420-355-LI - Programmation Web Cégep Limoilou Département d'Informatique Préparé par : **Jocelyn Goulet** Adapté par : **Martin Simoneau** Automne 2025	Tp1 (10 %) Jeu-Questionnaire (comme un quiz)

Objectifs

- Construire une petite application Web, client, en JavaScript offrant une interface HTML et CSS, dont les fonctionnalités dynamiques sont faites en JavaScript.
- Séparer correctement les différents composants de votre application (HTML, css, code dans des fonctions et objets).
- Valider correctement le fonctionnement de l'application (Structures de données, navigation, résultats …).

Lecture

- Je mets ici des références au sujet des formulaires HTML :
  - Formulaire HTML :
    https://developer.mozilla.org/fr/docs/Learn/Forms
    ET
    https://www.w3schools.com/html/html_forms.asp.

  - Pour valider une partie des valeurs d'un formulaire avec du HTML (faire attention à la compatibilité avec votre « browser ») :
    https://developer.mozilla.org/fr/docs/Learn/Forms/Form_validation.
    Et
    https://www.w3schools.com/js/js_validation.asp.

Conditions et remise

- Le travail se fait **obligatoirement** en équipe de 2 (3 s'il y a une personne seule, elle sera placée aléatoirement par le professeur).
- Vous avez 3 semaines pour faire et remettre ce travail à partir de sa date de réception.
- Vous devez utiliser git pour gérer votre projet et partager votre code avec votre coéquipier. Pensez à saisir des messages de *commits* concis et pertinents et rappelez-vous que *git* n'est pas un chat.

Agenda – à mettre à votre agenda

Date de réception :

Groupe 3 - 24 septembre.

Semaine 1 - Vérification d'avancement par le prof :

Groupe 3 – le 3 octobre :

- Prise en note, par votre professeur, de l'avancement de votre travail. La présence de tous les équipiers est obligatoire. Cette vérification sera notée.

Semaine 2 - Vérification d'avancement par le prof :

Groupe 3 – le 10 octobre entre 12h et 14h

- Prise en note, par votre professeur, de l'avancement de votre travail. La présence de tous les équipiers est obligatoire. Cette vérification sera notée.

Semaine 2 - Remise du fichier JS de questions :

Groupe 3 – le 8 octobre avant 17h

- Remettre le fichier final JS « **questionJSON.js** » de questions produit dans ce travail. Renommez le fichier comme suit « **Noms des équipiers – questionJSON.js** » avant de le remettre. Vous avez un fichier de test

Semaine 3 - Remise du code :

Groupe 1 - le 15 octobre avant 17h
Groupe 2 - le 15 octobre avant 17h
Groupe 3 – le 15 octobre avant 17h

Remettre le projet « PhpStorm » et seulement le projet dans un archive zip nommée « **Noms des équipiers – Tp1-Questionnaire – A25** » sur Omnivox/Léa. Retirer l'énoncé de votre remise.

- NOTE IMPORTANTE :
  Un projet « *PHPStorm* » de départ pour ce travail vous a été remis avec cet énoncé.

Mise en situation

Réalisez une petite application qui permet de jouer à un jeu-questionnaire (quiz). Votre jeu-questionnaire sera basé sur un ensemble d'au moins 15 vraies questions. Cet ensemble de questions sera vos données d'entrée dans le programme.

Au départ, vous affichez le but du jeu à l'utilisateur et vous permettez à l'utilisateur de jouer. Vous choisissez n questions aléatoirement dans votre ensemble de questions, ce qui constituera votre questionnaire (quiz). Par la suite, vous présentez une question à la fois, lorsque la question posée est répondue, vous présentez sa correction et ainsi de suite pour les 5 questions. Lorsque les 5 questions ont été répondues, vous présentez à l'utilisateur ses points (sa note finale) et un commentaire personnalisé d'encouragement. Dans ce dernier affichage, vous permettez aussi à l'utilisateur de rejouer avec 5 nouvelles questions prises aléatoirement dans votre « pool ». À tout moment, il est possible d'abandonner, le programme retourne alors au message d'introduction.

