Sorbonne Université

Environnements Virtuels Hautement Interactifs

Conception Projet EVHI

Auteurs :
BIEGAS Emilie (3700036)
YANG Zitong (3872648)

01 Décembre 2021

Table des matières

1 Introduction de l'application

Nous proposons de développer une application d'apprentissage de langue sous forme de série de questions de vocabulaire (sous forme de question à choix multiples (QCM) ou en demandant à l'utilisateur d'écrire la réponse en entier). C'est un jeu sérieux destiné à toute personne voulant améliorer son vocabulaire en anglais.

Nous allons pour cela modéliser les connaissances de l'utilisateur afin de pouvoir lui proposer des questions adaptées (que ce soit au niveau de la forme de la question (QCM ou autre) ou au niveau du mot de vocabulaire demandé), notamment en s'appuyant sur la courbe de l'oubli (courbe prennant en compte la rétention (pas de pratique pendant un temps) dans l'apprentissage). En effet, chacun des mots de vocabulaire va être associé à une valeur entre 0 et 1 qui quantifie la probabilité d'acquisition de ce mot par l'utilisateur (1 lorsque le mot est totalement acquis et 0 s'il n'est pas acquis ou que l'on n'a pas encore testé l'utilisateur sur ce point). Cette valeur va évoluer au cours du temps et en particulier va diminuer en suivant la courbe de l'oubli (qui prendra également en compte le nombre de fois où le mot en question a été demandé).

Nous allons également utiliser le "Keystroke level Model" (afin de mesurer la vitesse du clic ou de l'entrée de texte de l'utilisateur en comparaison à ce modèle) ainsi que les traces récoltées par l'oculomètre afin de déterminer si l'utilisateur semble avoir répondu au hasard ou s'il connaissait réellement la réponse. En effet, on a tendance à regarder plusieurs fois toutes les réponses les unes après les autres et à entrer la réponse plus lentement lorsque l'on hésite et plutôt à regarder une seule réponse plus longtemps et à la sélectionner rapidement lorsque l'on est sûr de nous. Cette estimation d'hésitation va, entre autre, permettre d'adapter les probabilités d'acquisition du mot de vocabulaire.

En fonction des connaissances de l'utilisateur et des traces recueillies, notre système va adapter les questions posées à l'utilisateur et proposer un retour à celui-ci sous forme de fiche explicative ciblée sur l'origine de ses lacunes, c'est-à-dire, par exemple, lorsqu'il semble confondre deux notions proches ou lorsqu'il confond la nature des mots (s'il écrit un verbe alors que l'on lui a demandé un adjectif par exemple).

2 Modélisation des données

Notre système comporte différentes données : les données pour générer les questions et proposer la fiche d'aide et les données des traces des utilisateurs. Attardons-nous tout d'abord sur les données des traces utilisateur ainsi que les données de sauvegarde.

Nous sauvegardons les données dans des fichiers textes locaux grâce à la classe DataSaver.

Nous sauvegardons différentes données regroupées dans plusieurs classes : UserTraces (classe permettant de rassembler les données à stocker concernant les traces du joueur (ensemble des vitesses de sélection et d'entrée de texte) dans un seul type), UserInitialisation (classe permettant de rassembler les données à stocker concernant l'initialisation dans un seul type, ce qui va permettre de sauvegarder l'avancée de l'utilisateur dans le phase d'initialisation), UserStats (classe permettant de rassembler les données à stocker concernant les statistiques du joueur dans un seul type, ce qui va permettre de sauvegarder le niveau de l'utilisateur ainsi que son avancée dans le jeu), et UserSelectionMenu (classe permettant de rassembler les données à stocker concernant les temps de sélection dans les menus du joueur dans un seul type, ce qui va permettre d'avoir un point de repère quant au niveau de sélection de l'utilisateur).

Ces classes sont détaillées ci-après.

Ces données seront sauvegardées lorsque l'utilisateur revient dans le menu principal et seront chargées lorsque l'utilisateur lance ou re-lance sa partie (sauf pour UserSelectionMenu dont les données seront sauvegardées et chargées lors du lancement de la partie d'un joueur).

Les données simples (un entier, une chaîne de caractère ou un flottant) comme le pseudo de l'utilisateur ou le numéro du joueur en cours de jeu vont être sauvegardées et accessibles dans les PlayerPrefs.

L'ensemble des questions / réponses disponibles est stocké dans un fichier CSV local.

2.1 Classe DataSaver

Classe permettant de sauvegarder les données contenant beaucoup de variables (comme des instances de UserStats ou de UserInitialisation).

Méthodes :

Algorithm 1: static void saveData<T>
Fonction permettant de sauvegarder les données

```
Data: T dataToSave, string dataFileName
string tempPath = Path.Combine(Application.persistentDataPath, "data");
tempPath = Path.Combine(tempPath, dataFileName + ".txt");
// Convertir en Json puis en bytes
string jsonData = JsonUtility.ToJson(dataToSave, true); byte[] jsonByte =
 Encoding.ASCII.GetBytes(jsonData);
// Creer un dossier si il n'existe pas encore
if (!Directory.Exists(Path.GetDirectoryName(tempPath))) then
 Directory.CreateDirectory(Path.GetDirectoryName(tempPath));
try
{
// Sauvegarde des données
File.WriteAllBytes(tempPath, jsonByte);
} catch (Exception e)
{
Debug.LogWarning("Echec dans la sauvegarde des données au chemin : " + tempPath.Replace("/", "
"));
Debug.LogWarning("Erreur : " + e.Message);
}
```

Algorithm 2: public static T loadData<T>
Fonction permettant de charger les données

```
Data: string dataFileName
Result: donnée T
string tempPath = Path.Combine(Application.persistentDataPath, "data");
tempPath = Path.Combine(tempPath, dataFileName + ".txt");
// Quitter si le dossier ou le fichier n'existe pas
if (!Directory.Exists(Path.GetDirectoryName(tempPath))) then
 // Debug.Log("Le dossier n'existe pas"); return default(T);
if (!File.Exists(tempPath)) then
 return default(T);
// Charger les données sauvegardées en Json
byte[] jsonByte = null;
try
{
jsonByte = File.ReadAllBytes(tempPath);
// Debug.Log("Données chargées depuis le chemin : " + tempPath.Replace("/", "
"));
}
catch (Exception e)
{
Debug.LogWarning("Echec dans le chargement des données depuis le chemin : " + tempPath.Replace("/",
 "
"));
Debug.LogWarning("Erreur : " + e.Message);
}
// Convertir en string
string jsonData = Encoding.ASCII.GetString(jsonByte);
// Convertir en Object
object resultValue = JsonUtility.FromJson<T>(jsonData);
return (T)Convert.ChangeType(resultValue, typeof(T));
```

Algorithm 3: static bool deleteData
Fonction permettant d'effacer les données

```
Data: string dataFileName
Result: bool success
bool success = false;
// Charger les données
string tempPath = Path.Combine(Application.persistentDataPath, "data"); tempPath =
 Path.Combine(tempPath, dataFileName + ".txt");
// Quitter si le dossier ou le fichier n'existe pas
if (!Directory.Exists(Path.GetDirectoryName(tempPath))) then
 return false;
if (!File.Exists(tempPath)) then
 return false;
try
{
File.Delete(tempPath);
// Debug.Log("Données effacées sur le chemin : " + tempPath.Replace("/", "
"));
success = true;
}
catch (Exception e)
{
Debug.LogWarning("Echec dans la suppression des données : " + e.Message);
}
return success;
```

2.2 Classe UserInitialisation

Classe permettant de rassembler les données à stocker concernant l'initialisation dans un seul type.

Attributs :

- **public int nbBienRep** : Nombre de fois où l'utilisateur a bien répondu au QCM puis à la même question en entier
- **public int nbMalRep** : Nombre de fois où l'utilisateur a bien répondu au QCM puis à mal répondu à la même question en entier
- **public int numIte** : Numéro de l'itération dans l'initialisation
- **public bool inInitialisation** : Indique si on est encore dans l'initialisation (true) ou non (false)
- **public bool RepEntreeOK** : Indique si la réponse entrée est bonne (true) ou non (false)
- **public int NbQuestAvantNouvelle** : Le nombre de questions nécessaires sur des mots déjà rencontrés avant une question sur un mot non encore rencontré (modifié seulement à la fin d'un cycle de réponse)
- **public int NbQuestAvantNouvelleTemp** : Le nombre de questions nécessaires sur des mots déjà rencontrés avant une question sur un mot non encore rencontré (modifié à chaque réponse de l'utilisateur)
- **public int NbAncienneQuestion** : Le nombre de questions posées sur des mots déjà rencontrés depuis la dernière rencontre d'un nouveau mot
- **public int NbNouvelleQuestion** : Le nombre de questions posées sur des mots non encore rencontrés depuis la dernière rencontre d'un ancien mot
- **public int NbQuestionsTotales** : Le nombre de questions rencontrées au total
- **public bool inQCM** : Indique si la question à laquelle on vient de répondre est un QCM (true) ou non (false)

  Pour retrouver la question posée dernièrement :
- **public int QuestionCourrante** : Indice de la question courrante (càd qui est en train d'être posée)
- **public string TypeQuestion** : Le type de la question en cours (QCM ou Entier)
- **public int NbAncienneQuestionTemp** : Pour permettre de mettre à jour NbAncienneQuestion que quand l'utilisateur a répondu et pas avant (s'il a juste vu la question)
- **public int NbNouvelleQuestionTemp** : Pour permettre de mettre à jour NbNouvelleQuestion que quand l'utilisateur a répondu et pas avant (s'il a juste vu la question)
- **public List<int> IndQuestNonRencontrees** : Indices des questions non encores posées

2.3 Classe UserStats

Classe permettant de rassembler les données à stocker concernant les statistiques du joueur dans un seul type

Attributs :

- **float[] probaAcquisition** : Défini le tableau de probabilité d'acquisition de chaque mot de vocabulaire (de VocabUtilisateur)
- **int[] nbRencontres** : Défini le tableau du nombre de rencontres de chaque mot de vocabulaire (de VocabUtilisateur)
- **string[] dateDerniereRencontre** : Défini le tableau de date de dernière rencontre de chaque mot de vocabulaire (en format string "MM/dd/yyyy HH :mm :ss") (de VocabUtilisateur)
- **int nivSelection** : Défini le niveau en terme de vitesse de sélection de l'utilisateur (de HesitationManager)
- **int nivEntreeTexte** : Défini le niveau en terme de vitesse d'entrée de texte de l'utilisateur (de HesitationManager)
- **float[][] probaAcquisNature** : Défini les connaissances de l'apprenant quant à la nature des mots
- **List<Vector2> occulaireHesite** : Défini l'ensemble des données occulaires (vecteur à deux dimensions) de la classe "hésite"
- **List<Vector2> occulaireSur** : Défini l'ensemble des données occulaires (vecteur à deux dimensions) de la classe "sûr"

2.4 Classe UserTraces

Classe permettant de rassembler les données à stocker concernant les traces du joueur (ensemble des vitesses de sélection et d'entrée de texte) dans un seul type.

Attributs :

- **public List<Tuple<float, bool»[] vitessesSelection** : Tableau de liste : chaque case du tableau correspond à un mot de vocabulaire, pour chaque mot on a une liste de tuple dont le premier élément est le temps mis et le second est la véracité de la réponse (à chaque fois qu'on a rencontré ce mot)
- **public List<Tuple<float, bool»[] vitessesEntreeTexte** : Tableau de liste : chaque case du tableau correspond à un mot de vocabulaire, pour chaque mot on a une liste de tuple dont le premier élément est le temps mis et le second est la véracité de la réponse (à chaque fois qu'on a rencontré ce mot)

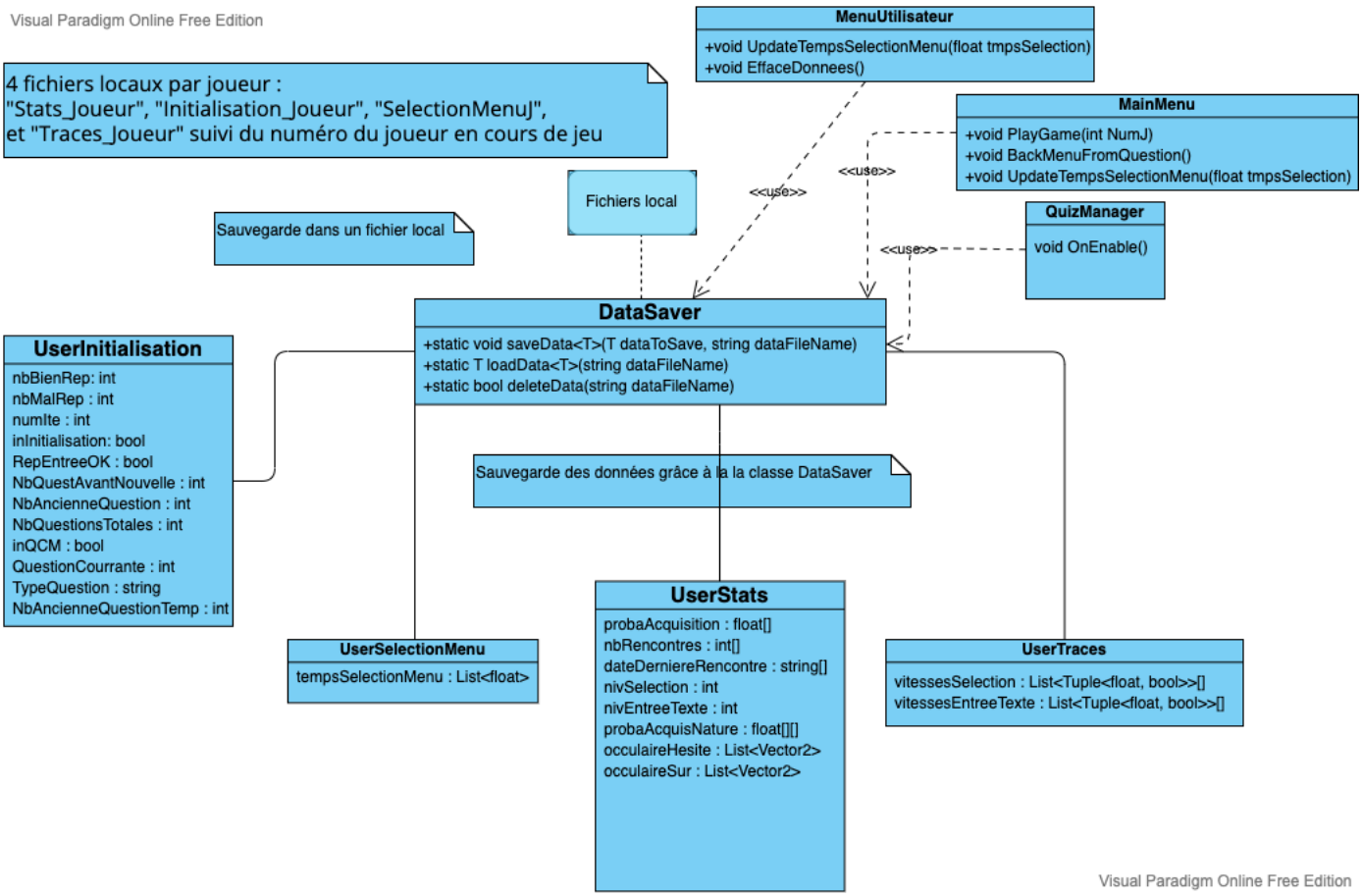
2.5 Classe UserSelectionMenu

Classe permettant de rassembler les données à stocker concernant les temps de sélection dans les menus du joueur dans un seul type.

