Ideas

Build du projet

Gradle ou **Maven** ?
Sachant que:

- On maîtrise plus la compilation avec Maven
- Il y a un projet gradle tout près sur moodle (Mais qui compile pas sur mon pc pour l'instant). Avec des imports pour **JavaFX**

Version du projet:

Java 17 ? Pour appliquer les nouvelles features du cours ?

GUI

Java Swing ou **JavaFX** ?
Remarque: Peut être plus simple sur **JavaFX** ? Et le projet gradle contient déjà les imports pour **JavaFX**. *A réfléchir*.

Classes

Voilà une idée de schéma de classes pour le projet. Bien sûr on peut faire totalement autrement, c'est juste pour avoir une première idée.

GUI:

Window.java (Si JavaSwing. Peut etre pas besoin si JavaFX)

____**GameView.java**: Attribut -> Game, WordListView

_____**WordListView.java**: Attribut -> WordList; Permet d'afficher les mots a l'écran. Hérite de JTextArea ou autre.

_____**WordView.java** ? Attributs -> Word; Fonctions -> changeColor() (Rachaichit la couleur du mot selon l'index: vert a gauche, rouge a droite par exemple si erreur)

Model:

Game.java: Attributs -> *enum type = {Normal, Jeu, Multiplayer} (pas sûr, voir la suite du pdf)*, WordList, int nbWords, int speed, int life, int level
____**WordList.java**: extends ArrayList<String ou Word ?>

- hérite de ArrayList ou alors contient une ArrayList. Ou alors une Stack<> (Pile) ? Et donc pas ArrayList
- Une arrayList de Word (Classe crée par nous) ou directement de String ?
- Des fonctions pour lire dans des fichiers et générer une WordList. Aléatoire ou pas.

_____**Word.java** ? : Attribut -> int curseur (position sur le mot. Permet aussi de vérifier si le mot a été à moitié remplie), static Color default , error, right; String mot; Color color; Fonctions -> compareWord(String str) (Compare str avec mot et définie la distance d'erreur ?)

____**NormalMode, GameMode, MultiplayerMode** extends **Game**
- Soit on utilise un enum type=NormalMode, GameMode, mutiplayermode dans Game.
- Soit on utilise ces trois classes qui hérite de Game pour faire un peu plus d'héritage.

____**MutliplayerMode** extends Game

- Communique en réseau. Peut être une utilisation des Thread. A voir comment on fait communiquer les joueurs.
- Pas de notion de niveau donc on se s'occupe pas du **int level** hérité de **Game**

Remarques

On aura surement besoin de plus de classes. Mais voilà un premier schéma. Une première vision du projet.

Attention aussi à respecter **l'encapsulation** pendant le projet.

- Renvoyer un objet.clone() dans les getters.
- Faire un objet.clone() dans les constructeurs pour initialiser les attributs.
- Utiliser **private** le plus possible.
- Utiliser **sealed** ou final sur les classes qui n'ont pas besoin d'être hérité.