Comme dit précédemment, les questions sont choisies aléatoirement. Ce qui veut dire que, lorsqu'une question est choisie pour un questionnaire (quiz) de 5 questions, elle ne peut être redemandée dans le même questionnaire. Il est important aussi que chaque question soit posée au moins une fois après avoir répondu à quelques questionnaires.

Contraintes au niveau des données

- Chaque semaine, selon l'agenda proposé ci-haut, votre professeur vérifiera au prêt de chaque équipe l'avancement de votre projet. Ce qui est important, c'est d'être présent (tous les membres de l'équipe) et de répondre aux questions de votre enseignant.

- Comme projet de départ, copiez le projet « **Noms des équipiers – Tp1-JeuQuestionnaire – A25** » qui est fourni dans le dossier du travail « Tp 1 » et renommez-le correctement.
- Le code HTML de départ se trouve dans le fichier « **tp1-JeuQuestionnaire.html** » à la racine du projet.

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>TP 1 - Jeu Questionnaire - A24</title>
  <link href="css/tp1-JeuQuestionnaire.css" rel="stylesheet">
</head>
<body>

<div id="zoneDeDonnees">
  <fieldset>
    <legend>Questionnaire</legend>
    <h1 id="titre2">Mini quiz </h1>
    <div id="contenu"></div>
    <div>
      <button id="boutonValider">Valider</button>
      <!--Mettre ici tous les autres boutons nécessaires-->
    </div>

  </fieldset>

  <span id="isactif">
     <input type="checkbox" id="actifToggle" name="isactif" value="false">
    <label for="actifToggle">Interface verrouillé</label>
  </span>
</div>
<span id="programmeurs">
  <h3>Programmé par</h3>
  <p>Auteur 1</p>
  <p>Auteur 2</p>
</span>

<script src="data/questionsJSONTest.js" type="text/javascript" defer></script>
<script src="js/classes/Question.js" type="text/javascript" defer></script>
<script src="js/classes/Quiz.js" type="text/javascript" defer></script>
<script src="js/elementHTML.js" type="text/javascript" defer></script>
<script src="js/tp1-JeuQuestionnaire.js" type="text/javascript" defer></script>

