# L'algorithme d'Ariane

Sommaire :

## I) Introduction

L'algorithme d'Ariane, est un cas particulier d'algorithme de guidage, visant à conduire un objet mobile jusqu'à son but à travers un parcours d'obstacles.
L'algorithme d'Ariane est inspiré de la mythologie grec dans laquelle Thésée, cherchant à tuer le minotaure pour mettre fin à des sacrifices, se retrouve perdu dans son labyrinthe et cherche la sortie.

Notre programme va guider Thésée et l'aider à sortir du labyrinthe, cette version n'inclus pas le minotaure.

## II) Installation et commandes

1) Installer le JRE (java runtime environment)

https://openjdk.java.net/install/ (version open-source)

https://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html (version oracle)

2) Installer le jeu

Aller dans le dossier PT21_APL2018/

Exécuter la commande make

3) Lancer le jeu

Exécuter la commande make run

4) Générer la documentation (javadoc)

Exécuter la commande make javadoc

5) Ouvrir la javadoc (firefox)

Exécuter la commande make doc

6) Désinstallation

Exécuter la commande make clean

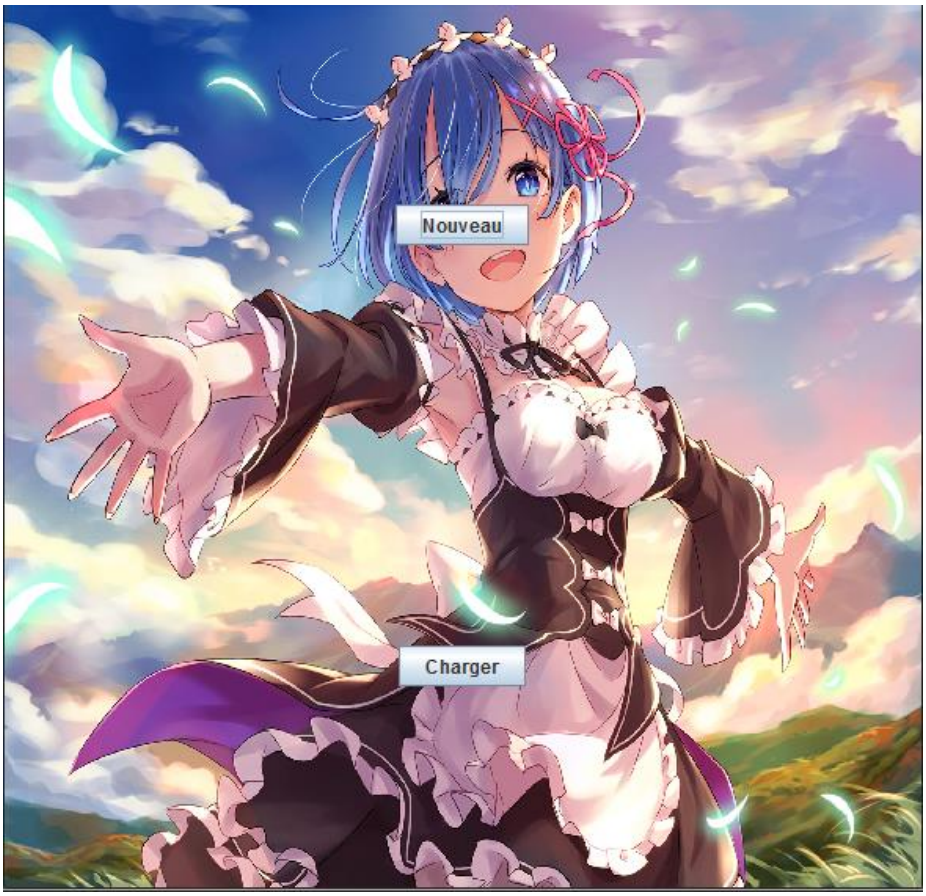
## III) Fonctionnalités

Thésée sera représenté par un joueur  , la sortie par un coffre  et les obstacles par des murs  .

