LU2IN002 : Programmation Orientée Objet JAVA

# Rapport de projet

## Objet : Jeu de labyrinthe


**Groupe :** 09 – DM IM

**Réalisé par :**

- AOUAOUCHE Louiza 3802084
- FREY Arthur 3802891

**Encadrés par :** Armel Jacques Nzekon


**Année universitaire :** 2019-2020

# Table des matières

# I. Introduction

Dans le cadre de la deuxième année de licence d'informatique au sein de Sorbonne Université, l'UE LU2IN002 nous a amenés à réaliser un projet JAVA initiant à la programmation objet. Après une grande délibération nous avons conclu sur le choix d'un jeu qui peut résumer toutes les notions vues en cours, TD et TME de l'UE : le jeu du labyrinthe.

# II. Jeu du labyrinthe

## 1. Objectif

Réalisation d'un jeu de labyrinthes de trois niveaux de difficultés (facile, normal et difficile) où le principe reste le même que le labyrinthe classique c'est à dire à travers une entrée il faut aboutir à la sortie avant que le temps ne s'écoule.

## 2. Problématique

La principale problématique de départ était : *comment générer des labyrinthes jouables et différents sur chaque partie ?*

Pour cela, nous avons consacré une partie de notre temps pour trouver la façon la plus intelligente pour cette construction et nous avons finalement opté pour une technique très souvent utilisée en informatique appelée le « retour récursif », back tracking en anglais.

L'application sur notre cas de labyrinthes est la suivante :

- Choix d'une case de départ
- Choix aléatoire d'une direction, création d'un chemin vers la case adjacente (en brisant le mur entre les deux) si et seulement si cette case voisine n'a pas été visitée. Elle devient la case courante à son tour.
- Si toutes les cases adjacentes ont été visitées c'est là que l'on fait un retour récursif vers les cases précédentes
- Le processus s'arrête au moment où toutes les cases ont fait leur retour récursif (de toutes les directions) jusqu'au point de départ.

Une fois que nous avons eu notre pièce maitresse, nous avons alors défini nos classes.

## 3. Classes

N.B :

Les attributs dont on ne spécifie pas la portée est probablement déclarée private afin qu'elle ne puisse être utilisée qu'à l'intérieur de la classe considérée.

La plupart des classes ont des constructeurs, getters et setters basiques.

---

- **Classe Case :** implémente une case du plateau de jeu et les différentes méthodes qui peuvent s'appliquer dessus

  - **Attributs :**

  *public static final Int IDVIDE=0* type par défaut d'une case, **static** car il concerne la classe case, **final** car sa valeur ne change pas et **public** pour faciliter la manipulation par les autres classes vu que sa valeur reste inchangée
  *public static final Int IDJOUEUR=2* si elle est occupée par le joueur , idem
  *Boolean murOuest,murNord,murEst,murSud* :vrai si la case contient le mur en question
  *int type_case : contient le type de la case, par défaut vide, un entier sinon*
  *boolean visited : pour savoir si la case a été visitée*
  *int x,y : position de la case dans le plateau de jeu*

---

- **Classe Joueur :** implémente la position du joueur à tout moment

  - **Attributs :**

  *Position p : position du joueur sur le plateau de jeu*

---

- **Classe Position :** implémente une position donnée sur le plateau de jeu

  - **Attributs :**

  *int x* :position sur les lignes
  *int y :position sur les colonnes*

---

- **Classe Direction :** implémente les directions vers lesquelles le joueur peut bouger

  - **Attributs :**

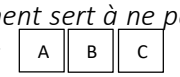
  Nous avons choisi d'initialiser les directions sur les puissances de 2 afin de faire facilement des calculs binaires et éviter les débordements
  final int N=1 : *direction nord*
  final int S=2 : *direction sud*
  final int E=4 : *direction est*
  final int W=8 : *direction ouest*

final int[ ] dx=new int[9] : *tableau qui résume les pas à ajouter selon x en cas de déplacement vers une direction donnée, la dimension 9 est juste pour simplifier l'initialisation et l'utilisation* *ex : pour l'Est on initialise dx[E]=1 (c'est équivalent à dx[4] au lieu de dx[2] vu que c'est la troisième donnée mais ça facilite la manipulation)*
final int[ ] dy=new int[9] : *tableau qui résume les pas à ajouter selon y, en cas de déplacement vers une direction donnée*
final int[ ] opposite=new int[9] *: tableau qui stocke pour chaque direction son opposée (S <->N , E<->W)*

