**Documentation Technique**

Dans ce document, vous trouverez les différentes méthodes et classes réalisé pour le fonctionnant des exercices.

1. **Partie Web**
2. **Exercice `Type-Script`**

**Les variables globales** (ces dernières ne seront pas représenté car elles sont les mêmes pour tous les exercices) :

/\*\* @type {boolean} \*/

let hasExerciceStarted = false; // L'exercice a commencé ?

/\*\* @type {boolean} \*/

let hasExerciceEnded = false; // L'exercice est terminé ?

/\*\* @type {Date} \*/

let tempsDebut = new Date(); // Date de début de l'exercice

/\*\* @type {boolean} \*/

let isFetching = false; // Est-ce qu'on est en train de récupérer une phrase depuis l'API python ?

/\*\* @type {string} \*/

let langue = 'fr'; // Langue de la phrase à récupérer

**Variables relatives à l’exercice :**

/\*\* @type {boolean} \*/

let caseSensitive = true; // Majuscules sensibles ?

/\*\* @type {boolean} \*/

let accentSensitive = true; // Accents sensibles ?

/\*\* @type {number} \*/

let nbreMotsTapes = 0; // Nombre de mots tapés

/\*\* @type {number} \*/

let nbreMotsTotal = 0; // Nombre de mots total

/\*\* @type {Array<string>} \*/

$: letters = sentences.split(''); // La phrase à taper, séparée en lettres

/\*\* @type {string} \*/

const ESPACE = ' ⸱ '; // Caractère de remplacement pour les espaces

/\*\* @type {string}\*/

let sentences = ''; // La phrase à taper

/\*\* @type {string} \*/

let source = ''; // La source de la phrase

// Résultats de l'exercice calculé par l’API Python

/\*\* @type {{

 \* tempsMisString: string,

 \* nbErreurs: number,

 \* nbCaracteres: number,

 \* vitesse: number,

 \* precision: number,

 \* score: number}} \*/

let resultats = {}

**Méthodes relatives au fonctionnement de l’exercice :**

/\*\*

 \* Appelé lorsqu'une touche est appuyée

\* Ici est vérifié si la lettre tapé par l’utilisateur correspond bien à la lettre attendue.

 \* @param {KeyboardEvent} event

 \*/

function keyDown(event) {

}

/\*\*

 \* Appelé lorsque l'utilisateur a terminé de taper les phrases.

\* Envois les résultats à Python pour récupérer le score de l’

 \* @returns {void}

 \*/

function handleSentencesCompleted() {

}

/\*\*

 \* Démarre l'exercice

 \* Récupère une phrase aléatoire depuis l'API python

 \* @returns {Promise<void>}

 \*/

async function startExercice() {

}

1. **Exercice `Sténographie`**

**Variables relatives à l’exercice :**

/\*\*

 \* Les audios des phrases ainsi que les phrases au format [(texte, audio file path), ...]

 \* @type {Array<[string, string]>}

 \*/

let audios = [];

/\*\* @type {string} \*/

let tapedSentence = ''; // La phrase tapée par l'utilisateur

/\*\* @type {number | null} \*/

let similitude = null; // Le pourcentage de similitude entre la phrase tapée et la phrase de l'audio

/\*\* @type {boolean} \*/

let checkMaj = false; // Autoriser les majuscules dans les chaines de caractères ?

/\*\* @type {boolean} \*/

let checkOrthographe = true; // Autoriser les accents dans les chaines de caractères ?

/\*\* @type {boolean} \*/

let checkPonctuations = false; // Autoriser les caractères spéciaux dans les chaines de caractères ?

**Méthodes relatives au fonctionnement de l’exercice :**

/\*\*

 \* Appelée lorsqu'une touche est appuyée

 \* Si la touche est ENTRÉE et que l'exercice n'a pas encore commencé, démarre l'exercice

 \* @param {KeyboardEvent} event

 \*/

function keyDown(event) {

}

/\*\*

 \* Démarre l'exercice

 \* Récupère depuis l'API python les phrases et leurs audios

 \*/

async function startExercice() {

}

/\*\*

 \* Joue l'audio suivant à retrancrire

 \* Si il n'y a plus d'audios, récupère de nouvelles phrases depuis l'API python

 \*/

function playNextAudio() {

}

/\*\*

 \* Appelée lorsqu'on clique sur le bouton de soumission

 \* Envoye la phrase tapée par l'utilisateur à l'API python pour vérifier si elle est correcte en fonction des options choisies