1. Menu principal

Lors du lancement du jeu, on accède à un premier menu, on peut choisir entre :

- "nouveau" qui permet de crée un nouveau labyrinthe (voir 2. )
- "charger" un labyrinthe depuis une sauvegarde. (voir 3. )

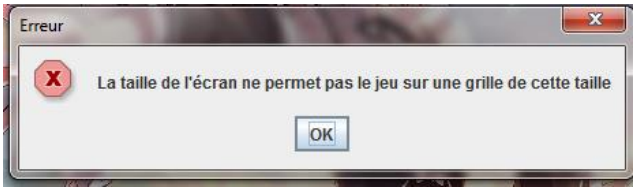



(annexe 1 - menu principal)

## 2. Nouvelle partie

On peut choisir de crée une nouvelle partie, on commencera par donner la taille de la grille, des messages aideront l'utilisateur s'il rentre une valeur invalide. Les messages apparaitront si :

- la taille de la grille est inférieure ou égale à 1 donc Thésée et la sortie sont confondus ou alors l'un des deux voir les deux ne seront pas dans le labyrinthe.
- la taille demandée est supérieur à la taille de la grille maximale (100x100)
- la taille demandée est inférieure à la taille de la grille maximale mais cet écran ne permet pas l'affichage d'une telle grille.
  Ce problème survînt car la fenêtre va s'agrandir (pas rétrécir) si et seulement si elle peut pour essayer d'afficher la grille (agrandissement limité par taille de l'écran et la taille max de la grille).
- la taille rentrée n'est pas un nombre



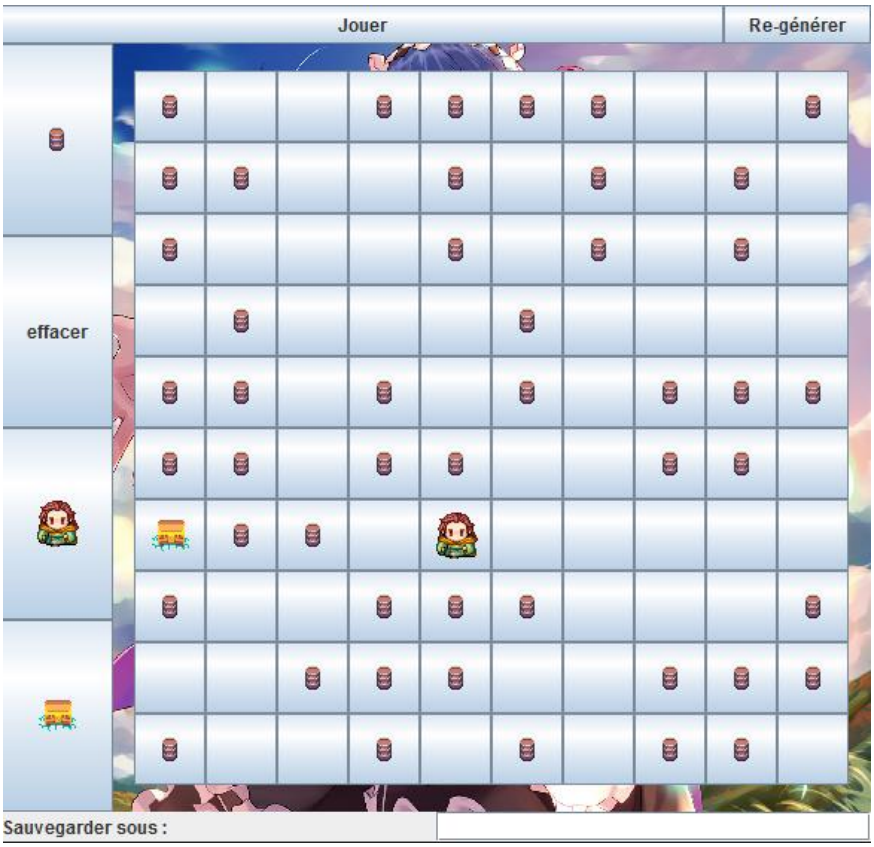

(annexe 2)


(annexe 3)

Vous aurez ensuite le choix pour remplir la grille (annexe 4) :

- "aléatoire" : on remplit aléatoirement la grille.
- "vide" : toutes les cases sont laissées vides.

L'algorithme de remplissage génère la position du joueur puis celle de la sortie en prenant en compte qu'elles doivent être distinctes puis pendant un nombre aléatoirement de fois va prendre un case différente du joueur et de la sortie et va aléatoirement changer son état entre vide et bloquée. Le bouton Re-générer permet de lancer/relancer l'algorithme de remplissage aléatoire de la grille).




(annexe 4 - création du jeu, remplissage aléatoire)

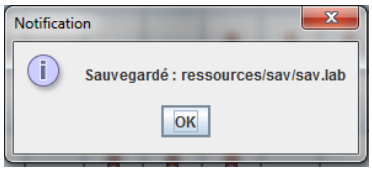
Remarque : il n'existe pas forcément de chemin allant à la sortie.