- **Classe Labyrinthe :** implémente le plateau de jeu, c'est la classe principale du jeu

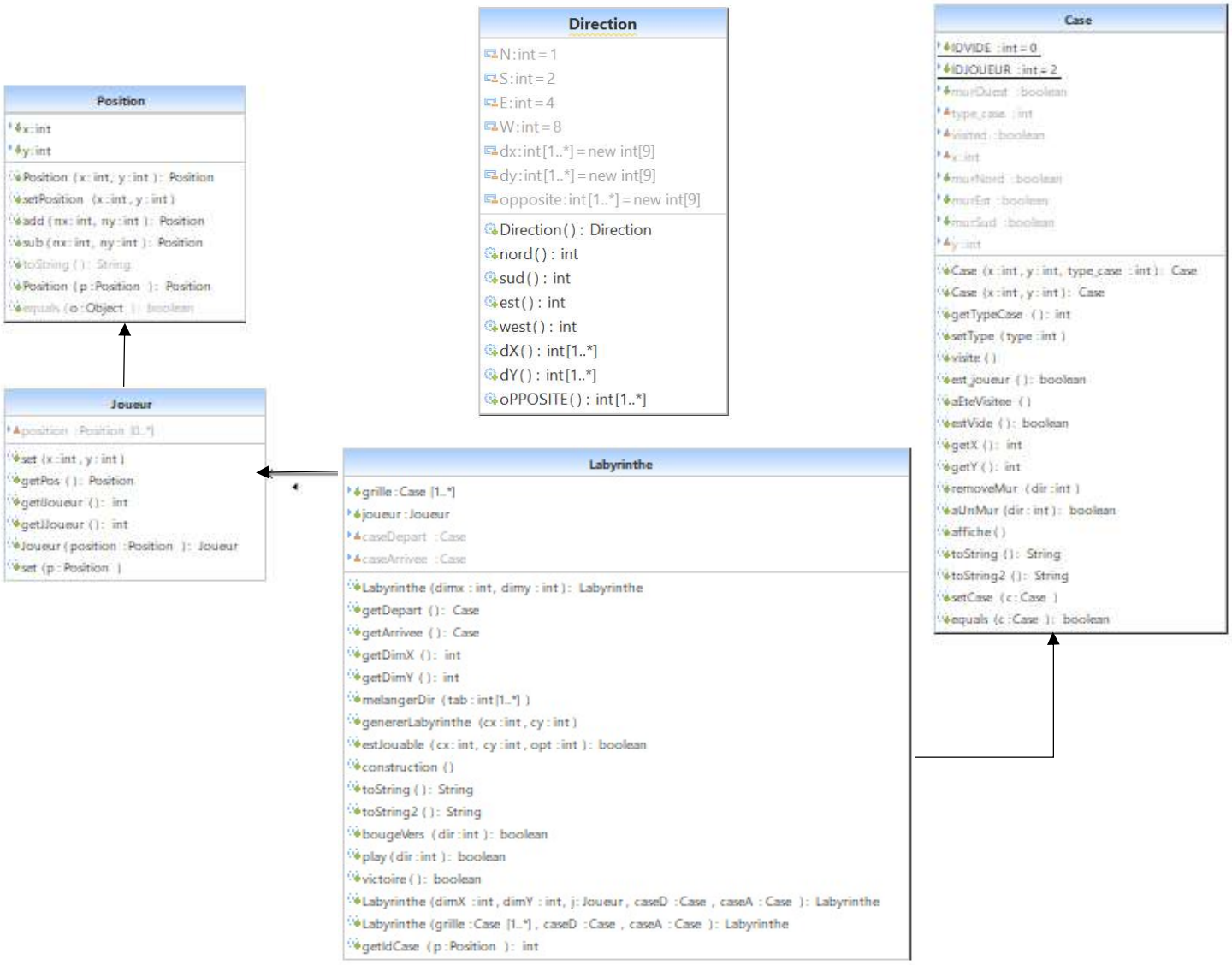
  - Attributs :