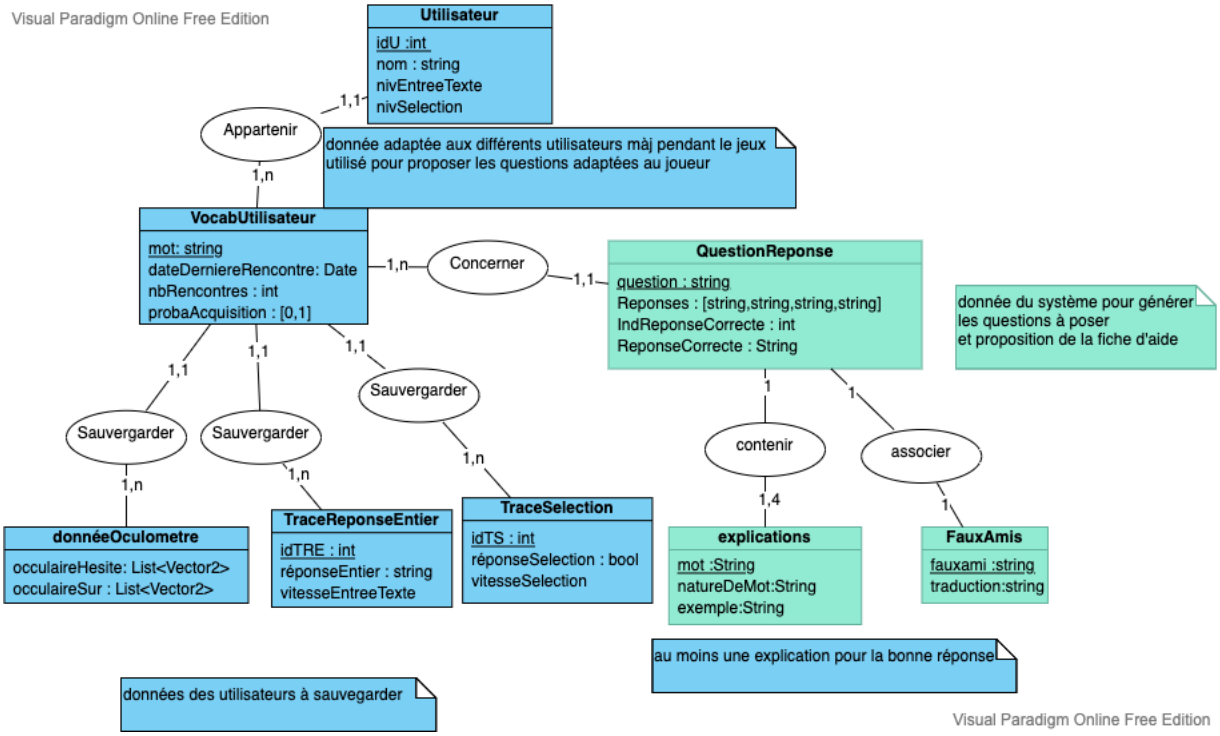
Attributs :

- **List<float> tempsSelectionMenu** : Une liste de temps de sélection dans les menus permettant d'initialiser le niveau de l'utilisateur en terme de temps de sélection.

2.6 Modèle conceptuel des données de sauvegarde et des traces

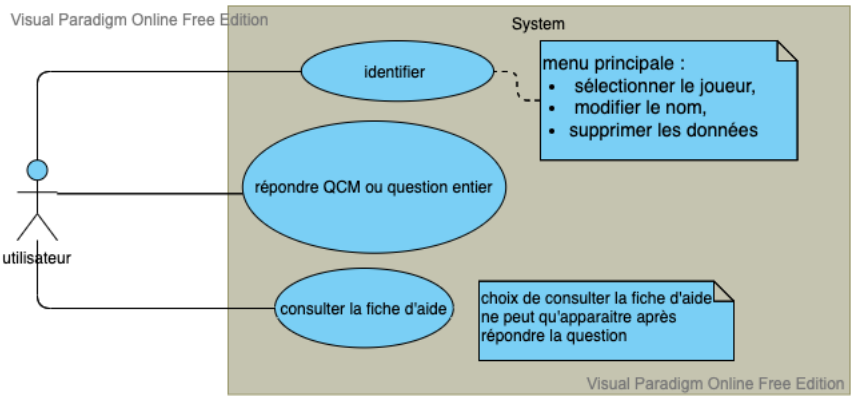


2.7 Modèle conceptuel des données de génération de question et de fiche d'aide

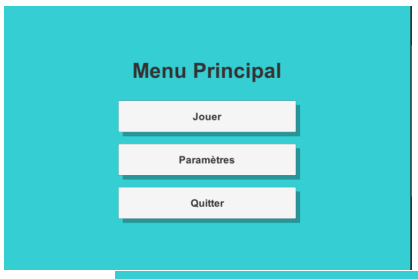


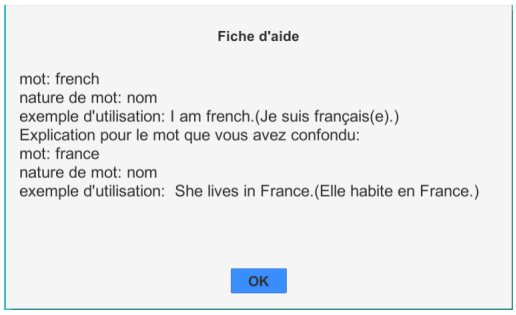
3 Modélisation de l'utilisateur et de l'environnement

3.1 Diagramme des cas d'utilisation

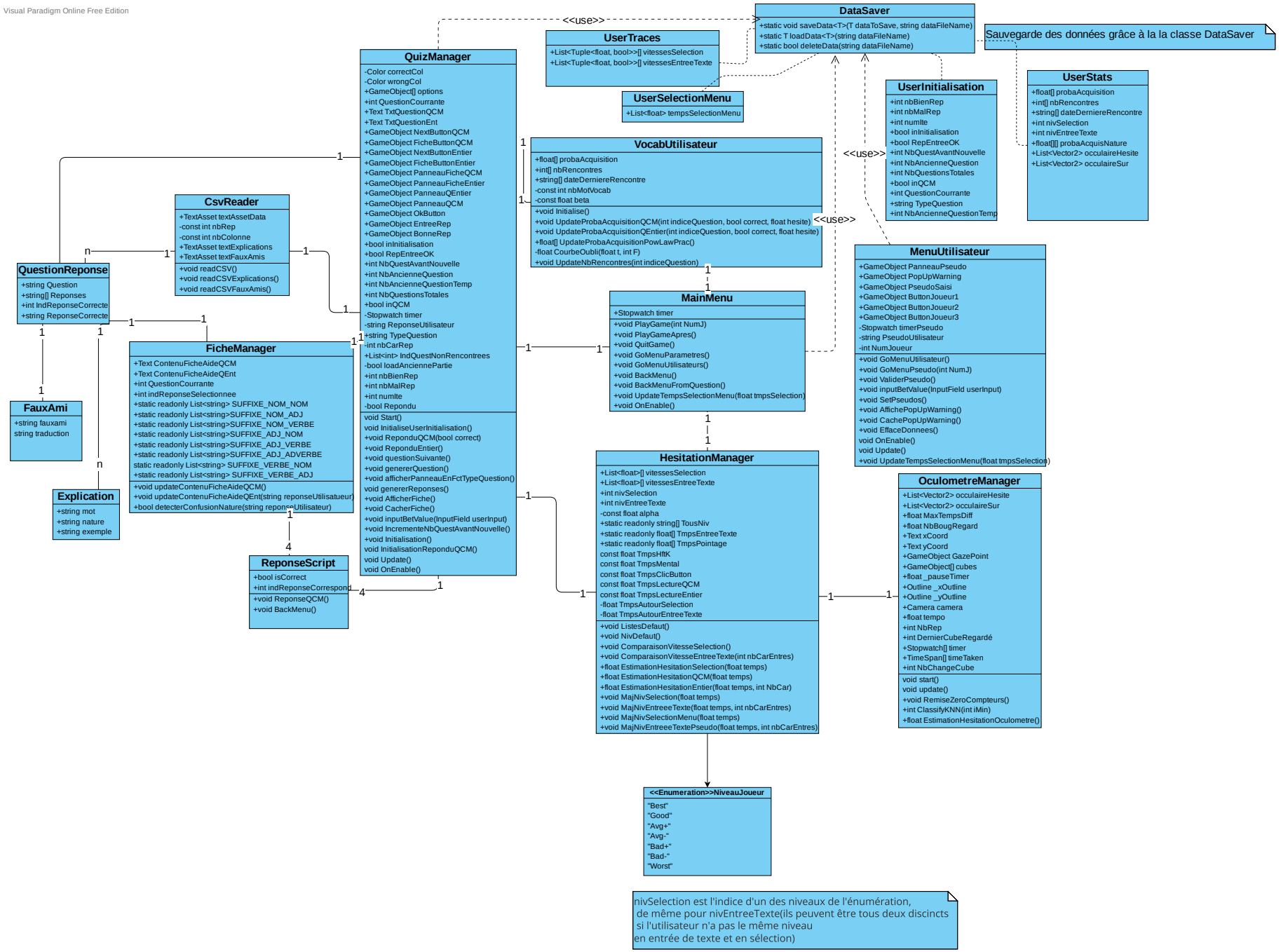


3.2 Principales scènes de jeux





3.3 Diagramme de classes



3.4 Classes de l'environnement

3.4.1 Classe MainMenu

Classe permettant de gérer le menu principal.

Méthodes :

Algorithm 4: void PlayGame
Associé au bouton jouer

```
Data: int NumJ
PlayerPrefs.SetInt("NumJoueur", NumJ);// On précise quel joueur joue pour pouvoir récupérer ses données
   et définir les données de ce joueur
// On sauvegarde les données de temps de sélection dans les menus de ce joueur
// On récupère l'ancienne liste de tempsSelectionMenu et on la complète avec les données récupérées
var tempsSelectionMenuTemp = new List<float>();
// On charge l'ancienne liste de tempsSelectionMenu du joueur concerné
UserSelectionMenu loadedDataSelectionJ = DataSaver.loadData<UserSelectionMenu>("SelectionMenuJ"
 + PlayerPrefs.GetInt("NumJoueur"));
if (loadedDataSelectionJ == null or
 EqualityComparer<UserSelectionMenu>.Default.Equals(loadedDataSelectionJ,
 default(UserSelectionMenu))) then
tempsSelectionMenuTemp = loadedDataSelectionJ.tempsSelectionMenu;
// On ajoute les données récupérées à ce tour
// On charge la liste de tempsSelectionMenu de ce tour
UserSelectionMenu loadedDataSelection = DataSaver.loadData<UserSelectionMenu>("SelectionMenu");
if (loadedDataSelection == null or
 EqualityComparer<UserSelectionMenu>.Default.Equals(loadedDataSelection,
 default(UserSelectionMenu))) then
tempsSelectionMenuTemp.AddRange(loadedDataSelection.tempsSelectionMenu);
/
UserSelectionMenu saveDataSelectionMenu = new UserSelectionMenu();
saveDataSelectionMenu.tempsSelectionMenu = new List<float>();
saveDataSelectionMenu.tempsSelectionMenu = tempsSelectionMenuTemp;
DataSaver.saveData(saveDataSelectionMenu, "SelectionMenuJ" + PlayerPrefs.GetInt("NumJoueur"));
SceneManager.LoadScene("QCM"); // On load la scène des QCM à 4 choix
```

Algorithm 5: void PlayGameApres
Utilisé lorsque l'on accède au jeu par le menu de changement de pseudo (on a donc déjà mis à jour NumJoueur en accédant au menu de pseudo)

```
// On load la scène des QCM à 4 choix
SceneManager.LoadScene("QCM");
```

Algorithm 6: void GoMenuParametres
Associé au bouton paramètre

```
// On arrête le chronomètre
timer.Stop();
TimeSpan timeTaken = timer.Elapsed;// On regarde le temps passé sur le chronomètre
// On transforme la durée obtenue en float (nombre de secondes/minutes/heures écoulées) afin de l'enregistrer
   int days, hours, minutes, seconds, milliseconds;
days = timeTaken.Days;
hours = timeTaken.Hours;
minutes = timeTaken.Minutes;
seconds = timeTaken.Seconds;
milliseconds = timeTaken.Milliseconds;
// Temps passé en secondes :
float floatTimeSpan = ((float)days*24*3600) + ((float)hours*3600) + ((float)minutes*60) + (float)seconds
 + ((float)milliseconds/1000);
// On enregistre le temps obtenu
UpdateTempsSelectionMenu(floatTimeSpan);
SceneManager.LoadScene("MenuParametres"); // On load la scène des paramètres
```

Algorithm 7: void GoMenuUtilisateurs
Lorsque l'on clique sur jouer depuis le menu principal, on choisit d'abord le profil que l'on souhaite jouer

```
// On arrête le chronomètre
timer.Stop();
TimeSpan timeTaken = timer.Elapsed;// On regarde le temps passé sur le chronomètre
int days, hours, minutes, seconds, milliseconds;
days = timeTaken.Days;
hours = timeTaken.Hours;
minutes = timeTaken.Minutes;
seconds = timeTaken.Seconds;
milliseconds = timeTaken.Milliseconds;
// Temps passé en secondes :
float floatTimeSpan = ((float)days*24*3600) + ((float)hours*3600) + ((float)minutes*60) + (float)seconds
 + ((float)milliseconds/1000);
// On enregistre le temps obtenu
UpdateTempsSelectionMenu(floatTimeSpan);
SceneManager.LoadScene("MenuUtilisateurs");// On load la scène de menu utilisateurs
```

Algorithm 8: void BackMenu
Associé au bouton menu accessible depuis un autre menu que le menu principal

```
SceneManager.LoadScene("MainMenu"); // On load la scène de menu principal
```

