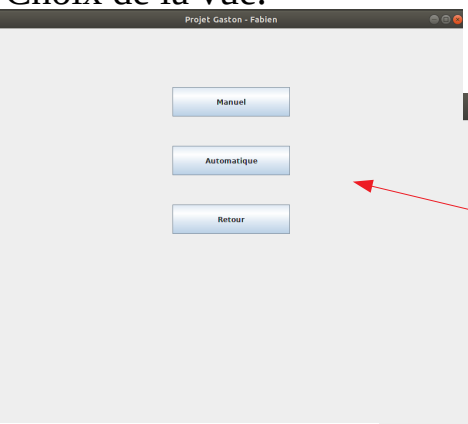
Année 2018-2019

Fonctionnalités du programme :

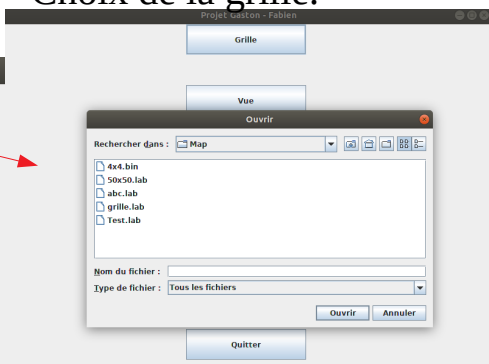
Le programme se lance en affichant ce menu.
C'est à partir de celui ci que tous les paramètres nécessaires pour le lancement vont se faire.



Choix de la vue.



Choix de la grille.

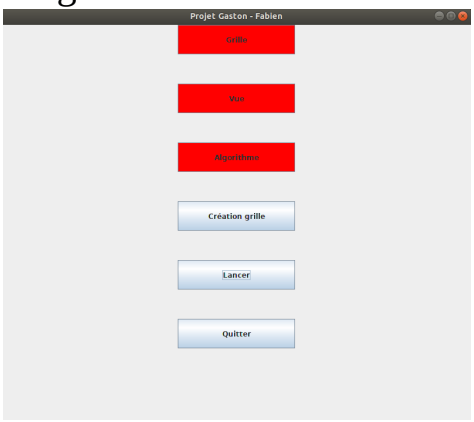


Choix de l'algorithme.

Ceci ouvre une fenêtre d'édition (présentée un peu plus tard dans le rapport)

Permettent bien évidemment de lancer et de quitter le programme

Si vous appuyer sur « Lancer », une méthode va vérifier si une grille, une vue et un algorithme ont bien été choisi. Si ce n'est pas le cas, l'élément manquant s'affichera en rouge.



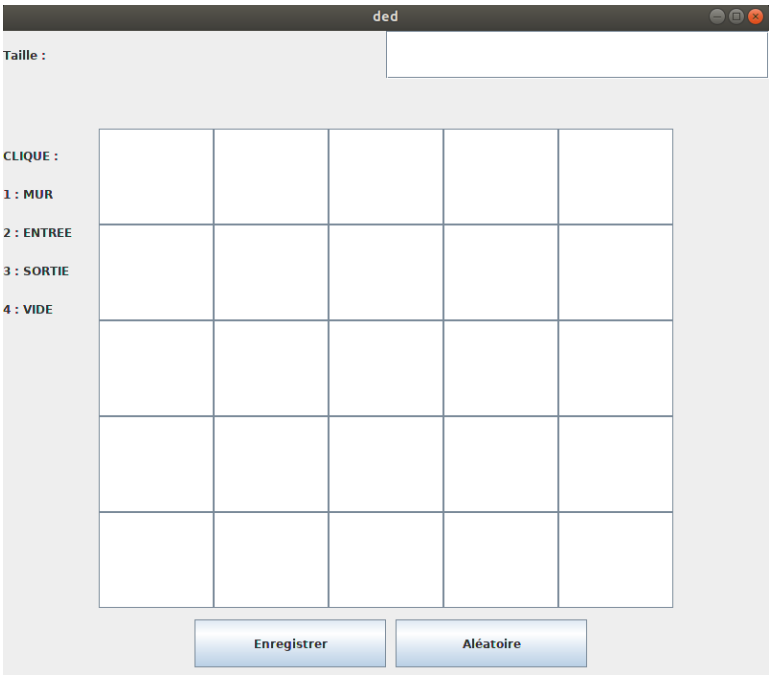
Édition de grille :