*Cases[ ][ ] grille* :tableau 2D qui représente les cases du labyrinthes
*Joueur joueur :* représente le joueur actuel sur le labyrinthe
*Case caseDepart* : c'est une case particulière à chaque labyrinthe qui représente l'unique case de départ du joueur, elle est initialisée aléatoirement dans le constructeur sur la première colonne du labyrinthe
*Case caseArrivee* : c'est une case particulière à chaque labyrinthe qui représente l'unique case d'arrivée du joueur, elle est initialisée aléatoirement dans le constructeur sur la dernière colonne du labyrinthe

  - Methodes importantes :
    - **genererLabyrinthe(int cx, int cy)** *:* c'est cette méthode qui permet d'implémenter notre algorithme de retour récursif à partir d'une cellule de départ [cx,cy]
    - **estJouable( int cx, int cy , int opt)** *: renvoie vrai si le labyrinthe généré est jouable (il existe un chemin entre la case de départ [cx,cy]et la case d'arrivée), le dernier argument sert à ne pas refaire une direction qui mène à une case déjà visitée ( ex :* 

*A -> B /vers l'est, on ne refait pas B->A /vers l'ouest pour éviter des chemins inutiles qui sont déjà visités )*
    - **Construction()** *: elle génère un nouveau labyrinthe tant que le précédent n'est pas jouable*

## 4. Diagramme UML



## 5. Graphisme

Etant donné que le côté graphique n'a pas été traité en détails en cours, TD et TME (les fenêtres d'affichage nous ont souvent été données), ce projet nous a vraiment initié au monde graphique offert par JAVA qui est immense.
Nous nous sommes beaucoup inspirés de ce que nous avons déjà fait en TME, nous avons donc utilisé **la classe SimpleInterface** dont nous avons l'habitude d'utiliser, fournie par Mr Vincent GUIGUE.

Nous avons alors opté pour plusieurs fenêtres pour notre jeu :

- **Classe Menu extends JFrame implement ActionListener :** elle implémente la fenetre du menu principal, qui hérite donc de JFrame
  - **Attributs principaux :**
    *JButton playButton* : bouton qui permet de lancer le jeu
    *JButton exitButton* : bouton qui permet de fermer le jeu
    *JButton helpButton* : bouton qui permet de lancer le jeu

  - On construit un panel pour la fenêtre du menu avec la méthode **BuildContentPane()** où nous faisons l'affichage grâce un **GridBagConstraints**() qui contient nos bouttons
  - La méthode **Help()** crée une nouvelle SimpleInterface qui affiche des instructions pour le jeu, cette fenêtre est construite à partir d'une photo préalablement crée qui sert de background
  - Finalement on gère les clicks sur les boutons avec un **ActionListener()**

---

- **Classe Menu2 extends JFrame implement ActionListener :** elle implémente la fenêtre du menu des niveaux de jeu, elle fait pratiquement la même chose que la classe Menu sauf qu'ici nous gérons :
  - **Attributs principaux :**
    *JButton facile* : bouton pour choisir le niveau facile
    *JButton normal* : bouton pour choisir le niveau normal
    *JButton difficile* : bouton pour choisir le niveau difficile

---

- **Classe Minuteur :** elle implémente **un compte à rebours pour chaque partie**
  - Attributs
    *Int seconds=0* : nombre de secondes écoulées depuis le lancements du minuteur par la méthode **start()**
    *Timer timer* : minuteur standard de java.util