Algorithm 9: void BackMenuFromQuestion

Appelée lorsque l'on retourne au menu principal depuis une question, on sauvegarde à ce moment les données du joueur

```
// On enregistre les données statistique utilisateur
UserStats saveDataStat = new UserStats();
saveDataStat.probaAcquisition = vocabUt.probaAcquisition;
saveDataStat.nbRencontres = vocabUt.nbRencontres;
saveDataStat.dateDerniereRencontre = vocabUt.dateDerniereRencontre;
saveDataStat.nivSelection = hesitationManager.nivSelection;
saveDataStat.nivEntreeTexte = hesitationManager.nivEntreeTexte; // Sauvegarde des données de UserStats
   dans un fichier nommé Stats_Joueur suivi du numéro du joueur
DataSaver.saveData(saveDataStat, "Stats_Joueur" + PlayerPrefs.GetInt("NumJoueur"));
// On enregistre les données d'initialisation utilisateur
UserInitialisation saveDataInit = new UserInitialisation();
saveDataInit.nbBienRep = quizManager.nbBienRep;
saveDataInit.nbMalRep = quizManager.nbMalRep;
saveDataInit.numIte = quizManager.numIte;
saveDataInit.inInitialisation = quizManager.inInitialisation;
saveDataInit.RepEntreeOK = quizManager.RepEntreeOK;
saveDataInit.NbQuestAvantNouvelle = quizManager.NbQuestAvantNouvelle;
saveDataInit.NbQuestAvantNouvelleTemp = quizManager.NbQuestAvantNouvelleTemp;
saveDataInit.NbAncienneQuestion = quizManager.NbAncienneQuestion;
saveDataInit.NbNouvelleQuestion = quizManager.NbNouvelleQuestion;
saveDataInit.NbQuestionsTotales = quizManager.NbQuestionsTotales;
saveDataInit.inQCM = quizManager.inQCM;
saveDataInit.QuestionCourrante = quizManager.QuestionCourrante;
saveDataInit.TypeQuestion = quizManager.TypeQuestion;
saveDataInit.NbAncienneQuestionTemp = quizManager.NbAncienneQuestionTemp;
saveDataInit.NbNouvelleQuestionTemp = quizManager.NbNouvelleQuestionTemp;
saveDataInit.IndQuestNonRencontrees = quizManager.IndQuestNonRencontrees;
// Sauvegarde des données de UserInitialisation dans un fichier nommé Initialisation_Joueur suivi du numéro du
   joueur
DataSaver.saveData(saveDataInit, "Initialisation_Joueur" + PlayerPrefs.GetInt("NumJoueur"));
// On load la scène de menu principal
SceneManager.LoadScene("MainMenu");
```

Algorithm 10: void UpdateTempsSelectionMenu
On sauvegarde les données de temps de sélection dans les menus de ce joueur

```
Data: float tmpsSelection
// On récupère l'ancienne liste de tempsSelectionMenu et on la complète avec les données récupérées
var tempsSelectionMenu = new List<float>();// Une liste de temps de sélection dans les menus permettant
    d'initialiser le niveau de l'utilisateur en terme de temps de sélection
// On charge l'ancienne liste de tempsSelectionMenu
UserSelectionMenu loadedDataSelection =
    DataSaver.loadData<UserSelectionMenu>("SelectionMenu");// On ne sait pas qui est en train de jouer
    pour l'instant
if (loadedDataSelection == null or
  EqualityComparer<UserSelectionMenu>.Default.Equals(loadedDataSelection,
  default(UserSelectionMenu))) then
else
  tempsSelectionMenu = loadedDataSelection.tempsSelectionMenu;
// On ajoute les données récupérées à ce tour
tempsSelectionMenu.Add(tmpsSelection);
UserSelectionMenu saveDataSelectionMenu = new UserSelectionMenu();
saveDataSelectionMenu.tempsSelectionMenu = new List<float>();
saveDataSelectionMenu.tempsSelectionMenu = tempsSelectionMenu;
// Sauvegarde des données de selection dans les menus dans un fichier nommé SelectionMenu
DataSaver.saveData(saveDataSelectionMenu, "SelectionMenu");
```

Algorithm 11: void OnEnable
Fonction appelée lors de l'ouverture de la scène MainMenu

```
// On lance le chronomètre pour calculer la vitesse de selection de l'utilisateur
timer = new Stopwatch();
timer.Start();
```

3.4.2 Classe MenuUtilisateur

Classe permettant de gérer le menu utilisateur.

Attributs :

- **public GameObject PanneauPseudo** : La page de choix de pseudo / effacage de données
- **public GameObject PopUpWarning** : Le pop-up qui apparaît lorsque l'on souhaite effacer des données utilisateur pour prévenir que c'est irréversible
- **public GameObject PseudoSaisi** : Le champs de saisie du pseudo
- **public GameObject ButtonJoueur1** : Le bouton de sélection du joueur 1
- **public GameObject ButtonJoueur2** : Le bouton de sélection du joueur 2
- **public GameObject ButtonJoueur3** : Le bouton de sélection du joueur 3
- **private Stopwatch timerPseudo** : Le chronomètre permettant de chronomètrer le temps que l'utilisateur met pour entrer son pseudo
- **private string PseudoUtilisateur** : Le pseudo choisi par l'utilisateur
- **private int NumJoueur** : Le numéro du joueur en cours de jeu ou qui souhaite changer son pseudo (trouvable également dans les PlayerPrefs)
- **private Stopwatch timerSelection** : Le chronomètre permettant de mesurer le temps de sélection de l'utilisateur dans le menu

Méthodes :

Algorithm 12: void GoMenuUtilisateur
Retourner dans le menu utilisateur (pas dans le menu pseudos), appuyer sur Retour cache le panneau pseudo

```
PanneauPseudo.SetActive(false);// On cache le panneau pseudo
SetPseudos();// On màj les pseudos
```

Algorithm 13: void GoMenuPseudo
Permet d'accéder au menu pseudos

```
Data: int NumJ
// On arrête le chronomètre de sélection
timerSelection.Stop();
TimeSpan timeTaken = timerSelection.Elapsed; // On regarde le temps passé sur le chronomètre de sélection
// On transforme la durée obtenue en float (nombre de secondes/minutes/heures écoulées) afin de l'enregistrer
int days, hours, minutes, seconds, milliseconds;
days = timeTaken.Days;
hours = timeTaken.Hours;
minutes = timeTaken.Minutes;
seconds = timeTaken.Seconds;
milliseconds = timeTaken.Milliseconds;
// Temps passé en secondes :
float floatTimeSpan = ((float)days*24*3600) + ((float)hours*3600) + ((float)minutes*60) + (float)seconds
  + ((float)milliseconds/1000);
PanneauPseudo.SetActive(true);// On affiche le panneau pseudo
// On lance le chronomètre de saisie de pseudo
timerPseudo = new Stopwatch();
timerPseudo.Start();
// On fait en sorte que le champs de saisie soit automatiquement sélectionné (pas besoin de cliquer dessus)
PseudoSaisi.GetComponent<InputField>().Select();
// On "écoute" la saisie de l'utilisateur
PseudoSaisi.GetComponent<InputField>().onEndEdit.AddListener(delegate
  inputBetValue(PseudoSaisi.GetComponent<InputField>()); );
NumJoueur = NumJ; // On précise quel joueur a défini son pseudo
PlayerPrefs.SetInt("NumJoueur", NumJ); // On précise quel joueur joue pour pouvoir associer les actions à
   ce joueur en particulier, et pour pouvoir cliquer sur jouer dans ce menu sans problèmes
// On enregistre le temps obtenu et on le
sauvegarde UpdateTempsSelectionMenu(floatTimeSpan);
```

Algorithm 14: void inputBetValue
Fonction permettant de lire la saisie de l'utilisateur

```
Data: InputField userInput
PseudoUtilisateur = userInput.text;
```

Algorithm 15: void AffichePopUpWarning
Fonction permettant d'afficher le pop-up warning de l'écrasement des données utilisateur

```
PopUpWarning.SetActive(true);
```

Algorithm 16: void CachePopUpWarning
Fonction permettant de cacher le pop-up warning de l'écrasement des données utilisateur

```
PopUpWarning.SetActive(false);
```

Algorithm 17: void ValiderPseudo
Appellée lorsque l'utilisateur valide son choix de pseudo

```
// Calcul de la vitesse d'entrée de texte (de son pseudo) de l'utilisateur
timerPseudo.Stop();
TimeSpan timeTaken = timerPseudo.Elapsed;
// On transforme la durée obtenue en float (nombre de secondes/minutes/heures écoulées) afin de l'enregistrer
   dans le gérant de l'hésitation
int days, hours, minutes, seconds, milliseconds;
days = timeTaken.Days;
hours = timeTaken.Hours;
minutes = timeTaken.Minutes;
seconds = timeTaken.Seconds;
milliseconds = timeTaken.Milliseconds;
// Temps passé en secondes :
float floatTimeSpan = ((float)days*24*3600) + ((float)hours*3600) + ((float)minutes*60) + (float)seconds
 + ((float)milliseconds/1000);
PlayerPrefs.SetFloat("TmpsEntreePseudoJ" + NumJoueur, floatTimeSpan); // On ajoute le temps de
   saisie du pseudo du joueur en cours de jeu pour pouvoir initialiser le temps d'entrée de texte
string foo = "Temps d'entrée de pseudo du joueur " + NumJoueur + " = " +
 timeTaken.ToString(@"m ssfff");
// On met à jour le pseudo du joueur en question
// Enregistrer le pseudo de l'utilisateur et le mettre dans l'emplacement sélectionné
if (NumJoueur == 1) then
    ButtonJoueur1.GetComponentInChildren<Text>().text = PseudoUtilisateur;
        PlayerPrefs.SetString("PseudoJ1", PseudoUtilisateur); // Enregistrer le pseudo du joueur dans les
        PlayerPrefs pour pouvoir y accéder à chaque lancement de jeu
if (NumJoueur == 2) then
    ButtonJoueur2.GetComponentInChildren<Text>().text = PseudoUtilisateur;
     PlayerPrefs.SetString("PseudoJ2", PseudoUtilisateur);
if (NumJoueur == 3) then
    ButtonJoueur3.GetComponentInChildren<Text>().text = PseudoUtilisateur;
     PlayerPrefs.SetString("PseudoJ3", PseudoUtilisateur);
// On se rend sur la page de choix d'utilisateur
GoMenuUtilisateur();
```

Algorithm 18: void SetPseudos
// Fonction permettant d'afficher les bons pseudos (grâce à la sauvegarde des pseudos dans les PlayerPrefs)

```
// On affiche les pseudos précédemment sauvegardés
if (PlayerPrefs.HasKey("PseudoJ1")) then
    ButtonJoueur1.GetComponentInChildren<Text>().text = PlayerPrefs.GetString("PseudoJ1");
else
    ButtonJoueur1.GetComponentInChildren<Text>().text = "Joueur 1";// Affichage par défaut
if  (PlayerPrefs.HasKey("PseudoJ2")) then
    ButtonJoueur2.GetComponentInChildren<Text>().text = PlayerPrefs.GetString("PseudoJ2");
else
    ButtonJoueur2.GetComponentInChildren<Text>().text = "Joueur 2";// Affichage par défaut
if  (PlayerPrefs.HasKey("PseudoJ3")) then
    ButtonJoueur3.GetComponentInChildren<Text>().text = PlayerPrefs.GetString("PseudoJ3");
else
    ButtonJoueur3.GetComponentInChildren<Text>().text = "Joueur 3"; // Affichage par défaut
```

Algorithm 19: void UpdateTempsSelectionMenu
On sauvegarde les données de temps de sélection dans les menus de ce joueur

```
Data: float tmpsSelection
// On récupère l'ancienne liste de tempsSelectionMenu et on la complète avec les données récupérées
var tempsSelectionMenu = new List<float>(); // Une liste de temps de sélection dans les menus permettant
   d'initialiser le niveau de l'utilisateur en terme de temps de sélection
// On charge l'ancienne liste de tempsSelectionMenu du joueur concerné
UserSelectionMenu loadedDataSelection = DataSaver.loadData<UserSelectionMenu>("SelectionMenuJ" +
 PlayerPrefs.GetInt("NumJoueur"));
if (loadedDataSelection == null or
 EqualityComparer<UserSelectionMenu>.Default.Equals(loadedDataSelection,
 default(UserSelectionMenu))) then
else
 ⌊ tempsSelectionMenu = loadedDataSelection.tempsSelectionMenu;
// On ajoute les données récupérées à ce tour
tempsSelectionMenu.Add(tmpsSelection);
UserSelectionMenu saveDataSelectionMenu = new UserSelectionMenu();
saveDataSelectionMenu.tempsSelectionMenu = new List<float>();
saveDataSelectionMenu.tempsSelectionMenu = tempsSelectionMenu;
// Sauvegarde des données de selection dans les menus dans un fichier nommé SelectionMenuJ suivi du numéro du
   joueur
DataSaver.saveData(saveDataSelectionMenu, "SelectionMenuJ" + PlayerPrefs.GetInt("NumJoueur"));
```

Algorithm 20: void EffaceDonnees
Associée au bouton d'écrasement des données utilisateur

```
// On efface toutes les données du joueur en question
DataSaver.deleteData("Stats_Joueur" + NumJoueur); // Efface les statistiques du joueur
DataSaver.deleteData("Initialisation_Joueur" + NumJoueur); // Efface les données d'initialisation du
   joueur
DataSaver.deleteData("SelectionMenuJ" + NumJoueur);// Efface les données de selection dans les menus
   du joueur
PlayerPrefs.DeleteKey("PseudoJ" + NumJoueur); // Efface son pseudo
PlayerPrefs.DeleteKey("TmpsEntreePseudoJ" + NumJoueur); // Efface le temps de saisie de son pseudo
// On se rend sur la page de choix d'utilisateur (en cachant le warning)
CachePopUpWarning();
GoMenuUtilisateur();
```

Algorithm 21: OnEnable()
Fonction appelée lors de l'ouverture de la scène MenuUtilisateurs

```
SetPseudos(); // On affiche les bons pseudos lorsque l'on ouvre le menu utilisateurs
// On lance le chronomètre pour calculer la vitesse de selection de l'utilisateur
timerSelection = new Stopwatch();
timerSelection.Start();
```

Algorithm 22: void Update()

```
// Detecter lorsqu'on appui sur la touche entrée pour valider le pseudo entré
if (Input.GetKeyDown(KeyCode.Return))  ValiderPseudo();
```

3.4.3 Classe CsvReader

Classe permettant de lire un CSV de question/réponses/explications et de récupérer la liste des question/réponses/explications.