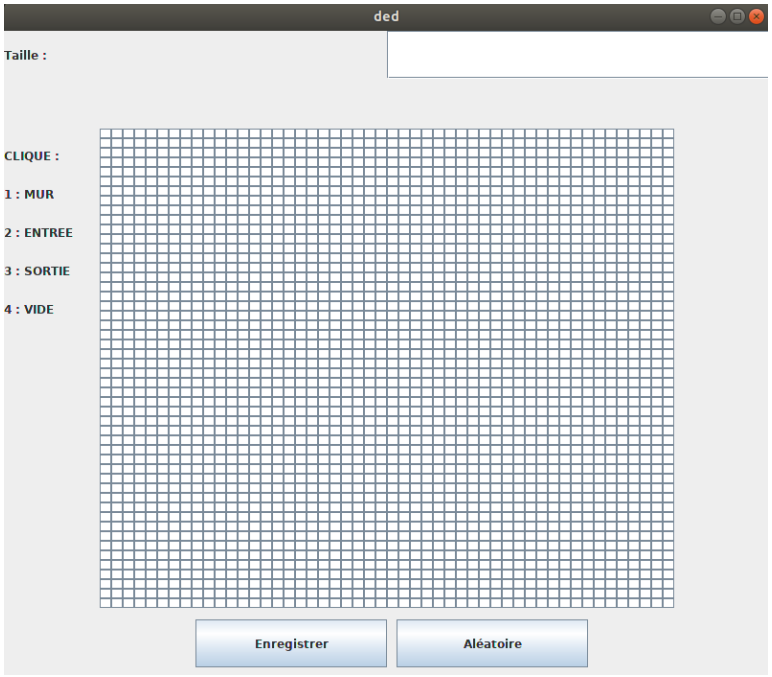
Taille 5

Taille 50

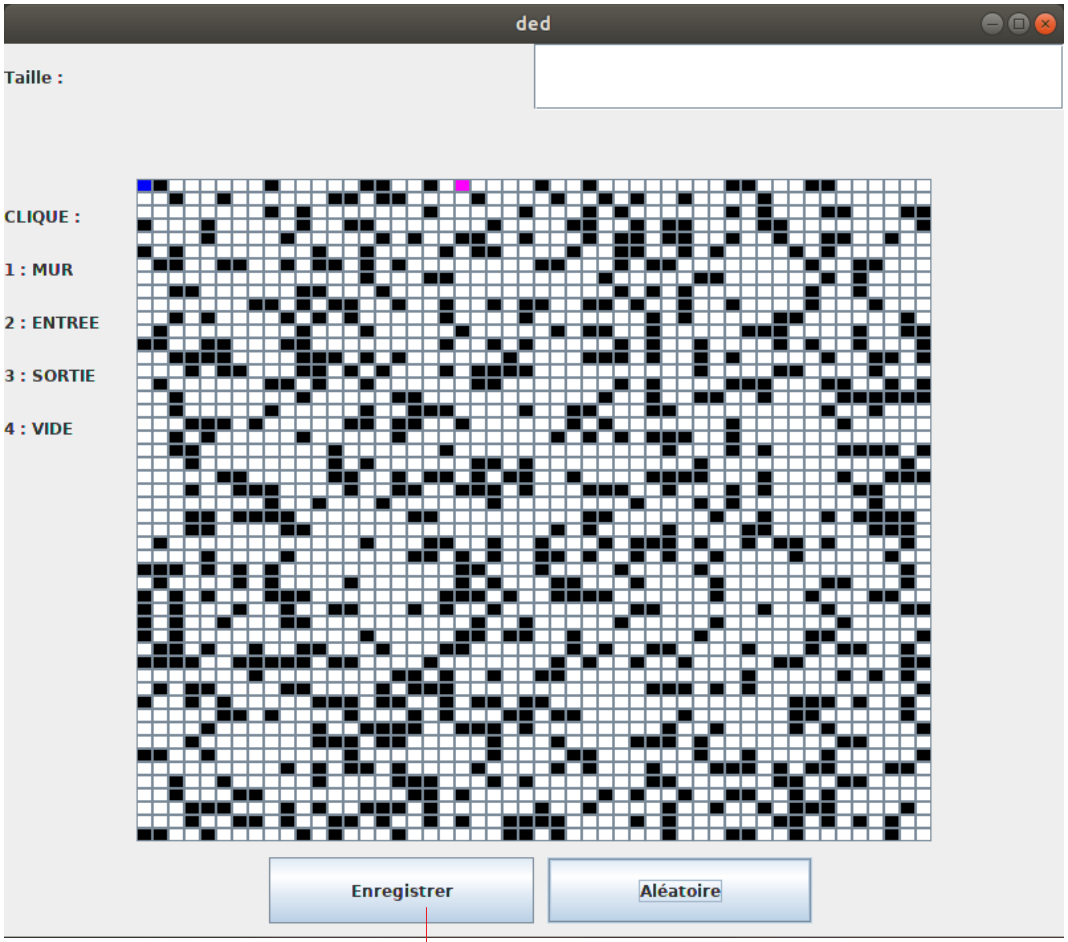
Lorsqu'on lance le mode édition on se trouve sur une fenêtre (présentée à gauche). L'utilisateur devra donc commencer par saisir une taille. Une grille s'affichera par la suite selon la saisit et l'utilisateur pourra changer l'état d'une case en cliquant dessus, comme indiqué sur la légende. Il a aussi la possibilité de remplir aléatoirement la grille.

