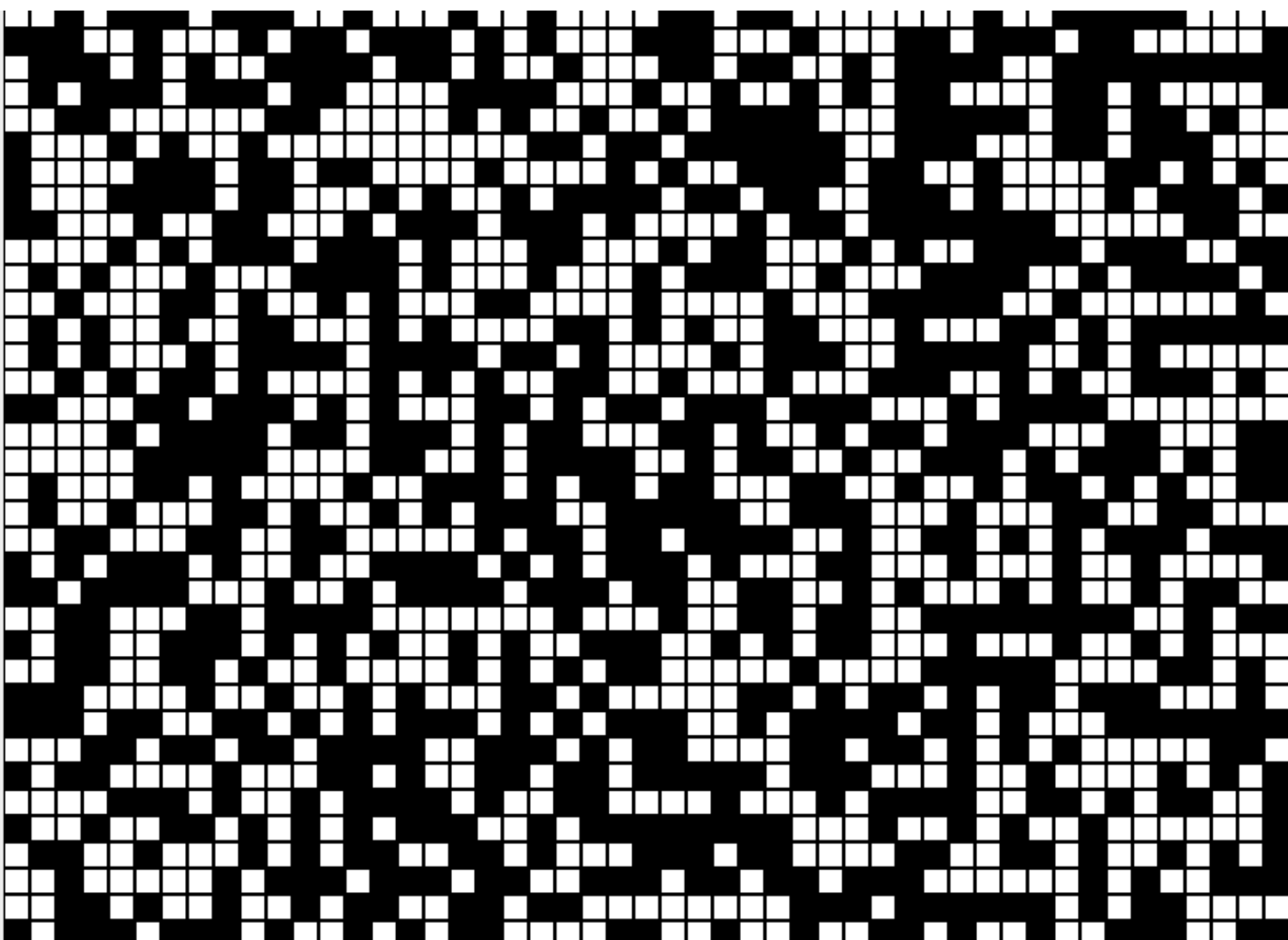


SAE21\_2022

# **ALGORITHME D'ARIANNE EN JAVA**



**BLANCHON Baptiste & DURAN Samet**

# SOMMAIRE

- 01** Introduction
- 02** Fonctionnalités
- 03** Structure du programme
- 04** Exposition de l'algorithme
- 05** Conclusions personnelles

# INTRODUCTION

Le but de l'algorithme est de modéliser un objet mobile du nom de "Thésée" ainsi qu'une sortie et, à travers un labyrinthe permettre à Thésée d'atteindre la sortie de plusieurs façons.

Le programme fonctionne de la manière suivante :

- Au lancement du programme, l'utilisateur peut choisir entre charger une grille ou en construire une nouvelle.
- Lorsqu'il choisit de la construire, il peut décider du remplissage de la grille, de la position de Thésée et de la sortie, mais aussi gérer la position des obstacles (cases noires) et des chemins (cases blanches).
- Si l'utilisateur clique sur "Algorithme aléatoire", Thésée se déplace de manière totalement aléatoire dans les chemins (représentés par des cases blanches).
- En revanche, si l'utilisateur clique sur "Algorithme déterministe", Thésée parcourt le chemin le plus court menant à la sortie.
- Le programme se termine lorsque la sortie a été trouvée par Thésée.

Le but de cette SAé était de coder cet algorithme avec le langage Java sans bibliothèque graphique particulière.

