

# Algo / Git

## Projet Backlog

labgen

# Générateur + Solver de labyrinthe

# Démo

# ✿ Libertés ✿

- Vous avez des idées de features ? n'hésitez pas à me les proposer
- Vous voulez implémenter d'autres algos ? go for it
- Vous voulez faire une interface graphique ? c'est possible (mais pas demandé dans le backlog)
- Vous n'êtes pas à l'aise en anglais ?
  - Le code en anglais (toujours)
  - Les commentaires en français

# ⛔ Contraintes ⛔

- Projet sur Git obligatoire
  - Si vous avez le moindre problème Git, **appelez-moi immédiatement**
  - Je vous aide + explique le sujet + on partage avec toute la classe si possible
- Utilisation d'un Board (Trello ou autre)

# ⛔ Contraintes ⛔

- Projet sur Git obligatoire
- Une partie de la note sera sur :
  - Qualité du code (nommage des variables, découpage des fonctions, etc)
  - Qualité de la doc

# ⛔ ChatGPT ⛔

- Utilisez le intelligemment, comme un *copilote*, pas comme votre *cerveau*
- Si au moment de la review vous êtes incapable d'expliquer votre morceau de code
  - 0/20 instantané sur la note individuelle (50% de la note finale)
  - Je n'applique pas cette règle bêtement, mais je me réserve le droit de le faire



# Story #01

**EN TANT QUE** Dev

**JE VEUX** Générer un labyrinthe aléatoire

**AFIN DE** préparer mon futur jeu rogue-like /  
FPS / Hack'n'Slash (rayer la mention inutile)

**DoD** Un labyrinthe aléatoirement généré  
avec entrée et sortie unique

## Info

- Plusieurs algos possibles (Recursive Backtrack, Krusal)
- Commencez par le plus simple : Recursive Backtrack
- Le labyrinthe est un `char[][]`, chaque case peut être :
  - Un mur : `'#'`
  - Un chemin : `' '`
  - Une entrée : `'E'`
  - Une sortie: `'S'`



Info



- Simplifiez au maximum pour démarrer :
  - Taille fixe du labyrinthe, ex 5x5
  - Taille impaire pour faciliter la vie de l'algo pour les murs
  - Entrée / Sortie toujours au même endroit (ex: entrée en haut a gauche, sortie en bas a droite)

# Story #02

**EN TANT QUE User**

**JE VEUX** Voir le labyrinthe généré

**AFIN DE** Voir ce qui m'attend et vérifier que  
le labyrinthe est solvable

**DoD** Labyrinthe dessiné en console quand on  
lance le programme

## Info

- Vous aurez besoin de plusieurs boucles (lignes / colonne)
- Regardez bien la différence entre `System.out.println()` et `System.out.print()`

# Story #03

**EN TANT QUE Dev**

**JE VEUX** Rendre le labyrinthe paramétrable

**AFIN DE** Adapter à plusieurs difficultés

**DoD** Largeur, hauteur, position entrée, sortie  
paramétrables



- Des variables, des variables, des variables
- Les dimensions sont toujours impaires

# Story #04

**EN TANT QUE User**

**JE VEUX** Configurer la génération en CLI

**AFIN DE** Rendre les paramètre interactifs au  
lancement du programme

**DoD** Le programme demande les différents  
paramètres au lancement avant de faire la  
génération, il gère les erreurs de saisie





- Vous aurez besoin de récupérer la saisie clavier avec la class

Scanner

```
Scanner scan = new Scanner(System.in);  
int number = scan.nextInt();
```

- Paramètres
  - largeur (ex: 5)
  - hauteur (ex: 5)
  - position entrée (ex: 0, 1 pour ligne 0, colonne 1)
  - position entrée (ex: 4, 4 pour ligne 4, colonne 4)

## Info

- Il faut gérer les erreurs et ne pas planter
- Si vous demandez un nombre et que l'utilisateur saisi autre chose, afficher une erreur claire et redemander
- Après avoir demandé tous les params afficher un récap puis :
  - Si l'utilisateur saisi **GEN** on génère et affiche le labyrinthe
  - Si l'utilisateur saisi **CONF** on recommence la saisie des paramètres
  - Si l'utilisateur saisi **QUIT** on QUIT le programme

# Story #05

**EN TANT QUE Dev**

**JE VEUX** Résoudre le labyrinthe  
automatiquement


**AFIN DE** Vérifier qu'il est valide sans épuiser  
un humain à le faire manuellement

**DoD** Après génération, labyrinthe  
automatique résolu et affiché

## Info

- Plusieurs algos possibles (Depth First Search, A\*)
- Commencez par DFS qui est beaucoup plus simple
- Une fonction qui prend en paramètre le labyrinthe généré et va le parcourir :
  - Commence toujours à l'entrée
  - Ne peux pas traverser les murs
  - Chaque chemin visité est marqué par un `v`
  - S'il arrive à la sortie, renvoie `true`, sinon `false`