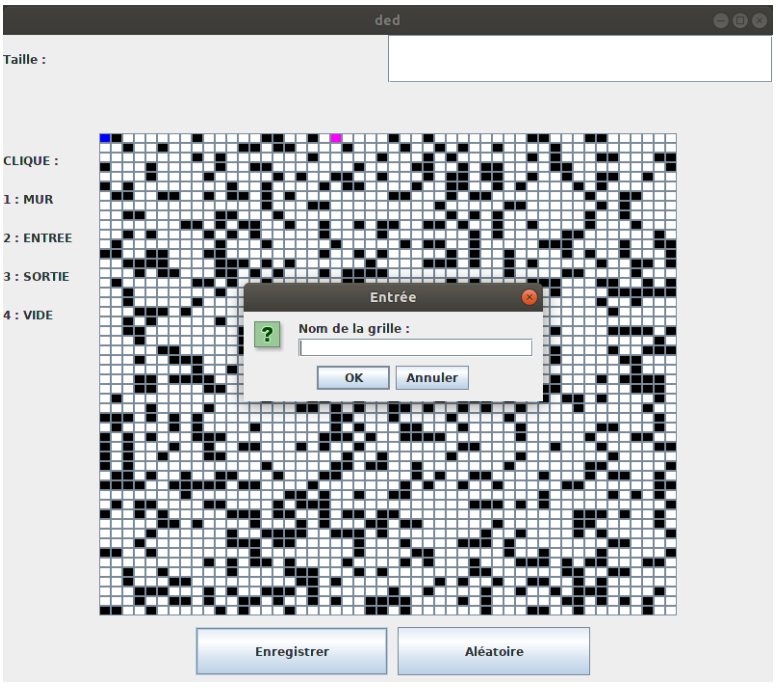
Lorsque l'utilisateur appuie sur le bouton « enregistrer », une méthode vérifie que la grille possède une seule entrée et une seule sortie. Si c'est le cas une petite fenêtre s'ouvre où l'utilisateur saisira le nom qu'il souhaite donner à sa grille, qui va ensuite s'enregistrer dans un fichier portant son nom dans le dossier « map ».





Voici à quoi ressemble une grille remplit (ici aléatoirement).





Les erreurs pouvant être rencontrées pendant l'édition :

L'utilisateur n'a pas entrer un entier pour la taille.

La taille doit être comprise entre 2 et 100.

L'utilisateur n'a pas entrer de nom pour la grille.

La grille ne possède pas une seule entrée et/ou une seule sortie.

Vous trouvez ci-dessous la liste des classes utilisées dans le programme ainsi que leurs attributs et méthodes. Le diagramme de classes se trouve à la page 9 (réalisé en 2 étapes pour plus de lisibilité).

DessinGrille
Taille : int NbrCases : int Algo : int Fenetre : Jframe Fichier : Fichier Thesee : Player Sortie : Point Finish : boolean PosMur : boolean[][] Visited : boolean[][] Solution : boolean[][]
SetCases() : void SetVisited() : void IsVisited() : boolean IsFinish() : void SetSolution() : void PaintComponent() : void

Algorithme
Taille : int Movcount : int Manuel : boolean Thesee : Player Sortie : Point Infogrille : boolean[][] Visited : boolean[][] Fichier : fichier Xpoint : int Ypoint : int Backtrack : Point Fenetre : Jframe Dessin : DessinGrille Rand : Random Chemin : Stack<Point> Monitor : Object
Equals() : boolean TryMove() : void Sleep() : void Aleatoire() : int Deterministe() : boolean ResetMovCount() : void ResoudreAleatoire() : int ResoudreDeterministe() : int

ActionManuelle
fenetre : JFrame Monitor : Object
KeyPressed() : void KeyReleased() : void Keytyped() : void

ActionMenu
Fenetre : JFrame Element : JPanel Bouton : JButton GestionMenu : Menu
ActionPerformed() : void

ActionMapEditor
ZoneText : JTextField Taille : int Fenetre : JFrame Editor : MapEditor NbrCases : int ZoneGrille : JPanel Bouton : Bouton Mode : String NbrClick : int Valider : JButton GroupeB : Bouton[] Indice : int
ActionPerformed() : void

MapEditor
Fenetre : Fenetre Taille : int TableauB : Bouton[] Rand : Random Mur : boolean[][] PosEntree : Point PosSortie : Point NbrSortie : int NbrEntree : int
CreateGrille() : void RefreshGrille() : void RempRand() : void SetVariable() : void GetMur() : boolean[][] GetEntree() : Point GetSortie() : Point GetTaille() : int SetEntree() : void SetSortie() : void VerifSortie() : int VerifEntree() : int

Fichier
Nom : String TailleLaby : int Entree : Point Sortie : Point Thesee : Player PosMur : boolean[][]
GetFile() : void GetTailleLaby() : int GetThesee() Player GetSortie() : Point GetPosMur() : boolean[][] WriteFile() : void OpenFile() : void

