

Rapport de Projet

HAI405I

SAVY Lisa, GILLES Eric, GALLERNE Romain, NAVEL Morgan
Groupe S
https://github.com/eric-gilles/Projet_Prog
L2 informatique
Faculté des Sciences, Université de Montpellier

16 avril 2022



Résumé

Jeu Qui Est-Ce ? avec un générateur et un mode double personnage
réalisé par le Groupe S de L2 Informatique

Table des matières

1 Technologies utilisées et organisation	3
1.1 Choix du langage	3
1.2 Organisation du travail	3
1.2.1 Jeu et Extension	3
1.2.2 Générateur	3
1.2.3 Rapport	3
1.2.4 Rythme de travail, mode de fonctionnement	3
2 Jeu principal : Permettre à l'utilisateur de jouer	4
2.1 Fonctionnalités de l'application	4
2.1.1 Mode de jeu Normal	4
2.1.2 Mode Facile	6
2.2 Le format du fichier JSON	6
2.3 Formatage des requêtes	7
2.4 Structures de données utilisées	7
2.4.1 Données web, modification du DOM	7
2.4.2 Données utilitaires, du JSON au programme	8
2.5 Traitements des requêtes	8
2.5.1 Requêtes des questions	8
2.5.2 Requêtes d'estimation et retournement automatique	9
2.5.3 Requêtes d'hypothèse	9
3 Générateur	10
3.1 Description des formats de données utilisés et résultat	10
3.1.1 Les personnages	10
3.1.2 Le nom des personnages	10
3.1.3 Les images	11
3.1.4 Les attributs	11
3.1.5 Les valeurs	11
3.2 Les différentes interactions possibles	11
3.2.1 Importer une image	11
3.2.2 Créer les attributs	13
3.2.3 Les valeurs d'attributs des personnages (et son nom)	14
3.2.4 Valider et supprimer un personnage	14
4 Extension : Mode Double Personnages	15
4.1 Description de l'extension	15
4.2 Description de la mise en oeuvre de l'extension	16
4.2.1 Initialisation du mode Double personnage	16
4.2.2 Crédit des Questions	17
4.2.3 Traitement des questions	17
5 Conclusion	19
5.1 Problèmes rencontrés	19
5.2 Bilan	19

1 Technologies utilisées et organisation

Le projet consiste en la réalisation d'une réplique du jeu *Qui est-ce ?*, ainsi que d'un générateur de fichiers JSON, qui pourront par la suite être utilisés dans le jeu.

Il faudra également ajouter une extension parmi les choix suivants :

- Jouer contre l'ordinateur
- Jouer à deux
- Jouer avec deux personnages cachés

1.1 Choix du langage

Pour notre projet, nous avons fait le choix d'utiliser JavaScript. Plusieurs raisons nous ont poussé vers ce langage :

- Dans ce projet, nous allons manipuler un fichier JSON (JavaScript Object Notation) et le JavaScript est donc un langage particulièrement adapté pour cela.
- Concernant la partie graphique, nous avons fait le choix de nous orienter vers une application web et ce grâce à JavaScript. En effet, on peut modidier des objets dans le DOM d'un site web et même utiliser des bibliothèques graphiques (par exemple D3JS).
- Il existe dans JavaScript beaucoup de bibliothèques particulièrement puissantes, notamment JQuery qui nous sera très utile pour accéder aux différents éléments de la page.

1.2 Organisation du travail

1.2.1 Jeu et Extension

- Morgan s'est occupé de l'affichage et l'initialisation des questions.
- Lisa s'est occupé du CSS et du choix aléatoire du personnage.
- Morgan et Lisa se sont, en parallèle de la construction des questions, occupés de l'extension Mode Double Personnage.
- Romain a développé les fonctions liées au mode Facile et l'affichage des personnages ainsi que le retournement des images et le traitement des questions.
- Éric a développé les fonctions liées à la sauvegarde, au chargement du jeu en cours et le test de choix de personnage.

1.2.2 Générateur

- Morgan et Lisa ont développé les fonctions liées aux attributs.
- Romain a développé les fonctions liées la création de personnages.
- Éric a développé les fonctions d'import d'images et la conversion des objets Javascript en un fichier JSON.

1.2.3 Rapport

- Morgan s'est occupé de rédiger la partie sur l'extension.
- Lisa s'est occupé de rédiger la partie sur le générateur et du rendu final du rapport.
- Romain s'est occupé de rédiger la partie sur le jeu principal.
- Éric s'est occupé de rédiger l'introduction et la conclusion.

1.2.4 Rythme de travail, mode de fonctionnement

Travail pendant 3h au moins par semaine pendant les périodes prévus dans l'emploi du temps plus les week-end et toutes les après-midi des vacances.

2 Jeu principal : Permettre à l'utilisateur de jouer

Afin de permettre à l'utilisateur de jouer à un jeu fidèle au jeu d'origine, nous avons incorporé de nombreuses fonctionnalités à notre application, nous allons les détailler ici.

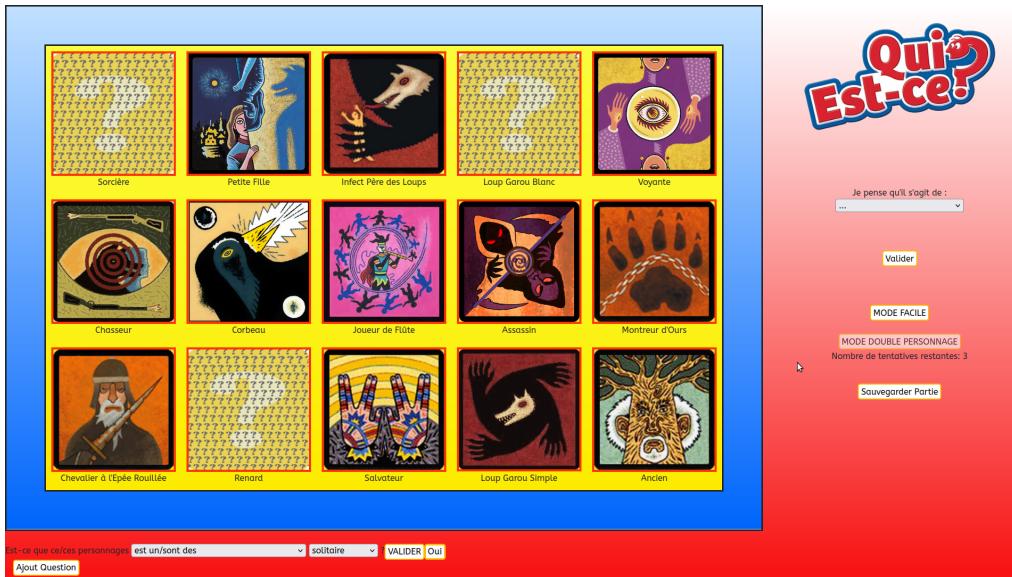


FIGURE 1 – Rendu du Jeu Qui est-ce?
Capture d'écran de la page de jeu principal en cours de partie.