 \*/

async function handleSubmissionButtonClicked() {

}

1. **Exercice `Rythme`**

**Variables relatives à l’exercice :**

/\*\* @type {Array<import('$lib/classes/Niveau').Niveau>} \*/

let niveaux = []; // Les différents niveaux récupérés depuis python

/\*\* @type {import('$lib/classes/Niveau').Niveau} \*/

let selectedLevelObj; // Le niveau sélectionné

/\*\* @type {number} \*/

let audioPosition = 0; // La position actuelle de l'audio

/\*\* @type {number} \*/

let score = 0; // Le score actuel

/\*\* @type {number} \*/

let scoreMax = 0; // Le score maximum possible

/\*\* @type {Array<string>} \*/

let keys = []; // Les lettres dans l'ordre du clavier

/\*\* @type {Array<any>} \*/

let keysScored = []; // Les lettres déjà scorés

/\*\* @type {Array<string>} \*/

let keyboardLayout = ['a', 'z', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', 'q', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 'm', 'w', 'x', 'c', 'v', 'b', 'n']; // Le layout du clavier pour trier les touches dans l'ordre

/\*\* @type {Array<string>} \*/

let touchesAppuyees = []; // Les touches actuellement appuyées

**Méthodes relatives au fonctionnement de l’exercice :**

/\*\*

 \* Appelée lorsque la fenêtre est chargée

 \* Récupère les niveaux depuis python

\*/

onMount(async () => {

}

/\*\*

 \* Appelée lorsqu'une touche est appuyée

 \* Si la touche est ENTRÉE et que l'exercice n'a pas encore commencé, démarre l'exercice

 \* Sinon, vérifie si la touche appuyée est dans les touches à appuyer pour le temps actuel

 \* @param {KeyboardEvent} event

 \*/

function keyDown(event) {

}

/\*\*

 \* Démarre l'exercice

 \* Récupère le niveau sélectionné et démarre l'audio

 \*/

function startExercice() {

}

1. **Exercice `Réaction`**

**Variables relatives à l’exercice :**

/\*\* @type {number} \*/

let nombreDeReactions = 6; // Le nombre de réactions à afficher

/\*\* @type {number} \*/

let nombreDeReactions = 6; // Le nombre de réactions à afficher

/\*\* @type {boolean} \*/

let hasExerciceEnded = false; // L'exercice est terminé ?

/\*\* @type {boolean} \*/

let allowUppercase = true; // Autoriser les majuscules dans les chaines de caractères ?

/\*\* @type {boolean} \*/

let allowAccents = true; // Autoriser les accents dans les chaines de caractères ?

/\*\* @type {boolean} \*/

let allowSpecialCharacters = false; // Autoriser les caractères spéciaux dans les chaines de caractères ?

/\*\* @type {string}\*/

let reaction = ''; // La chaine de caractères à écrire

/\*\* @type {number} \*/

let tempsDebut; // Temps auquel la réaction a été affichée (pour calculer le temps de réaction de l'utilisateur)

/\*\* @type {string} \*/

let typedReaction = ''; // La chaine de caractères tapée par l'utilisateur

/\*\* @type {Array<Array<any>>} \*/

let tempsDeReactions = []; // Les temps de réaction de l'utilisateur en millisecondes

/\*\* @type {number} \*/

let indexReaction = 0; // Index de la réaction actuelle

/\*\* @type {number} \*/

let score = 0; // Le score du joueur

/\*\* @type {number} \*/

let temps\_moyen\_difficulte = 0; // Le temps moyen de réaction par rapport à la difficulté

/\*\* @type {number} \*/

let temps\_moyen\_total = 0; // Le temps moyen de réaction total

**Méthodes relatives au fonctionnement de l’exercice :**

/\*\*

 \* Appelée depuis l'API python lorsqu'une réaction est à afficher

 \* @param {string} reaction\_a\_afficher

 \*/

function afficherReaction(reaction\_a\_afficher) {

}

/\*\*

 \* Démarre l'exercice

 \* Envoie les paramètres choisis à l'API et attend que l'API initialise l'exercice