Bouton
NbrClique : int
SetClique() : void GetClique() : int

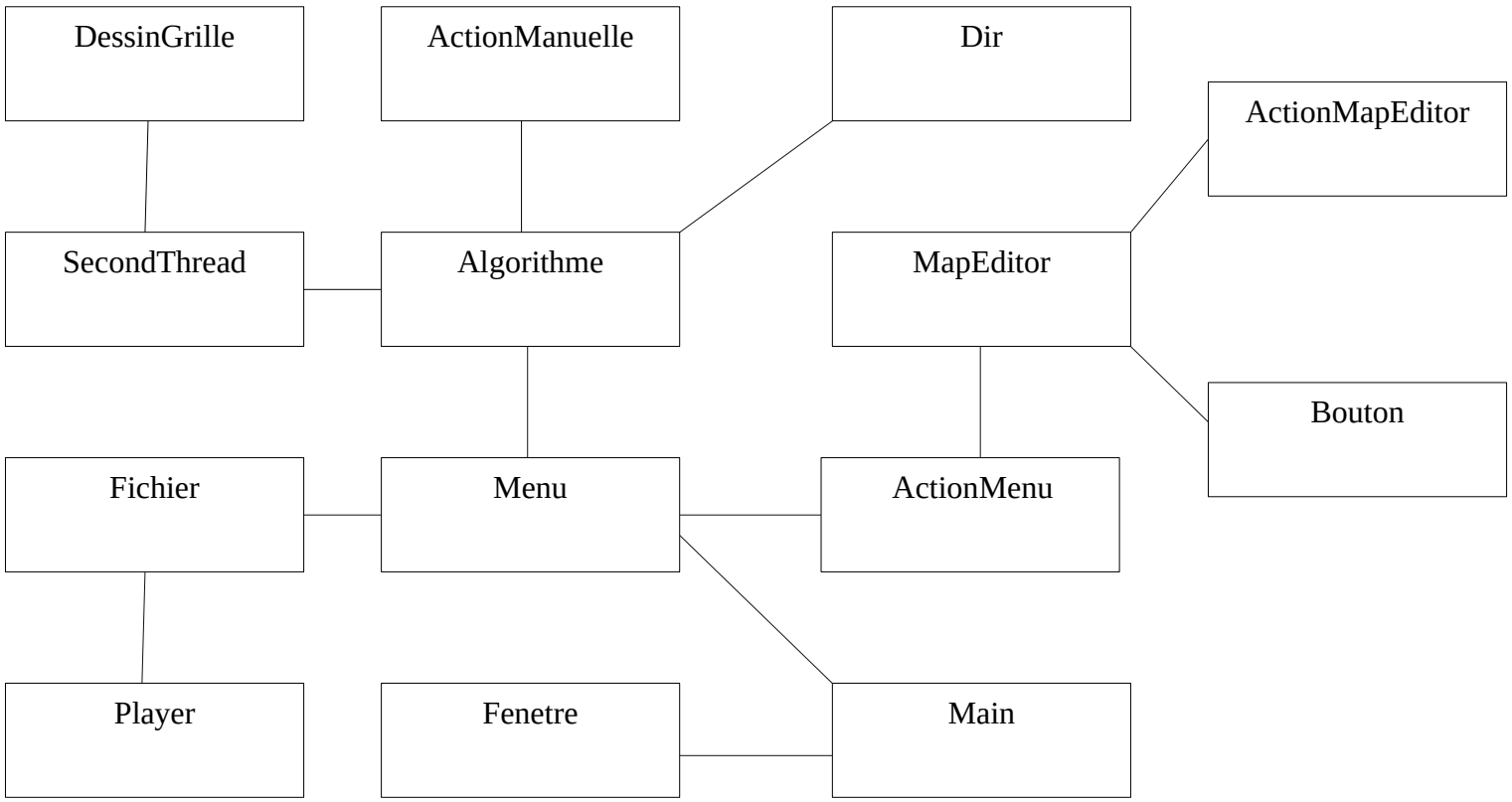
Player
InitialThesee : Point
ResetPosition() : void GetLigne() : int GetColonne() : void GetPosition() : Point GetProchaineLigne() : int GetProchaineColonne() : int SetPosition() : void MovePosition() : void

SecondThread
ChoixGrille : Fichier ChoixAlgo : int
Run() : void

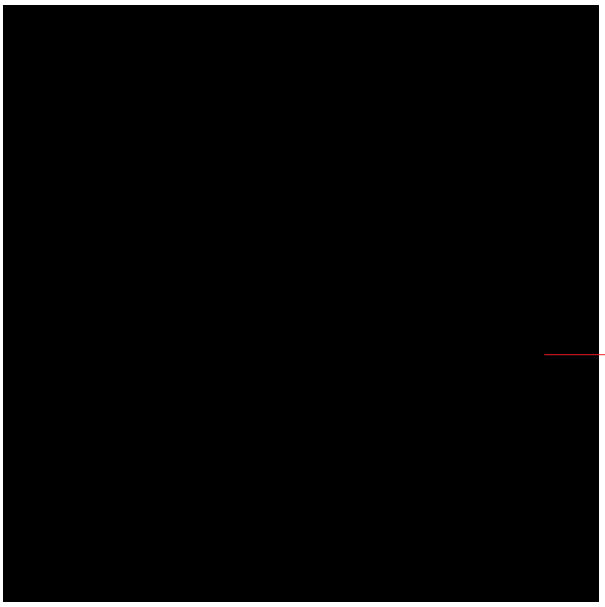
Dir
GetX() : int GetY() : int QuartDeTour() : Dir QuartDeTourAnti() : Dir

