# Rapport du groupe SB-2-GC

## Sommaire :



## 1. Intro, présentation du sujet

Présentation du ReadMe. (Voir fichier sur git)

Présentation du sujet de base. (Voir fichier sur git)

## 2. Présentation des outils (LIUM, Whisper)

État de l'art de LIUM(voir fichier sur Git ).

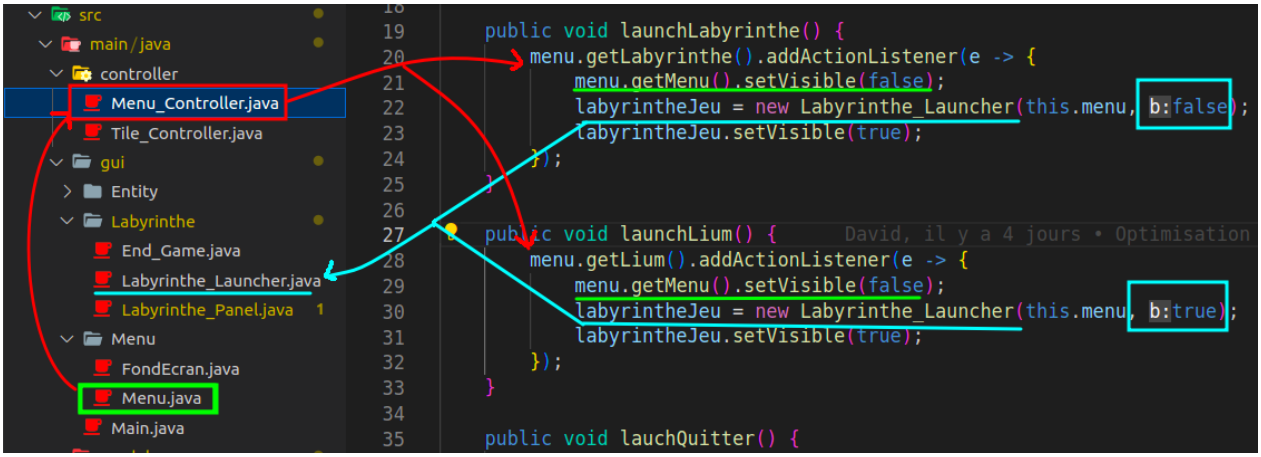
État de l'art de Whisper(voir fichier sur Git ).

## 3. Lancement du jeu, écran d'accueil

### - Interface du jeu sur l'accueil :

Au commencement du jeu, la première fenêtre JFrame lancée par le main est la classe Menu. Celle-ci est simplement constituée d'un fond visuel qui rappel de thème du jeu et de 3 JButtons indiquant les 3 possibilités offertes au joueur. Un bouton exit permet de quitter le jeu et de fermer la fenêtre et les deux autres boutons déterminent le choix du mode de jeu que l'on souhaite lancer. Le fonctionnement de ces boutons est géré à l'aide d'action listener par la classe Menu_Controller, qu'on active l'un ou l'autre des boutons le programme qui suivra sera le même à un détail prêt, dans les 2 cas le programme fermera la fenêtre Menu et appellera la seule autre classe qui extends JFrame du projet à savoir Labyrinthe_Launcher,

mais en prenant en argument un booléen modeJeu dont la valeur déterminera lequel des modes de jeu est lancé. Un fois arrivé sur la fenêtre Labyrinthe_Launcher, il suffit d'appuyer sur échap pour revenir au menu.




# 4. principe du jeu

## -mode Lium :

L'utilisateur est sur une carte en ligne droite et doit appuyer sur espace pour activer le micro pendant une durée définie. Chaque personne doit parler séparément pendant 20 secondes (un timer est affiché en bas de l'écran). Une fois l'enregistrement terminé un fichier wav sera créé de la durée définie et sera fusionné à un autre audio servant de base qui permet d'entraîner LIUM afin d'obtenir un résultat plus précis. Après ça, LIUM se lance et réalisé une segmentation de l'audio indiquant à quel moment le locuteur parle, pendant combien de temps, son genre...