2.1 Fonctionnalités de l'application

Le point le plus important d'un logiciel d'un point de vue utilisateur est bien sûr les fonctionnalités offertes par le dit logiciel (cf. FIGURE 1). Nous avons donc mis l'accent sur les diverses possibilités que notre application offrirait au joueur, à commencer par différents mode de jeu.

2.1.1 Mode de jeu Normal

Tout d'abord, dès son arrivée sur l'application, l'utilisateur doit en premier lieu importer un fichier JSON de son choix, compatible avec le jeu.

Le formalisme de ces fichiers et leur générateur seront détaillés dans d'autres parties.

Une fois le fichier choisi par l'utilisateur, il suffit à celui-ci de presser le bouton IMPORTER (cf. FIGURE 2). Le logiciel traite alors le fichier JSON et affiche de lui-même la grille de jeu. Il pré-génère également les questions qui pourront être posées en fonction des attributs dans le fichier.

Afin d'empêcher toute fausse manipulation, nous retirons le bouton d'importation lorsque tout a été convenablement généré.



FIGURE 2 – Bouton d'import du fichier JSON
Situé en bas à gauche de la page, il permet de choisir un fichier JSON et déclenche automatiquement toutes les fonctions de traitements afin d'afficher la grille de jeu.

Une fois le fichier JSON importé, le jeu se met en place. La grille de jeu est affichée et les questions sont pré-chargées. L'utilisateur peut alors poser diverses questions dépendantes des attributs des personnages. Le joueur peut demander la valeur précise de chaque attribut, par exemple il pourra demander

Est-ce que ce/ces personnages a/ont les cheveux blonds ? (L'écriture au pluriel prévoyant plusieurs personnages est utile pour le mode Double Personnage, nous y reviendrons par la suite).

Une fois la question validée, l'application répond par "OUI" ou "NON", c'est alors à l'utilisateur de déduire les personnages à retourner.

Lorsque l'utilisateur le souhaite, il peut retourner une case (cf. FIGURE 3). L'opération est évidemment réversible.

Cette action n'a aucun effet sur le fonctionnement du programme et est purement dédiée à l'utilisateur afin qu'il puisse trier les personnages librement.

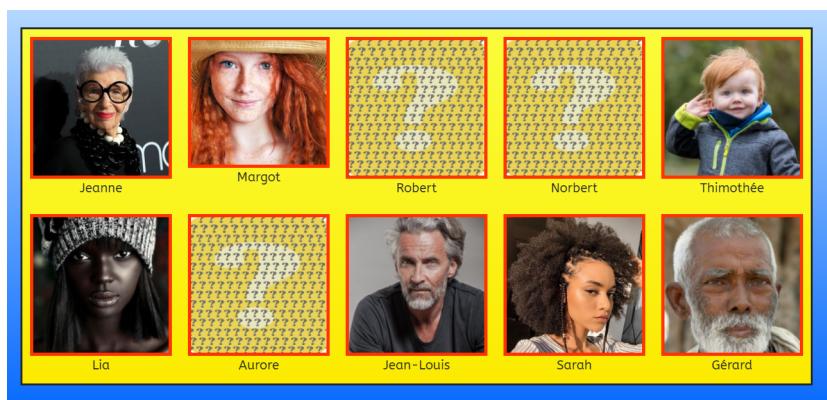


FIGURE 3 – Face retournée des personnages
La carte se retourne pour laisser place à un point d'intérogation.

Quand le joueur pense avoir une idée de la réponse, il peut formuler une hypothèse dans le menu de sélection des réponses.

Le joueur doit alors choisir le personnage qu'il pense être le bon et presser le bouton VALIDER (cf. FIGURE 4). Si le joueur aboutit à la bonne réponse, alors une fenêtre est affichée pour le féliciter et le jeu se relance automatiquement après fermeture de la dite fenêtre.

Dans le cas contraire, le nombre d'essais restants (3 dans le mode Normal) est décrémenté. Si celui-ci atteint 0, c'est perdu.

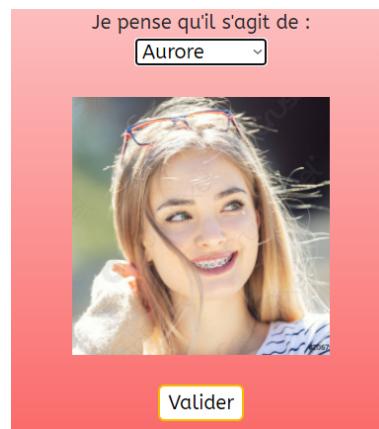


FIGURE 4 – Menu de sélection d'une réponse
Ici, le joueur pense qu'Aurore est la bonne réponse, son image s'affiche alors au dessous afin d'éviter toute erreur avant la validation

L'une des dernières fonctionnalités principales de notre application est la sauvegarde de la partie. Si l'utilisateur ferme la page durant la partie, à sa prochaine visite, il lui sera possible de recharger en l'état la partie quittée précédemment. Les personnages retournés le seront ainsi que le nombre de tentatives et le mode de jeu. Nous utilisons ici la sauvegarde locale des pages web afin de gérer cette sauvegarde.

2.1.2 Mode Facile

Juste en-dessous du menu de sélection de la réponse, le joueur peut activer à tout moment de la partie le mode **Facile**, nous allons détailler les changements qu'entraîne l'activation du mode **Facile** dans le déroulement de la partie.

L'impact le plus notable du mode **Facile** sur le fonctionnement du jeu est l'apparition d'un bouton **ESTIMER** à côté du bouton de validation des questions. De fait, ce bouton permet, après formulation d'une question, de retourner au joueur le nombre de personnages que cette question éliminera, c'est-à-dire le nombre de cases qui pourront être retournées. Cette estimation prend évidemment en compte la logique des questions multiples énoncée ci-après.