## 6. Main

- Classe MainMaze : c'est la classe qui contient le main pour tester le jeu, tous ses attributs sont **static** car ils sont spécifiques à cette classe
    - Attributs :
      *Menu menu :* menu principal
      *SimpleInterface ui* : la fenêtre sur laquelle on dessine notre labyrinthe
      *final int sec1 =10, sec2=20,sec3=30 :* nombre de secondes pour une partie de niveau 1 (2 ou 3), final car on ne veut pas qu'elle soit changée au cours du programme
      *BufferedImage[ ] sprites, lignedepart, lignearrivee,particuliers :*image pour dessiner le jeu
    - *Méthodes importantes :*
        - **Show :** permet construire le labyrinthe et l'afficher sur la fenêtre ui
        - **playUI :** permet au joueur de jouer au clavier pour déplacer son personnage sur le labyrinthe
        - **vicMessage(), echecMessage()** : implémente une fenêtre affichant à l'utilisateur s'il a gagné (atteinte de la case d'arrivée) ou perdu (temps expiré)

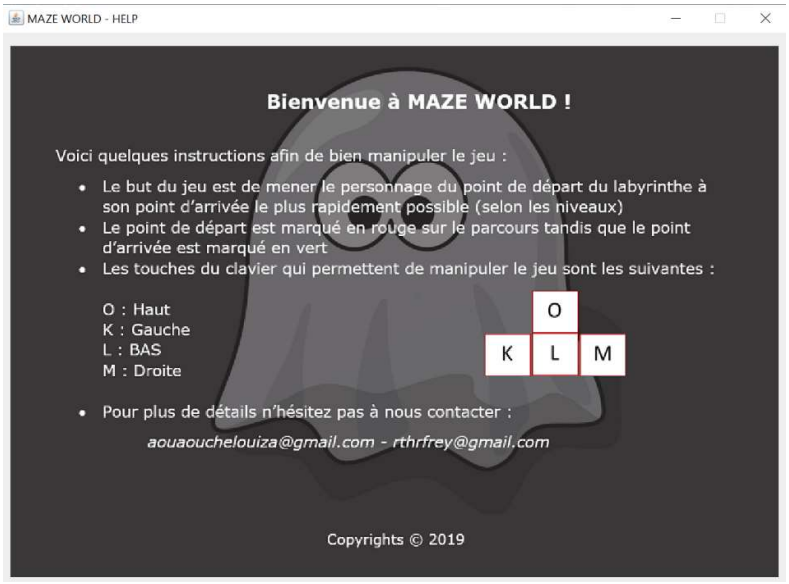
## 7. Résultat

Nous avons appelé notre jeu : MAZE WORLD qui signifie le monde des labyrinthes

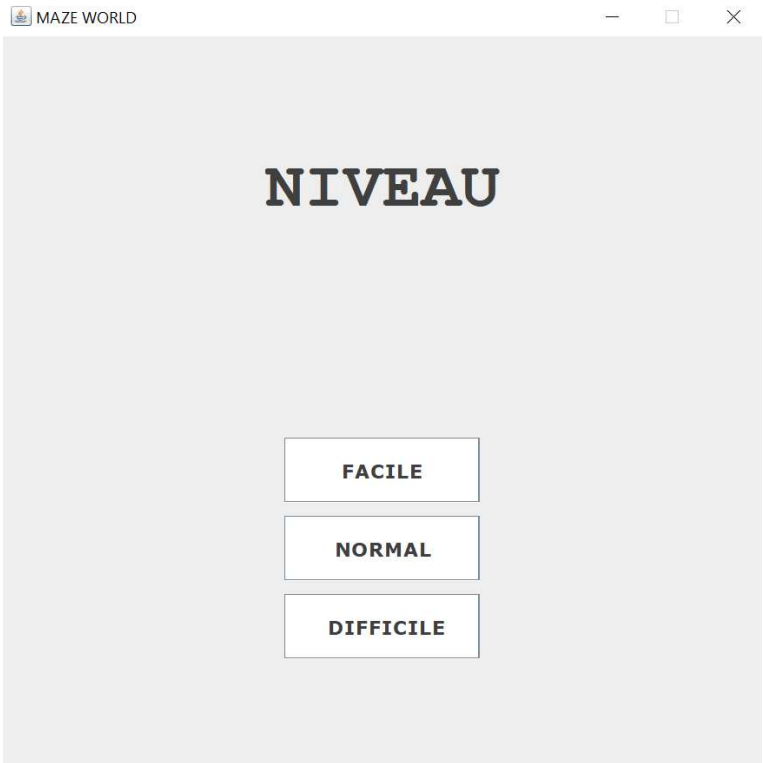
- Menu principal :