# FONCTIONNALITÉS

## CHOIX DE LA GRILLE :

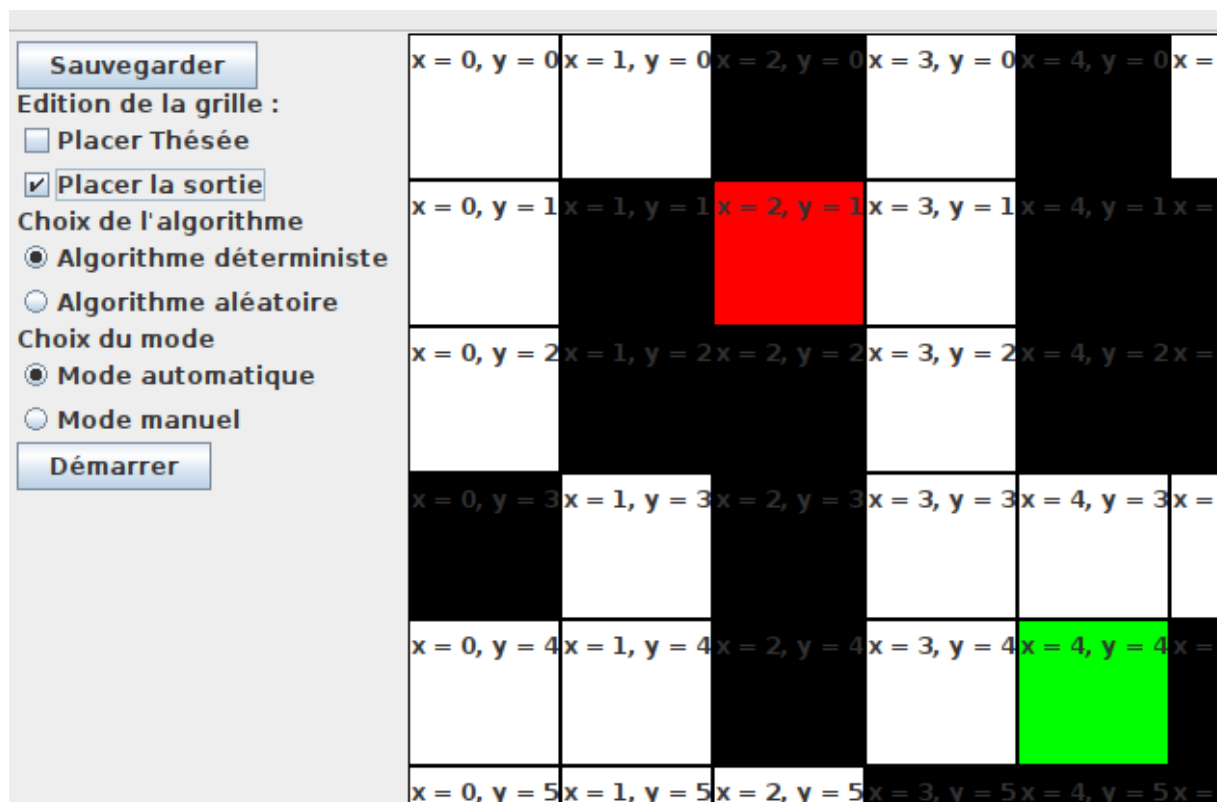
L'utilisateur a la possibilité de choisir la grille (en construire une ou en charger une) :



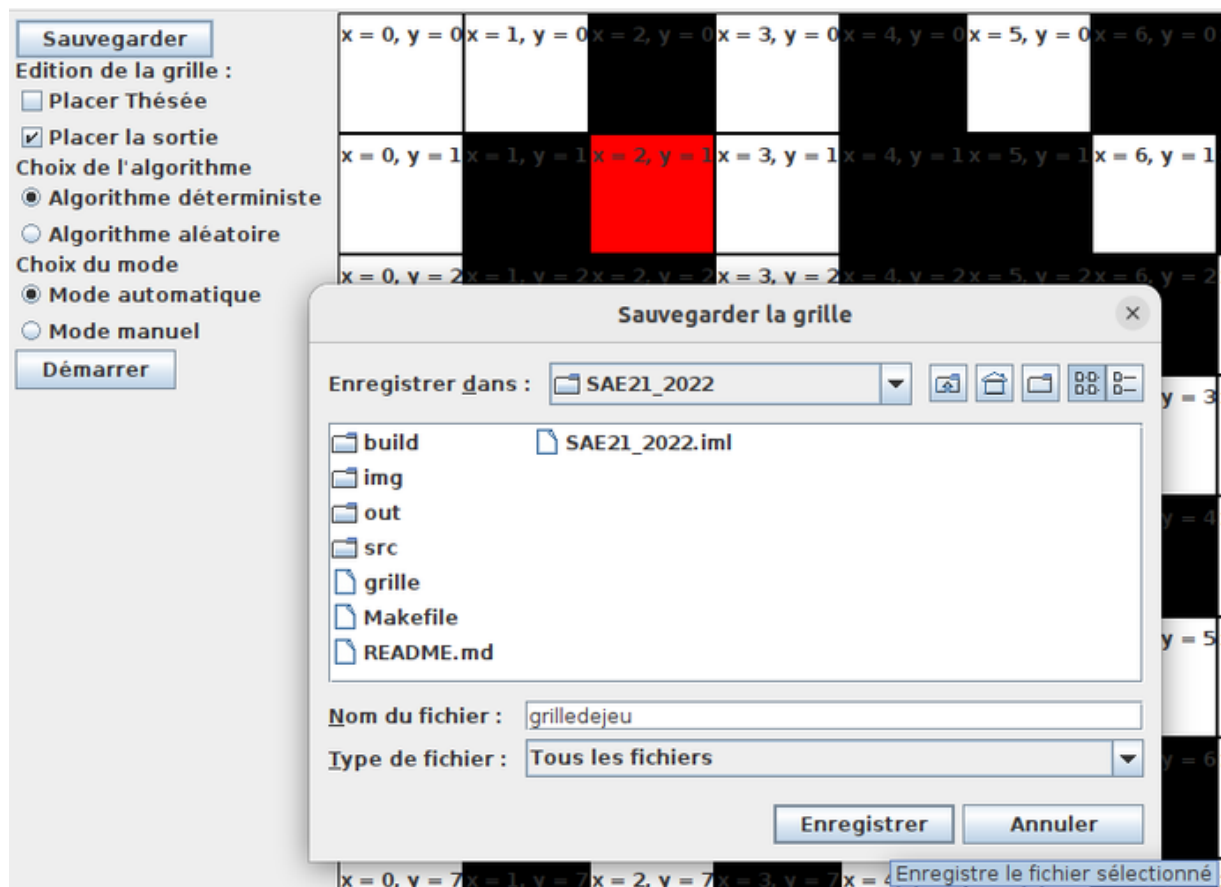
S'il choisit de construire une nouvelle grille, il décide de sa taille et s'il souhaite qu'elle soit initialement vide :



