rapport du groupe SB-2-GC

patricia fournier

Robic Matthieu

Yu David

Leconte Yohan

Guillaume (désolé je m'en souviens plus il est 3h aie pitié)

- déroulement du jeu :

l'utilisateur est en premier lieu devant le menu qui lui propose 2 choix par l'intermédiaire de deux bouttons sur un fond d'écran rappelant le théme du jeu.sur le premier bouton situé au centre, il y'a marqué lium et sur le second il est écrit labyrinthe. En fonction du bouton utilisé par l'utilisateur nous aurons deux versions différentes du jeu qui se lanceront. En effet c'est le menu_controller du code qui va réagir a l'aide d'action listener et qui va fermé le Jframe du Menu précédent et lancer les différents mode de jeux avec labyrinthe_launcher. Si l'utilisateur a appuyé sur le bouton lium du menu alors menu_controlleur créera le labyrinthe_launcher avec le boolean false ce qui donnera la version du jeux ou le personnage se meut par la voix de l'utilisateur grâce a la librairie Lium et whisper. Et quand le bouton labyrinthe est utilisé alors le labyrinthe_launcher est créé avec le boolean true et on peut le déplacer par les touches "Z" pour aller vers le haut, "Q" pour aller vers la gauche, "D" pour aller vers la droite et le "S" pour aller vers le bas. Le niveau est fait de maniére a ce le joueur se déplace jusqu'a la fin ou il y a un portail qui émménera le joueur dans la deuxiéme map prévu.

le mode de jeu avec lium lance les même labyrinthe mais le personnage se déplace avec la voix de l'utilisateur a droite, a gauche, en haut ou a droite grâce a la librairie whisper et il reconnait le nombre de personne qui parle et se déplace ou moins en fonction du genre de l'utilisateur.

-Interface du jeu :

A présent intéréssons nous plus en détail a l'interface du jeux. Labyritneh_launcher.java extend jframe et est segmenté en borderLayout(), nous avons mis 4 jpanels de sur lequel on a configuré leur background afin de les mettrent de la même couleurs que les arbres du labyrinthe pour les bordures et utiliser la méthode add pour les rajouter borderLayout.NORTH,borderLayout.SOUTH,borderLayout.EAST,borderLayout.WEST!

le BorderLayout.CENTER est prévue pour le labyrinthe_Panel qui extends un JPanel et implemente Runnable.

Penchons nous sur a 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 seront interprétés de maniére a faire la map en fonction d'eux. Une tile leur sera attribué en fonction de leur valeur. Les tiles sont des objets constitués d'une BufferedImage et d'un boolean collsion.Par la fonction getTileImage() du fichier Tile_controller.java on va pouvoir faire un tableau de tile qui va répertorié toutes les tiles possible en fonction des valeurs donc 7 tiles différents.

//screen des 7 images .png et screen de la fonction getTileImage() pour iullustrer propos

Comme nous pouvons le voir juste au dessus chaque tiles va être crée, leur attribuer une image situé dans src/ressources/labyrinthe/ et initialisé le boolean collision a true sauf Tile[1]. Le joueur se déplaceront donc sur les 1 des fichiers map.txt 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 la galleries 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.

//partie sur l'implémentation du player dans labyrinthe_panel

maintenant qu'on 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 vérifié que les dimensions du héros (ça hitbox) ne rentre 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 héro c'est a dire up,dwon,leftt et right et agir en conséquence dans le cas pou il rentre en collision avec un objet en stoppant le héro.

//screen du switch(e.direction){ de la fonction checkTile du fichier collision_checker.java

//parler des portails si estimés nécéssaires

//parler de runnable et thread aussi

-implémentation de Lium et whisper:

//screen des arguments de lium au niveau du process builder quand on parle

process builder dans le texte ci dessous (je savais pas comment l'ammener.

Pour commencer voyons comment lium sont sensé marcher! Quand le mode de jeu lium est lancé, l'utilisateur doit appuyer sur espace ce qui va activer le micro qui va rester activé pendant une durée définie. A la fin de ce délai un durée .wav sera créer de la durée définie et ce fichier .wav sera fusionner par la fonction (je sais pas encore je dois la chercher) pour entrainer lium pour une précision plus précise.Ce fichier audio sera alors utiliser dans un process builder (détailler plus).Lium va ainsi créer un fichier .txt qu'on appelle cluster qui va montré les résultats de lium en montrant les différentes signatures sonores de chaque personne qui parle, a quel moment il parle et pendant quel durée (une segmentation audio de chaque locuteur)

// screen d'exemple de cluster avec fléches sur les données et se qu'on comprend.

j'utilise ensuite les méthodes nbrLocuteur() et nbrHommesFemmes() du fichier Analyse_audio.java qui va utilisé un scanner poour copmpter. dans le cas du nombres locuteur il suffit de regarder chaque ligne terminant par ";"

//screen de la fonction nbrLocuteur()

dans le cas de la fonction nbrHommeFemme() nous allons renvoyer un tableau de taille 2 dans laquelle la case 0 représente le nombres d'hommes et la case 1 le nombre de femme. On utilise ensuite aussi un scanner qui pour chaque ligne contenant ";" va regarder la ligne suivante et additionnera 1 au case correspondante du tableau en fonction de si le scanner trouve "F" ou "M" qui correspond au genre du locuteur.

//screen de la fonction nbrHommeFemme

Dans le cas de whisper, celui ci va aussi faire un fichier .txt mais juste réécrire ce qu'il a compris dans ce fichier. On peut ainsi utiliser un scanner et utiliser ce fichier pour comprendre ce que dit l'utilisateur et faire avancer notre personnage.Cela sera fait par la fonction direction() qui va renvoyer un string du mot utilisé par l'utilisateur si c'est bien "gauche","droite","haut" ou "bas".

Pour que lium soit plus efficace on a réglé les paramètres du fichier .wav en 16000 hz,mono

//screen de ces paramètrages

//screen de la méthode direction d'analysse_audio.java

//partie sur les paramétres vraiment important de lium et son paramétrage