Peu importe votre choix, vous pouvez cliquer sur l'un des boutons de gauche, et toutes les prochaines cases de la grille cliquée prendront la forme de ce bouton (annexe 5,6). Notez qu'il ne peut y avoir qu'exactement un joueur par grille et un coffre par grille (donc placer le joueur/sortie revient va les déplacer et non en ajouter un nouveau).

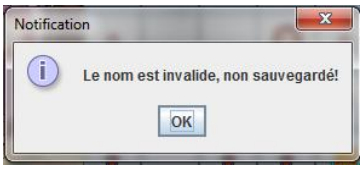

(annexe 5 - sélection)


(annexe 6 - place le joueur)

On peut ensuite choisir de sauvegarder en rentrant le nom de la sauvegarde, cependant seuls les caractères alphanumériques sont autorisés donc caractères imprimables.




(annexe 8 - sauvegarde)



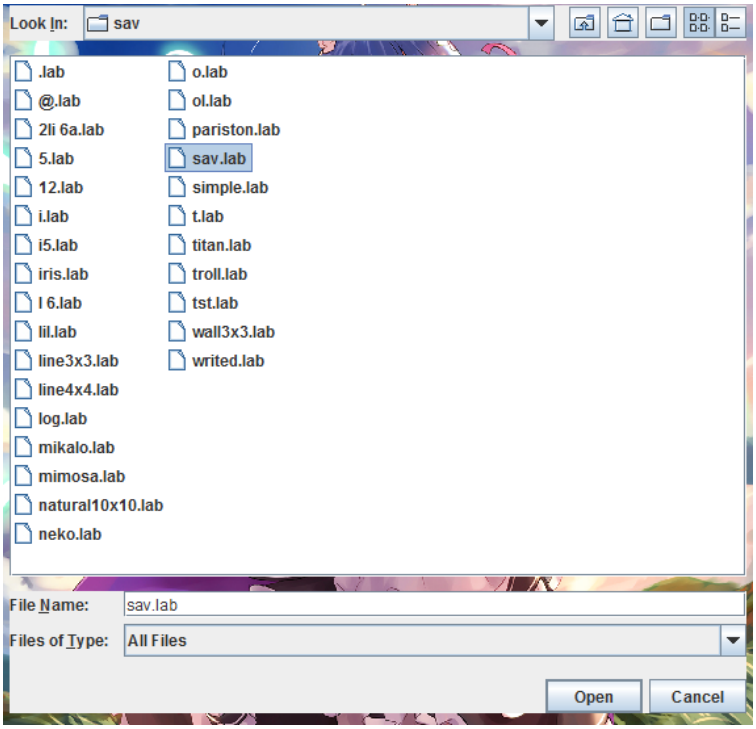

(annexe 9 - erreur)

Le bouton jouer permet de lancer l'exécution de la simulation (voir 4.). Notez que la simulation ne se lance que si la grille contient un joueur et une sortie sinon un message d'erreur s'affichera.


annexe 7 - erreur lancement du jeu

## 3. Charger une partie

Charger une partie, vous permet de charger un fichier de sauvegarde (sous la forme indiqué dans la javadoc de la classe SaveLoader). Les données du fichier doivent être correctes (elles sont vérifiées telles que la position de la sortie, du joueur ou encore qu'il y ait autant de valeurs 0/1 qu'il y ait de cases). Si un fichier incorrect est lu, alors une erreur est affichée (annexe 9) sinon le jeu est lancé (voir 4. ).
Le bouton annuler, permet de revenir en arrière, open d'ouvrir la sauvegarde (ou double clic dessus).




(annexe 8 - choix de la sauvegarde)


(annexe 9 - erreur de lancement du jeu)

## 4. Lancement du jeu

Le lancement du jeu se déroule en deux temps, on choisit d'abord si on veut un guidage aléatoire (on essaye aléatoirement de se déplacer et par chance trouver la sortie) ou déterministe (on cherche

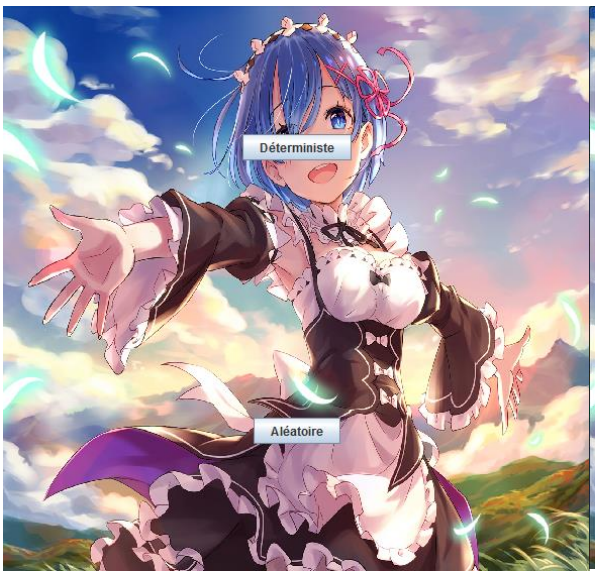
intelligemment la sortie en utilisant une méthode). Le cas de l'algorithme déterministe est expliqué dans V car il a pour objectif de toujours trouver la sortie s'il y en a une et avec le moins de déplacements possibles.