Quand la grille est générée, l'utilisateur a plusieurs choix, comme placer Thésée ainsi que la sortie où il veut, placer des obstacles (cases noires) en cliquant sur les cases blanches ou bien les enlever. Par la suite, il a la possibilité de sauvegarder cette grille et de l'enregistrer où il le souhaite. Quand cette grille est sauvegardée, l'utilisateur relance le programme, clique sur le bouton "Charger une grille" puis sélectionne le fichier qu'il a sauvegardé auparavant et la grille réapparaît. Il a l'obligation de cocher un algorithme ainsi qu'un mode de réalisation, sinon une erreur apparaîtra lors d'un clic sur le bouton "Démarrer".



- Illustration des différents boutons présent à côté de la grille de l'algorithme



- Démonstration de l'enregistrement d'un fichier qui permet de sauvegarder l'état de la grille, pour éventuellement rejouer sur la même.

# CHOIX DES ALGORITHMES

L'utilisateur choisit entre l'algorithme déterministe (Thésée va rejoindre la sortie en empruntant le plus court chemin possible) ou l'algorithme aléatoire (c'est un algorithme qui permet de se déplacer aléatoirement dans les chemins, représentés par des cases blanches) :

## ALGORITHME ALÉATOIRE :

**Le mode automatique** : ce mode permet à Thésée de se déplacer aléatoirement entre les cases disponibles (blanches) de la grille. Le nombre d'étapes pour atteindre la sortie change à chaque fois que l'on appuie sur le bouton "Algorithme aléatoire". Exemple illustré ci-dessous :

```
Coordonnées Thésée : x = 2, y = 2
Coordonnées sortie : x = 1, y = 2
GAUCHE (VALIDE)
sortie trouvée
Nombre d'essais = 7
```