 \*/

async function startExercice() {

}

/\*\*

 \* Appelée lorsque l'utilisateur a tapé une chaine de caractères dans l'input de la réaction

 \* Compare la chaine de caractères tapée par l'utilisateur avec la chaine de caractères attendue

 \*/

$: if (typedReaction) {

}

/\*\*

 \* Appelée lorsque l'exercice est terminé

 \* Envoie les temps de réaction à l'API python pour récupérer les statistiques

 \*/

async function exerciceEnded() {

}

1. **Exercice `Jeu du Bac`**

**Variables relatives à l’exercice :**

/\*\* @type {Array<string>} \*/

let themes = []; // Les thèmes disponibles. Récupérés depuis l'API python lors du montage du composant

/\*\* @type {Array<string>} \*/

let selectedThemes = ['Animaux', 'Pays', 'Fruits', 'Légumes', 'Métiers']; // Les thèmes sélectionnés par l'utilisateur (par défaut les 5 ci-gauche)

/\*\* @type {number} \*/

let nombreDeRound = 3; // Le nombre de ligne de l'exercice

let chronometre = 0; // Le chronometre de la round actuelle (en secondes)

let max\_temps = 60; // Le temps maximum pour chaque round (en secondes)

let chronometreTotal = 0; // Le temps total pour finir l'exercice (en secondes)

/\*\* @type {Array<JeuBacRow>} \*/

let rows = []; // Les lignes du tableau

**Classes relatives au fonctionnement de l’exercice :**

/\*\*

 \* La classe JeuBacRow représente une ligne de la grille du jeu du Bac.

 \* Elle contient une lettre et une liste de mots entrés par les joueurs.

 \*/

export default class JeuBacRow {

}

 \* La classe Mot représente un mot entré par un joueur dans une grille du jeu du Bac.

 \* L'attribut "valide" est null si le mot n'a pas encore été validé, "true" si le mot est valide et "false" s'il ne l'est pas.

 \*/

export class Mot {

}

**Méthodes relatives au fonctionnement de l’exercice :**

/\*\*

 \* Appelée lorsqu'on appuie sur ENTRÉE

 \* Vérifie que l'utilisateur a sélectionné 5 thèmes

 \* Si oui, démarre l'exercice

 \*/

async function startExercice() {

}

/\*\*

 \* Appelée lorsqu'un thème est sélectionné ou désélectionné

 \* Si le thème est déjà sélectionné, le retire des thèmes sélectionnés

 \* Sinon, l'ajoute aux thèmes sélectionnés

 \* Si l'utilisateur a déjà sélectionné 5 thèmes, l'empêche de sélectionner plus de thèmes

 \* @param {any} e

 \* @param {string} theme

 \*/

function handleThemeSelected(e, theme) {

}

/\*\*

 \* Démarre un round

 \* Récupère une lettre aléatoire et crée une ligne pour cette lettre

 \*/

async function startRound() {

}

/\*\*

 \* Démarre la roulette

 \* Retourne une lettre aléatoire après un certain nombre de tours de roulette

 \* @returns {Promise<string>}

 \*/

function startRoulette() {

}

/\*\*

 \* Appelée lorsqu'on clique sur le bouton de validation

 \* Envoie les réponses à l'API python pour vérifier si les mots sont corrects depuis la base de données

 \*/

async function validateRow() {

}

/\*\*

 \* Appelée lorsqu'on a fini l'exercice

 \* Trouve le nombre de mots valides

 \*/

function endExercice() {

}

1. **Partie Python**
2. **Clavier**
3. **Exercice `Jeu du Bac`**

**Variables relatives à l’exercice :**

self.connexion # Connexion à la base de donnée SQLite

**Méthodes relatives au fonctionnement de l’exercice :**

def \_\_init\_\_(self):

    """Initialise la connexion à la base de données.

    """

def chargement(self):

    """Depuis le fichier mots.json, récupère les diffèrents thèmes et leurs mots et les

    ajoutes dans la base de donnée

    """

def clean\_mot(self, mot: str):

    """Nettoie un mot en enlevant les accents et en le mettant en minuscule.

    Tous les caractères qui ne sont pas des lettres uniquement (a-z) sont supprimés.

    afin de pouvoir le comparer avec les mots écrits par le joueur.