On choisit ensuite le mode de défilement entre :

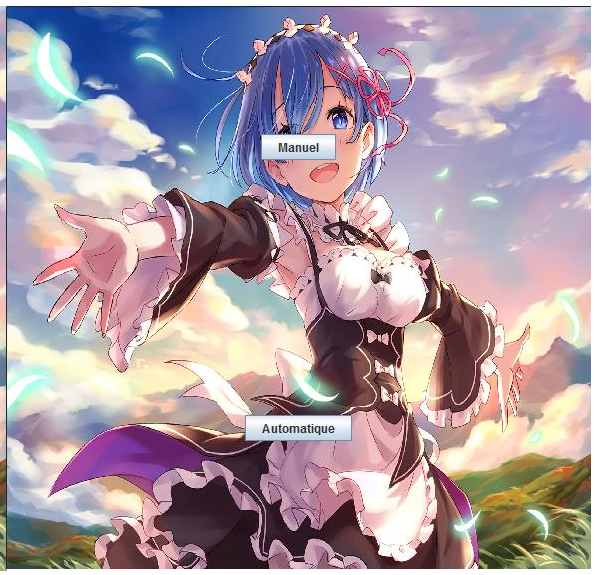
- manuel = on appuie sur n'importe quel touche, et on voit l'algorithme s'exécuter pas à pas, la direction conseillée par l'algorithme est affichée en haut (annexe 12)
- automatique = on ne voit pas l'algorithme tourner, dans le cas de l'algorithme déterministe, le nombre de déplacements est affiché et dans le cas de l'algorithme aléatoire, il s'agît d'une moyenne sur 100 essais.

Le résultat est ensuite affichée ou un message disant qu'il n'y a pas de sortie s'il n'y en a pas.

L'algorithme aléatoire est limité par le fait qu'il n'est lancé que s'il existe une sortie, et continue de tourner seulement si le nombre de déplacements est inférieur à 100 000 (fixé).




(annexe 10 - choix algorithme)




(annexe 11 - choix mode défilement)


(annexe 12 - mode manuel)


(annexe 13 - résultats)

## IV) Structure du programme

Le programme est composé de 32 classes selon 2 rôles principaux :

- les classes qui sont dans le package engine et qui sont faites pour être réutilisable sans le package game.
- Les classes du package game qui représentent notre jeu qui lui sont alors spécifiques et ne peuvent pas être réutilisées.

Le diagramme de classes du projet se trouve à la fin (pages 9 et 10).

Le programme est structuré de la façon suivante :

Le Main sert de lanceur, il ne connaît pas le jeu qu'il va lanceur mais s'attends à ce qu'il possède certaines méthode (Classe abstraite Game) pour pouvoir lancer le jeu.

Le jeu hérite de Game possède des composants (implémente GameComponant) tels le menu (Classe Menu) qui charge une sauvegarde ou choisi de créer une grille et le cœur du jeu (GameCore) qui lit une sauvegarde (SaveLoader), Crée une nouvelle partie (CreateGame) et lance l'algorithme de parcours en fonction des choix de l'utilisateur.

Le jeu est lancé dans une fenêtre (Window) qui a la particularité d'exister à partir de son fond d'écran (Background) chargé par ImageLoader.

Le labyrinthe est représenté sous la forme d'une grille (Grille) qui contient des cases (Cases).

Si des erreurs surviennent ou on veut afficher des messages, on utilisera les classes de popup et d'exceptions (…Popup/…Exception).

## V) Algorithme déterministe

L'algorithme déterministe est basé sur les algorithmes de parcours en largeur et profondeur (Breadth-first search et Depth-first search) sur les graphes. Il va aller dans une direction, dans l'ordre :

1. gauche
2. droite
3. haut
4. bas

On utilisera un système de liste, où on ajoute la case actuelle au lancement de l'algorithme, puis chaque case où on tente d'aller/va.