Un autre changement notable est le retournement automatique des cases. En effet, dès l'appui du bouton **VALIDER**, le programme retournera alors de lui-même l'entièreté des cases qui doivent l'être, facilitant grandement la tâche du joueur. À noter qu'il n'est pas difficile de démontrer qu'on peut aboutir à un unique personnage non-retourné en un nombre fini de questions.

Enfin, le nombre de tentatives restantes augmente de 2, totalisant 5 tentatives pour laisser plus de possibilités.

L'activation du mode **Facile** empêche celle du mode **Double Personnage** (notre extension) que nous avons considéré comme étant un mode **Difficile**, il n'est donc pas compatible avec le mode dont nous parlons.

2.2 Le format du fichier JSON

Le format du fichier JSON a été décidé de façon à respecter le langage naturel. Nous avons donc opté pour un format fixe de question .

```
0 json3.json ×
C > Users > miraj > OneDrive > Bureau > projet > projetprog > Rendu > ProjetProg_R.Gallerie-E.Gilles-L.Savy-M.Navel > json > 0 json3.json > ...
1 {
2   "personnages": [
3     {"nom": "Sorcière", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/sorciere.png", "attributs": {"est un/son des": ["villageois"]},
4     {"nom": "Petite Fille", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/pf.png", "attributs": {"est un/son des": ["villageois"]},
5     {"nom": "Infect Père des Loups", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/ipld.png", "attributs": {"est un/son des": ["loup-garou"]},
6     {"nom": "Loup Garou Blanc", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/lgb.png", "attributs": {"est un/son des": ["solitaire"]},
7     {"nom": "Voyante", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/voyante.png", "attributs": {"est un/son des": ["villageois"]},
8     {"nom": "Chasseur", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/chasseur.png", "attributs": {"est un/son des": ["villageois"]},
9     {"nom": "Corbeau", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/corbeau.png", "attributs": {"est un/son des": ["villageois"]},
10    {"nom": "Joueur de Flûte", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/jdf.png", "attributs": {"est un/son des": ["solitaire"]},
11    {"nom": "Assassin", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/assassin.png", "attributs": {"est un/son des": ["solitaire"]},
12    {"nom": "Montreuil d'Ours", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/montreuil.png", "attributs": {"est un/son des": ["villageois"]},
13    {"nom": "Chevalier à l'Epee Rouillée", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/chevalier.png", "attributs": {"est un/son des": ["villageois"]},
14    {"nom": "Renard", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/renard.png", "attributs": {"est un/son des": ["villageois"]},
15    {"nom": "Salvateur", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/salvateur.png", "attributs": {"est un/son des": ["villageois"]},
16    {"nom": "Loup Garou Simple", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/lg.png", "attributs": {"est un/son des": ["loup-garou"]}},
17    {"nom": "Ancien", "image": "https://raw.githubusercontent.com/eric-gilles/Projet_Prog/main/images/jsonlg/ancien.png", "attributs": {"est un/son des": ["villageois"]}, "e
18  ]
19 }
```

FIGURE 5 – Exemple d'un fichier JSON

Comme on peut le voir sur la FIGURE 5, le fichier JSON contient un seul attribut : **"personnages"**, ayant pour valeur un tableau de dictionnaires où chaque dictionnaire représente un personnage de la grille. Chaque dictionnaire de personnage possède trois attributs :

- **"nom"**, ayant pour valeur le nom du personnage,
- **"image"**, ayant pour valeur le lien de l'image et
- **"attributs"**, ayant pour valeur un dictionnaire représentant les attributs du personnage et leurs valeurs.

2.3 Formatage des requêtes

Nous avons fait le choix de nous rapprocher au maximum du langage naturel afin de permettre, même aux utilisateurs les moins aguerri, d'utiliser sans soucis notre logiciel.

Pour cela, nous avons organisé nos données de façon à ce que les questions puissent s'assembler en langage naturel. Nous avons donc, sur la page, le début de la question déjà affichée lorsque la cette dernière est créée : *Est-ce que ce/ces personnages*.

Ensuite, c'est à l'utilisateur de choisir parmi la liste des attributs celui qu'il souhaite utiliser.

L'ensemble des attributs sont ainsi noté ici, mais encore une fois sous la forme du langage naturel, on ne notera donc pas l'attribut *cheveux* mais bien *a/ont les cheveux*. On précisera ensuite la valeur de cet attribut, ici, par exemple, *blonds* afin de former la question *Est-ce que ce/ces personnages a/ont les cheveux blonds ?* (cf. FIGURE 6).

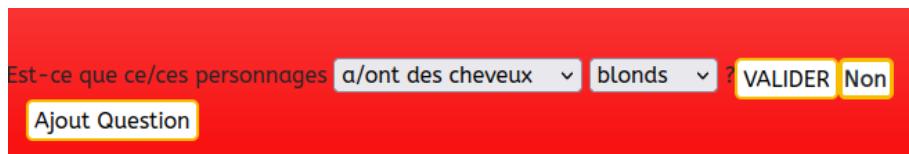


FIGURE 6 – Sélection d'une question

On voit ici que les noms d'attributs et leurs valeurs s'agencent afin de former une question correcte en langue française. Nous avons cependant dû prévoir la présence du pluriel pour l'extension double personnage que nous détaillerons dans la Partie 4.

2.4 Structures de données utilisées

Notre programme étant réalisé en JavaScript, donc en format Web, nous sommes amenés à manipuler à de très nombreuses reprises des objets au format DOM pour l'affichage. Concernant la logique, nous utilisons majoritairement des variables et objets JavaScript.

2.4.1 Données web, modification du DOM

Le DOM (Document Object Model) est le format des objets présents sur la page Web. Celui-ci est généré directement par le navigateur dépendamment du code HTML fourni.

Afin de gérer les différents objets de la page tels que la liste des questions, l'affichage des cases (retournées ou non), l'affichage du personnage sélectionné ou encore la désactivation des boutons, nous sommes sans cesse contraints de modifier les objets du document. C'est pour cela que, pour la majorité de ces données, nous n'avons nullement besoin de les stocker dans un format de données autre puisque nous les positionnons directement dans le DOM là où elles sont à la fois disponibles pour l'utilisateur mais également consultables et modifiables à tout moment par le programme comme des variables.

C'est donc dans ce format que nous chargeons les données importantes telles que les images des personnages ou bien la liste des attributs et leur valeurs possibles directement depuis le fichier JSON importé.