Fenetre

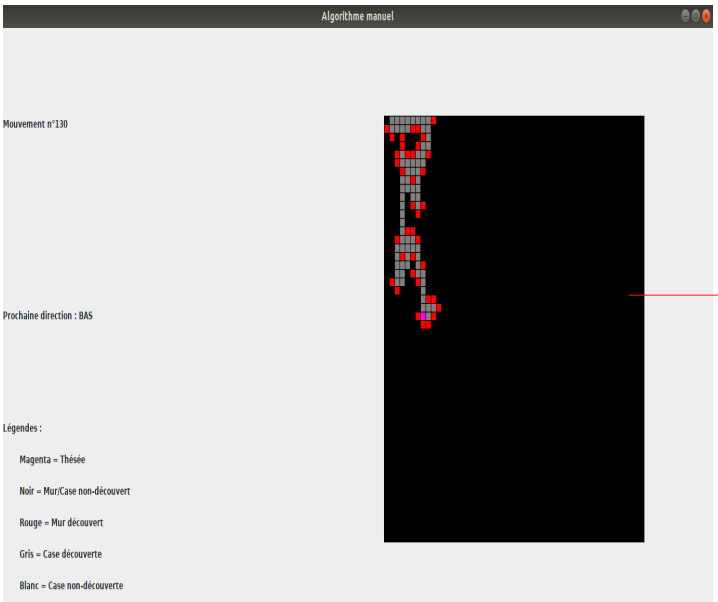
Main
Main() : void


DessinGrille
ActionManuelle
Dir
ActionMapEditor
SecondThread
Algorithme
MapEditor
Bouton
Fichier
Menu
ActionMenu
Player
Fenetre
Main

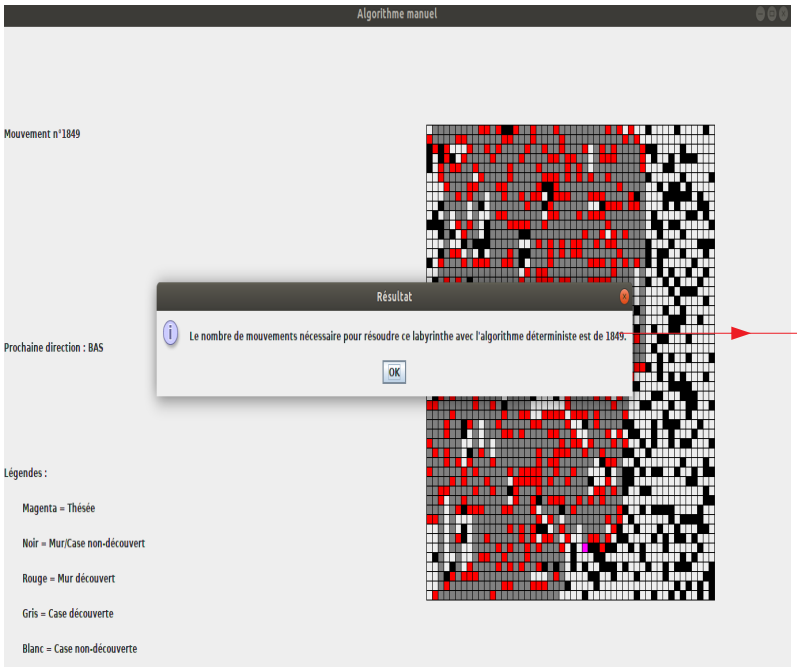
Démonstration en image de l’algorithme :