Attributs :

- public TextAsset textAssetData : Le CSV à lire en format texte
- public TextAsset textExplications : Le CSV à lire en format texte pour les explications de mots
- public TextAsset textFauxAmis : Le CSV à lire en format texte pour les faux amis de mots
- public List<QuestionReponse> myQr : La liste des question/réponses lu d'après le CSV

— private const int nbRep : Le nombre de proposition de réponse dans chaque ligne du CSV
— private const int nbColonne : Le nombre de colonne par mot de vocabulaire : une colonne question et une colonne réponse correcte en plus

Méthodes :

Algorithm 23: void readCSV

```
string[] data = textAssetData.text.Split(new string[]",","\n", StringSplitOptions.None);// Le CSV sous
   forme de float
// On met en forme les différentes question/réponses
for (int i = 0; i < tableSize; i++) do
    // On parcourt les lignes (càd les différentes questions)
    // Mise en ordre aléatoire des réponses
    int[] indices = new int[nbRep]; // Tableau qui associe la place de la réponse à l'indice de la réponse
    for (int l = 0; l < nbRep; l++) do
        indices[l] = l; // Ils sont tous à la même place que l'ordre dans lequel ils sont cités dans le CSV
    System.Random rnd = new System.Random(i);
    int indGood = rnd.Next(0, nbRep); // On tire au hasard l'indice de la bonne réponse (qui est au début
       dans la première colonne)
    indices[0] = indGood; // On inverse l'indice de la première colonne avec l'indice tiré pour que la bonne
       réponse soit placée à un indice aléatoire
    indices[indGood] = 0;
    // On ajoute donc la question/réponses tout juste lue (avec la place de la bonne réponse modifiée)
    myQr.Add(new QuestionReponse() Question = data[(nbColonne)*i], Reponses = new
     string[nbRep]data[nbColonne*i+1+indices[0]],
     data[nbColonne*i+1+indices[1]],data[nbColonne*i+1+indices[2]],data[nbColonne*i+1+indices[3]],
     ReponseCorrecte = data[nbColonne*i+1], IndReponseCorrecte = indGood);
```

Algorithm 24: void readCSVExplications
associer l'explication de bonne réponse et les explications supplémentaires des réponses pour chaque question

```
// On ajoute l'explication de bonne réponse au courant et les explications supplémentaires pour les autres
   réponses s'ils existent
string[] data = textExplications.text.Split(new string[]"\n", StringSplitOptions.None); // séparer les lignes
for (int i = 0; i < data.Length; i++) do
    string[] explications=data[i].Split(',');// séparer chaque case d'une ligne
    string[] explicationBR=explications[0].Split('\');// l'explication de bonne réponse, séparer
       mot\nature\exemple d'une explication
    // ajouter l'explication de bonne réponse
    myQr[i].explicationBonneReponse=new
     Explication()mot=explicationBR[0],nature=explicationBR[1],exemple=explicationBR[2];
    if (explications.Length>1) then
        // existe les explications pour les autres réponses
        // ajouter les explications pour les autres réponses
        myQr[i].explicationsSupplement=new Explication[explications.Length-1];
        for (int j=1;j<explications.Length;j++) do
            string[] explication=explications[j].Split('\');
            myQr[i].explicationsSupplement[j-1]=new
             Explication()mot=explication[0],nature=explication[1],exemple=explication[2];
```

Algorithm 25: void readCSVFauxAmis
associer faux ami d'une question

```
string[] data = textFauxAmis.text.Split(new string[]"\n", StringSplitOptions.None) ;
for (int i = 0 ; i < data.Length ; i++) do
    string[] ligne_fauxami=data[i].Split(',') ;// séparer ligne correspondant, faux ami,traduction
    // ajouter faux-ami pour le mot correspondant
    myQr [int.Parse(ligne_fauxami[0])].fauxAmi = newFauxAmi(){fauxami =
     ligne_fauxami[1], traduction = ligne_fauxami[2]} ;
```

3.4.4 Classe ReponseScript

Classe permettant de gérer la réponse de l'utilisateur à un QCM.

Attributs :

- **public bool isCorrect** : Indique si la réponse indiquée par l'utilisateur est correcte ou non.
- public int indReponseCorrespond : indice de reponse associé

Algorithm 26: void ReponseQCM
Appellée lorsque l'utilisateur répond à un QCM

```
ficheManager.indReponseSelectionnee=indReponseCorrespond ;// update l'indice de réponse selectionné par
   utilisateur
if (isCorrect) then
    quizManager.ReponduQCM(true) ; // On traite la réponse de l'utilisateur dans le quizManager
       (affichage des couleurs etc.)
    quizManager.DecrementeNbQuestAvantNouvelle() ;// On augmente l'approfondissement des connaissances
    quizManager.RepEntreeOK = true ; // Indiquer dans quizManager que l'utilisateur a bien répondu
else
 e
lse  quizManager.ReponduQCM(false) ; // On traite la réponse de l'utilisateur dans le quizManager
   (affichage des couleurs etc.)
quizManager.IncrementeNbQuestAvantNouvelle() ;// On augmente le renforcemment des connaissances
quizManager.RepEntreeOK = false ; // Indiquer dans quizManager que l'utilisateur a mal répondu
```

3.4.5 Classe QuestionReponse

Classe permettant de gérer la réponse de l'utilisateur à un QCM.

Attributs :

- public string Question
- public string[] Reponses
- public int IndReponseCorrecte
- public string ReponseCorrecte
- public Explication explicationBonneReponse
- public Explication[] explicationsSupplement : liste d'explications pour chaque réponse
- public FauxAmi fauxAmi

3.4.6 Classe QuizManager

Classe permettant de gérer le quiz.

Attributs :

- **[SerializeField] private Color correctCol, wrongCol** : Les couleurs correspondant aux réponses fausses (rouge) et à la bonne (vert)
- **public List<QuestionReponse> QnA** : Liste des questions/réponses
- **public GameObject[] options** : Liste des 4 boutons pour le QCM
- **public int QuestionCourrante** : Indice de la question courrante (càd qui est en train d'être posée)

— **public Text TxtQuestionQCM** : Le texte qui formule la question pour le QCM (pour pouvoir le changer)
— **public Text TxtQuestionEnt** : Le texte qui formule la question pour la réponse entière (pour pouvoir le changer)
— **public VocabUtilisateur vocUt** : L'objet gérant les données associées au vocabulaire de l'utilisateur
— **public GameObject NextButtonQCM** : Bouton "question suivante" pour les QCM
— **public GameObject FicheButtonQCM** : Bouton permettant d'accéder à la fiche pour les QCM
— **public GameObject NextButtonEntier** : Bouton "question suivante" pour les questions à réponse entière
— **public GameObject FicheButtonEntier** : Bouton permettant d'accéder à la fiche pour les questions à réponse entière
— **public GameObject PanneauFicheQCM** : Le panneau détaillant la fiche pour les QCM
— **public GameObject PanneauFicheEntier** : Le panneau détaillant la fiche pour les questions à réponse entière
— **public GameObject PanneauQEntier** : Le panneau contenant la question et le champs de saisie pour les questions à réponse entière
— **public GameObject PanneauQCM** : Le panneau contenant la question et les réponses pour les QCM
— **public GameObject OkButton** : Le bouton OK permettant de valider la réponse dans les questions à réponse entière
— **public GameObject EntreeRep** : Le champs de saisie de la réponse dans les questions à réponse entière
— **public GameObject BonneRep** : Le champs de saisie (en mode lecture) permettant d'afficher la bonne réponse lorsque l'utilisateur s'est trompé dans les questions à réponse entière
— **public bool inInitialisation** : Indique si on est encore dans l'initialisation (true) ou non (false)
— **public bool RepEntreeOK** : Indique si la réponse entrée est bonne (true) ou non (false)
— **public int NbQuestAvantNouvelle** : Le nombre de questions nécessaires sur des mots déjà rencontrés avant une question sur un mot non encore rencontré (modifié seulement lorsque le cycle en cours est terminé)
— **public int NbQuestAvantNouvelleTemp** : Le nombre de questions nécessaires sur des mots déjà rencontrés avant une question sur un mot non encore rencontré (modifié à chaque réponse de l'utilisateur)
— **public int NbAncienneQuestion** : Le nombre de questions posées sur des mots déjà rencontrés depuis la dernière rencontre d'un nouveau mot
— **public int NbAncienneQuestionTemp** : Pour permettre de mettre à jour NbAncienneQuestion que quand l'utilisateur a répondu et pas avant (s'il a juste vu la question)
— **public int NbNouvelleQuestion** : Le nombre de questions posées sur des mots non encore rencontrés depuis la dernière rencontre d'un ancien mot
— **public int NbNouvelleQuestionTemp** : Pour permettre de mettre à jour NbNouvelleQuestion que quand l'utilisateur a répondu et pas avant (s'il a juste vu la question)
— **public int NbQuestionsTotales** : Le nombre de questions rencontrées au total
— **public bool inQCM** : Indique si la question à laquelle on vient de répondre est un QCM (true) ou non (false)
— **public List<int> IndQuestNonRencontrees** : Indices des questions non encores posées
— **private Stopwatch timer** : Le chronomètre permettant de mesurer le temps de sélection ou d'entrée de texte de l'utilisateur
— **private string ReponseUtilisateur** : La réponse entrée par l'utilisateur lors d'une question à réponse entière **public string TypeQuestion** : Le type de la question en cours (QCM ou Entier)
— **private int nbCarRep** : Le nombre de caractères entrés par l'utilisateur pour sa réponse en champs de saisie lors d'une question à réponse entière
— **private bool loadAncienne Partie** : Booléen qui indique si on a load une ancienne partie (true) ou non (false)
— **private bool Repondu** : Booléen qui indique si l'utilisateur a répondu à la question courrante ou non

Méthodes :

Algorithm 27: void start

```
// On commence par lire le CSV contenant les mots de vocabulaire
CsvLu.readCSV();
CsvLu.readCSVExplications();
CsvLu.readCSVFauxAmis();
QnA = CsvLu.myQr; // On récupère la liste de question/réponses obtenue avec le CsvReader
// Le nombre de mots de vocabulaire est donné par la taille de la liste de question/réponses
PlayerPrefs.SetInt("NbMotsVocab", QnA.Count); // On enregistre cette donnée dans les PlayerPrefs afin de
   pouvoir y accéder dans tous les scripts
// Pour le calcul de la vitesse de sélection timer = new Stopwatch();
// pour avoir l'indice de reponse selectionnee
for (int i = 0; i < options.Length; i++)
 options[i].GetComponent<ReponseScript>().indReponseCorrespond=i;
Initialisation();
```

Algorithm 28: void InitialiseUserInitialisation

Fonction permettant d'initialiser les valeurs des attributs pour l'initialisation dans le cas d'un tout nouveau joueur

```
// On initialise les attributs caractérisant la dynamique du quiz NbQuestAvantNouvelle = 1;
NbQuestAvantNouvelleTemp = 1; // Le nombre de questions nécessaires sur des mots déjà rencontrés avant
   une question sur un mot non encore rencontré (modifié à chaque réponse de l'utilisateur)
NbAncienneQuestion = 0; // Le nombre de questions posées sur des mots déjà rencontrés depuis la dernière
   rencontre d'un nouveau mot
NbNouvelleQuestion = 0; // Le nombre de questions posées sur des mots pas encore rencontrés depuis la
   dernière rencontre d'un ancien mot
NbQuestionsTotales = 0; // Le nombre de questions rencontrées au total
// Instantiation des variables pour l'initialisation
nbBienRep = 0; // Nombre de fois où l'utilisateur a bien répondu au QCM puis à la même question en entier
nbMalRep = 0; // Nombre de fois où l'utilisateur a bien répondu au QCM puis à mal répondu à la même
   question en entier
numIte = 0; // Numéro de l'itération dans l'initialisation
// On initialise les indices des questions non encore rencontrées
IndQuestNonRencontrees = new List<int>();
for (int i = 0; i < PlayerPrefs.GetInt("NbMotsVocab"); i++) do
 IndQuestNonRencontrees.Add(i);
inInitialisation = true; // On commence l'initialisation
QuestionCourrante = 0; // Initialisation, la valeur va être modifiée très prochainement
```

Algorithm 29: public void ReponduQCM
Fonction appellée lorsque l'utilisateur vient de répondre à un QCM

```
Data: bool correct : correct vaut true si l'utilisateur a donné la bonne réponse et false sinon
// Calcul de la vitesse de clic de l'utilisateur
timer.Stop();// On arrête le chronomètre
vocUt.dateDerniereRencontre[QuestionCourrante] = DateTime.Now.ToString("MM/dd/yyyy
    HH :mm :ss");// On met à jour la date de dernière rencontre du mot
Repondu = true// On a répondu à la question
TimeSpan timeTaken = timer.Elapsed; // On regarde le temps passé sur le chronomètre
// On transforme la durée obtenue en float (nombre de secondes/minutes/heures écoulées) afin de l'enregistrer
    dans le gérant de l'hésitation
int days, hours, minutes, seconds, milliseconds;
days = timeTaken.Days;
hours = timeTaken.Hours;
minutes = timeTaken.Minutes;
seconds = timeTaken.Seconds;
milliseconds = timeTaken.Milliseconds;
// Temps passé en secondes :
float floatTimeSpan = ((float)days*24*3600) + ((float)hours*3600) + ((float)minutes*60) + (float)seconds
    + ((float)milliseconds/1000); // On enregistre ce temps dans le gérant d'hésitation
hesitationManager.vitessesSelection[QuestionCourrante].Add(floatTimeSpan);
inQCM = true; // On indique que la réponse à laquelle on vient de répondre est un QCM
// On indique que l'on vient de rencontrer cette question, on enlève donc l'indice de cette question à la
    liste d'indice des questions non encore rencontrées
if (IndQuestNonRencontrees.Contains(QuestionCourrante)) then
    // On ne met ça que quand on a répondu à un QCM puisqu'on répond toujours à un QCM avant de répondre à la
        question entière PB si ça change
    IndQuestNonRencontrees.Remove(QuestionCourrante);
vocUt.UpdateNbRencontres(QuestionCourrante); // On indique que l'on a rencontré une fois de plus la
    question
NbQuestionsTotales += 1; // On indique que l'on a rencontré une question de plus
NbAncienneQuestion = NbAncienneQuestionTemp; // On màj le NbAncienneQuestion mtn que l'utilisateur a
    répondu
NbNouvelleQuestion = NbNouvelleQuestionTemp; // On màj le NbNouvelleQuestion mtn que l'utilisateur a
    répondu
float hesite = hesitationManager.EstimationHesitationQCM(floatTimeSpan); // On estime l'hésitation de
    l'utilisateur
vocUt.UpdateProbaAcquisitionQCM(QuestionCourrante, correct, hesite);// On met à jour les probas
    d'acquisition
// On affiche la bonne réponse en vert et les autres en rouge
for (int i = 0; i < options.Length; i++) do
    // Rendre les boutons non cliquables
    options[i].GetComponent<Button>().interactable = false;
    if (options[i].GetComponent<ReponseScript>().isCorrect == true) then
        // Change la couleur du bouton (en mode non cliquable)
        var colors = options[i].GetComponent<Button>().colors;
        colors.disabledColor = correctCol;
        options[i].GetComponent<Button>().colors = colors;
    else
        // Change la couleur du bouton (en mode non cliquable)
        var colors = options[i].GetComponent<Button>().colors;
        colors.disabledColor = wrongCol;
        options[i].GetComponent<Button>().colors = colors;
if (inInitialisation and RepEntreeOK == false) then
    numIte += 1; // L'utilisateur a fini de répondre à cette question et on peut donc incrémenter le numéro
        de l'itération de l'initialisation
// On affiche les boutons "suivant" permettant de passer à la question suivante et "fiche d'aide" permettant
    de consulter la fiche
NextButtonQCM.SetActive (true);
FicheButtonQCM.SetActive (true);
```