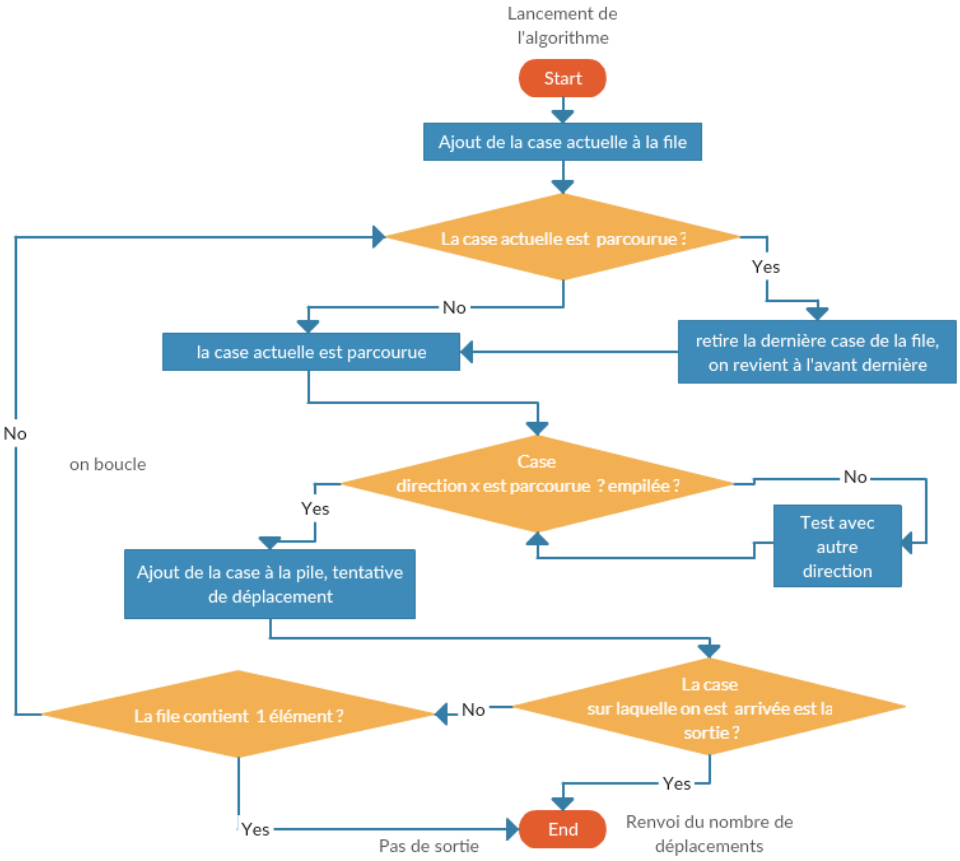
Thésée commence par définir la case où il se trouve comme étant parcourue. Il va ensuite regarder les directions (voir ordre supra) et va choisir dans qu'elle direction il va tenter de se déplacer.

Il ne va accepter de se déplacer dans une direction, uniquement si :

- il n'est jamais allé sur une case (non parcourue)
- il n'a pas classé la case dans la direction x comme étant empilée (ie statut donné à une case pour laquelle Thésée à essayer de s'y déplacer mais à fait du surplace, ou alors si c'est cas menait à un cul-de-sac.

Il empile la case qu'il veut atteindre s'il a choisi une direction sinon (toutes les cases adjacentes sont parcourues/empilées), il marque sa case actuelle comme étant empilée pour ne plus y aller (la retire de la file) et revient en arrière (revient à la nouvelle dernière case de la file).

Cet algorithme tourne tant que Thésée n'a pas trouvé la sortie ou qu'il revient à sa position de départ et que les 4 cases adjacentes sont empilées.