Un exemple :

```
;; cluster:S0 [ score:FS = -33.11774629783916 ] [ score:FT = -33.91721107604137 ] [ score:MS = -33.463377418823306 ] [ score:MT = -34.178170192108895 ]
test 1 60 749 F S U S0
test 1 879 1467 F S U S0
test 1 4809 503 F S U S0
;; cluster:S11 [ score:FS = -32.02331219743305 ] [ score:FT = -33.043068413209355 ] [ score:MS = -33.64992806854904 ] [ score:MT = -34.027734795283855 ]
test 1 5312 621 F S U S11
test 1 5933 1573 F S U S11
test 1 7506 842 F S U S11
test 1 8352 1319 F S U S11
test 1 9728 683 F S U S11
test 1 10413 1635 F S U S11
test 1 12048 1305 F S U S11
test 1 13353 696 F S U S11
test 1 14049 1366 F S U S11
test 1 17122 1578 F S U S11
;; cluster:S22 [ score:FS = -32.23371507159226 ] [ score:FT = -33.34550324897421 ] [ score:MS = -33.709148581591094 ] [ score:MT = -34.20012627980649 ]
test 1 15415 1707 F S U S22
;; cluster:S27 [ score:FS = -32.69725434547528 ] [ score:FT = -33.141849157168316 ] [ score:MS = -31.40049573519943 ] [ score:MT = -32.296428841818425 ]
test 1 18712 1240 M S U S27
test 1 19952 1725 M S U S27
test 1 21677 1212 M S U S27
test 1 22889 1929 M S U S27
test 1 24818 1835 M S U S27
test 1 26653 224 M S U S27
test 1 26877 819 M S U S27
test 1 27696 1531 M S U S27
test 1 29227 1514 M S U S27
test 1 30741 1233 M S U S27
test 1 31974 1118 M S U S27
;; cluster:S3 [ score:FS = -32.687516989034464 ] [ score:FT = -33.69741444716024 ] [ score:MS = -33.4382800027181 ] [ score:MT = -34.28049395091387 ]
test 1 2346 339 F S U S3
test 1 2685 932 F S U S3
test 1 3617 1192 F S U S3
```

Le résultat est renvoyé dans un fichier txt qui sera lu et analysé par plusieurs fonctions. Celles-ci renverront le nombre de locuteurs ainsi que le nombre d'hommes et de femmes reconnus.

En ce qui concerne les règles du jeu, le nombre de pas du personnage est défini en fonction du genre : si la voix reconnue est celle d'une femme, le personnage se déplacera de deux cases. Si la voix reconnue est celle d'un homme, le personnage se déplacera d'une seule case.



## -Regles de jeu whisper :

Quand ce mode-ci se lance, l'utilisateur est envoyé dans une carte créée aléatoirement (dont on expliquera le développement plus tard) avec des règles de jeu à respecter qui sont les suivantes :

-But : Aller d'un portail à l'autre du labyrinthe.

-Étape 1 : Pour donner la direction, lancer le micro, dire une action avec le mot (gauche, droite, haut ou bas) compris dedans puis couper le micro puis lancer la touche J.

(Attention : ne pas changer de direction avec le mode debug en plein déplacement)

-Étape 2 : Pour obtenir le nombre de pas à effectuer, parler (de préférence) 20 secondes par personnes puis lancer la touche K.

-Étape 2Bis : Il arrive que LIUM commette des erreurs, dans ce cas refaites étape 2.

-Étape 3 : Appuyer sur L pour faire avancer le personnage, s'il tombe sur un mur, refaite l'étape 1, puis le personnage continuera.

-Étape 4 : Utiliser O pour récupérer le nombre de pas déjà généré par LIUM puis refaites l'étape 3.

Et voici les différentes touches pour que l'utilisateur puisse interagir avec le jeu :

-Espace : ouvrir fermer le micro

-J : Lancement de Whisper

-K : Lancement de LIUM

-L : Faire avancer le personnage du nombre de pas enregistré

-U : Active les touches ZQSD (mode débug)