    >>> b = Bac()

    >>> b.clean\_mot("Éléphant")

    'elephant'

    >>> b.clean\_mot("Marie-Antoinette")

    'marieantoinette'

    >>> b.clean\_mot("École 123")

    'ecole'

    """

def executer\_sql(self, requete):

    """ Exécute une requête SQL sur la base de données locale.

    """

def \_\_del\_\_(self):

"""Ferme la connexion à la base de données lors de la destruction de l'objet.

    """

def verifier\_mot(self, reponses, lettre: str):

    """Vérifie si les réponses données par le joueur sont correctes.

    Args:

        reponses (list): Liste des réponses données par le joueur.

  Exemple : [("Animaux", "éléphant"), ("Fruits et légumes", "épinard"), ("Pays", "Paris")]

        lettre (str): La lettre

    Returns:

        list: Liste des réponses correctes et incorrectes.

        Exemple : [True, False, False]

    """

def url\_to\_html(self, url):

    """Télécharge le contenu d'une page web et le retourne sous forme de texte.

    """

def get\_valid\_letters(self, themes):

    """

    Renvoie les lettres qui ont des mots pour tout les themes

    """

def get\_themes(self):

    """

    Récupère les themes de la base de donnée

    """

1. **Exercice `reaction`**

**Variables relatives à l’exercice :**

reactions = [] # Contient les reactions en str

ACCENTS = "éèàùçê" # Liste des accents

MAJUSCULES = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" # Liste des majuscules

SPECIAUX = ".,;:!?$%&()-\_@" # Liste des caractères speciaux

MINUSCULES = "abcdefghijklmnopqrstuvwxyz" # Liste des minuscules

**Méthodes relatives au fonctionnement de l’exercice :**

def init\_reactions(autoriser\_accent, autoriser\_maj, autoriser\_speciaux, nombre):

    """Renvoie une liste contenant les différents

    caractères à écrire le plus vite possible ainsi que leur temps d'apparition en ms

    Args:

        autoriser\_accent (boolean): Est-ce que les suites de caractères peuvent contenir des accents ?

        autoriser\_maj (boolean): Est-ce que les suites de caractères peuvent contenir des majuscules ?

        autoriser\_speciaux (boolean): Est-ce que les suites de caractères peuvent contenir des caractères spéciaux ?

        nombre (int): Le nombre de réactions à générer

    Returns:

        list: Liste de tuples contenant le temps d'apparition et la suite de caractères à écrire

    Exemple d'output : [(500, "h"), (1900, "zoi"), (4100, "vx"), (700, "lOi")]

    """

def lancer\_reaction(self, index, api):

    """Lance la réaction à l'index donné

    Args:

        index (int): L'index de la réaction à lancer

        api (Api): L'API pour communiquer avec le site

    """

def calculer\_score\_reaction(data):

    """Calcule le score de l'exercice 'Réaction' à partir des données de l'utilisateur

    Args:

        data (list): Les données de l'utilisateur :

                        Une liste au format [[reaction, temps], [reaction, temps], [reaction, temps], ...]

    """

1. **Exercice `rythme `**

**Variables relatives à l’exercice :**

self.current\_key = None # la lettre actuellement pressée

self.nom

self.difficulte

self.son

self.timer

self.touches

**Méthodes relatives au fonctionnement de l’exercice :**

def key\_pressed(self, key):

    """Appelée lorsque l'utilisateur appuie sur une touche

    Args:

        key (str): La touche appuyée

    """

def key\_released(self, key):

        """Appelée lorsque l'utilisateur relâche une touche

        Args:

            key (str): La touche relâchée

        """

def lancer\_compteur(self):

        """Lance le compteur de touches appuyées

        """

def load(self):

        """Charge un niveau depuis un fichier

        """

def create\_level(self):

        """Crée un niveau avec les paramètres donnés

        """

def register\_key(self, key, hold\_time):

        """Enregistre une touche appuyée par l'utilisateur

        Args:

            key (str): La touche appuyée

            time (float): Le temps pendant lequel la touche a été appuyée. 0 si la touche compte une seule fois

        """

def save(self):

        """Sauvegarde dans un fichier les données du niveau

        """

def recuperer\_niveaux():

        """Recupere les niveaux"""

1. **Exercice `Stenographie `**

**Variables relatives à l’exercice :**

**Méthodes relatives au fonctionnement de l’exercice :**

def get\_audios\_with\_texts(self, langue):

        """Récupère des audios et leur retranscription depuis voxforge.org

        Returns:

            list de tuple: Liste de tuple contenant le texte et le chemin d'accès vers l'audio.