Algorithm 30: void ReponduEntier

Fonction appellée lorsque l'utilisateur vient de répondre à une question à réponse entière

```
// Calcul de la vitesse d'entrée de texte de l'utilisateur
timer.Stop() ; // On arrête le chronomètre
vocUt.dateDerniereRencontre[QuestionCourrante] = DateTime.Now.ToString("MM/dd/yyyy
    HH :mm :ss") ; // On met à jour la date de dernière rencontre du mot
Repondu = true ; // On a répondu à la question
TimeSpan timeTaken = timer.Elapsed ; // On regarde le temps passé sur le chronomètre
// On transforme la durée obtenue en float (nombre de secondes/minutes/heures écoulées) afin de l'enregistrer
    dans le gérant de l'hésitation
int days, hours, minutes, seconds, milliseconds ;
days = timeTaken.Days ;
hours = timeTaken.Hours ;
minutes = timeTaken.Minutes ;
seconds = timeTaken.Seconds ;
milliseconds = timeTaken.Milliseconds ;
// Temps passé en secondes :
float floatTimeSpan = ((float)days*24*3600) + ((float)hours*3600) + ((float)minutes*60) + (float)seconds
    + ((float)milliseconds/1000) ; UnityEngine.Debug.Log("Temps d'entrée de texte : " + floatTimeSpan +
    "secondes") ; // On enregistre ce temps dans le gérant d'hésitation
hesitationManager.vitessesEntreeTexte[QuestionCourrante].Add(floatTimeSpan) ;
inQCM = false ; // On indique que la réponse à laquelle on vient de répondre n'est pas un QCM mais une
    question entière
vocUt.UpdateNbRencontres(QuestionCourrante) ; // On indique que l'on a rencontré une fois de plus la
    question
NbQuestionsTotales += 1 ;// On indique que l'on a rencontré une question de plus
NbAncienneQuestion = NbAncienneQuestionTemp ; // On màj le NbAncienneQuestion mtn que l'utilisateur a
    répondu
NbNouvelleQuestion = NbNouvelleQuestionTemp ; // On màj le NbNouvelleQuestion mtn que l'utilisateur a
    répondu
// On affiche le carré en vert si l'utilisateur a entré la bonne réponse et sinon on affiche en rouge son
    carré et la bonne réponse en vert
if (ReponseUtilisateur == QnA[QuestionCourrante].ReponseCorrecte) then
    float hesite = hesitationManager.EstimationHesitationEntier(floatTimeSpan, nbCarRep) ; // On estime
        l'hésitation de l'utilisateur
    vocUt.UpdateProbaAcquisitionQEntier(QuestionCourrante, true, hesite) ; // On met à jour les probas
        d'acquisition
    RepEntreeOK = true ; // On indique que l'utilisateur a donné la bonne réponse
    DecrementeNbQuestAvantNouvelle() ; // On augmente l'étendu de l'apprentissage (plutôt que le
        renforcement des connaissances)
    // Change la couleur du bouton (en mode non cliquable)
    var colors = EntreeRep.GetComponent<InputField>().colors ; colors.disabledColor = correctCol ;
    EntreeRep.GetComponent<InputField>().colors = colors ;
    // Rendre le champs de saisie et le bouton OK non cliquable
    EntreeRep.GetComponent<InputField>().interactable = false ;
    OkButton.GetComponent<Button>().interactable = false ;
else
```

```
    float hesite = hesitationManager.EstimationHesitationEntier(floatTimeSpan, nbCarRep) ; // On estime
      l'hésitation de l'utilisateur
    vocUt.UpdateProbaAcquisitionQEntier(QuestionCourrante, false, hesite) ; // On met à jour les probas
      d'acquisition
    RepEntreeOK = false ;// On indique que l'utilisateur a donné la mauvaise réponse
    IncrementeNbQuestAvantNouvelle() ;
    // On met la réponse rentrée en rouge et on affiche un deuxième carré contenant la bonne réponse en vert
    var colors = EntreeRep.GetComponent<InputField>().colors ; colors.disabledColor = wrongCol ;
    EntreeRep.GetComponent<InputField>().colors = colors ;
    colors = BonneRep.GetComponent<InputField>().colors ;
    colors.disabledColor = correctCol ;
    BonneRep.GetComponent<InputField>().colors = colors ;
    // Afficher le bouton bonne réponse contenant la bonne réponse dans ce cas
    BonneRep.SetActive(true) ;
    BonneRep.GetComponent<InputField>().text = QnA[QuestionCourrante].ReponseCorrecte ;
    // Rendre les champs de saisie et le bouton OK non cliquable
    EntreeRep.GetComponent<InputField>().interactable = false ;
    BonneRep.GetComponent<InputField>().interactable = false ;
    OkButton.GetComponent<Button>().interactable = false ;
if (inInitialisation) then
    numIte += 1 ;// L'utilisateur a fini de répondre à cette question et on peut donc incrémenter le numéro
      de l'itération de l'initialisation
    if (RepEntreeOK) then
        nbBienRep += 1 ;
    else
        nbMalRep += 1 ;
// On affiche les boutons "suivant" permettant de passer à la question suivante et "fiche d'aide" permettant
   de consulter la fiche
NextButtonEntier.SetActive (true) ;
FicheButtonEntier.SetActive (true) ;
```

Algorithm 31: void genererQuestion
Fonction permettant de choisir une question et de la générer

```
// NbQuestAvantNouvelle doit être non nul mais peut être supérieur à 0 (exprime le nb de questions ancienne
   avant d'en avoir une nouvelle) ou inférieur à 0 (exprime -le nb de questions nouvelle avant d'en avoir une
   ancienne)
if (NbQuestAvantNouvelle > 0) then
    if (NbAncienneQuestion < NbQuestAvantNouvelle) then
        // Dans ce cas, on pose encore une question sur un mot déjà rencontré
        NbAncienneQuestionTemp = NbAncienneQuestion + 1; // On ne le change que si l'utilisateur a
           répondu à la question et pas tout de suite
        // On choisit la question déjà rencontrée avec la plus faible proba d'acquisition
        // Calculer les probabilités d'acquisition actuelles (selon le temps passé depuis la dernière rencontre
           du mot et le nombre de rencontres de ce mot)
        float[] probaAcquisActuelles = new float[PlayerPrefs.GetInt("NbMotsVocab")];
        probaAcquisActuelles = vocUt.UpdateProbaAcquisitionPowLawPrac();
        // On cherche l'indice de la question avec la plus faible proba d'acquisition actuelle
        var minProba = -1.0;
        var minInd = -1;
        for (int i = 0; i < PlayerPrefs.GetInt("NbMotsVocab"); i++) do
            if (vocUt.nbRencontres[i] > 0 and i != QuestionCourrante) then
                // On ne considère la question que si elle a déjà été posée et qu'elle est différente de la
                   dernière question posée
                if (minInd == -1) then
                    // On initialise le minimum
                    minProba = probaAcquisActuelles[i];
                    minInd = i;
                if (minProba > probaAcquisActuelles[i]) then
                    minProba = probaAcquisActuelles[i];
                    minInd = i;
        if (minInd == -1) then
            // Si aucune question n'a encore été posée
            NbAncienneQuestionTemp = NbAncienneQuestion - 1; // On n'incrémente pas le nombre de
               questions anciennes
            if (QuestionCourrante != 0) then
                minInd = 0; // On pose la première question
            else
                minInd = 1; // On pose la seconde question
        QuestionCourrante = minInd;
    else
        // Si NbAncienneQuestion >= NbQuestAvantNouvelle
        // Dans ce cas, on pose une question sur un mot non encore rencontré
        NbAncienneQuestionTemp = 0; // On réinitialise le nombre de questions posées sur des mots déjà
           rencontrés depuis la dernière rencontre d'une nouvelle question
```

```
        if (IndQuestNonRencontrees.Count == 0) then
            // Si toutes les questions ont déjà été posées, on prend celle de proba d'acquisition minimum
            // Calculer les probabilités d'acquisition actuelles (selon le temps passé depuis la dernière
               rencontre du mot et le nombre de rencontres de ce mot)
            float[] probaAcquisActuelles = new float[PlayerPrefs.GetInt("NbMotsVocab")];
            probaAcquisActuelles = vocUt.UpdateProbaAcquisitionPowLawPrac();
            // On cherche l'indice de la question avec la plus faible proba d'acquisition actuelle
            var minProba = -1.0;
            var minInd = -1;
            // On choisit la question avec la proba d'acquisition la plus faible parmi toutes sauf la question
               qui vient d'être posée
            for (int i = 0; i < PlayerPrefs.GetInt("NbMotsVocab"); i++) do
                if (i != QuestionCourrante) then
                    // On ne considère la question que si elle est différente de la dernière question posée
                    if (minInd == -1) then
                        // On initialise le minimum
                        minProba = probaAcquisActuelles[i];
                        minInd = i;
                    if (minProba > probaAcquisActuelles[i]) then
                        minProba = probaAcquisActuelles[i];
                        minInd = i;
            QuestionCourrante = minInd;
        else
            // On choisit au hasard une question non rencontrée
            System.Random rnd = new System.Random();
            int IndListe = rnd.Next(0, IndQuestNonRencontrees.Count); // Créer un numéro entre 0 et la
               taille de la liste
            QuestionCourrante = IndQuestNonRencontrees[IndListe];
        NbQuestAvantNouvelle = NbQuestAvantNouvelleTemp; // On met à jour les caractèristiques du
           nouveau cycle maintenant que l'on en commence un nouveau (et pas avant)
    if (NbQuestAvantNouvelle < 0) then
        if if(NbNouvelleQuestion < (-1)*NbQuestAvantNouvelle) then
            // Dans ce cas, on pose encore une question sur un mot non encore rencontré
            NbNouvelleQuestionTemp = NbNouvelleQuestion + 1; // On ne le change que si l'utilisateur a
               répondu à la question et pas tout de suite
            if (IndQuestNonRencontrees.Count == 0) then
                // Si toutes les questions ont déjà été posées, on prend celle de proba d'acquisition minimum
                // Calculer les probabilités d'acquisition actuelles (selon le temps passé depuis la dernière
                   rencontre du mot et le nombre de rencontres de ce mot)
                float[] probaAcquisActuelles = new float[PlayerPrefs.GetInt("NbMotsVocab")];
                probaAcquisActuelles = vocUt.UpdateProbaAcquisitionPowLawPrac();
                // On cherche l'indice de la question avec la plus faible proba d'acquisition actuelle
                var minProba = -1.0;
                var minInd = -1; // On choisit la question avec la proba d'acquisition la plus faible parmi toutes
                   les questions différentes de la dernière posée
                for (int i = 0; i < PlayerPrefs.GetInt("NbMotsVocab"); i++) do
                    if (i != QuestionCourrante) then
                        // On ne considère la question que si elle est différente de la dernière question posée
                        if (minInd == -1) then
                            // On initialise le minimum
                            minProba = probaAcquisActuelles[i];
                            minInd = i;
                        if (minProba > probaAcquisActuelles[i]) then
                            minProba = probaAcquisActuelles[i];
                            minInd = i;
                QuestionCourrante = minInd;
            else
```

```
        // On choisit au hasard une question non rencontrée
        System.Random rnd = new System.Random();
        int IndListe = rnd.Next(0, IndQuestNonRencontrees.Count); // Créer un numéro entre 0 et la taille de la liste
        QuestionCourrante = IndQuestNonRencontrees[IndListe];
    else
        // Si (-1)*NbNouvelleQuestion >= NbQuestAvantNouvelle
        // Dans ce cas, on pose une question sur un mot déjà rencontré
        NbNouvelleQuestionTemp = 0;// On réinitialise le nombre de questions posées sur des mots déjà rencontrés depuis la dernière rencontre d'une nouvelle question
        // On choisit la question déjà rencontrée avec la plus faible proba d'acquisition
        // Calculer les probabilités d'acquisition actuelles (selon le temps passé depuis la dernière rencontre du mot et le nombre de rencontres de ce mot)
        float[] probaAcquisActuelles = new float[PlayerPrefs.GetInt("NbMotsVocab")];
        probaAcquisActuelles = vocUt.UpdateProbaAcquisitionPowLawPrac();
        // On cherche l'indice de la question avec la plus faible proba d'acquisition actuelle
        var minProba = -1.0;
        var minInd = -1;
        for (int i = 0; i < PlayerPrefs.GetInt("NbMotsVocab"); i++) do
            if (vocUt.nbRencontres[i] > 0 and i != QuestionCourrante) then
                // On ne considère la question que si elle a déjà été posée et qu'elle est différente de la dernière question posée
                if (minInd == -1) then
                On initialise le minimum  minProba = probaAcquisActuelles[i];
                minInd = i;
                if (minProba > probaAcquisActuelles[i]) then
                    minProba = probaAcquisActuelles[i];
                    minInd = i;
        if (minInd == -1) then
            // Si aucune question n'a encore été posée
            NbAncienneQuestionTemp = NbAncienneQuestion - 1; // On n'incrémente pas le nombre de questions anciennes
            if (QuestionCourrante != 0) then
                minInd = 0;
            else
                minInd = 1;
        QuestionCourrante = minInd;
        NbQuestAvantNouvelle = NbQuestAvantNouvelleTemp;// On met à jour les caractèristiques du nouveau cycle maintenant que l'on en commence un nouveau (et pas avant)
// Si la question a déjà été rencontrée et que l'utilisateur avait bien répondu à ce moment là, on propose alors la question sous forme Entier (et pas QCM) //PB a changer plus tard
if (vocUt.nbRencontres[QuestionCourrante] > 0 and vocUt.probaAcquisition[QuestionCourrante] > 0.75) then
    TypeQuestion = "Entier";
else
    TypeQuestion = "QCM";
// On affiche la question sélectionnée
afficherPanneauEnFctTypeQuestion();
```

Algorithm 32: void afficherPanneauEnFctTypeQuestion
Fonction permettant d'afficher la question sélectionnée et de lancer le chronomètre de temps de réponse

```
if (TypeQuestion == "QCM") then
    // La question choisie est un QCM
    // On affiche le panneau de QCM et pas de question Entier
    PanneauQEntier.SetActive(false);
    PanneauQCM.SetActive(true);
    TxtQuestionQCM.text = "Traduisez le mot : _n" + QnA[QuestionCourrante].Question;// La question
      est bien celle sélectionnée
    genererReponses();// On génère les réponses possibles de ce QCM
else
    // La question choisie est une question dont la réponse doit être entrée entièrement
    // On affiche le panneau de question Entier et pas de QCM
    PanneauQEntier.SetActive(true);
    PanneauQCM.SetActive(false);
    // On fait en sorte que le champs de saisie soit automatiquement sélectionné (pas besoin de cliquer dessus)
    EntreeRep.GetComponent<InputField>().Select();
    // Pour pouvoir lire ce qu'écrit l'utilisateur
    EntreeRep.GetComponent<InputField>().onEndEdit.AddListener(delegate
     inputBetValue(EntreeRep.GetComponent<InputField>()); );
    TxtQuestionEnt.text = "Traduisez le mot : \n" + QnA[QuestionCourrante].Question; // La question
      est bien celle sélectionnée
Repondu = false;// On n'a pas encore répondu à la question
// On lance le timer pour la prochaine question
timer.Reset();
timer.Start();
```