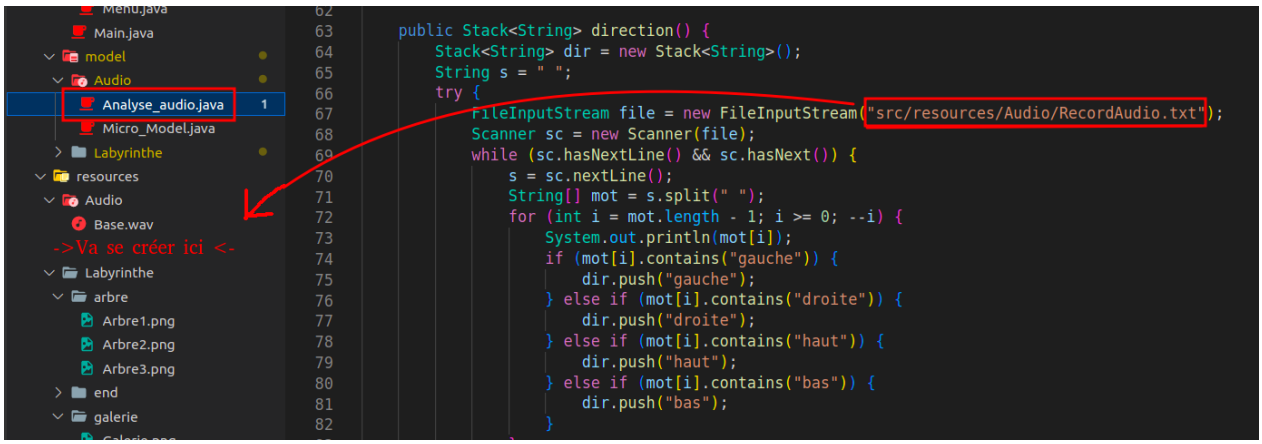
-I : Ajoute un pas dans le compteur générale (mode débug)

-O : Permet de réutiliser la valeur qui a été déterminer par LIUM

-Echap : retour au menu principal

## -Fonctionnement whisper :

Le micro se lance, afin de donner une direction au personnage il faut utiliser les indications « gauche », « droite », « haut » ou « bas » en fonction d'où on veut se déplacer. (Il est préférable d'utiliser les mots dans une phrase comme : « Je veux aller à gauche » pour que ça soit plus simple pour Whisper pour reconnaître. Ensuite Whisper se lance et renvoi un fichier txt de la retranscription de ce qui a été dit, une fonction va lire dans ce fichier afin de vérifier l'existence d'une direction. Si celle si existe bien, le programme transfert la direction au personnage qui s'oriente en fonction.



# 5. Interface du jeu.

## - Création du JPanel du jeu :

La fenêtre de jeu Labyrinthe_Launcher.java extends Fframe est segmentée en borderLayout(). 4 JPanels utilitaires ont été ajoutés autour du BorderLayout.CENTER. Leur background est configuré avec la même couleur que les arbres du Labyrinthe pour s'y accorder et constituer leur bordure. Ils contiennent chacun des informations de jeu importantes (à l'exception du bottom). Le JPanel add à borderLayout.NORTH affiche les règles du jeu, celui de borderLayoutWest affiche la direction dans laquelle pointe le personnage et celui de borderLayoutEast correspond au nombre de pas de déplacement disponible. Dans la première version du programme les déplacements du personnage s'effectuaient « de case en case » sans transition, à la suite d'une amélioration du code, le personnage se déplace de façon continue sur l'ensemble du chemin, le nombre de pas disponible correspond donc désormais à une distance de parcours autorisée et non plus à un nombre de case limité.

Le cœur du jeu se trouve dans le dans le borderLayout .CENTER dans lequel on add un Objet de la classe Labyrinthe_Panel extends JPanel implements Runnable. C'est là, au centre de l'écran que l'affiche en continu et en mouvement du Labyrinthe et du personnage a lieu.

## - Génération du personnage et de son fonctionnement :

Comme l'import de l'image et l'esthétique du personnage se font dans ce Labyrinthe_Panel. Je vais vous expliquer la mécanique qui se trouve derrière le personnage, tout d'abord le personnage a une hitbox de forme rectangle, qui est de proportion égale aux tuiles du jeu, ceci permet de positionner le personnage toujours au centre de chaque case avec une marge par rapport aux murs.

La décision sur l'orientation du personnage se fait justement ici, si nous utilisons les touches du clavier, il réagira et s'adaptera grâce à un switch, de même avec « Whisper » qui renvoie vers le personnage le mot clé et ensuite qui passera par un swich. Ainsi la prise en charge des collisions se fait aussi ici, la vérification se fait à chaque tentative de réorientation et de déplacement.