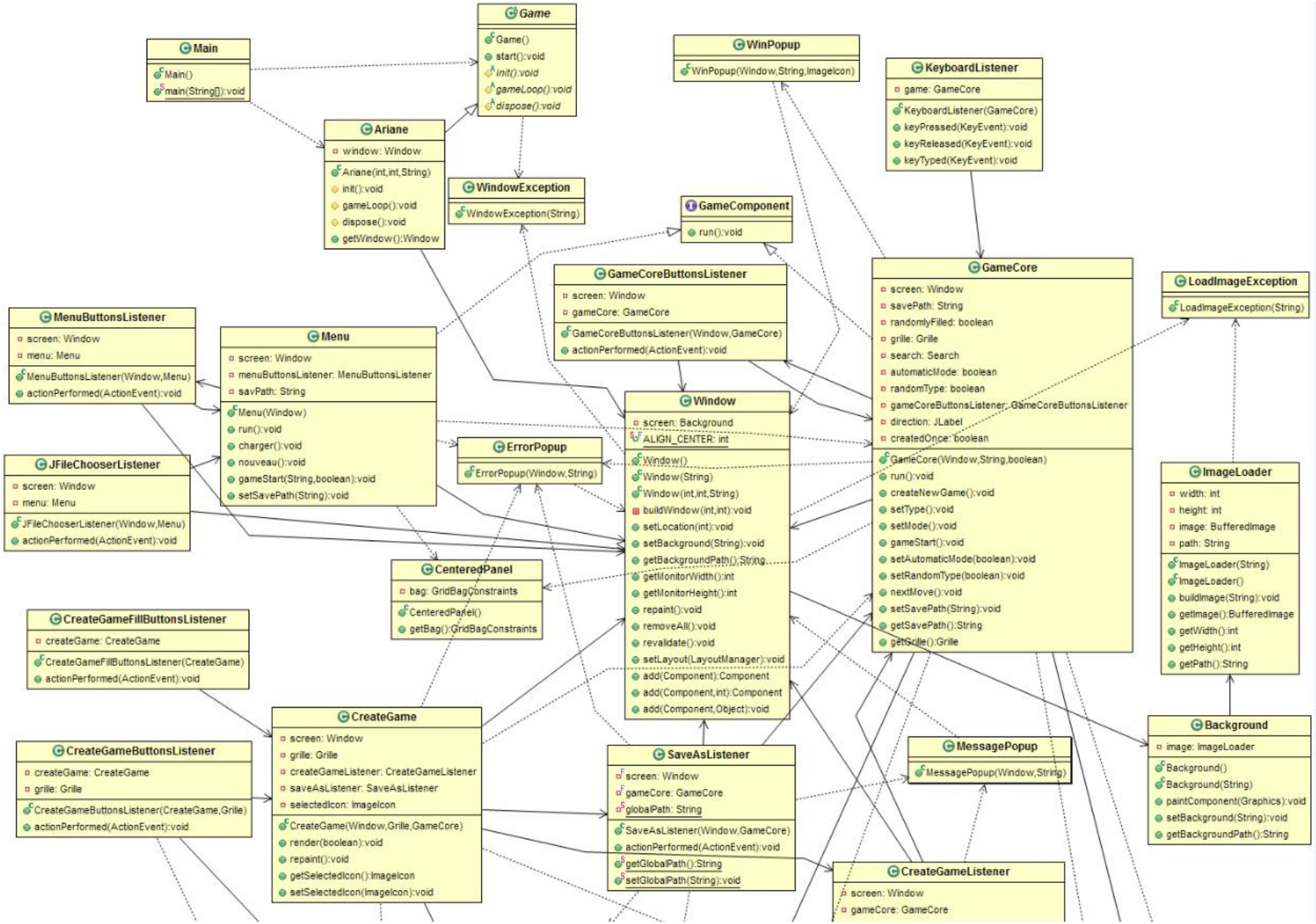

L'algorithme atteindra toujours la sortie ?