Algorithm 33: void genererReponses
Fonction permettant de générer les réponses pour le QCM et de leur associer la véracité de la réponse

```
for (int i = 0; i < options.Length; i++) do
    options[i].GetComponent<ReponseScript>().isCorrect = false;
    options[i].transform.GetChild(0).GetComponent<Text>().text = QnA[QuestionCourrante].Reponses[i];
    if (QnA[QuestionCourrante].IndReponseCorrecte == i) then
        options[i].GetComponent<ReponseScript>().isCorrect = true;
```

Algorithm 34: void AfficherFiche
Fonction d'affichage de la fiche d'aide

```
if (TypeQuestion == "QCM") then
    ficheManager.QR=QnA[QuestionCourrante];
    ficheManager.updateContenuFicheAideQCM();
    PanneauFicheQCM.SetActive(true);
else
    ficheManager.QR=QnA[QuestionCourrante];
    ficheManager.updateContenuFicheAideEnt(ReponseUtilisateur);
    PanneauFicheEntier.SetActive(true);
    // On doit cacher le champs de saisie de réponse, le champs de la bonne réponse et le bouton OK
    EntreeRep.SetActive(false);
    OkButton.SetActive(false);
    BonneRep.SetActive(false);
```

Algorithm 35: void CacherFiche
Fonction pour cacher la fiche d'aide

```
if PanneauFicheQCM.SetActive(false) ;
then TypeQuestion == "QCM"
    PanneauFicheEntier.SetActive(false) ;
    // On doit faire réapparaître le champs de saisie de réponse et le bouton OK
    EntreeRep.SetActive(true) ;
    // On doit aussi faire réapparaître le champs de la bonne réponse si il était précedemment affiché (càd si
       l'utilisateur avait mal répondu)
    if (RepEntreeOK == false) then
        BonneRep.SetActive(true) ; // Le réafficher seulement si on avait entré une mauvaise réponse
```

Algorithm 36: void inputBetValue
Fonction permettant de lire la saisie de l'utilisateur

Data: InputField userInput
ReponseUtilisateur = userInput.text ; nbCarRep = userInput.text.Length + 1 ;

Algorithm 37: void IncrementeNbQuestAvantNouvelle
Fonction pour incrémenter le nombre d'ancienne question rencontré

```
// NbQuestAvantNouvelle doit être non nul mais peut être supérieur à 0 (exprime le nb de questions ancienne
   avant d'en avoir une nouvelle) ou inférieur à 0 (exprime -le nb de questions nouvelle avant d'en avoir une
   ancienne)
if (NbQuestAvantNouvelleTemp < 10) then
    NbQuestAvantNouvelleTemp += 1 ;
if (NbQuestAvantNouvelleTemp == 0) then
    NbQuestAvantNouvelleTemp = 1 ;
```

Algorithm 38: void Initialisation

Permet de faire l'initialisation afin de cibler le niveau de l'utilisateur en lui posant les premières questions

```
var nbIteMax = 4 ;// Nombre maximum d'itérations avant de génerer les questions "normalement"
// On fait l'initialisation jusqu'à ce que l'on ai rencontré une fois les deux cas de figure ou jusqu'à un
   nombre défini d'itération
if ((nbBienRep < 1 or nbMalRep < 1) and numIte <= nbIteMax) then
    if (loadAnciennePartie) then
        loadAnciennePartie = false ; if (inQCM and RepEntreeOK) then
            // Dans ce cas on doit poser la même question que la dernière rencontrée en entier
            // Si l'utilisateur clique sur la bonne réponse, on lui repose la question sous forme d'écriture
               complète
            TypeQuestion = "Entier" ;
            // On garde la même QuestionCourrante
            // On affiche la question choisie
            afficherPanneauEnFctTypeQuestion() ;
        else
            // Si on a mal répondu au QCM ou que la dernière question était une question entière, on pose un
               nouveau QCM
            Initialisation() ;
    else
        // Sinon, on commence toujours par poser la question sous forme de QCM
        TypeQuestion = "QCM" ;
        QuestionCourrante =
            ((int)Math.Floor((double)numIte*(PlayerPrefs.GetInt("NbMotsVocab")-1)/nbIteMax)) ; // On
            prend des mots espacés dans la base de question (du mot numéro 0 au dernier d'indice (nbMotVocab-1)
            puisque les indices commencent à 0)
        // On affiche la question choisie
        afficherPanneauEnFctTypeQuestion() ;
else
    inInitialisation = false ;// On n'est plus dans l'initialisation
    // Une fois l'initialisation terminée, on génère les questions comme expliqué dans le cahier des charges
    genererQuestion() ;
```

Algorithm 39: InitialisationReponduQCM

```
if (RepEntreeOK) then
    // Si l'utilisateur clique sur la bonne réponse, on lui repose la question sous forme d'écriture complète
    TypeQuestion = "Entier" ;
    // On garde la même QuestionCourrante
    // On affiche la question choisie
    afficherPanneauEnFctTypeQuestion() ;
else
    Initialisation() ; // On continue l'initialisation
```

Algorithm 40: void Update

Fonction appelée toute les frame

```
// Detecter lorsqu'on appui sur la touche entrée if (Input.GetKeyDown(KeyCode.Return))
if (TypeQuestion == "Entier" and Repondu == false) then
    // Si on n'a pas encore répondu à la question entière, ça valide la réponse quand on appui sur entrée
    // Lorsque l'on appui sur la touche entrée, ça valide la réponse entrée
    ReponduEntier() ;
else
    if (Repondu) then
        // Si on appui sur entrée lorsqu'on a déjà répondu à la question (entière ou QCM), ça passe à la
           question suivante
        questionSuivante() ;
```

Algorithm 41: void OnEnable
Fonction appelée lors de l'ouverture de la scène QCM

```
// On charge les données sauvegardées
// Chargement des données de l'utilisateur en question
if (PlayerPrefs.HasKey("NumJoueur")) then
    var NumJoueur = PlayerPrefs.GetInt("NumJoueur");// On a le num du joueur qui joue, il faut ensuite
       charger ses données
    // On charge les données statistiques du joueur concerné
    UserStatsloadedData = DataSaver.loadData < UserStats >
     ("Statsjoueur" + PlayerPrefs.GetInt("NumJoueur")); if (loadedData == null or
     EqualityComparer<UserStats>.Default.Equals(loadedData, default(UserStats))) then
        // On initialise les données de vocabulaire de l'utilisateur
        vocUt.Initialise();
        hesitationManager.NivDefaut();
    else
        // Si il y a quelque chose dans les données chargées, on les charge
        // Mise à jour des données du joueur
        vocUt.probaAcquisition = new float[PlayerPrefs.GetInt("NbMotsVocab")];
        vocUt.nbRencontres = new int[PlayerPrefs.GetInt("NbMotsVocab")];
        vocUt.dateDerniereRencontre = new string[PlayerPrefs.GetInt("NbMotsVocab")];
        Array.Copy(loadedData.probaAcquisition, vocUt.probaAcquisition,
         PlayerPrefs.GetInt("NbMotsVocab")); Array.Copy(loadedData.nbRencontres,
         vocUt.nbRencontres, PlayerPrefs.GetInt("NbMotsVocab"));
        for (int i = 0; i < PlayerPrefs.GetInt("NbMotsVocab"); i++) do
            if (loadedData.dateDerniereRencontre[i] != null) then
                // On ne charge que les données qui existent (les autres restent vides mais on ne les consultera
                   pas donc pas de problèmes)
                vocUt.dateDerniereRencontre[i] = loadedData.dateDerniereRencontre[i];
        hesitationManager.nivSelection = loadedData.nivSelection;
        hesitationManager.nivEntreeTexte = loadedData.nivEntreeTexte;
    // On charge les données d'initialisation du joueur concerné
    UserInitialisation loadedDataInit = DataSaver.loadData<UserInitialisation>("Initialisation_Joueur" +
     PlayerPrefs.GetInt("NumJoueur"));
    if (loadedDataInit == null or EqualityComparer<UserInitialisation>.Default.Equals(loadedDataInit,
     default(UserInitialisation))) then
        // On indique que l'on n'a pas load d'ancienne partie (c'est une toute nouvelle partie)
        loadAncPartie = false;
        // On initialise les attributs du tout nouvel utilisateur pour qu'il puisse commencer sa partie
        InitialiseUserInitialisation();
    else
        // Si il y a quelque chose dans les données chargées, on les charge
        // Mise à jour des données du joueur
        nbBienRep = loadedDataInit.nbBienRep;
        nbMalRep = loadedDataInit.nbMalRep;
        numIte = loadedDataInit.numIte;
        inInitialisation = loadedDataInit.inInitialisation;
        RepEntreeOK = loadedDataInit.RepEntreeOK;
        NbQuestAvantNouvelle = loadedDataInit.NbQuestAvantNouvelle;
        NbAncienneQuestion = loadedDataInit.NbAncienneQuestion;
        NbNouvelleQuestion = loadedDataInit.NbNouvelleQuestion;
        NbQuestionsTotales = loadedDataInit.NbQuestionsTotales;
        inQCM = loadedDataInit.inQCM;
        QuestionCourrante = loadedDataInit.QuestionCourrante;
        TypeQuestion = loadedDataInit.TypeQuestion;
        NbAncienneQuestionTemp = loadedDataInit.NbAncienneQuestionTemp;
        NbNouvelleQuestionTemp = loadedDataInit.NbNouvelleQuestionTemp;
        IndQuestNonRencontrees = new List<int>();
        IndQuestNonRencontrees = loadedDataInit.IndQuestNonRencontrees;// On indique que l'on a load
           une ancienne partie (possiblement en cours d'initialisation)
        loadAncPartie = true;
```

```
    hesitationManager.ListesDefaut();
    // On initialise le niveau de vitesse d'entrée de texte de l'utilisateur avec les données récupérées lors
       de sa saisie de pseudo
    if (PlayerPrefs.HasKey("TmpsEntreePseudoJ" + NumJoueur)) then
        var temps = PlayerPrefs.GetFloat("TmpsEntreePseudoJ" + NumJoueur);
        hesitationManager.MajNivEntreeeTextePseudo(temps, PlayerPrefs.GetString("PseudoJ" +
            NumJoueur).Length + 1);// Le nombre de caractères du pseudo + la touche entrée
    // On initialise le niveau de vitesse de sélection de l'utilisateur avec les données récupérées lors de sa
       navigation dans les menus
    // On charge la liste de tempsSelectionMenu du joueur concerné
    UserSelectionMenu loadedDataSelection =
     DataSaver.loadData<UserSelectionMenu>("SelectionMenuJ" + PlayerPrefs.GetInt("NumJoueur"));
    var tempsSelectionMenuTemp = new List<float>();
    tempsSelectionMenuTemp = loadedDataSelection.tempsSelectionMenu;
    // On initialise le niveau de l'utilisateur en ce qui concerne le temps de sélection
    for (int i = 0; i < tempsSelectionMenuTemp.Count; i++) do
        hesitationManager.MajNivSelectionMenu(tempsSelectionMenuTemp[i]);
    // On retire les temps de sélection dans les menus puisqu'ils sont déjà traités
    DataSaver.deleteData("SelectionMenuJ" + NumJoueur);// Efface les données de selection dans les
       menus du joueur
else  UnityEngine.Debug.Log("PB PROBLEME, JOUEUR NON PRECISE!!");
```

3.4.7 Classe FicheManager

Classe permettant de gérer la fiche d'aide.

Attributs :

- **public Text ContenuFicheAideQCM**
- **public Text ContenuFicheAideQEnt**
- **public QuestionReponse QR** : question au courant
- **public int indReponseSelectionnee** : indice de la réponse QCM sélectionnée par l'utilisateur au courant

  les attributs pour détecter la confusion de mots, par ex le suffixe "dom" dans la liste SUFFIXE_NOM_NOM ,si la réponse correcte est "king" (nom) et l'utilisateur écrit "kingdom", alors on peut détecter cette confusion à l'aide de cette liste, pareille pour les autres types de confusions.
- **public static readonly List<string> SUFFIXE_NOM_NOM**(["dom","ship","hood",...])
- **public static readonly List<string> SUFFIXE_NOM_ADJ**
- **public static readonly List<string> SUFFIXE_NOM_VERBE**
- **public static readonly List<string> SUFFIXE_ADJ_NOM**
- **public static readonly List<string> SUFFIXE_ADJ_VERBE**
- **public static readonly List<string> SUFFIXE_ADJ_ADVERBE**
- **public static readonly List<string> SUFFIXE_VERBE_NOM**
- **public static readonly List<string> SUFFIXE_VERBE_ADJ**
- **public string typeConfusion** : peut être : "NOM_NOM","NOM_ADJ","NOM_VERBE","ADJ_NOM", "ADJ_VERBE","ADJ_ADVERBE","VERBE_NOM","VERBE_ADJ"

Méthodes :

Algorithm 42: void updateContenuFicheAideQCM
Fonction applée quand on veut afficher le contenu de fiche aide pour une qcm

```
// afficher toujours d'abord l'explication de la réponse correcte
ContenuFicheAideQCM.text="explication for "+ QR.ReponseCorrecte;
ContenuFicheAideQCM.text+="\n nature de mot : "+QR.explicationBonneReponse.nature;
ContenuFicheAideQCM.text+="\n exemple : "+QR.explicationBonneReponse.exemple;
// si l'utilisateur n'a pas sélectionné la bonne réponse
if (indReponseSelectionnee!=QR.IndReponseCorrecte) then
    ContenuFicheAideQCM.text+="\n vous avez confondu avec :
     "+QR.Reponses[indReponseSelectionnee];
    // s'il y a des explications supplémentaires sur les autres réponses
    if QR.explicationSupplement!=null then
        // cherche s'il y a l'explication supplémentaire sur la réponse que l'utilisateur a sélectionné
        for (inti = 0; i < QR.explicationsSupplement.Length; i++) do
            // afficher l'explication du mot que l'utilisateur a sélectionné
            if (QR.Reponses[indReponseSelectionnee]==QR.explicationsSupplement[i].mot) then
                ContenuFicheAideQCM.text+="\n nature de mot : "+QR.explicationsSupplement[i].nature;
                ContenuFicheAideQCM.text+="\n exemple : "+QR.explicationsSupplement[i].exemple;
                break;
```