D'autres éléments de la page, comme les questions, sont référencés par leur attributs "id" dans le DOM. Les valeurs de ces attributs sont générées par des compteurs globaux comme `NombreQuestions` ou `EssaisRestants`. Toutes ces données, qui doivent être accessible partout et à tout moment, sont référencées globalement. Par exemple, pour l'attribut de la seconde question, on lui attribuera l'id "attribut"+`NombreQuestions`, ce qui donnera "attribut1" (1 car nous commençons le compte à 0 et non à 1).

2.4.2 Données utilitaires, du JSON au programme

Nous devons récupérer toutes les données du fichier JSON afin de les transformer en un format Objet.

C'est ici que s'illustre tout particulièrement le choix de notre langage, le JavaScript. Effectivement, le JavaScript permet un traitement très complet du format JSON, le transformant aisément en objet JavaScript (JSObject) dont le format est le même. Ainsi, dès l'import du fichier JSON, nous l'extrayons dans un objet JavaScript global que nous utiliserons par la suite dans tout le programme afin à ne pas avoir à récupérer le fichier à chaque retouche.

Au passage, nous y choisissons aléatoirement un élément qui deviendra la réponse attendue, le personnage à deviner. Celui-ci sera stocké au format JSObject dans une variable globale à part.

2.5 Traitements des requêtes

On distingue dans notre jeu trois types de requêtes différentes pouvant être formulées par l'utilisateur :

- les requêtes de questions et leurs variantes.
- les requêtes d'estimation.
- les requêtes d'hypothèses.

2.5.1 Requêtes des questions

Afin de permettre de plus amples possibilités de jeu, il est possible ici d'ajouter d'autres questions, liées par une logique booléenne. Avec un maximum de 16 questions, le programme aboutit à une seule réponse générée logiquement avec une priorité pour la dernière question.

Par exemple, posons **c** le connecteur "ET" ou "OU", et "1", "2", ... "16" les 16 questions possibles. La logique sera traitée par le programme de la manière suivant : (((1 **c** 2) **c** 3) **c** 4 ...) **c** 16. Le joueur formule une ou des questions dans la zone appropriée et presse le bouton VALIDER, la logique des questions s'enclenche alors afin d'aboutir à une réponse unique.

Pour permettre cela, nous devons d'abord déterminer la réponse individuelle de chaque question. Dans une fonction dédiée appelée **TraitemtUneQuestion()** prenant en paramètres l'attribut ainsi que sa valeur, par exemple "*a/ont les cheveux*" et "*blonds*". On vérifie alors que, pour la bonne réponse, pour rappel, référencé dans une variable globale au format JSObject, à l'index passé en paramètres, ici, "*a/ont les cheveux*", on trouve bien la même valeur, ici, "*blonds*". Si c'est le cas alors on retourne la valeur booléenne **True**, sinon **False**.

Enfin, une fonction à part appellé **TraitemtQuestion()** s'occupe d'assembler la réponse de toutes ces questions d'après la logique formulée ci-dessus.

On aboutit donc à une réponse finale qui sera affichée à l'utilisateur :

- Oui, si le résultat est **True**.
- Non, si celui-ci est **False**.

2.5.2 Requêtes d'estimation et retournement automatique

Les requêtes d'estimations sont une variante des requêtes de questions puisqu'elles utilisent exactement la même logique, à ceci près que nous ne cherchons pas simplement à aboutir à une réponse mais bien à compter le nombre de personnages pour lesquels la réponse est différente de celle du personnage à trouver.

Par exemple, si la réponse à la question demandée pour le personnage caché est **Vrai**, on va chercher pour quels autres personnages la réponse est différente. On retournera au joueur le décompte de ces personnages. Pour notre exemple, ceux dont la réponse est **Faux**.

À noter que nous utilisons aussi ce système de comparaison en mode Facile afin de retourner automatiquement les cartes lors de l'appui du bouton VALIDER.

Ainsi, les requêtes d'estimation ont une plus grande complexité que les requêtes de questions. En effet, là où l'évaluation d'une question en mode Normal a une complexité en $O(1)$, celle de l'estimation s'évalue en complexité $O(n)$, n étant le nombre de questions.

2.5.3 Requêtes d'hypothèse

Pour finir, les requêtes d'hypothèse sont les plus simples. Celles-ci se contentent de vérifier si le personnage sélectionné est le même que le personnage caché. On ne compare ici que le nom du personnage, cela suffit puisqu'on interdit dans le générateur la création de deux personnages ayant le même nom.

- Si le nom est le même, alors on déclenche la fonction de victoire, affichant une notification pour féliciter le joueur et réinitialisant le jeu.
- Si le nom est différent, on décrémente le nombre d'essais.

3 Générateur

Le générateur (cf. FIGURE 7) permet à l'utilisateur de créer ses propres grilles de personnages qu'il pourra utiliser pour jouer au jeu du Qui est-ce ?