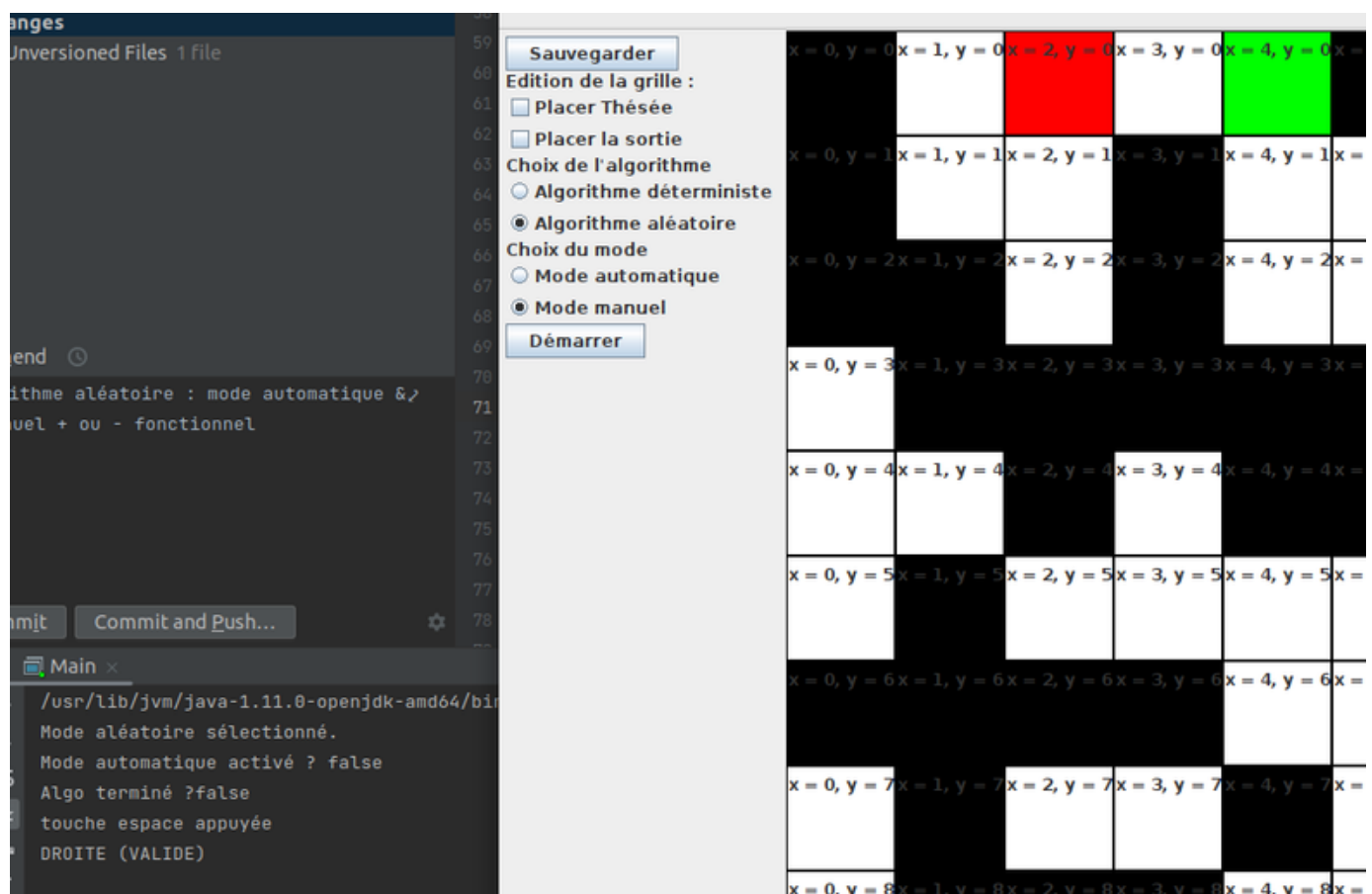
```
HAUT (REFUSE)
GAUCHE (VALIDE)
sortie trouvée
Nombre d'essais = 2
```

```
HAUT (REFUSE)
GAUCHE (VALIDE)
sortie trouvée
Nombre d'essais = 4
```

Le nombre d'essais passe de 7 à 2 puis 4 (exemple avec 3 tentatives). Cependant, durant la programmation de l'algorithme, nous avons remarqué que parfois Thésée restait bloquée et allait par exemple se bloquer dans une case de la grille, puis n'effectuait plus que des changements de direction à gauche ou à droite par exemple. L'affichage dans le terminal était donc infini et Thésée ne trouvait pas de solution (du moins nous n'avions pas l'espoir qu'elle en trouve une). **Notre mode automatique ne permet pas de faire tourner la simulation 100 fois (elle est également affichée).**

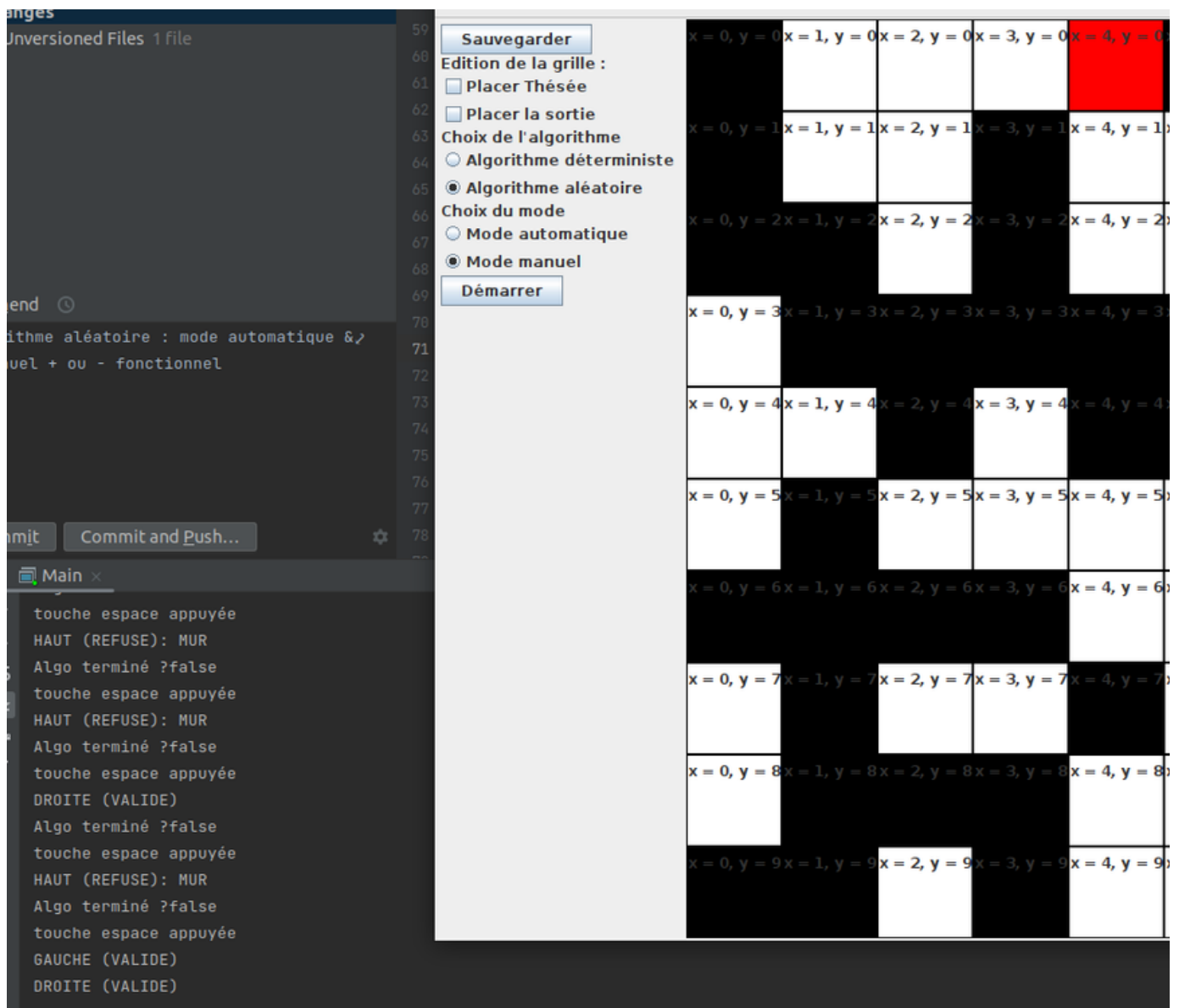
**Le mode manuel :** l'algorithme fonctionne de la même façon que le mode automatique, cependant l'avancement de Thésée est représenté sur la fenêtre : on voit les cases blanches devenir rouge (quand Thésée passe sur la case) puis redevenir blanche (Thésée est passée sur la case, puis continue son chemin). Le tout est réalisé grâce à une classe Java qui prend en compte les actions de l'utilisateur sur le clavier (KeyListener). Dans notre cas, la touche espace sera utilisée pour déplacer Thésée d'une case à chaque fois (ou aucun déplacement si elle rencontre une case noire ou un mur).