Algorithm 43: void updateContenuFicheAideQEnt
Fonction appelée quand on veut afficher le contenu de fiche aide pour une question entière

```
Data: string ReponseUtilisateur
Result: None
// afficher toujours d'abord l'explication de la réponse correcte
ContenuFicheAideQEnt.text="l'explication pour "+ QR.ReponseCorrecte;
ContenuFicheAideQEnt.text+="\n nature de mot : "+QR.explicationBonneReponse.nature;
ContenuFicheAideQEnt.text+="\n exemple : "+QR.explicationBonneReponse.exemple;
// si l'utilisateur n'écrit pas la bonne réponse
if (indReponseSelectionnee!=QR.IndReponseCorrecte) then
    ContenuFicheAideQEnt.text+="\n vous avez confondu avec :
     "+QR.Reponses[indReponseSelectionnee];
    // chercher si l'utilisateur se trompe avec le mot qui a l'explication
    if QR.explicationSupplement!=null then
        for (inti = 0; i < QR.explicationsSupplement.Length; i++) do
            // afficher l'explication du mot l'utilisateur a selectionnée
            if (QR.Reponses[indReponseSelectionnee]==QR.explicationsSupplement[i].mot) then
                ContenuFicheAideQEnt.text+="\n nature de mot : "+QR.explicationsSupplement[i].nature;
                ContenuFicheAideQEnt.text+="\n exemple : "+QR.explicationsSupplement[i].exemple;
                break;
    // si l'utilisateur se trompe avec le faux ami
    if (QR.fauxAmi!=null) then
        if (ReponseUtilisateur==QR.fauxAmi.fauxami) then
            ContenuFicheAideQEnt.text+="\n vous êtes confondu avec le faux ami\n
             "+QR.fauxAmi.fauxami+" traduction : "+QR.fauxAmi.traduction;
    // chercher si l'utilisateur se trompe sur la nature de mot
    if(detecterConfusionNature(ReponseUtilisateur)) ContenuFicheAideEnt.text+="vous êtes vous trompé
     sur la nature de mot"+"("+typeConfusion+")";
```

Algorithm 44: bool detecterConfusionNature

```
Data: string reponseUtilisateur
Result: true si l'utilisateur confondre la nature de mot
char[] charactersRU=ReponseUtilisateur.ToCharArray() ;// characteres reponse utilisateur
char[] charactersRC=QR.ReponseCorrecte.ToCharArray() ;// characteres reponse correcte
if charactersRU.Length<=charactersRC.Length then
    // impossible avoir un suffixe
    return false ;
// vérifier préfixe de reponseUtilisateur est motCorrecte
// comparer caractère par caractère
for (int i=0 ; i<charactersRC.Length ;i++) do
    if (charactersRC[i] !=charactersRU[i]) then
        return false ;
// préfixe de reponseUtilisateur est motCorrecte
// trouver le suffixe,càd les caractères dans reponseUtilisateur après motcorrecte
string suffixe=(new string(charactersRU)).Substring(charactersRC.Length) ;
// détecter le type de confusion
if (QR.explicationBonneReponse.nature=="nom") then
    if (SUFFIXE_NOM_NOM.Contains(suffixe)) then
        typeConfusion="NOM_NOM" ;
        return true ;
    if ( SUFFIXE_NOM_ADJ.Contains(suffixe)) then
        typeConfusion="NOM_ADJ" ;
        return true ;
    if ( SUFFIXE_NOM_VERBE.Contains(suffixe)) then
        typeConfusion="NOM_VERBE" ;
        return true ;
if (QR.explicationBonneReponse.nature=="adjectif") then
    if ( SUFFIXE_ADJ_NOM.Contains(suffixe)) then
        typeConfusion="ADJ_NOM.Contains(suffixe)" ;
        return true ;
    if ( SUFFIXE_ADJ_VERBE.Contains(suffixe)) then
        typeConfusion="ADJ_VERBE" ;
        return true ;
    if ( SUFFIXE_ADJ_ADVERBE) then
        typeConfusion="ADJ_ADVERBE" ;
        return true ;
if (QR.explicationBonneReponse.nature=="verbe") then
    if ( SUFFIXE_VERBE_NOM.Contains(suffixe)) then
        typeConfusion="VERBE_NOM" ;
        return true ;
    if ( SUFFIXE_VERBE_ADJ.Contains(suffixe)) then
        typeConfusion="VERBE_ADJ" ;
        return true ;
```

3.4.8 Classe FauxAmi

Classe permettant de donner faux ami de mot courant et la traduction.

Attributs :

- **public string fauxami**
- **public string traduction**

3.4.9 Classe Explication

Classe permettant de donner l'explication d'un mot

Attributs :

— public string mot : mot
— public string nature : nature de mot(peut être nom,adjectif,verbe ou adverbe)
— public string exemple : donne une utilisation de mot, une phrase anglais(et traduction français)

3.5 Classes pour la modélisation de l'utilisateur

3.5.1 Classe HesitationManager

Classe permettant de gérer l'hésitation de l'utilisateur (au niveau de sa vitesse de sélection, d'entrée de texte).

Attributs :

— **public List<float>[] vitessesSelection** : Tableau de liste : chaque case du tableau correspond à un mot de vocabulaire, pour chaque mot on a la liste des temps mis (à chaque fois qu'on a rencontré ce mot)
— **public List<float>[] vitessesEntreeTexte** : Tableau de liste : chaque case du tableau correspond à un mot de vocabulaire, pour chaque mot on a la liste des temps mis (à chaque fois qu'on a rencontré ce mot)
— **public int nivSelection** : Défini le niveau en terme de vitesse de sélection de l'utilisateur
— **public int nivEntreeTexte** : Défini le niveau en terme de vitesse d'entrée de texte de l'utilisateur
— **private const float alpha = 1** :Paramètre alpha permettant de prendre plus ou moins en compte l'estimation d'hésitation par l'oculomètre et par la vitesse de sélection.
— **public static readonly string[] TousNiv** = "Best", "Good", "Avg+", "Avg-", "Bad+", "Bad-", "Worst" ; D'après le Keystroke-Level Model (KLM), on a les teps suivants :
— **public static readonly float[] TmpsEntreeTexte** = 0.08f, 0.12f, 0.20f, 0.28f, 0.5f, 0.75f, 1.2f : Le temps d'entrée de texte (d'un caractère) en fonction du niveau de l'utilisateur
— **public static readonly float[] TmpsPointage = 0.8f, 1f, 1.1f, 1.3f, 1.4f, 1.5f, 1.6f** : Le temps de pointage à la souris de 0.8 à 1.6 secondes en fonction du niveau de l'utilisateur
— **const float TmpsHftK = 0.4f** : Le temps pour passer du clavier à un autre dispositif (souris) ou au statut inactif (pas sur le clavier ni sur le dispositif) et inversement
— **const float TmpsMental = 1.35f** : Le temps de préparation mentale allant de 1.35 à 1.62 (temps nécessaire à l'utilisateur pour réflechir à sa décision)
— **const float TmpsClicButton = 0.1f** : Le temps pour cliquer ou relaché un bouton (sur la souris)
— **const float TmpsLectureQCM = 0f** : Le temps de lecture de la question QCM est estimé à secondes
— **const float TmpsLectureEntier = 0f** : Le temps de lecture de la question à réponse entière est estimé à secondes
— **private float TmpsAutourSelection = TmpsLectureQCM + TmpsMental + 2*TmpsClicButton** : Ce qui doit être ajouté au temps de pointage pour estimer le temps de réponse du QCM : pour répondre à un QCM on a TmpsLectureQCM + TmpsMental + TmpsPointage + 2*TmpsClicButton
— **private float TmpsAutourEntreeTexte** = TmpsLectureEntier + TmpsMental + TmpsHftK : Ce qui doit être ajouté au temps d'entrée de texte pour estimer le temps de réponse entière

Méthodes :

Algorithm 45: void ListesDefaut
Initialisation des listes de vitesses de sélection et d'entrée de texte (à n'appeler que pour un nouvel utilisateur)

```
vitessesSelection = new List<float>[PlayerPrefs.GetInt("NbMotsVocab")];
vitessesEntreeTexte = new List<float>[PlayerPrefs.GetInt("NbMotsVocab")];
for (int i = 0; i < PlayerPrefs.GetInt("NbMotsVocab"); i++) do
    vitessesSelection[i] = new List<float>();
    vitessesEntreeTexte[i] = new List<float>();
```

Algorithm 46: void NivDefaut
Initialisation des niveaux de l'utilisateur (à n'appeler que pour un nouvel utilisateur)

```
// Valeur par défaut des niveaux de l'utilisateur
nivSelection = -1 ;
nivEntreeTexte = -1 ;
```

Algorithm 47: void ComparaisonVitesseSelection
Fonction permettant d'afficher les différentes vitesses de selection selon le niveau de l'utilisateur

```
string message = "vitessesSelection : " ;
for (int i = 0 ; i < TousNiv.Length ; i++) do
    message += "pour " + TousNiv[i] + " est " + (TmpsAutourSelection + TmpsPointage[i]) + " ; " ;
UnityEngine.Debug.Log(message) ;
```

Algorithm 48: void ComparaisonVitesseEntreeTexte
onction permettant d'afficher les différentes vitesses d'entrée de texte selon le niveau de l'utilisateur

```
Data: int nbCarEntres
string messageNormal = "vitessesEntreeTexte : " ;
string messageAppuiSaisie = "vitessesEntreeTexte avec appui sur barre saisie : " ;
string messageOK = "vitessesEntreeTexte avec appui sur touche OK : " ;
for (int i = 0 ; i < TousNiv.Length ; i++) do
    messageNormal += "pour " + TousNiv[i] + " est " + (TmpsAutourEntreeTexte +
     nbCarEntres*TmpsEntreeTexte[i]) + " ; " ;
    messageAppuiSaisie += "pour " + TousNiv[i] + " est " + (TmpsAutourEntreeTexte +
     (TmpsPointage[i] + 2*TmpsClicButton) + nbCarEntres*TmpsEntreeTexte[i]) + " ; " ;
    messageOK += "pour " + TousNiv[i] + " est " + (TmpsAutourEntreeTexte +
     nbCarEntres*TmpsEntreeTexte[i] + (TmpsHftK + TmpsPointage[i] + 2*TmpsClicButton)) + " ; " ;
UnityEngine.Debug.Log(messageNormal) ;
UnityEngine.Debug.Log(messageAppuiSaisie) ;
UnityEngine.Debug.Log(messageOK) ;
```

Algorithm 49: float EstimationHesitationSelection
Fonction estimant et retournant la probabilité d'hésitation de l'utilisateur en fonction de la vitesse de sélection

```
Data: float temps
Result: float hesitation
MajNivSelection(temps) ;// On met à jour le niveau de l'utilisateur en fonction du temps qu'il a mis à
   répondre
float tempsPredit = TmpsAutourSelection + TmpsPointage[nivSelection] ;
if (temps<tempsPredit) then
    return 0 ; // Si il met moins de temps que ce qui est prévu, il ne présente aucune hésitation
float hesitation = ((temps - tempsPredit)/temps) ;
return hesitation ;
```

Algorithm 50: float EstimationHesitationQCM
Fonction estimant et retournant la probabilité d'hésitation de l'utilisateur (lorsqu'il a répondu à un QCM

```
Data: float temps
// Nous devons prendre en compte la vitesse de sélection et les données oculomètriques
float oculometre = oculometreManager.EstimationHesitationOculometre() ;
float selection = EstimationHesitationSelection(temps) ;
return alpha*oculometre + (1-alpha)*selection ;
```

Algorithm 51: float EstimationHesitationEntier
Fonction estimant et retournant la probabilité d'hésitation de l'utilisateur (lorsqu'il a répondu à une question à réponse entière)

```
Data: float temps, int NbCar
MajNivEntreeeTexte(temps, NbCar) ; // On met à jour le niveau de l'utilisateur en fonction du temps qu'il
    a mis à répondre
UnityEngine.Debug.Log("Niveau d'entrée de texte de l'utilisateur = " + nivEntreeTexte) ;
float tempsPredit = TmpsAutourEntreeTexte + NbCar*TmpsEntreeTexte[nivEntreeTexte] ;
if (temps<tempsPredit) then
    return 0 ; // Si il met moins de temps que ce qui est prévu, il ne présente aucune hésitation
float hesitation = ((temps - tempsPredit)/temps) ;
return hesitation ;
```

Algorithm 52: void MajNivSelection
Fonction permettant de mettre à jour le niveau de l'utilisateur en ce qui concerne la vitesse de sélection

```
Data: float temps
// On met à jour le niveau de sélection de l'utilisateur dès que celui-ci sélectionne une réponse
// On compare alors le temps qu'il a mis avec le temps prédit
if (nivSelection == -1) then
    // Si on n'a pas encore initialisé le niveau de l'utilisateur
    nivSelection = TousNiv.Length-1 ; // On commence par mettre le pire niveau à l'utilisateur (cela va se
        mettre à jour juste après)
// S'il sélectionne plus vite que son temps prédit et que son niveau n'est pas le meilleur, c'est peut-être
    qu'il a un meilleur niveau
while (nivSelection > 0 and temps < TmpsAutourSelection + TmpsPointage[nivSelection]) do
    // Si son niveau n'est pas le meilleur, on augmente de 1 le rang de son niveau
    nivSelection -= 1 ;
```

Algorithm 53: void MajNivEntreeeTexte
Fonction permettant de mettre à jour le niveau de l'utilisateur en ce qui concerne la vitesse d'entrée de texte

```
Data: float temps, int nbCarEntres
Result: None
// On met à jour le niveau d'entrée de texte de l'utilisateur dès que celui-ci entre du texte
// On compare alors le temps qu'il a mis avec le temps prédit
if (nivEntreeTexte == -1) then
    // Si on n'a pas encore initialisé le niveau de l'utilisateur
    nivEntreeTexte = TousNiv.Length-1 ; // On commence par mettre le pire niveau à l'utilisateur (cela va
        se mettre à jour juste après)
// S'il écrit plus vite que son temps prédit et que son niveau n'est pas le meilleur, c'est peut-être qu'il a
    un meilleur niveau
while (nivEntreeTexte > 0 and temps < TmpsAutourEntreeTexte +
 nbCarEntres*TmpsEntreeTexte[nivEntreeTexte]) do
    // Si son niveau n'est pas le meilleur, on augmente de 1 le rang de son niveau
    nivEntreeTexte -= 1 ;
```

Algorithm 54: void MajNivSelectionMenu
Fonction permettant de mettre à jour le niveau de l'utilisateur en ce qui concerne la vitesse de sélection lorsqu'il navigue dans les menus

```
Data: float temps
// On met à jour le niveau de sélection de l'utilisateur dès que celui-ci sélectionne un bouton
// On compare alors le temps qu'il a mis avec le temps prédit par son niveau
if (nivSelection == -1) then
    // Si on n'a pas encore initialisé le niveau de l'utilisateur
    nivSelection = TousNiv.Length-1 ;
while (nivSelection > 0 and temps < TmpsMental + TmpsPointage[nivSelection] + 2*TmpsClicButton)
 do
    nivSelection -= 1 ;
```

Algorithm 55: void MajNivEntreeeTextePseudo
Fonction permettant de mettre à jour le niveau de l'utilisateur en ce qui concerne la vitesse d'entrée de texte lorsqu'il entre son pseudo

```
Data: float temps, int nbCarEntres
// On met à jour le niveau d'entrée de texte de l'utilisateur dès que celui-ci entre son pseudo
// On compare alors le temps qu'il a mis avec le temps prédit
if (nivEntreeTexte == -1) then
    // Si on n'a pas encore initialisé le niveau de l'utilisateur
    nivEntreeTexte = TousNiv.Length-1 ;
while (nivEntreeTexte > 0 and temps < TmpsMental + nbCarEntres*TmpsEntreeTexte[nivEntreeTexte])
 do
    nivEntreeTexte -= 1 ;
```

3.5.2 Classe VocabUtilisateur

Classe permettant de gérer les données associées au vocabulaire de l'utilisateur (statistiques sur son apprentissage et sur son interaction avec le jeu).