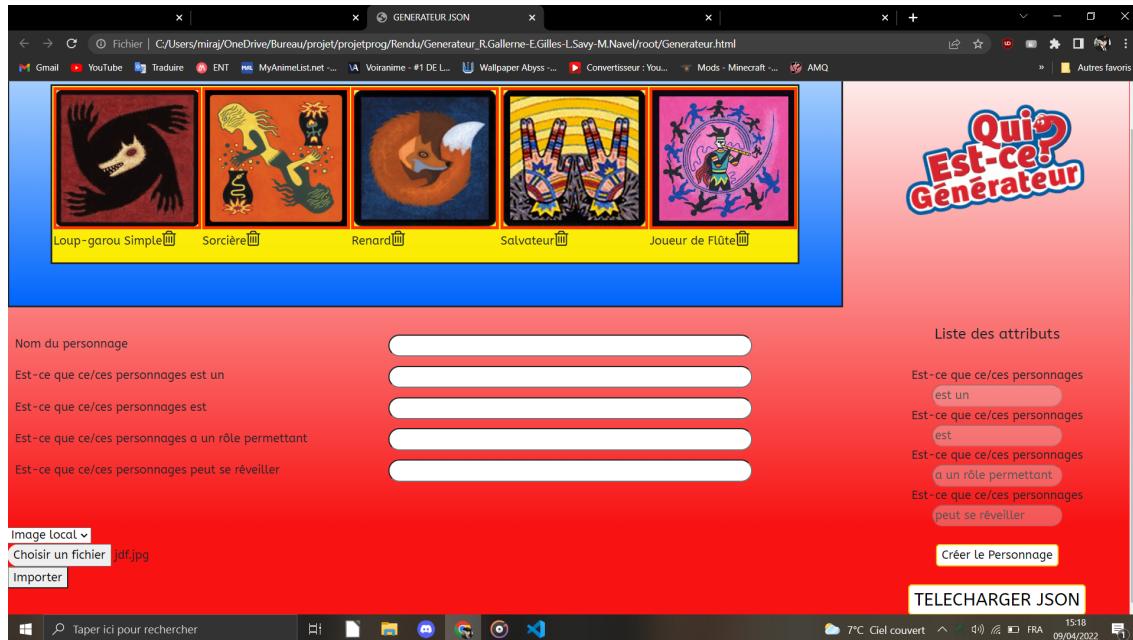


FIGURE 7 – Rendu du générateur

3.1 Description des formats de données utilisés et résultat

L'objectif de ce logiciel et de traiter des données entrées par l'utilisateur afin de rendre à la fin un fichier au format JSON pour qu'il puisse être utilisé sur notre jeu principal Qui est-ce ?

Le format JSON est formaté comme décrit dans la Partie 2 : Jeu principal et c'est en fonction de lui que nous avons choisi nos différents formats de données.

3.1.1 Les personnages

Quand l'utilisateur crée un personnage, un objet JavaScript est créé. Celui-ci contient 3 attributs :

- "nom" : une chaîne de caractère (String),
- "image" : une chaîne de caractère (String) et
- "attributs" : un objet JavaScript indexé par les noms des attributs renseignés par l'utilisateur.

Les valeurs de ces attributs vont être décrisées dans les sous-parties suivantes.

Quand l'utilisateur appuiera sur le bouton CRÉER LE PERSONNAGE, l'objet du personnage sera stocké dans un autre objet Javascript indexé par les noms des personnages créés.

3.1.2 Le nom des personnages

Quand l'utilisateur rentrera le nom de son personnage, celui-ci ne sera pas définitif tant que l'utilisateur ne créera pas son personnage. Il en sera de même pour toutes les autres données entrées par l'utilisateur, elles ne seront stockées que quand le personnage sera créé.

Quand le personnage sera créé, le nom va être stocké dans l'attribut "nom" de l'objet JavaScript du personnage qui vient d'être créé.

Nous avons également décidé d'afficher dynamiquement les entrées de l'utilisateur comme le nom, les attributs ou les valeurs. Les entrées ne sont pas réellement stockées, nous manipulons seulement le DOM.

3.1.3 Les images

L'utilisateur aura le choix d'importer une image locale ou via un lien URL. Si l'image est importée via un lien URL, le lien est tout simplement stocké sous forme de chaîne de caractères (String) dans l'attribut "image" de l'objet JavaScript du personnage qui vient d'être créé.

3.1.4 Les attributs

Quand l'utilisateur éditera son premier personnage, il devra renseigner tous les attributs qu'il souhaitera donner aux personnages. Tous les personnages auront les mêmes attributs, seules leurs valeurs différeront. Quand ce premier personnage sera validé, les attributs ne seront plus modifiables et stockés dans l'attribut "attributs". Chaque attribut rentré par l'utilisateur deviendra une clé de l'objet JavaScript "attributs" de chaque personnage, les valeurs associées à chaque attribut seront celles renseignées dans les zones de saisie des valeurs comme expliqué après.

3.1.5 Les valeurs

A chaque personnage que l'utilisateur créera, il devra lui renseigner des valeurs pour chaque attribut (celles-ci peuvent être vides mais chaque combinaison de valeurs doit être unique). Ces valeurs seront stockées dans l'attribut "attributs" du personnage qui vient d'être créé. Chaque clé de l'attribut "attributs" sera associé à la valeur correspondante renseignée par l'utilisateur.

3.2 Les différentes interactions possibles

Pour créer une grille de personnages, il faudra que l'utilisateur crée chaque personnage un à un. Pour créer un personnage, il pourra soit commencer par importer une image ou directement rentrer les données du personnage qu'il souhaite créer et importer l'image après ou durant la création du personnage.

3.2.1 Importer une image

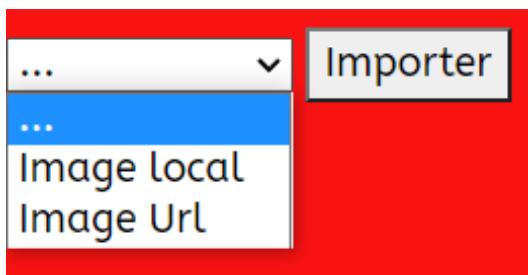


FIGURE 8 – Choix du mode d'importation d'image

Attention ! Il est recommandé d'utiliser des images au format carré pour éviter que celles-ci soient compressées.

Il y a deux façons d'importer des images (cf. FIGURE 8) :

— Importer une image avec un URL

Quand l'utilisateur choisira cette option, une zone d'entrée de texte apparaîtra juste en-dessous. Il suffira à l'utilisateur de saisir l'URL de l'image désirée et de cliquer sur importer pour que l'image s'affiche (cf. FIGURE 9). *Il sera nécessaire d'avoir accès à Internet pour voir l'image.*



FIGURE 9 – Importer une image avec un URL

— Importer une image locale

Quand l'utilisateur choisira cette option, un bouton apparaîtra et il pourra ainsi sélectionner l'image désirée dans son ordinateur et cliquer sur le bouton importer pour que l'image s'affiche (cf. FIGURE 10). *Choisir d'utiliser des images locales alourdira le fichier JSON car le lien de chaque image est entièrement recodée en caractères (environ 60000 caractères).*

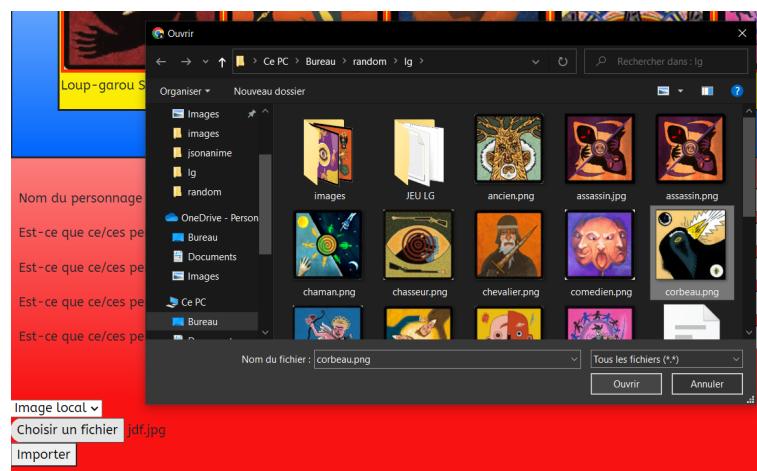


FIGURE 10 – Importer une image locale

Tant que le personnage n'est pas créé, le bouton Importer restera grisé.