Exemple illustré ci-dessous :



- On lance ici l'algorithme aléatoire en mode manuel, puis observons la position de Thésée qui évolue (représentée en rouge) :

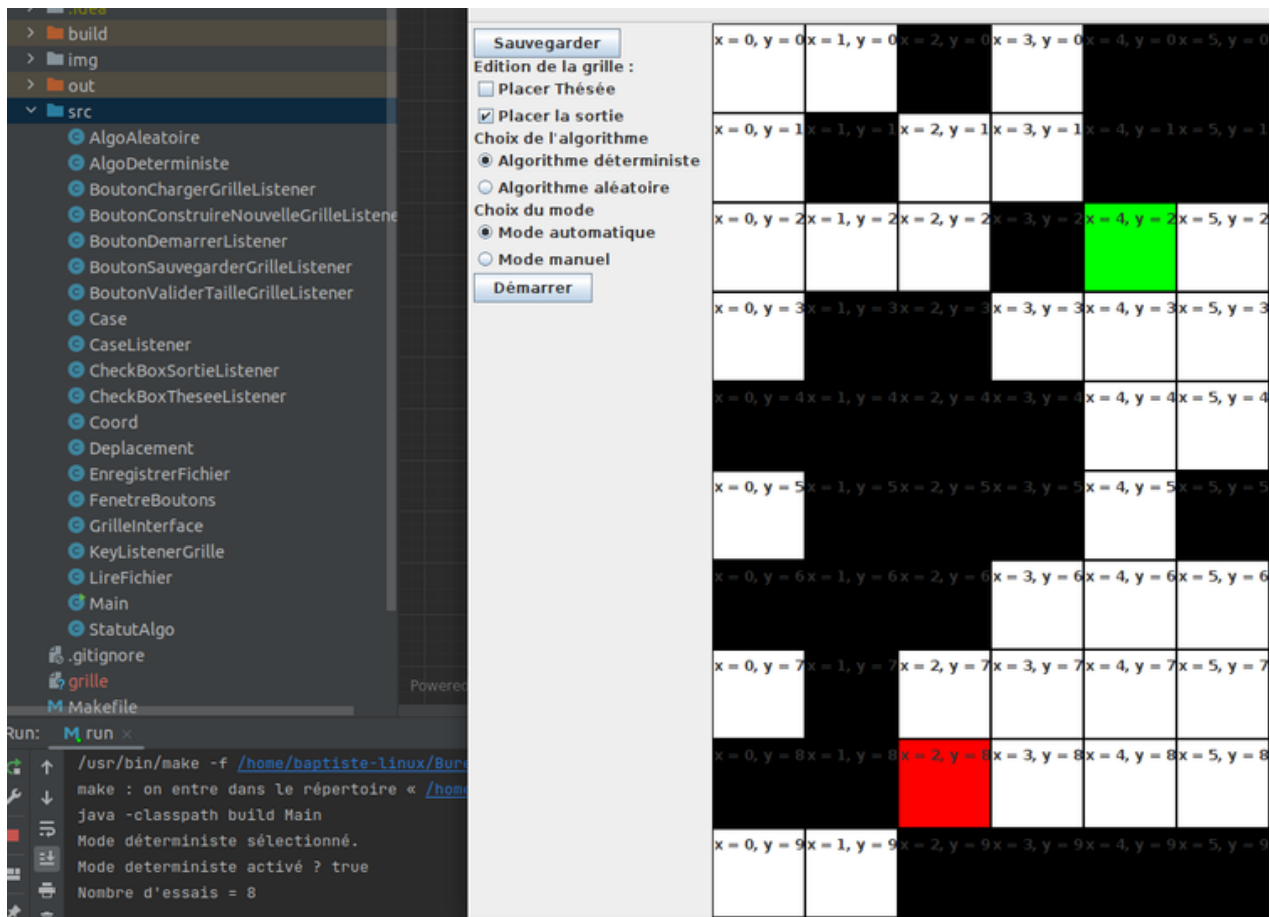




- Thésée s'est déplacée, on le remarque grâce à l'historique des positions présent en bas à gauche. Le programme s'arrête car Thésée a trouvé la sortie. Cependant, il y a une erreur dans notre code au niveau de l'affichage (avec les `System.out.println`) car Thésée trouve la sortie et ce n'est pas affiché. Thésée se situe bien sur l'arrivée si on compare avec la capture d'écran précédente mais elle sera obligée de se re-déplacer encore une fois dans les cases blanches (imaginons à gauche), avant que le programme s'arrête. Le message apparaîtra enfin dans le terminal, pour indiquer que la sortie a été trouvée.

# ALGORITHME DÉTERMINISTE :

**Le mode automatique :** Thésée va se déplacer automatiquement de la position initiale à la sortie, avec le plus court chemin possible. Exemple illustré ci-dessous :



- On voit dans le terminal que le chemin le plus court à été trouvé : Thésée a parcouru 8 cases grâce à l'algorithme déterministe avant de trouver la sortie.

**Le mode manuel :** par manque de temps et de compétences, le mode manuel n'est pas disponible pour l'algorithme déterministe. Cependant il aurait fonctionné de la même façon que le mode aléatoire, c'est à dire avec un KeyListener.

# STRUCTURE DU PROGRAMME

Le projet est séparé en plusieurs fichiers source, qui représentent différentes classes. On peut y trouver des classes qui permettent de gérer toute la partie graphique de l'algorithme, la génération de la grille, la gestion des événements ou encore bien évidemment le Main.