        """

def delete\_old\_audios(self):

        """Supprime les anciens fichiers audios

        """

def get\_static\_folder\_path(self):

        """Renvoie le chemin d'accès vers le dossier static du site

        """

def get\_html(self, url):

        """Télécharge le contenu d'une page web et le retourne sous forme de texte.

        """

def get\_random\_voxforge\_page\_url(self, langue):

        """Renvoie une page aléatoire de voxforge.org

        Entre 1 et 76

        """

def get\_links\_in\_voxforge\_page(self, html):

        """Récupère les liens dans une page de voxforge.org

        """

def get\_tgz\_download\_link(self, url):

        """Récupère le lien de téléchargement du .tgz sur une page voxforge.org

        Args:

            url (str): url de la page (ex: https://www.voxforge.org/home/downloads/speech/french-speech-files/nbara-20160203-pkd#zMNM6jv5nOFCo0Wz1eM\_0Q)

        """

def download\_tgz(self, url):

        """Télécharge un fichier .tgz depuis une url

        Args:

            url (str): url du fichier .tgz

        Returns:

            (str): Le path vers le fichier téléchargé

        """

def extract\_tgz(self, filepath, where):

        """Décompresse un fichier .tgz

        Args:

            path (str): chemin d'accès vers le fichier .tgz

        Returns:

            (str): Le nom du dossier où le contenu a été extrait

        """

def verifier\_phrase(self, phrase\_original, phrase\_tapee, majs, orthographe, ponctuations):

        """Vérifie si la réponse donnée par l'utilisateur est correcte

        Args:

            phrase\_original (str): La phrase originale

            phrase\_tapee (str): La phrase tapée par l'utilisateur

            majs (bool): True si les majuscules sont prises en compte, False sinon

            orthographe (bool): True si l'orthographe est prise en compte, False sinon

            ponctuations (bool): True si les ponctuations sont prises en compte, False sinon

        Returns:

            bool: True si la réponse est correcte, False sinon

        """

def similarity(self, a, b):

        """Calcul le pourcentage de ressemblance entre deux phrases

        Args:

            a (str): La première phrase

            b (str): La deuxième phrase

        Returns:

            float: Le pourcentage de ressemblance

        """

def enlever\_accent(self, phrase):

        """Enlève les accents d'une phrase

        Args:

            phrase (str): La phrase

        Returns:

            str: La phrase sans accents

        """

def enlever\_terminaisons(self, phrase):

        """Met toutes les terminaisons `ées`, `és`, `ée`, `er`, `ai`, `aient`, `ais`, `ait` en `er`

        et met toutes les termisaisons `is`, `it`, `ie`, `ient` en `i`

        Args:

            phrase (str): La phrase

        Returns:

            str: La phrase modifiée

        """

1. **Exercice `Type Script`**
2. **Récupération d’un article aléatoire sur Wikipedia**

**Variables relatives à l’exercice :**

MIN\_LENGTH # Longueur minimale de la phrase aléatoire

MAX\_LENGTH # Longueur maximale de la phrase aléatoire

**Méthodes relatives au fonctionnement de l’exercice :**

def \_\_init\_\_(self, langue = "fr") -> None:

        """Initialise le générateur de texte aléatoire

        Args:

            langue (str, optional): La langue du texte. Defaults to "fr".

        """

def get\_random\_article(self):

        """Récupère le code HTML d'un article wikipedia aléatoire

        Returns:

            str: Le code HTML de l'article

        """

def get\_article\_text(self, html):

        """Récupère le texte de l'article wikipedia

        Args:

            html (str): Le code HTML de l'article

        Returns:

            str: Le texte de l'article

        """

def get\_random\_sentences\_from\_text(self, paragraphe: str):

        """Récupère une ou des phrases aléatoires d'un paragraphe

        pour avoir un texte d'une certaine longueur

        Args:

            paragraphe (str): Le paragraphe à analyser

        Returns:

            str: Une phrase aléatoire

        """

def clean\_text(self, text):