Le nombre de case à déplacer se fait par rapport aux résultats de « LIUM », il le calcule comme précisé dans la consigne : 1 case pour un homme et 2 pour une femme, le calcul est toujours adapté aux proportions de la taille des tuiles.

À la suite de ces traitements, pour éviter de devoir lancer « LIUM » à chaque fois pour le faire avancer, il pré-enregistre et met à disposition des valeurs de sauvegardes, ce sont les « steps » que vous voyez à droite de l'écran du jeu, il s'actualise à toute actions.


## - Génération de la carte par lecture de texte :

Penchons-nous sur à présent sur la matérialisation du plateau. Les tableaux sont des fichiers .txt de 38 colonnes et 22 lignes composé de 0,1,2,3,4,5 et de 6. Ces numéros déterminent le type de tile auquel correspondra cette case du tableau, et donc le type d'image qui sera affiché à cet endroit de l'écran, puisque les tiles sont des objets constitués d'une BufferedImage et d'un boolean collision. Par la fonction getTileImage() du fichier Tile_controller.java on va pouvoir faire un tableau de tile qui va répertorier toutes les tiles possible en fonction des valeurs donc 7 tiles différents. Celles qu'on ne voit pas dans l'image ci-dessous correspondent aux illustrations de portail qui permettent de passer d'une carte à une autre.


Comme nous pouvons le voir juste au-dessus chaque tile va être créée, une image situé dans src/ressources/labyrinthe/ va être leur être attribuée et initialisée le booléen collision sera initialisé à true sauf pour Tile[1]. Le joueur se déplaceront donc sur les tyles de type 1 qui correspondent aux chemins alors que les autres valeurs permettront d'implémenter les arbes autour du personnage et les portails de

début et fin de niveau. En effet le 0 représente le haut des arbres, les 1 sont le chemin ou le joueur peut passer, le 2 le bas de l'arbre, le 3 le tronc de l'arbre, le 4 le haut du portail, le 5 le milieu du portail et le 6 le bas du portail.

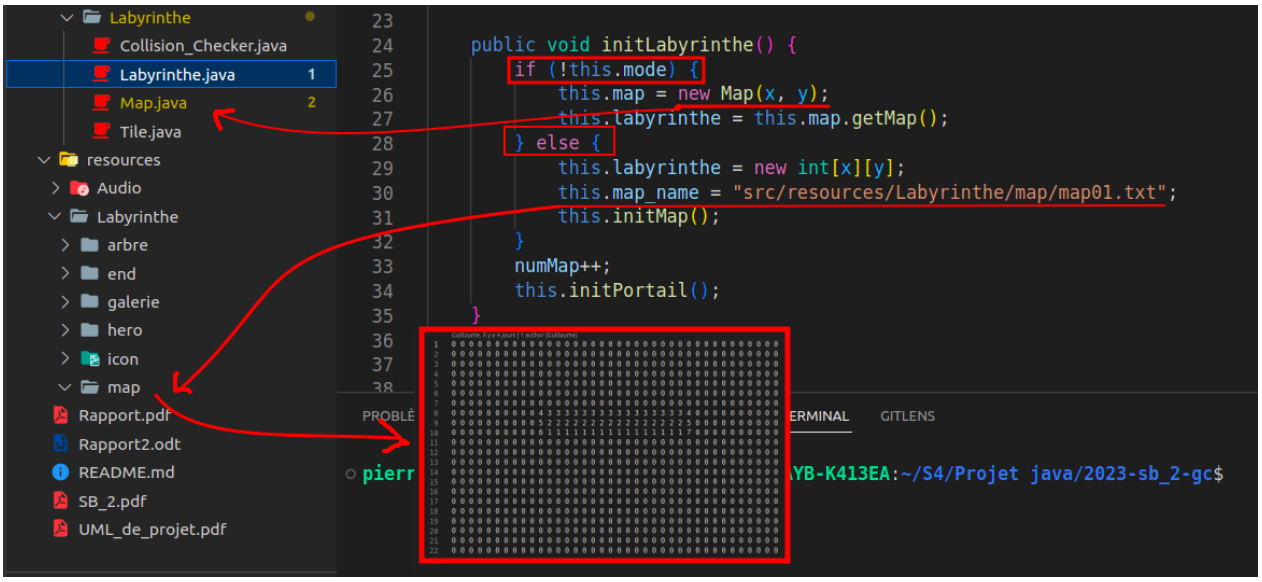
## -collision de la carte et du héros :