Avant de commencer le projet, il a été important de réfléchir à la structure du code. Nous avons choisi de découper le programme en plusieurs classes, toutes contenant du code qui effectue des actions différentes chacune des autres. Nous allons voir brièvement ce qu'elles contiennent, ce qu'elles permettent de faire et donc nous allons y apporter des explications :

- Le fichier source **Main.java** permet l'affichage de la fenêtre (JFrame) en appelant la classe FenetreBoutons. Ce fichier contient tous les éléments graphiques permettant l'affichage du côté utilisateur.
- La classe **FenetreBoutons** s'occupe d'ajouter tous les boutons nécessaires pour que les interactions entre l'utilisateur et le programme soient optimales. Les lignes de codes sont normalement déjà assez explicites, on ajoute juste à ce panneau (JPanel) les boutons ainsi que des écouteurs d'événements, dont on verra le contenu détaillé dans la description des classes correspondantes (les classes "Listener").

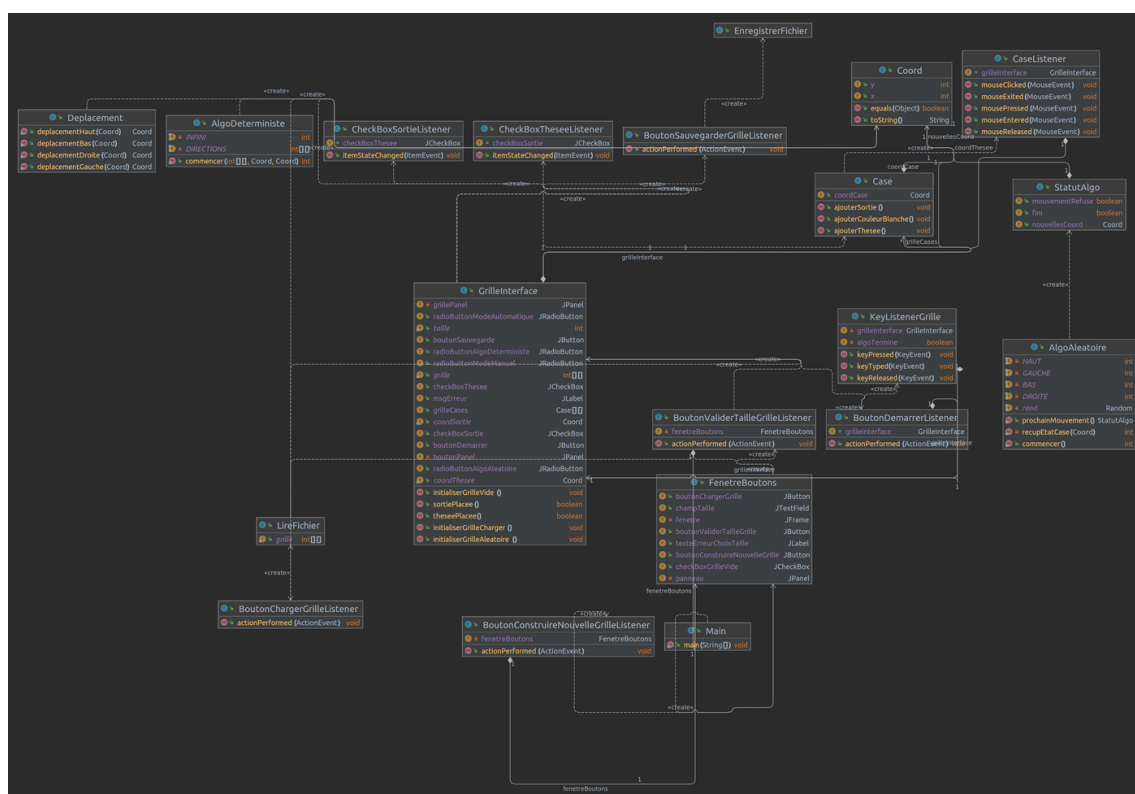
- La classe **GrilleInterface** est l'une des bases de notre code, celui-ci s'occupe de générer la grille de l'algorithme. On y retrouve des fonctions qui permettent l'initialisation de la grille, qu'elle soit construite aléatoirement ou totalement vide ; également une fonction permettant l'initialisation de Thésée et de la sortie et une méthode permettant de vérifier qu'elles soient bien placées. Pour couronner le tout, une méthode permet d'afficher le tout à l'écran.