3.2.2 Créez les attributs

Quand l'utilisateur commencera à créer son premier personnage, il devra rentrer la liste **définitive** des attributs qu'auront tous les personnages dans la zone de saisie à droite de la page (cf. FIGURE 11). L'utilisateur peut supprimer l'attribut de son choix ou en ajouter jusqu'à 15 tant qu'il est en train de créer son premier personnage. Quand celui-ci sera validé, l'utilisateur ne pourra plus du tout toucher à cette liste. Il ne doit pas y avoir d'attribut **vide** !



FIGURE 11 – Liste d'attributs avant et après création

Il est conseillé à l'utilisateur de créer des attributs cohérents avec le format de questions du jeu (notamment de considérer le pluriel au vu du mode Double Personnage). Les questions en mode **Normal** commencent toujours par *Est-ce que le personnage* (cf. FIGURE 12). Pour que la phrase est un sens, l'utilisateur doit rentrer un attribut commençant par un verbe conjugué.

FIGURE 12 – Exemple d'un question en mode Normal. La figure montre une interface de saisie de question en mode Normal. La question "Est-ce que ce/ces personnages possède" est suivie d'un curseur, puis "un stand", puis d'un curseur, puis d'un bouton "VALIDER". Ensuite, il y a deux boutons : "Oui" et "VRAI".

FIGURE 12 – Exemple d'un question en mode Normal

Quand le mode **Double Personnage** est sélectionné, la question commence toujours par *Est-ce que* suivi de aucun *des personnages*, *au moins un des personnages* ou *les deux personnages* (cf. FIGURE 13). Pour que la phrase est un sens, l'attribut doit commencer par un verbe conjugué prenant en compte le pluriel. Le pluriel n'est pas nécessaire mais conseillé pour se rapprocher au mieux du langage naturel.

FIGURE 13 – Exemple d'un question en mode Double Personnage. La figure montre une interface de saisie de question en mode Double Personnage. La question "Est-ce que [les deux personnages] est un/sont des" est suivie d'un curseur, puis "solitaire", puis d'un curseur, puis d'un bouton "VALIDER". Ensuite, il y a deux boutons : "VRAI" et "FAUX".

FIGURE 13 – Exemple d'un question en mode Double Personnage

Attention ! La question ne se finit pas juste après l'attribut mais après la valeur ! Il ne faut donc pas créer une question qui oblige que les valeurs d'attributs soit *oui* ou *non* sinon les réponses à ce genre de questions seront peu claires et porteront à confusion (cf. FIGURE 14).

Est-ce que ce/ces personnages	<input type="text" value="est un homme"/>	<input type="text" value="oui"/>	?	<input type="button" value="VALIDER"/>	<input type="button" value="Non"/>
-------------------------------	---	----------------------------------	---	--	------------------------------------

FIGURE 14 – Exemple de question mal pensée

3.2.3 Les valeurs d'attributs des personnages (et son nom)

Les valeurs d'un personnage sont entrées lors de sa création et ne pourront plus être modifiées après. La zone pour entrer les valeurs se situe juste en-dessous du rendu des images (cf. FIGURE 15).

Nom du personnage	<input type="text" value="Sorcière"/>
Est-ce que ce/ces personnages est/sont un/des	<input type="text" value="villageois"/>
Est-ce que ce/ces personnages est/sont	<input type="text" value="humain(s)"/>
Est-ce que ce/ces personnages a/ont un rôle qui peut	<input type="text" value="tuer n'importe qui,sauver,obtenir des informations"/>
Est-ce que ce/ces personnages peu(ven)t se réveiller	<input type="text" value="toutes les nuits"/>

FIGURE 15 – Exemple de liste de valeurs

Afin d'aider à la compréhension, le début de la question (Est-ce que + attribut affiché dynamiquement) est indiqué juste devant les zones de saisie.

Quand le nom est rentré par l'utilisateur, il s'affiche dynamiquement sous l'image du personnage en cours d'édition.

Comme vu en FIGURE ..., l'utilisateur peut rentrer plusieurs valeurs pour un même attribut, celles-ci peuvent contenir des espaces et des apostrophes mais il faut bien faire attention à séparer chaque valeur par une virgule et seulement une virgule (si l'utilisateur rentre un espace juste après il sera pris en compte dans la deuxième valeur d'attribut).

3.2.4 Valider et supprimer un personnage

Quand l'utilisateur sera satisfait de sa grille de personnages, il lui suffira de cliquer sur le bouton "Conversion JSON" et le rendu sous format JSON de la grille se téléchargera et sera stockée dans le dossier Téléchargement de l'utilisateur.

4 Extension : Mode Double Personnages

Pour notre projet, nous avons décidé d'ajouter le mode **Double Personnage** en tant qu'extension.

L'objectif de cette extension est d'augmenter la difficulté du jeu et d'utiliser la logique des questions de notre jeu de base afin de voir comment complexifier la façon dont l'utilisateur posera des questions.

4.1 Description de l'extension

L'utilisateur, en activant ce mode, devra trouver deux personnages, avec des questions qui sont formés différemment. En effet, comme on peut le voir sur la FIGURE 16 , la forme est très similaire aux questions du jeu de base, mais avec en plus une information pour savoir si "au moins un des personnages", "les deux personnages" ou "aucun des personnages" vérifient la question demandée, tout en utilisant les connecteurs logiques "OU" et "ET" du jeu de base.

Lorsque le bouton d'activation du mode **Double Personnage** est activé, nous avons fait le choix de

The screenshot shows a red-themed user interface for creating questions. At the top, there's a dropdown menu with the text "Est-ce que au moins un des personnages" followed by a dropdown for selecting a character type ("villageois"). To the right is a yellow "VALIDER" button. Below this, there are three stacked conditional statements using the "ET" connector:

- Est-ce que [les deux personnages] a un rôle/ont des rôles permettant de [obtenir des informations] ?
- Est-ce que [aucun des personnages] peut/peuvent se réveiller [toutes les nuits] ?