Attribut :

- **float[] probaAcquisition** :Un tableau de proba d'acquisition de la même taille que QnA de QuizManager ou myQr de CsvReader (càd le nombre de mots de vocabulaire)
- **int[] nbRencontres** :Un tableau du nombre de fois où le mot a été rencontré de la même taille que QnA de QuizManager ou myQr de CsvReader (càd le nombre de mots de vocabulaire)
- **string[] dateDerniereRencontre** : Un tableau de date de la dernière rencontre au format "MM/dd/yyyy HH :mm :ss" pour pouvoir appliquer la Power Law of Practice
- **const float beta = 0.5f** : Paramètre beta permettant de prendre plus ou moins en compte la correction de la réponse donnée et l'hésitation de l'utilisateur dans la proba d'acquisition du mot de vocabulaire

Méthodes :

Algorithm 56: void Initialise
Initialisation des statistiques (à n'appeler que pour un nouvel utilisateur)

```
probaAcquisition = new float[PlayerPrefs.GetInt("NbMotsVocab")] ;
nbRencontres = new int[PlayerPrefs.GetInt("NbMotsVocab")] ;
// On initialise toutes les données à 0
for (int i = 0 ; i < PlayerPrefs.GetInt("NbMotsVocab") ; i++) do
    probaAcquisition[i] = 0 ;
    nbRencontres[i] = 0 ;
dateDerniereRencontre = new string[PlayerPrefs.GetInt("NbMotsVocab")] ;
```

Algorithm 57: void UpdateProbaAcquisitionQCM
Fonction qui màj la proba d'acquisition lorsque l'on répond à un QCM

```
Data: int indiceQuestion, bool correct, float hesite
// correct vaut true si l'utilisateur a donné la bonne réponse et false sinon
// hesite est déterminé par la probabilité d'hésitation de l'utilisateur, et vaut 1 si l'utilisateur hésite totalement (d'où le 1-cette proba)
int intCorrect = correct ? 1 : 0; // Vaut 1 si true et 0 si false
probaAcquisition[indiceQuestion] = beta*intCorrect + (1-beta)*(1-hesite)*intCorrect; // On veut que ça vaille 0 si l'utilisateur a mal répondu
```

Algorithm 58: float[] UpdateProbaAcquisitionPowLawPrac

```
Result: float[] newProbaAcquisition
float[] newProbaAcquisition = new float[PlayerPrefs.GetInt("NbMotsVocab")];
Array.Copy(probaAcquisition, newProbaAcquisition, PlayerPrefs.GetInt("NbMotsVocab"));// On fait une copy pour ne pas perdre les valeurs de proba d'acquisition au temps de la dernière rencontre
// On converti les dates en DateTime depuis le type string
DateTime[] DerniereRencontreDATE = new DateTime[PlayerPrefs.GetInt("NbMotsVocab")];
TimeSpan[] TempsEntreDerniereDateEtAuj = new TimeSpan[PlayerPrefs.GetInt("NbMotsVocab")];
   // Temps écoulé depuis la dernière rencontre du mot
float[] floatTimeSpan = new float[PlayerPrefs.GetInt("NbMotsVocab")];// Temps écoulé depuis la dernière rencontre du mot en float (en seconde par ex.)
for (int i = 0; i < PlayerPrefs.GetInt("NbMotsVocab"); i++) do
    if (dateDerniereRencontre[i] != null and dateDerniereRencontre[i] != "") then
        // On récupère la date de dernière rencontre en type DateTime
        DerniereRencontreDATE[i] = System.DateTime.ParseExact(dateDerniereRencontre[i],
          "MM/dd/yyyy HH :mm :ss", null);
        // On calcule le temps écoulé entre ce temps et maintenant
        TempsEntreDerniereDateEtAuj[i] = DateTime.Now - DerniereRencontreDATE[i]; // Sous forme HH:mm:ss
        // On transforme la durée obtenue en float (nombre de secondes/minutes/heures écoulées)
        int days, hours, minutes, seconds, milliseconds;
        days = TempsEntreDerniereDateEtAuj[i].Days;
        hours = TempsEntreDerniereDateEtAuj[i].Hours;
        minutes = TempsEntreDerniereDateEtAuj[i].Minutes;
        seconds = TempsEntreDerniereDateEtAuj[i].Seconds;
        milliseconds = TempsEntreDerniereDateEtAuj[i].Milliseconds;
        // Temps obtenu en secondes :
        floatTimeSpan[i] = ((float)days*24*3600) + ((float)hours*3600) + ((float)minutes*60) +
          (float)seconds + ((float)milliseconds/1000);
        floatTimeSpan[i] /= 24*3600; // On met le résultat en jours plutôt comme c'est plus cohérent pour la fonction d'oubli
        // On applique la fonction choisie (ici, courbe de l'oubli)
        float ValApresFct = CourbeOubli(floatTimeSpan[i], nbRencontres[i]);
        // On met à jour la proba d'acquisition actuelle (toujours située entre 0 et 1)
        newProbaAcquisition[i] = ValApresFct * probaAcquisition[i];// On prend également en compte la proba d'acquisition déjà établie au dernier temps de rencontre du mot
return newProbaAcquisition;
```

Algorithm 59: float CourbeOubli
Fonction choisie pour caractériser l'oubli en fonction du temps et du nombre de rappels de la connaissance

```
Data: float t, int F
Result: (float)Math.Exp(-t/F)
// Retourne une valeur entre 0 et 1 comme t > 0 et F > 0 soit t/F > 0 donc -t/F va de -inf à 0 et donc exp(-t/F) va de 0 à 1
```

Algorithm 60: void UpdateNbRencontres
Fonction permettant d'indiquer que l'on a rencontré une fois de plus la question d'indice indiceQuestion

Data: int indiceQuestion
Result: None
nbRencontres[indiceQuestion] += 1

3.5.3 Classe OculometreManager

Classe permettant de générer les données oculometre.

- **public List<Vector2> occulaireHesite** : Défini l'ensemble des données occulaires (vecteur à deux dimensions) de la classe "hésite"
- **List<Vector2> occulaireSur** : Défini l'ensemble des données occulaires (vecteur à deux dimensions) de la classe "sûr"
- **public float MaxTempsDiff** : La différence maximum de temps passé sur un mot entre deux mots
- **public float NbBougRegard** : Le nombre de fois total où l'utilisateur a bougé son regard sur un autre mot divisé par le nombre de propositions (donc la somme du nombre de fois où il a regardé chaque mot divisé par le nombre de propositions)
- **public List<Vector2>[] occulaire** : Défini l'ensemble des données oculaires (vecteur à deux dimensions) de la classe "hésite"à la case 0 et l'ensemble des données oculaires (vecteur à deux dimensions) de la classe "sûr" à la case 1
- **int numClasses = 2** : Il y a deux classes : hésite et sûr
- **public Text xCoord**
- **public Text yCoord**
- **public GameObject GazePoint**
- **public GameObject[] cubes** : Liste des 4 boutons pour le QCM
- **public float _pauseTimer**
- **public Outline _xOutline**
- **public Outline _yOutline**
- **public Camera camera**
- **public float tempo**
- **public int NbRep** : Le nombre de propositions de réponses aux QCM
- **public int DernierCubeRegardé** : L'indice du dernier cube regardé
- **public Stopwatch[] timer** : Les chronomètres des différentes propositions de réponse
- **public TimeSpan[] timeTaken** : Les durées sur chacune des réponses
- **public int NbChangeCube** : Le nombre de fois où l'utilisateur a changé de cube regardé

Algorithm 61: void Start

```
tempo = 0;
camera = Camera.main;
_xOutline = xCoord.GetComponent<Outline>();
_yOutline = yCoord.GetComponent<Outline>();
NbRep = 4; // 4 est le nombre de propositions de réponses
DernierCubeRegardé = -1; // On initialise le cube dernièrement regardé à -1 pour indiquer que c'est le
    début
timer = new Stopwatch[NbRep];// On initialise les timers des différentes propositions
timeTaken = new TimeSpan[NbRep]; // On initialise les durées passées sur les différentes propositions
NbChangeCube = 0;// On initialise le nombre de fois où l'utilisateur a changé de cube regardé
```

Algorithm 62: void Update

```
if (_pauseTimer > 0) then
    _pauseTimer -= Time.deltaTime;
    return;
GazePoint.SetActive(false);
_xOutline.enabled = false;
_yOutline.enabled = false;
GazePoint gazePoint = TobiiAPI.GetGazePoint();
if (gazePoint.IsValid) then
    // Coordonnée yeux de l'eye tracking
    Vector2 gazePosition = gazePoint.Screen;
    yCoord.color = xCoord.color = Color.white;
    Vector2 roundedSampleInput = new Vector2(Mathf.RoundToInt(gazePosition.x),
     Mathf.RoundToInt(gazePosition.y));
    xCoord.text = "x (in px) : " + roundedSampleInput.x;
    yCoord.text = "y (in px) : " + roundedSampleInput.y;
    // Hit box des rectangles sur unity
    Bounds[] bounds = new Bounds[NbRep];
    for (int i = 0; i < NbRep; i++) do
        bounds[i] = cubes[i].GetComponent<BoxCollider2D>().bounds;
    // Scale à la taille de l'ecran global
    Vector3[] origin = new Vector3[NbRep];
    Vector3[] extents = new Vector3[NbRep];
    for (int i = 0; i <NbRep; i++) do
        origin[i] = camera.WorldToScreenPoint(new Vector3(bounds[i].min.x, bounds[i].min.y, 0.0f));
        extents[i] = camera.WorldToScreenPoint(new Vector3(bounds[i].max.x, bounds[i].max.y, 0.0f));
    // Redefinition de la hitbox adapte à la taille de l'ecran global
    Rect[] goodBound = new Rect[NbRep];
    for (int i = 0; i < NbRep; i++) do
        goodBound[i] = new Rect(origin[i].x, origin[i].y, extents[i].x - origin[i].x, extents[i].y - origin[i].y);
            // point de référence (haut gauche), longueur, hauteur
    // Verifie si le point de l'eye tracking est dans le rectangle hitbox
    for (int i = 0; i < NbRep; i++) do
        if (goodBound[i].Contains(new Vector3(roundedSampleInput.x, roundedSampleInput.y, 0))) then
            // Si c'est le ces, changer la couleur du bouton regardé
            tempo += 5f * Time.deltaTime;
            float tempo2 = (float) (Math.Sin(tempo) + 1f) / 2;
            float tempo3 = (float)(Math.Cos(tempo) + 1f) / 2;
            cubes[i].GetComponent<Image>().color = new Color(1f, tempo2, tempo3); // On fait varier la
               couleur du cube regardé
            if (DernierCubeRegardé != i) then
                // Dans ce cas, l'utilisateur a bougé son regard sur un autre cube
                DernierCubeRegardé = i; // On indique que le dernier cube regardé est le cube i
                NbChangeCube += 1; // On a changé de cube regardé
if (Input.GetKeyDown(KeyCode.Space) and gazePoint.IsRecent()) then
    _pauseTimer = 3f;
    GazePoint.transform.localPosition = (gazePoint.Screen - new Vector2(Screen.width, Screen.height) /
     2f) /GetComponentInParent<Canvas>().scaleFactor;
    yCoord.color = xCoord.color = new Color(0 / 255f, 190 / 255f, 255 / 255f);
    GazePoint.SetActive(true);
    _xOutline.enabled = true;
    _yOutline.enabled = true;
```

Algorithm 63: void RemiseZeroCompteurs
Fonction permettant de remettre tous les compteurs à zéro

```
DernierCubeRegardé = -1 ; // On réinitialise le cube dernièrement regardé à -1 pour indiquer que c'est le
   début
timer = new Stopwatch[NbRep] ; // On réinitialise les timers des différentes propositions
timeTaken = new TimeSpan[NbRep] ; // On réinitialise les durées passées sur les différentes propositions
NbChangeCube = 0 ; // On réinitialise le nombre de fois où l'utilisateur a changé de cube regardé
```

Algorithm 64: int ClassifyKNN
Fonction permettant de classifier un point en entrée

```
Data: Vector2 pointObtenu
Result: int iMin
// On compare la distance du point en entrée avec chacune des classes
var distMin = -1 ; // La distance minimale du point avec une des classes
var iMin = -1 ; // La classe dont la distance avec le point en entrée est minimale
for (int i = 0 ; i < numClasses ; i++) do
    var disti = Vector2.Distance(pointObtenu, occulaire[i]) ; // On regarde la distance à la classe i
    if distMin == -1 then
        // On initialise la distance minimum
        distMin = disti ; iMin = i ;
    if distMin > disti then
        // On met à jour la distance minimum
        distMin = disti ; iMin = i ;
return iMin ;
```

Algorithm 65: float EstimationHesitationOculometre
Fonction estimant et retournant la probabilité d'hésitation de l'utilisateur selon l'oculomètre

```
// On calcule les deux critères
// La différence maximum de temps passé sur un mot entre deux mots et le nombre de fois total où l'utilisateur
   a bougé son regard sur un autre mot divisé par le nombre de propositions (donc la somme du nombre de fois
   où il a regardé chaque mot divisé par le nombre de propositions)
var tmpsMin = -1 ; // Le temps minimum passé sur un mot
var tmpsMax = -1 ; // Le temps maximum passé sur un mot
for (int i = 0 ; i < NbRep ; i++) do
    timeTaken[i] = timer[i].Elapsed ; // On regarde le temps passé sur le chronomètre
    if tmpsMin == -1 then
        // On initialise le temps minimum
        tmpsMin = timeTaken[i] ;
    if tmpsMax == -1 then
        // On initialise le temps maximum
        tmpsMax = timeTaken[i] ;
    if tmpsMin > timeTaken[i] then
        // On met à jour le temps minimum
        tmpsMin = timeTaken[i] ;
    if tmpsMax < timeTaken[i] then
        // On met à jour le temps maximum
        tmpsMax = timeTaken[i] ;
// On a donc le point sur deux critère suivant :
Vector2 pointObtenu = new Vector2(tmpsMax-tmpsMin, NbChangeCube/NbRep) ;
// On estime l'hésitation de l'utilisateur
int predicted = ClassifyKNN(pointObtenu) ;
// On ajoute le point obtenu à la base de donnée
occulaire[predicted].Add(pointObtenu) ;
// On remet les compteurs à 0
RemiseZeroCompteurs() ; return (1-predicted) ; // On renvoie l'hesitation de l'utilisateur
```

Nous reprenons aussi les classes CloudPointVisualizer, GazePlotter, et ShowNoGazeDetection du TME qui

permettent de gérer l'oculomètre.