Passons ensuite à l'une des fonctionnalités qui était demandée, la sauvegarde de la grille, sous la forme d'un fichier binaire. Ce fichier est généré lorsque l'utilisateur clique sur "Sauvegarder" après avoir construit une grille, ce qui permet de l'enregistrer dans un fichier. Après avoir relancé l'algorithme, s'il clique sur charger, il peut ouvrir l'explorateur de fichier afin de retrouver son fichier. La grille sauvegardée auparavant sera bien importée.

- La classe **LireFichier** ouvre un flux d'entrée qui permet de lire la taille de la grille, puis ouvre deux autres flux d'entrée permettent de stocker la position de Thésée et de la sortie. Le dernier flux stocke grâce à une boucle la position et la couleur de chaque case de la grille.
- La classe **EnregistrerFichier** permet d'enregistrer la position de Thésée et de la sortie sous forme d'octet (comme demandé dans le sujet, l'ordre est respecté). Le tout est ensuite converti en binaire, et ces données sont stockées dans la classe `ByteArrayOutputStream`, qui hérite d'un `Objet`. Ces octets permettent d'écrire dans le flux de sortie grâce à "write". Pour finir, ces données binaires seront collectées dans une variable, prise en paramètre par le constructeur, pour afficher la grille sauvegardée.

- La classe **Case** permet d'ajouter Thésée, la sortie ou alors de supprimer l'une des deux grâce à trois méthodes. Les cases sont définies par des coordonnées.
- La classe **Coord** permet de stocker les coordonnées de Thésée et de la sortie. Cela va être particulièrement utile si par exemple l'utilisateur souhaite modifier leurs positions, ce qui les supprimera de leur endroit initial, et laissera une case blanche ou noire.
- La classe **Déplacement** va utiliser des coordonnées de type Coord, qui permettront par la suite de déplacer Thésée selon la grille de jeu.
- La classe **KeyListenerGrille** permet de faire appelle à des méthodes qui vont permettre de prendre en compte les interactions de l'utilisateur qui sont effectuées sur le clavier, pour ensuite déplacer Thésée.
- La classe **AlgoAleatoire** permet de réaliser le parcours du labyrinthe en utilisant plusieurs méthodes, qui permettront à Thésée de se déplacer de manière totalement aléatoire.
- La classe **StatutAlgo** définit juste des données qui vont permettre d'être utilisées par d'autres classes pour donner des informations sur l'algorithme.

- La classe **AlgoDeterministe** permet de réaliser le parcours du labyrinthe en parcourant le plus court chemin possible amenant Thésée à la sortie.
- Pour finir, l'ensemble des classes restantes comportent dans leur nom "**Listener**", dans lesquelles nous allons y retrouver l'ensemble de la gestion des événements (lorsque l'utilisateur va interagir avec le programme). On y retrouve par l'exemple l'ouverture de l'explorateur de fichier lors de la sauvegarde, la génération de la grille après avoir appuyer sur un bouton. Toutes ces actions ne font rien si elles ne sont pas prises en compte par un ActionListener, un ItemListener etc... qui étendent de la classe EventListener.
- Un diagramme de classes a été réalisé après la programmation de l'algorithme :

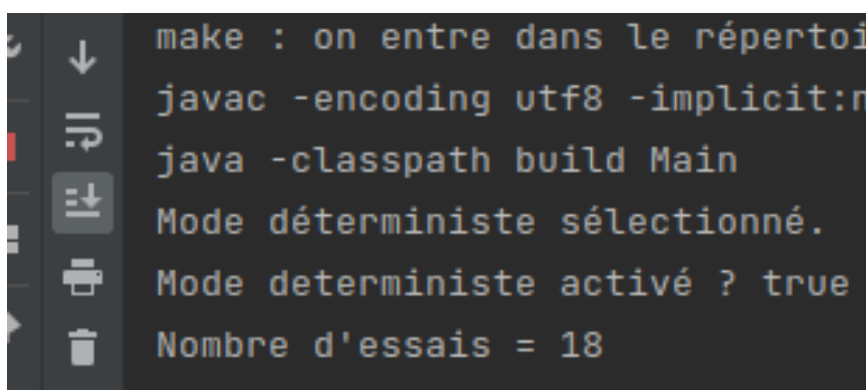


- Il est disponible de manière plus visible et de meilleure qualité dans le dossier **GIT** de notre projet : **SAE21\_2022/img**