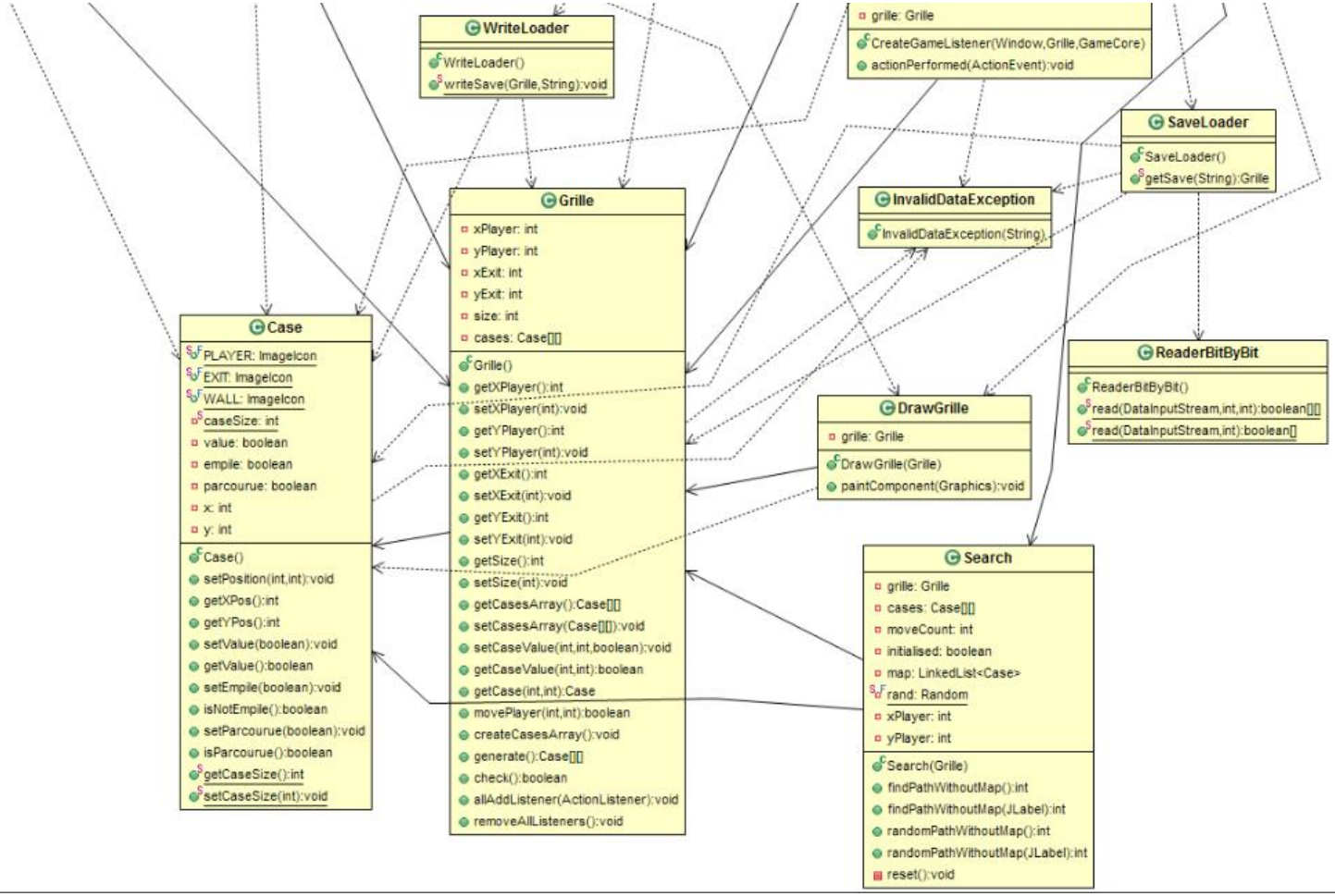
Thésée marque les cases qu'il parcourt, donc il ne tournera pas en rond car le seul moment où il revient sur ces pas, c'est lors qu'il est bloqué et condamne alors la case. Thésée va explorer au maximum chaque chemin et seulement lorsqu'il arrive dans un cul-de-sac fait demi-tour, « empile » toutes les cases jusqu'à la dernière intersection et recommence. Thésée parcours toutes les cases de tous les chemins et donc forcément trouver la sortie s'il y en a une.

## VI) Conclusion personnelle

Je pense que pour savoir écrire du code, il faut écrire du code. J'ai passé énormément de temps à comprendre, analyser, méthodiquement réfléchir à mon projet : Quels sont mes objets ? Comment les représenter ? Comment mon programme va fonctionner ? Quels comportements pourraient avoir les utilisateurs ? J'ai fait en sorte d'écrire directement la javadoc et de simuler que je faisais le projet à plusieurs ce qui m'a obligé à écrire des classes qui soit plus universelles et à penser mon code énormément différemment que je l'aurais fait en C par exemple. J'espère que tout futur utilisateur prendra autant de plaisir à tester mon application, que j'en ai eu à la concevoir.