At the bottom left is a blue "Ajout Question" button.

FIGURE 16 – Exemple de la formation de questions en mode double personnage

ne pas pouvoir revenir en arrière lorsque celui-ci est activé. On ne peut donc pas le désactiver (voir FIGURE 17).

De plus une petite ligne apparaît pour l'informer du nombre d'essais restant. Quand ce mode est activé, le nombre de tentatives passe de 3 à 6. Nous avons aussi fait le choix de ne pas pouvoir activer le mode Facile car cela prendrait beaucoup plus de temps que ce qu'on nous permet.

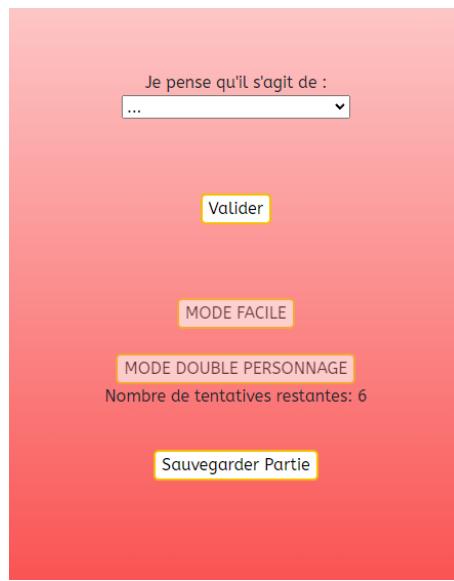


FIGURE 17 – Image des boutons après activation du mode Double personnage

De plus, quand l'utilisateur fait une tentative et trouve un des deux personnages un message l'informant qu'il a bien trouvé un des deux personnages apparaît (voir FIGURE 18). La condition de victoire étant d'avoir trouvé le deuxième personnage, le message affiché sera comme dans la FIGURE 19

Cette page indique
Vous avez trouvé un des personnages ! Bravo plus que 1 !

OK

FIGURE 18 – Message lors de la découverte d'un des deux personnages

Cette page indique
Fin du jeu ! vous avez trouvé les deux personnages ! Bravo !

OK

FIGURE 19 – Message lors de la découverte du deuxième personnage

4.2 Description de la mise en oeuvre de l'extension

4.2.1 Initialisation du mode Double personnage

```

437 //active le mode double personnage (qui désactive le mode facile)
438 function LancementDoublePersonnages(){
439     document.getElementById("MODE_FACILE").className = "disabled";
440     document.getElementById("MODE_FACILE").onclick = "";
441     document.getElementById("boutonDouble").className = "disabled";
442     document.getElementById("boutonDouble").onclick = "";
443     ChoixPersonnage2();
444     console.log("Personnage caché 1 :" + persoCache1);
445     console.log("Personnage caché 2 :" + persoCache2);
446     ModeDouble=true;
447     $("#" + valider).attr("onclick", "traitementQuestionsModeDouble(1)");
448     nbEssais=nbEssaisModeDouble;
449     $("#" + NbEssai).empty();
450     $("#" + NbEssai).append("Nombre de tentatives restantes: " + nbEssais);
451
452 }
453 }
```

FIGURE 20 – Extrait du code du lancement du mode

La fonction `LancementDoublePersonnages()` (voir Figure 20) va faire principalement des modifications dans le DOM. Elle va dans un premier temps désactiver les boutons DOUBLE PERSONNAGE et MODE FACILE présent sur la page comme expliqué précédemment, puis elle va exécuter la fonction `ChoixPersonnage2()` qui se charge de donner un personnage aléatoire pour le deuxième personnage. `LancementDoublePersonnages()` actualise aussi le nombre d'essais de 3 à 6 et ajoute une variable global booléenne `ModeDouble` qui permet de savoir à n'importe quel moment dans notre programme si ce mode est activé.

Et enfin, cette fonction change l'événement `onclick()` du bouton de validation d'une question de la manière dont on doit traiter une question dans ce mode.

4.2.2 Crédation des Questions

Dans ce mode, les questions sont un peu différentes, ce qui nous oblige à modifier le DOM pour avoir la bonne formulation pour la première question affichée, et pour les questions qui seront ajoutées à la suite comme dans le jeu de base.

Pour celà, nous avons le code en FIGURE 21 et 22 qui font respectivement la création de la première question et la liste des attributs possibles comme dans le jeu de base avec, en plus, la liste contenant "au moins un des personnages", "les deux personnages" et "aucun des personnages" pour avoir la base dans le HTML.

Enfin, les fonctions d'ajout et de suppression de questions qui sont très similaire à celles du jeu de base. Dans la fonction d'ajout de question, nous avons ajouté une variable global `firstTime` afin de pouvoir remettre à zero le compteur de questions du programme pour pouvoir indexer correctement les questions. La fonction de suppression quant à elle supprime la division où se trouve la question, diminue le compteur de questions et actualise l'événement `onclick()` du bouton VALIDER.

```

203 //Construit les questions de la bonne façon pour jouer au Mode Double Personnage Caché
204 function ConstruitQuestionModeDouble(){
205     $("#Questions").empty();
206     let html = "<div>Est-ce que <select id='Predicat0'><option>...</option><option>au moins un des personnages</option><option>les deux personnages</option><option>aucun des perso
207     $("#Questions").append(html);
208     for(let a in jsonobject.personnages[0].attributs){
209         html = "";
210         html += "<option id='"+a+">" + a + "</option>";
211         $("#attr0").append(html);
212     }
213 }
214
215
216

```

FIGURE 21 – Extrait du code de la création de la première question de la page

```

200 //ajout Question Mode Double
201 $('#add_button').click(function(e) {
202     e.preventDefault();
203     var html = "<option>...</option>";
204     if(modeDouble && firstTime){ //x doit revenir à la première question quand on change en mode double
205         x=1;
206         firstTime=false;
207     }
208     if(modeDouble){
209         $(div).append("<div><select id='c'+"+(x-1)+">\n\t<option>ET</option>\n\t<option>OU</option>\n</select>\n<br>Est-ce que <select id='Predicat'+"+(x-1)+"><option>...</option></select> ?");
210         $('#valider').attr("onclick","traitementQuestionsModeDouble('"+x+"')");
211     }
212 });
213 //suppression Question Mode Double
214 $(div).on("click", ".delete", function(e) {
215     e.preventDefault();
216     $(this).parent('div').remove();
217     x--;
218     if(modeDouble){
219         $('#valider').attr("onclick","traitementQuestionsModeDouble('"+x+"')");
220     }
221 });

```

FIGURE 22 – Extrait du code de l'ajout et suppression de question

4.2.3 Traitement des questions

La fonction de traitement est aussi légèrement différent du jeu de base, car il faut traduire "au moins un des personnages", "les deux personnages" et "aucun des personnages" en logique.