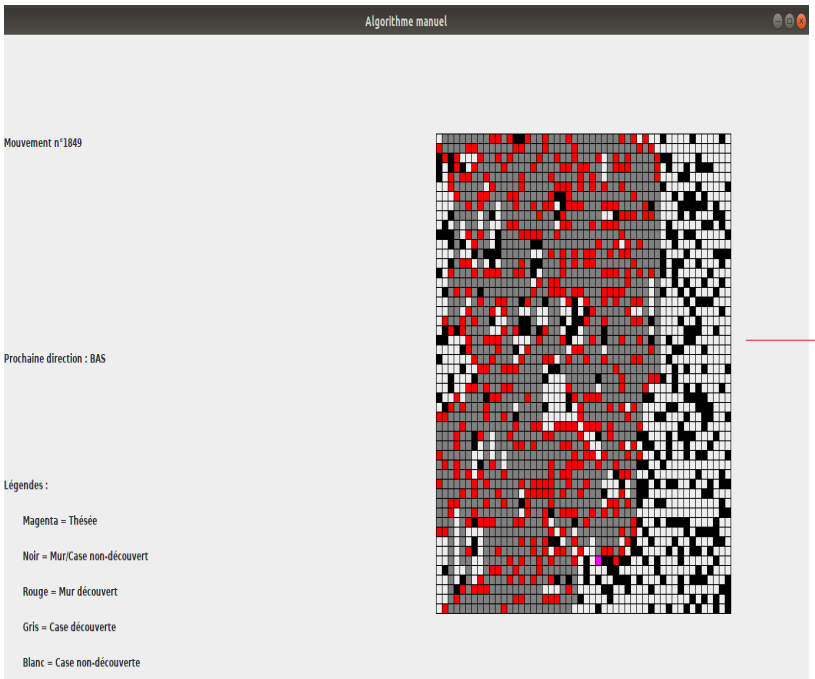
Lorsque l’algorithme se lance, on voit un bloc noir au centre de la page. Celui ci représente le labyrinthe vue par Thésée qui ne connaît uniquement les cases visitées. Donc au départ aucune.



Le labyrinthe s’affiche donc au fur et à mesure de l’avancé de Thésée.
En rouge : les murs rencontrés.
En gris : les cases découvertes.
En rose : la position de Thésée.
En noir : les cases non découvertes.



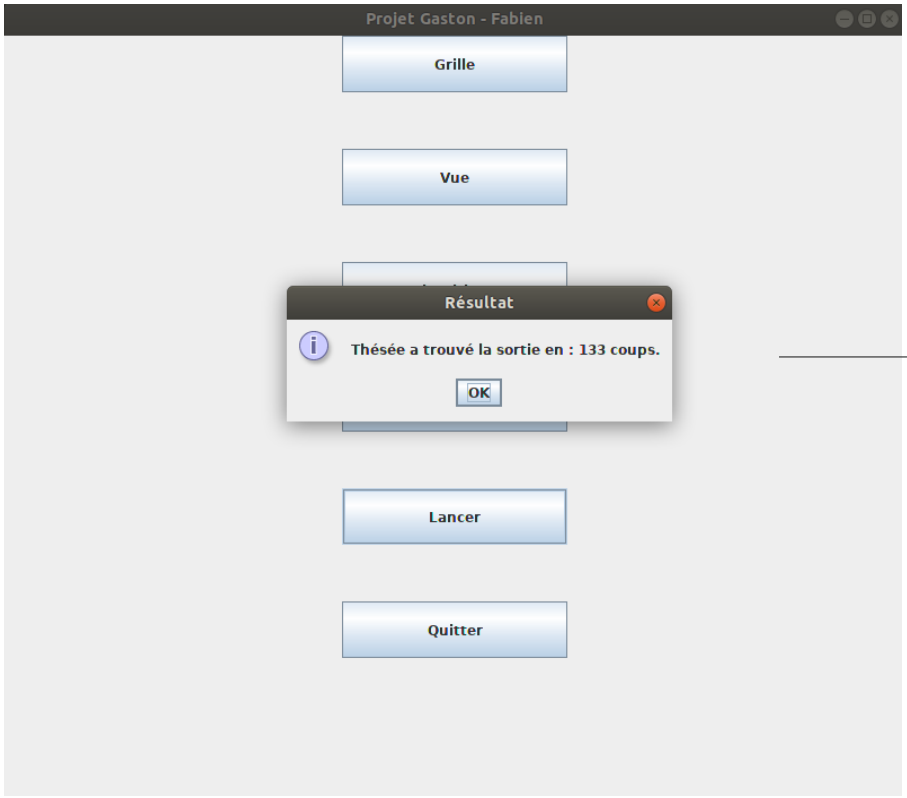
Une fois que l'algorithme a finit son travail il affiche un message affichant le nombre de mouvements qui ont été nécéssaire pour trouver la sortie.
Si aucune sortie n'a été trouvée, le message indique qu'il n'y a aucune solution.



Une fois le message affiché, la grille entière est découverte. Les cases noirs indiquents un mur qui n'a pas été découvert et les cases blanches les cases n'ayant pas été découverte.