        """Applique plusieurs nettoyages sur le texte afin de le rendre plus lisible

        Args:

            text (str): Le texte à nettoyer

        Returns:

            str: Le texte nettoyé

        """

def get\_ramdom\_sentences\_from\_random\_article(self):

        """Récupère une phrase aléatoire d'un article wikipedia aléatoire

        Returns:

            str: Une phrase aléatoire d'une certaine longueur

        """

1. **Génération de l’exercice**

**Méthodes relatives au fonctionnement de l’exercice :**

def get\_random\_sentence(self, langue):

        """Récupère une phrase aléatoire d'un article wikipedia aléatoire

        Args:

            langue (str): La langue de la phrase

        Returns:

            str: Une phrase aléatoire

        """

def calculer\_score\_typescript(self, data):

        """Cette fonction est appelé par le code javascript

        lorsque l'utilisateur a fini l'exercice "Type Script"

        Returns:

            obj: Les différentes données de l'exercice : le temps mis, le nombre d'erreurs et le nombre de caractères total

        """

1. **Souris**
2. **Exercice `SM`**

**Classes relatives à l’exercice :**

1. **Sound Manager**
   1. **Déclaration**

class SoundManager:

    """ Class permettant de gérer les sons du jeu

    """

* 1. **Attributs**

self.sounds = {} # Dictionnaire contenant les sons du jeu (clé : nom du son, valeur : path)

* 1. **Méthodes**

def play(self, sound):

    """ Fonction permettant de jouer un son

    Args:

        sound (str): Nom du son à jouer

    """

1. **SM**
   1. **Déclaration**

class SM:

    """ Classe permettant de gérer le menu du jeu

    """

* 1. **Méthodes**

def start(self):

    """ Fonction permettant de lancer le menu de l'exercice

    Elle dessine le menu avec pygame et permet de lancer l'exercice

    """

1. **Maze**
   1. **Déclaration**

class Maze:

    """

    Classe permettant de gérer les différents niveaux du labyrinthe

    """

* 1. **Attributs**

self.spawn # Position de départ du curseur

self.start # Permet de savoir si le niveau a commencé ou non

* 1. **Méthodes**

def start\_maze(self):

    """

    Fonction permettant de lancer le labyrinthe

    Elle fait, en boucle, les actions suivantes :

    - Initialisation de Pygame

    - Boucle principale

        > Initialisation des dimensions de l'écran

        > Initialisation des variables de la souris

        > Initialisation des variables de victoire

        > Initialisation des niveaux

        > Pour chaque niveau

            - Affichage des rectangles (murs, chemins, etc.)

            - Affichage du début et de la fin du labyrinthe

            - Initialisation des carrés de début et fin

            - Initialisation du curseur

            - Boucle de vérification du niveau

            - Condition de victoire

            - Condition de téléportation

            - Condition de mouvement

    """

1. **Exercice `Rosu !`**

**Classes relatives à l’exercice :**

1. **Menu**
   1. **Déclaration**

class Rosu:

    """ Classe permettant de lancer le menu de ROSU!

    """

* 1. **Méthodes**

def start\_rosu(langue="fr"):

    """ Méthode permettant de lancer le menu Rosu!

    Args:

        langue (str): La langue utilisée pour l'affichage des textes. Peut être "fr" ou "en". Par défaut, "fr".

    """

1. **Niveau**
   1. **Déclaration**

class Niveau:

   """ Classe permettant de stocker les niveaux

   """

1. **Attributs**

self.nom = nom # Nom du niveau

self.musique = musique # Chemin du fichier de la musique du niveau

self.image\_fond = image\_fond # Chemin du fichier de l'image de fond du niveau

self.difficulte = difficulty # Difficulté du niveau (sur 20)

self.data = data # Liste des cercles à cliquer (format: [(temps, (x, y))])

1. **Méthodes**

def \_\_init\_\_(self, nom, musique, image\_fond, difficulty, data) -> None:

   """ Constructeur de la classe Niveau

"""

1. **Sauvegarde**
   1. **Déclaration**

class Sauvegarde:

    """Classe Sauvegarde. Permet de stocker les données de sauvegarde du joueur.