## Info

- Afficher le labyrinthe résultant avec les 
- Plus le labyrinthe est grand, moins il aura de chance de visiter toutes les cases

# Story #06

**EN TANT QUE User**

**JE VEUX** Choisir si le solver est appelé ou non

**AFIN DE** Pouvoir se garder la surprise >\_>

**DoD** Une option dans les paramètres console  
pour appeler le solver ou non

## Info

- N'hésitez pas à mettre des valeurs par défaut pour les saisies, ça vous fera gagner du temps pour debug
- Ex :
  - Largeur du labyrinthe (5) ?
  - Résoudre le labyrinthe (Y) ?
- Prevoyez un mode "non-interactive" qui saisi automatiquement tous les paramètres sans poser de questions à l'utilisateur
- Ca vous fera gagner *beaucoup* de temps de debug

# Story #07

**EN TANT QUE User**

**JE VEUX** Enregistrer le labyrinthe dans un fichier

**AFIN DE** Pouvoir le réutiliser plus tard

**DoD** Un fichier lab01.labgen est généré avec le labyrinthe



# Info

- ⚠️ Ajoutez `.labgen` dans le `.gitignore`
- Les fichiers se créent automatiquement à la suite (existant non écrasé) :
- `lab01.labgen`
- `lab02.labgen`
- `lab03.labgen`
- C'est un fichier, techniquement vous écrivez ce que vous voulez dedans
- Le but est de pouvoir le lire par la suite, le plus serait d'écrire comme à l'affichage

# Story #08

**EN TANT QUE User**

**JE VEUX** Lancer le solver sur un fichier généré

**AFIN DE** Vérifier massivement des  
labyrinthes

**DoD** Un `MainSolve.java` qui ouvre un fichier  
labgen, le lit et lance le solver

## Info

- De la même façon que vous avez *écrit* le fichier, il faut maintenant le *lire*
- Une fonction `static char[][] readLabFile(String filepath)`
  - Elle lit le fichier passé en paramètre, et renvoie ensuite le labyrinthe dans votre format

# Story #09

**EN TANT QUE Dev**

**JE VEUX** Afficher la complexité de  
génération de Recursive Backtracking

**AFIN DE** Connaitre le "coût" de l'algo

**DoD** Après la génération, afficher ces stats

## Info

- Complexité en "temps"
- Combien de fois l'algo visite une case
- Complexité en "espace"
- Combien de fois la fonction se rappelle elle même

# Story #10

**EN TANT QUE** Dev

**JE VEUX** Afficher la complexité du solver DFS

**AFIN DE** Connaitre le "coût" de l'algo

**DoD** Après la génération, afficher ces stats

## Info

- Complexité en "temps"
- Combien de fois l'algo visite une case
- Complexité en "espace"
- Combien de fois la fonction se rappelle elle même

# Story #11

**EN TANT QUE Dev**

**JE VEUX** Se déplacer dans le labyrinthe

**AFIN DE** Sortir !

**DoD** on peut jouer et sortir du labyrinthe



# Story #B01

**EN TANT QUE User**

**JE VEUX** Afficher un beau labyrinthe



**AFIN DE** Faire une CLI stylé

**DoD** Pimper l'affichage, mettre des couleurs,  
emojis, etc

## Info

- Vous pouvez utiliser les codes couleurs ANSI
  - Attention, pas compatible avec Windows sur un terminal classique
- Vous pouvez utiliser la lib [Jansi](#) qui est compatible Windows
  - Téléchargeable directement depuis IntelliJ (`org.fusesource.jansi:jansi:2.4.1`)
- Si vous trouvez une autre lib qui est cool, dites-moi !

## Info

- L'utilisateur est symbolisé par un 
- A chaque fois que l'utilisateur appuie sur W,A,S,D
  - Vérifier si le déplacement est autorisé, si non afficher ""Déplacement interdit""
  - Si oui, déplacer  à la case
  - Afficher le nouvel état du labyrinthe

# Story #B02

**EN TANT QUE** Dev

**JE VEUX** Générer un labyrinthe aléatoire  
avec l'algo Krusal

**AFIN DE** Comparer les résultats (perfs +  
motif) avec Recursive Backtracking

**DoD** Le labyrinthe est généré avec l'algo  
Krusal

## Info

- Attention, cet algo est beaucoup plus complexe à implémenter, **ne le faites QUE si le 1er algo est déjà fonctionnel**
- Si votre code est bien organisé, normalement il y a juste 1 appel de fonction à changer

# Story #B03

**EN TANT QUE** User

**JE VEUX** Résoudre le labyrinthe avec l'algo A\*

**AFIN DE** Comparer les résultats (perfs + motif) avec DFS

**DoD** Le labyrinthe est résolu avec l'algo A\*



- **Ne le faites QUE si le 1er algo est déjà fonctionnel**
- Si votre code est bien organisé, normalement il y a juste 1 appel de fonction à changer
- Découper bien le sujet :
  - D'abord calculer la distance de Manhattan, vérifier qu'elle est bonne
  - ENSUITE seulement, vous implémentez l'algo  $A^*$