- **EXIT :** ferme le programme

- **HELP :**

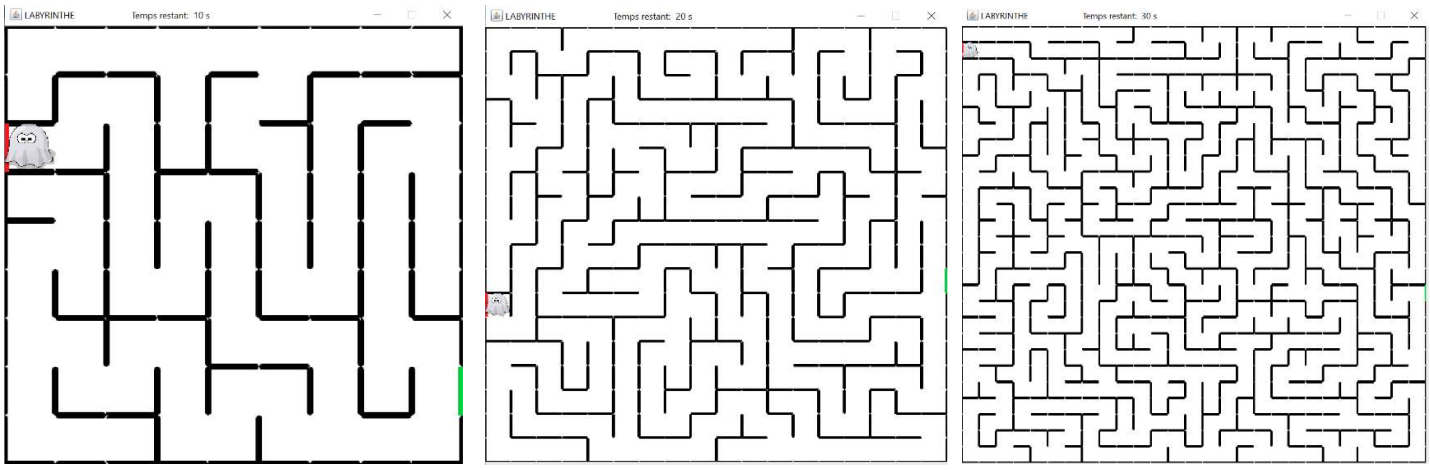



- **Play :** affichage du menu des niveaux




NB : vu que nous générons le labyrinthe à chaque fois que l'on clique sur le bouton Play cela nous permet de générer une infinité de labyrinthes différents à chaque partie

**Facile** **Normal** **Difficile**




**En cas de victoire** (case d'arrivée atteinte) **En cas de perte** (temps écoulé)



## III. Pistes d'amélioration

- Améliorer le rendu graphique pour qu'il soit plus dynamique
- Améliorer l'affichage du chronomètre qui est actuellement en haut de la fenêtre
- Ajouter des sonorités
- Simplifier les classes qui ont été faites
- Développer un moyen d'afficher le chemin du labyrinthe en cas de perte du joueur
- Ajouter un mode d'échec : nombre de pas effectués

## IV. Conclusion

Malgré les difficultés rencontrées comme le manque de temps et le manque de connaissances graphiques, nous avons pu réaliser notre idée principale en ayant touché à pratiquement toutes les notions vues en cours. En plus de cela, nous avons réellement appris de nouvelles fonctionnalités du langage JAVA, c'est donc avec une grande satisfaction et joie que nous concluons ce projet.

# V. Bibliographie


- http://www-connex.lip6.fr/~guigue/wikihomepage/pmwiki.php?n=Course.CourseLI230
- https://data.brains-master.com/pdf/10601-apprenez-a-programmer-en-java.pdf
- http://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking
- https://en.wikipedia.org/wiki/Maze_generation_algorithm#Depth-first_search
- https://www.youtube.com/