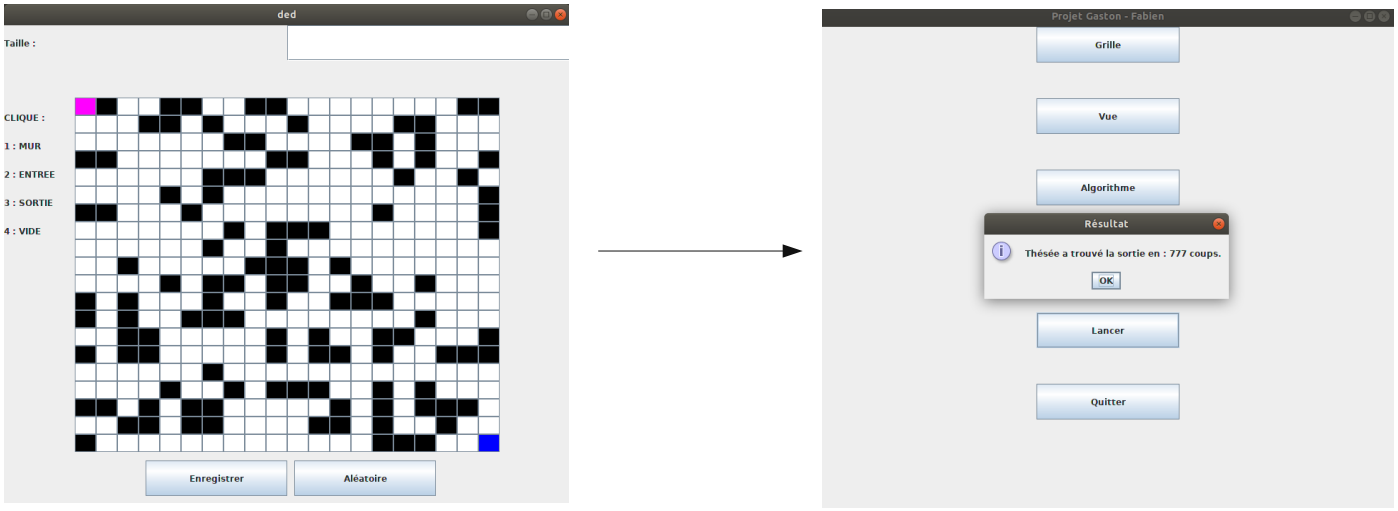
Les images ci-dessous prouvent que l'algorithme déterministe est plus performant que l'aléatoire.

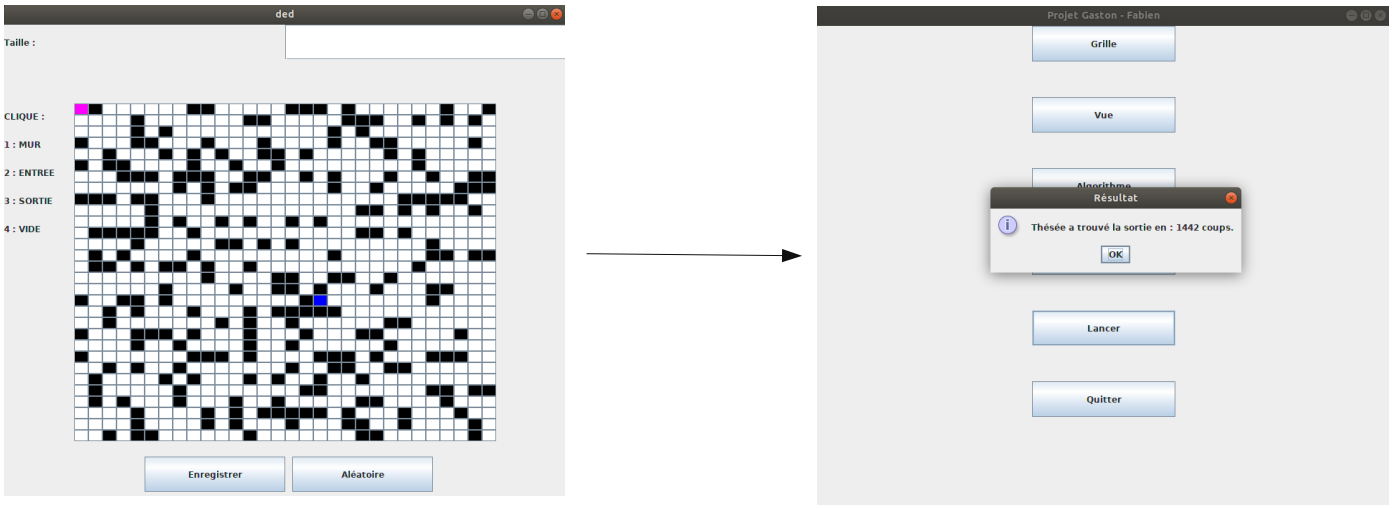
Aléatoire



Déterministe

Prenons maintenant 3 exemples de grilles, deux dans lesquels Thésée peut atteindre la sortie, puis une où ce n’est pas le cas :