    """

* 1. **Attributs**

self.nom\_niveau = nom\_niveau # Nom du niveau de la sauvegarde

self.meilleur\_score = meilleur\_score  # Meilleur score du joueur

self.note = note # Note du joueur

self.precision = precision # Précision du joueur

* 1. **Méthodes**

def \_\_init\_\_(self, nom\_niveau, meilleur\_score = "", note = "", precision = 0) -> None:

    """Constructeur de la classe Sauvegarde"""

1. **Map Creator**
   1. **Déclaration**

class MapCreator:

    """ Classe permettant de créer un niveau pour l'exercice ROSU!

    """

* 1. **Méthodes**

def CreateMap():

    """ Affiche une fenêtre permettant de créer un niveau pour l'exercice ROSU!

    Elle permet de sauvegarder les clics de l'utilisateur pour créer un niveau.

    """

1. **Engine**
   1. **Déclaration**

class Engine:

    """ Classe permettant de lancer un niveau de l'exercice ROSU!

    """

* 1. **Méthodes**

def \_\_init\_\_(self, langue) -> None:

    """Constructeur de la classe Engine

    Args:

        langue (str): La langue utilisée pour l'affichage des textes. Peut être "fr" ou "en".

    """

def start\_level(self, niveau: Niveau):

    """ Méthode permettant de lancer un niveau de l'exercice ROSU!

    Args:

        niveau (Niveau): Le niveau à lancer.

    Execute dans l'ordre :

    - Affichage du fond

    - Lancement de la musique

        - Affichage des cercles en fonction du temps

        - Vérification des clics

        - Affichage des erreurs

    - Affichage du score

    - Enregistrement du score

    """

1. **Exercice combinant souris et clavier**
2. **Exercice `Verbal Warfare`**
3. **Classe `Menu`**

**Déclaration**

class VW\_Menu:

    """Affiche le menu du jeu Verbal Warfare

    """

**Attribut**

self.folder # Chemin du dossier contenant les assets

**Méthode**

def start(self):

    """Lance le menu de l'exercice

    """

1. **Classe `Engine`**

**Déclaration**

class Engine:

    """Gère l'execution de l'exercice Verbal Warfare

    """

**Attributs**

self.connexion # Connexion à la base de données

self.dictionnaires # Tout les mots de la base de données

self.chargeur # Chargeur de mots

self.temps\_avant\_nouveau\_mot = 1 # Temps avant d'ajouter un nouveau mot

self.score = 0 # Score de l'utilisateur

self.rate = 0 # Nombre de mots ratés

self.combo = 0 # Combo de mots réussis

**Méthodes**

def verifier\_mot\_clique(self, pos):

    """

    Vérifie si un mot a été cliqué par l'utilisateur

    Args:

        pos (tuple): La position du clic de l'utilisateur

    """

def remplir\_mot(self, mot\_clique):

    """

    Remplis le mot cliqué par l'utilisateur avec les lettres écrites

    Args:

        mot\_clique (str): Le mot cliqué par l'utilisateur (ex: "B\_\_jour")

    Returns:

        str: Le mot à compléter (ex: "Bonjour")

    """

def check\_existance(self, mot):

    """

    Vérifie si le mot existe dans la base de données

    Args:

        mot (str): Le mot à vérifier

    Returns:

        bool: True si le mot existe, False sinon

    """

def incrementer\_score(self, mot):

    """

    Incrémente le score de l'utilisateur

    Args:

        mot (str): Le mot qui a été complété (ex: "B\_\_jour")

    """

def handle\_mouse\_click(self, pos):

    """

    Gère le clic de l'utilisateur

    Args:

        pos (tuple): La position du clic de l'utilisateur

    """

def ajouter\_mot(self):

    """ Ajoute un mot à l'écran toutes les 3 secondes

    """

def deplacer\_mots(self):

    """Déplace les mots d'un côté à l'autre de l'écran

    """

def afficher\_viseur(self, pos):

    """Affiche l'image du viseur à la position de la souris

    Args:

        pos (tuple): La position de la souris

    """

def executer\_sql(self, requete):

    """ Exécute une requête SQL sur la base de données locale.

    """

def generer\_mot(self):

    """Génère un mot aléatoire à compléter.

    Returns:

        tuple: Dans la forme (mot\_complet, mot\_incomplet)

        exemple: ("Bonjour", "B\_\_jour")

    """