Dans un premier temps, nous avons regardé comment traiter UNE question en mode Double Personnage. Ceci est fait par la fonction `TraitementUneQuestionModeDouble()` (voir FIGURE 23 1.437).

Finalement, le traitement d'une question c'est simplement le traitement d'une question utilisant la même fonctionnalité que le jeu de base (on l'appellera traitement simple), mais avec une différence c'est que l'on a deux personnages, donc on différencie les cas pour savoir quel connecteur logique utilisé ("ET" si le joueur a choisi "les deux personnages", "OU" si le joueur a choisi "au moins un des personnages" et "NON(OU)" si le joueur a choisi "aucun des personnages personnes") et ensuite on applique ce connecteur logique sur le retour booléen du traitement simple sur le premier personnage avec le retour booléen du traitement simple du deuxième personnage.

Cette fonction prend donc en paramètre le Prédicat sélectionné (sous entendu "au moins un des personnages", "les deux personnages" ou "aucun des personnages"), l'attribut sélectionné et la valeur sélectionnée, et rend en fonction de ces paramètres une valeur booléen VRAI ou FAUX.

Enfin, pour plusieurs questions effectuées par la fonction traitementQuestionsModeDouble (voir Figure 23 1.405 - 434), il suffit de prendre une valeur tampon que l'on initialise au retour booléen de la première question indexée par 0, et ensuite, en fonction du connecteur "ET" ou "OU", on applique ce connecteur binaire entre le retour booléen "tampon" et la valeur booléenne de la question courante. La fonction prend en entrée le nombre de questions dans la page (d'où le besoin d'actualiser le nombre de questions) et elle renvoie un valeur booléenne.

```

385 //Logique des questions pour le Mode Double: rend une valeur booléenne
386 ∵ function traitementQuestionsModeDouble(nbQuestions){
387   let SelectedAttributs=$("#attr0").val();
388   let SelectedValue=$("#valeur0").val();
389   let SelectedPredicat=$("#Predicat0").val();
390   let reponse=TraitemEntUneQuestionModeDouble(SelectedPredicat,SelectedAttributs,SelectedValue);
391
392 ∵ if(nbQuestions > 1){ //pour plusieurs questions
393   ∵ for(let i=1;i<nbQuestions;i++){
394     SelectedAttributs=$("#attr"+i).val();
395     SelectedValue=$("#valeur"+i).val();
396     SelectedPredicat=$("#Predicat"+i).val();
397     ∵ if(document.getElementById("c"+i).options[document.getElementById("c"+i).selectedIndex].value=="ET"){
398       reponse=(reponse&& TraitemEntUneQuestionModeDouble(SelectedPredicat,SelectedAttributs,SelectedValue));
399     }
400     ∵ else{
401       reponse=(reponse || TraitemEntUneQuestionModeDouble(SelectedPredicat,SelectedAttributs,SelectedValue));
402     }
403   }
404
405   $("#reponse").empty();
406   ∵ if(reponse){
407     $("#reponse").append("VRAI");
408   }
409   ∵ else{
410     $("#reponse").append("FAUX");
411   }
412
413   console.log("réponse finale pour une question : "+reponse);
414 }
415
416 |
417
418 //Traite une seule question en mode Double
419 ∵ function TraitemEntUneQuestionModeDouble(SelectedPredicat,SelectedAttributs,SelectedValue){
420   let reponse=false;
421   ∵ if(SelectedPredicat=="aucun des personnages"){
422     reponse= (TraitemEntUneQuestion(SelectedAttributs,SelectedValue,persoCache1)==false) && (TraitemEntUneQuestion(SelectedAttributs,SelectedValue,persoCache2)==false);
423   }
424   ∵ else if(SelectedPredicat=="au moins un des personnages"){
425     reponse= (TraitemEntUneQuestion(SelectedAttributs,SelectedValue,persoCache1)|| TraitemEntUneQuestion(SelectedAttributs,SelectedValue,persoCache2));
426   }
427   ∵ else{
428     reponse= (TraitemEntUneQuestion(SelectedAttributs,SelectedValue,persoCache1) && TraitemEntUneQuestion(SelectedAttributs,SelectedValue,persoCache2));
429   }
430   return reponse;
431 }
432

```

FIGURE 23 – Extrait du code du traitement de plusieurs questions

5 Conclusion

5.1 Problèmes rencontrés

Nous avons eu des problèmes liés au développement du projet de programmation :

- Le premier problème qui s'est présenté à nous très vite a été la gestion et récupération du fichier JSON que nous avions prévu de gérer dynamiquement. Nous avons finalement choisi de le récupérer dès le début dans une variable globale.
- Lors de la suppression des questions, nos indices se décalaien systématiquement, nous avons donc du réécrire entièrement cette fonction.
- Pour la sauvegarde de la partie de jeu, nous avons voulu utiliser les cookies de JavaScript mais nous avons changé pour les LocalStorage pour une facilité de développement. Le stockage local est majoritairement utilisé pour les applications clients et n'ont pas de dates d'expiration comparés aux cookies.
- Enfin, nous avons malgré nous passé beaucoup de temps sur de l'HTML et du CSS afin de rendre notre application la plus agréable possible à utiliser.

5.2 Bilan

Ce projet nous a permis d'acquérir et de renforcer des notions en développement Web que nous avons vu en cours et précisément en JavaScript et en Jquery pour la réalisation d'une application Web d'un jeu Qui est-ce ?.

Cela nous a aussi donné la possibilité de modéliser des maquettes de la page de jeu et de création de JSON que nous devions réaliser pour permettre de disposer facilement les différents éléments graphiques et les emplacements des menus et ainsi faciliter la réalisation de l'application Web.

Au niveau de la gestion de projet, cela nous a donné l'occasion de travailler en groupe sur un projet avec un but bien défini et de nous répartir sur les différentes tâches à réaliser.