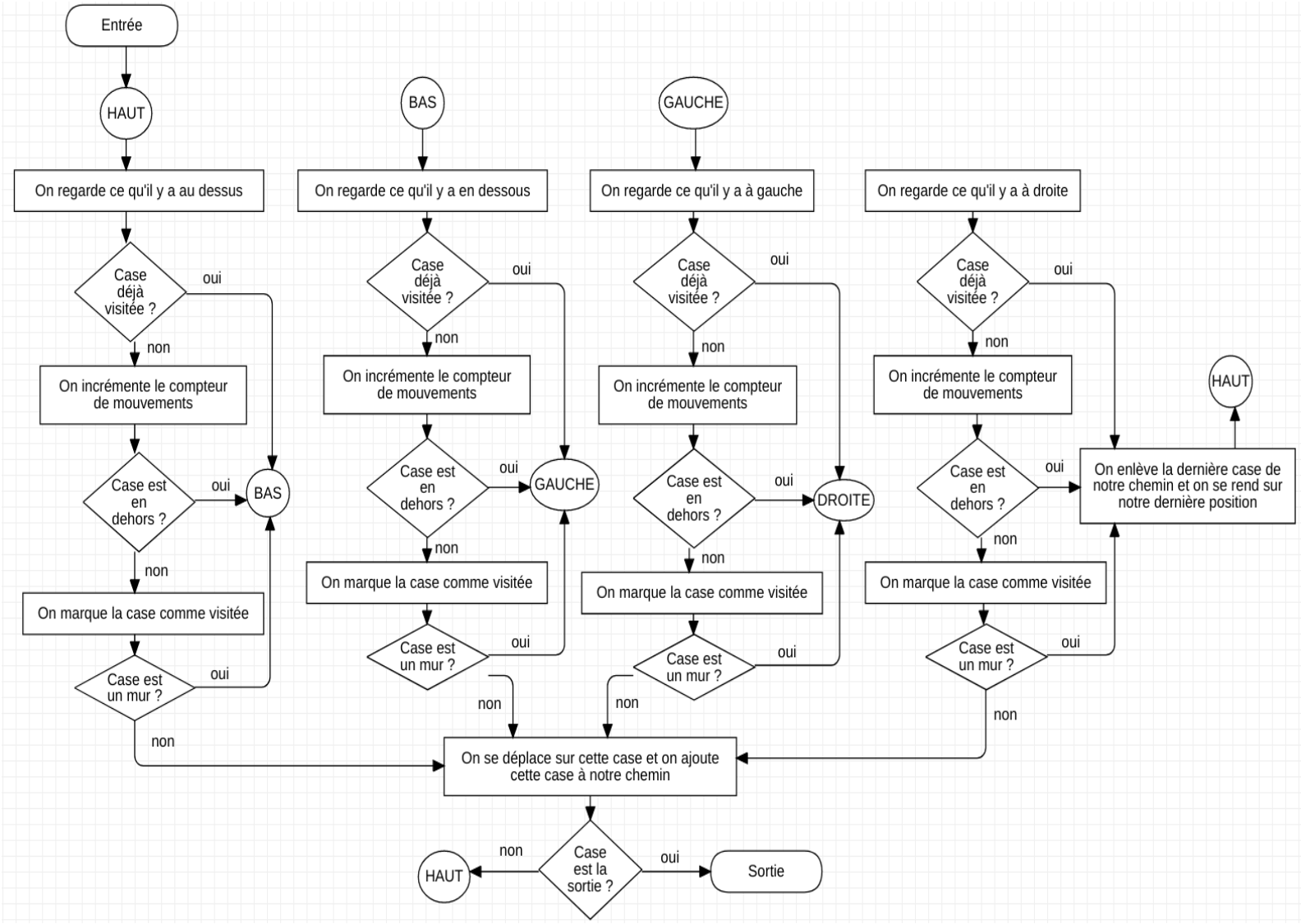
On remarque bien que l’algorithme déterministe trouve toujours si il y a une sortie ou non.
Pour ce qui est de l’algorithme aléatoire si le nombre de mouvements est supérieur à la taille de la grille * 1 000 000 on considère que le labyrinthe n’a pas de solution .

Algorithme déterministe

Pour l'algorithme déterministe nous avons décidé de créer un algorithme récursif disposant d'un système de 'backtrack' c'est-à-dire que l'algorithme va revenir sur ses pas lorsqu'il va rencontrer un chemin sans issue.

Nous avons représenté le fonctionnement de notre algorithme déterministe sous la forme d'un algorigramme (flowchart) réalisé avec la version gratuite de StarUML pour une meilleure compréhension du déroulement de celui-ci.

A noter que c'est notre tout premier algorigramme, il se peut donc qu'il y ait quelques coquilles.



Conclusion personnelle

Gaston : En ce qui me concerne, la réalisation de ce projet m'a plu, puisque c'est la première fois que l'on utilisait un langage orienté objet lors d'un projet. Il a été enrichissant puisque j'ai pu apprendre à travailler en équipe, m'organiser. Ce fut donc bénéfique.

Fabien : Pour ma part ce projet a été mon premier projet en binôme. J'ai compris que le travail en groupe était bien différent du travail seul. Il faut dès le début bien se répartir les tâches pour ne pas que l'un reste sans rien faire, cela demande beaucoup d'organisation notamment en ce qui concerne le git.
A part ça le projet m'a apporté de nombreuses connaissances en programmation orienté objet mais dans la compréhension en détails du fonctionnement d'une fonction récursive (algorithme récursif).