Quentin Ramsamy- -Ageorges, Groupe 6


Game
Game()
start():void
init():void
gameLoop():void
dispose():void
Main
Main()
main(String[]):void
WinPopup
WinPopup(Window,String,ImageIcon)
KeyboardListener
game: GameCore
KeyboardListener(GameCore)
keyPressed(KeyEvent):void
keyReleased(KeyEvent):void
keyTyped(KeyEvent):void
Ariane
window: Window
Ariane(int,int,String)
init():void
gameLoop():void
dispose():void
getWindow():Window
WindowException
WindowException(String)
GameComponent
run():void
GameCoreButtonsListener
screen: Window
gameCore: GameCore
GameCoreButtonsListener(Window,GameCore)
actionPerformed(ActionEvent):void
GameCore
screen: Window
savePath: String
randomlyFilled: boolean
grille: Grille
search: Search
automaticMode: boolean
randomType: boolean
gameCoreButtonsListener: GameCoreButtonsListener
direction: JLabel
createdOnce: boolean
GameCore(Window,String,boolean)
run():void
createNewGame():void
setType():void
setMode():void
gameStart():void
setAutomaticMode(boolean):void
setRandomType(boolean):void
nextMove():void
setSavePath(String):void
getSavePath():String
getGrille():Grille
LoadImageException
LoadImageException(String)
MenuButtonsListener
screen: Window
menu: Menu
MenuButtonsListener(Window,Menu)
actionPerformed(ActionEvent):void
Menu
screen: Window
menuButtonsListener: MenuButtonsListener
savPath: String
Menu(Window)
run():void
charger():void
nouveau():void
gameStart(String,boolean):void
setSavePath(String):void
Window
screen: Background
ALIGN_CENTER: int
Window()
Window(String)
Window(int,int,String)
buildWindow(int,int):void
setLocation(int):void
setBackground(String):void
getBackgroundPath():String
getMonitorWidth():int
getMonitorHeight():int
repaint():void
removeAll():void
revalidate():void
setLayout(LayoutManager):void
add(Component):Component
add(Component,int):Component
add(Component,Object):void
ErrorPopup
ErrorPopup(Window,String)
JFileChooserListener
screen: Window
menu: Menu
JFileChooserListener(Window,Menu)
actionPerformed(ActionEvent):void
ImageLoader
width: int
height: int
image: BufferedImage
path: String
ImageLoader(String)
ImageLoader()
buildImage(String):void
getImage():BufferedImage
getWidth():int
getHeight():int
getPath():String
CenteredPanel
bag: GridBagConstraints
CenteredPanel()
getBag():GridBagConstraints
CreateGameFillButtonsListener
createGame: CreateGame
CreateGameFillButtonsListener(CreateGame)
actionPerformed(ActionEvent):void
CreateGame
screen: Window
grille: Grille
createGameListener: CreateGameListener
saveAsListener: SaveAsListener
selectedIcon: ImageIcon
CreateGame(Window,Grille,GameCore)
render(boolean):void
repaint():void
getSelectedIcon():ImageIcon
setSelectedIcon(ImageIcon):void
Background
image: ImageLoader
Background()
Background(String)
paintComponent(Graphics):void
setBackground(String):void
getBackgroundPath():String
CreateGameButtonsListener
createGame: CreateGame
grille: Grille
CreateGameButtonsListener(CreateGame,Grille)
actionPerformed(ActionEvent):void
SaveAsListener
screen: Window
gameCore: GameCore
globalPath: String
SaveAsListener(Window,GameCore)
actionPerformed(ActionEvent):void
getGlobalPath():String
setGlobalPath(String):void
MessagePopup
MessagePopup(Window,String)
CreateGameListener
screen: Window
gameCore: GameCore


WriteLoader
WriteLoader()
writeSave(Grille,String):void
grille: Grille
CreateGameListener(Window,Grille,GameCore)
actionPerformed(ActionEvent):void
SaveLoader
SaveLoader()
getSave(String):Grille
InvalidDataException
InvalidDataException(String)
Grille
xPlayer: int
yPlayer: int
xExit: int
yExit: int
size: int
cases: Case[][]
Grille()
getXPlayer():int
setXPlayer(int):void
getYPlayer():int
setYPlayer(int):void
getXExit():int
setXExit(int):void
getYExit():int
setYExit(int):void
getSize():int
setSize(int):void
getCasesArray():Case[][]
setCasesArray(Case[][]):void
setCaseValue(int,int,boolean):void
getCaseValue(int,int):boolean
getCase(int,int):Case
movePlayer(int,int):boolean
createCasesArray():void
generate():Case[][]
check():boolean
allAddListener(ActionListener):void
removeAllListeners():void
Case
PLAYER: ImageIcon
EXIT: ImageIcon
WALL: ImageIcon
caseSize: int
value: boolean
empile: boolean
parcourue: boolean
x: int
y: int
Case()
setPosition(int,int):void
getXPos():int
getYPos():int
setValue(boolean):void
getValue():boolean
setEmpile(boolean):void
isNotEmpile():boolean
setParcourue(boolean):void
isParcourue():boolean
getCaseSize():int
setCaseSize(int):void
DrawGrille
grille: Grille
DrawGrille(Grille)
paintComponent(Graphics):void
ReaderBitByBit
ReaderBitByBit()
read(DataInputStream,int,int):boolean[][]
read(DataInputStream,int):boolean[]
Search
grille: Grille
cases: Case[][]
moveCount: int
initialised: boolean
map: LinkedList<Case>
rand: Random
xPlayer: int
yPlayer: int
Search(Grille)
findPathWithoutMap():int
findPathWithoutMap(JLabel):int
randomPathWithoutMap():int
randomPathWithoutMap(JLabel):int
reset():void