Maintenant qu'on a implémenté le labyrinthe et le héros il faut s'assurer que les collisions fonctionnent et on va s'assurer de ça a l'aide du fichier Collision_checker.java. Ce fichier va nous permettre de vérifier que les dimensions du héros (sa hitbox) ne rentrent jamais dans un tiles ou il n'est pas sensé aller. Elle va faire cela a l'aide de la fonction checkTile(entity e) qui va voir en fonction de la direction que prend le héros c'est à dire up,down,leftt et right et agir en conséquence dans le cas où il rentrerait en collision avec un objet devant le stopper.

```
switch (e.direction){
    case "up":
        topRow = (topY - e.speed)/lp.getTileSize();
        tileNum1 = lp.getTileController().getLabyrinthe().getVal(topRow, leftCol);
        tileNum2 = lp.getTileController().getLabyrinthe().getVal(topRow, rightCol-1);
        if (lp.getTileController().getTile()[tileNum1].collision || lp.getTileController().getTile()[tileNu
            e.collisionOn = true;
        }
        break;
    case "down":
        bottomRow = (bottomY + e.speed)/lp.getTileSize();
        tileNum1 = lp.getTileController().getLabyrinthe().getVal(bottomRow-1, leftCol);
        tileNum2 = lp.getTileController().getLabyrinthe().getVal(bottomRow-1, rightCol-1);
        if (lp.getTileController().getTile()[tileNum1].collision || lp.getTileController().getTile()[tileNu
            e.collisionOn = true;
        }
        break;
    case "left":
        leftCol = (leftX - e.speed)/lp.getTileSize();
        tileNum1 = lp.getTileController().getLabyrinthe().getVal(topRow, leftCol);
        tileNum2 = lp.getTileController().getLabyrinthe().getVal(bottomRow-1, leftCol);
        if ((lp.getTileController().getTile()[tileNum1].collision || lp.getTileController().getTile()[tileN
            e.collisionOn = true;
        }
        break;
    case "right":
        rightCol = (rightX + e.speed)/lp.getTileSize();
```

## -Génération aléatoire de la carte :

Alors que dans le mode Lium, la map est générée en faisant appel à un fichier.txt déjà déterminé dans les fichiers du jeu (en l'occurence une ligne droite), dans le mode complet Layrinthe, la classe Labyrinthe fait appel à une classe map qui va initialiser une nouvelle carte aléatoirement à chaque partie.




Dans la classe Map, l'attribut map (un tableau de tableau de int) qui va déterminer la nouvelle carte va être initialisé de façon aléatoire à l'aide de la fonction genererLabyrinthe(int longueur, int largeur) qui se base sur le « backtracking », c'est à dire le fait de revenir en arrière dans le code lorsqu'on rencontre un blocage. Succinctement, celle-ci fonctionne de la façon suivante :

Après avoir défini les dimensions du labyrinthes avec les deux attribut de la fonction, puis choisi aléatoirement le point de départ et le point d'arrivée du labyrinthe, le programme établi un chemin en stockant la case courante(là où en est le chemin) dans une pile, puis ses cases dites « voisines »(à deux cases d'elles et dans les limites du terrain). Une de ces cases voisines est choisie aléatoirement comme nouvelle case courante et la case intermédiaire entre l'ancienne et la nouvelle case courante est également convertie en chemin. Lorsque l'algorithme atteint une situation où aucune case voisine non visitée n'est disponible, cela signifie qu'il s'est retrouvé dans une impasse. À ce stade, il retire la case courante de la pile et revient à la case précédente pour explorer d'autres chemins possibles.En revenant en arrière, l'algorithme continue à retirer des cases de la pile jusqu'à ce qu'il atteigne une case qui a encore des cases voisines non visitées. À partir de cette case, il choisit à nouveau une case voisine aléatoire et poursuit le processus d'exploration.

## 6. Conclusion, présentation graphique des appels de fonctions

- Présentation d'UML du jeu : pour une meilleur compréhension du déroulement de notre jeu au niveau du code c'est à dire quel fichier est appelé à quel moment nous avons construit un UML (Unified Modeling Language).