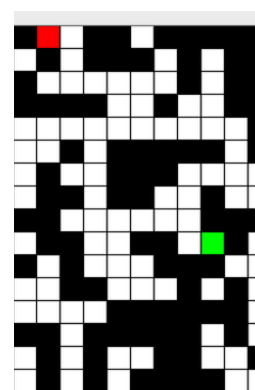
# EXPOSITION DE L'ALGORITHME

L'algorithme utilisé est basé sur la méthode de résolution décisive. Il consiste à parcourir toutes les possibilités pour trouver la sortie la plus courte d'un labyrinthe représenté par une grille. L'algorithme utilise une file d'attente pour stocker les cases à parcourir. Le point de départ est ajouté à la file d'attente avec une distance de 0. Tant que la file n'est pas vide, on retire le premier élément de la file, on explore ses voisins et on les ajoute à la file avec une distance correspondante à celle du point actuel + 1. L'algorithme utilise également une matrice de distances pour stocker les distances entre le point de départ et les autres points de la grille. Cette matrice est initialisée avec une valeur infinie pour toutes les cases, sauf pour le point de départ qui est initialisé à 0. Lorsque l'on trouve la sortie, on renvoie / affiche la distance correspondante. Si aucune sortie n'est trouvée, on renvoie -1. L'algorithme est intelligent puisqu'il trouve la solution la plus courte possible à chaque fois que l'utilisateur génère une nouvelle grille.

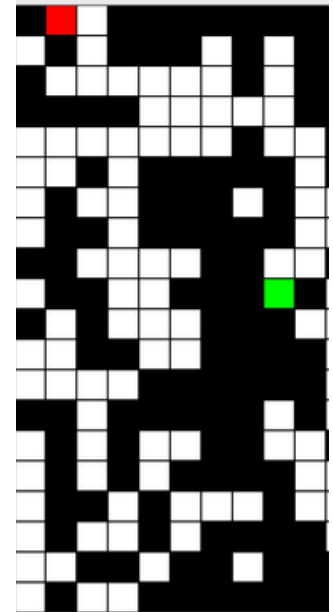
- Comme nous pouvons le voir sur les captures d'écran suivantes, l'algorithme est toujours capable de trouver le chemin le plus court, qu'on ajoute ou qu'on supprime des obstacles (cases noires) :



```
make : on entre dans le répertoire
javac -encoding utf8 -implicit:n
java -classpath build Main
Mode déterministe sélectionné.
Mode deterministe activé ? true
Nombre d'essais = 18
```



```
↑ make : on entre dans le répertoire
↓ javac -encoding utf8 -implicit:none
  java -classpath build Main
  Mode déterministe sélectionné.
  Mode deterministe activé ? true
  Nombre d'essais = 18
  Mode déterministe sélectionné.
  Mode deterministe activé ? true
  Nombre d'essais = 18
```



- Peu importe la position de la sortie ou de Thésée, la sortie est toujours atteignable par la plus petite distance grâce à l'algorithme déterministe.



# CONCLUSIONS PERSONNELLES

Baptiste: Ce deuxième projet du semestre a été pour moi assez difficile, puisqu'il m'a forcé à me documenter encore plus sur les pages manuel des différentes classes Java car il en existe énormément, que ce soit pour gérer l'affichage avec des JFrame, JPanel etc.. ou encore me documenter pour que les différents fichiers sources interagissent bien entre eux. Au départ, j'appréhendais moins le Java que le C, je trouvais que visuellement ça avait l'air plus facile, puis au fur à mesure de la programmation de la SAE, je me suis rendu compte qu'il existait quand même beaucoup de différences entre les deux langages, et qu'il était parfois difficile de s'y retrouver. La partie graphique avec l'ajout de boutons, de checkboxes ou encore de JRadioButton a été la partie la plus simple pour moi. Ce projet m'a surtout permis d'apprendre à résoudre un problème grâce aux outils qui m'entourent (Internet par exemple). Le plus dur a été la création de classes car avec l'accumulation de code, il était difficile de s'y retrouver pour réaliser ce que l'on voulait vraiment faire. Ce projet a cependant été un excellent compromis pour m'améliorer en Java.

Samet: Ce deuxième projet que j'ai réalisé était assez complexe en raison du grand nombre de classes utilisées, ce qui a nécessité une documentation approfondie. Le passage du langage C au langage Java a également été difficile pour moi, contrairement à ce que les gens disent sur la facilité de la transition si on a des bases en C. Cependant, grâce à ce projet, j'ai acquis une meilleure compréhension du Java et j'ai pu améliorer mes compétences en programmation orientée objet. La partie interface graphique du projet était assez simple à utiliser, mais malheureusement nous n'avons pas eu suffisamment de temps pour terminer ce projet entièrement. Cela m'a permis de me rendre compte de mes erreurs dans la gestion du temps et du travail, ainsi que de mon plus grand problème : la gestion des bugs. En effet, lorsque des erreurs apparaissaient, j'avais tendance à passer trop de temps à les chercher et à les corriger, ce qui ralentissait considérablement mon travail. En fin de compte, ce projet a été une expérience très formatrice pour moi, qui m'a permis de développer mes compétences en Java et en programmation orientée objet, ainsi que de prendre conscience de mes lacunes en matière de gestion du temps et de résolution de bugs.