</body>
</html>
```

- La section des boutons est incomplète. Vous devrez y ajouter les boutons nécessaires.
- La division avec l'id «*contenu*» servira à contenir le code HTML des questions que vous devez générer par programmation js.
- Toute la logique est dans des fichiers et/ou classes JavaScript. Utilisez des structures de données, des fonctions et des classes pour gérer vos questions et votre questionnaire (quiz).
- Le fichier CSS vous est fourni avec le TP.

- 2 fichiers JavaScript JSON vous ont été remis.
    - questionsJSON.js contenant des questions de départ que vous devrez changer par les vôtres avant la remise
    - questionsJSONTest.js contentant des questions simples pour tester vos algorithmes.
- Ces fichiers JSON simulent les données qui seraient fournies par le serveur. Il contient un tableau associatif de format JSON avec toutes les questions possibles. Cet objet JSON sera utilisé pour remplir un tableau **indexé** d'objets « *Question* » au démarrage de votre application. C'est avec le tableau d'objets « *Question* » que votre application devra travailler, et non avec l'objet JSON directement.
- Donc on veut :
    - Au moins 2 classes :
        - Une classe « ***Question*** » qui doit être utilisée pour contenir l'information pour **une question.** Un objet *Question* doit être instancié par question possible.
        - Une classe «**Quiz** » qui doit
            - choisir les 5 questions au hasard à partir de la banque de questions. Si la banque de questions comporte moins de 5 question, votre quiz aura évidemment moins de 5 questions.
            - Faire le compte des points totaux.
            - Gérer l'avancement des questions lorsque l'utilisateur passe à la question suivante.
            - Générer le message de fin.
            - Autres fonctionnalités que vous jugerez nécessaires.
    - Créez les getter et setter au besoin. N'accédez jamais directement aux attributs.

    - Un tableau indexé d'objet « Question » pour contenir toutes les questions du fichier de données JSON. On prend l'information du tableau associatif JSON (fichier questionsJSON.js) et on fabrique des objets « Question » que l'on place dans un tableau indexé (« Array » [ ]).

- Les textes et les fenêtres présentés **ci-après** représentent l'affichage et le fonctionnement minimal attendu, si vous le voulez, vous pouvez faire mieux, mais il ne faut pas altérer le fonctionnement demandé.

Contraintes au niveau du fonctionnement (étapes de développement)

1. Fichier de questions

Le fichier principal de questions doit porter obligatoirement le nom de « **questionJSON.js** ». Il est au format JSON (tableau associatif JavaScript) et contient au **minimum** la définition de 15 questions **réelles portant sur la matière du cours**.

Une question, dans ce fichier, doit être définie selon les informations **minimums** suivantes :

- Le texte de la question.
- Le texte de 5 réponses (5 obligatoires et seulement 5).
- Les points accordés pour chaque réponse. La somme des points positifs doit donner 100, la somme des points négatifs doit donner -100 et évidemment, la somme totale doit donner 0.

- La valeur de la question. Chaque question possède son propre poids. Pour connaître les points obtenus par le joueur, il faut calculer
    - (somme des *points accordés*)/100 * la *valeur*

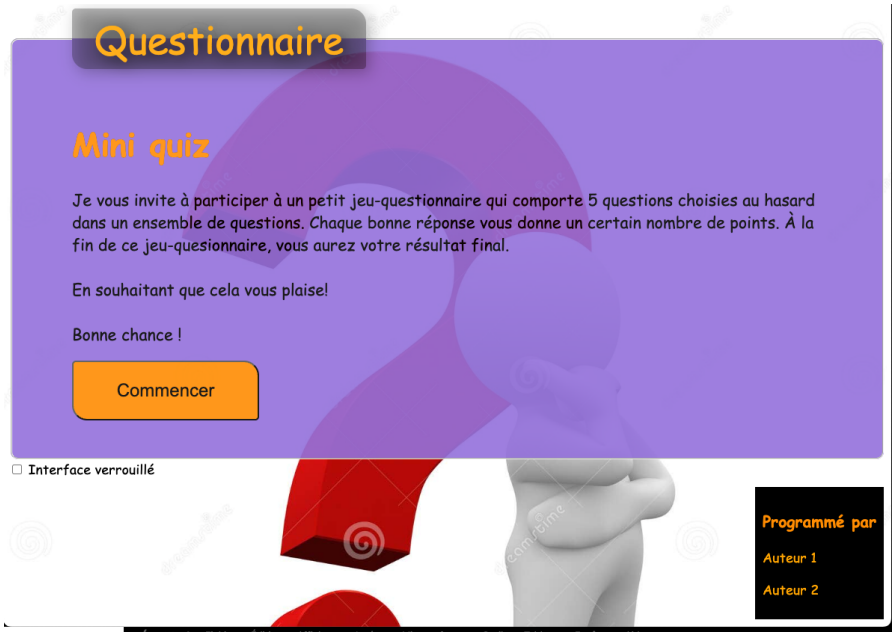
IMPORTANT : Les questions doivent être réelles et toutes composées en relation avec la matière du cours. Utilisez la matière JavaScript vue dans les 6 premières semaines de cours (à partir des notes de cours, des formatifs et des sites Web identifiés dans le cours).

2. Chargement des questions dans l'application

   Au départ de l'application, les questions contenues dans le tableau associatif JSON (fichier questionsJSON.js) doivent être transférées dans un tableau indexé (« Array ») d'objets « *Question* ». Cette structure (le array) servira de tableau « pool » de questions par la suite. Ce n'est pas le questionnaire (Quiz).

3. Premier affichage de l'application

   Le premier affichage de votre application devrait présenter à l'utilisateur vos règles de fonctionnement et la possibilité de jouer. Voici l'exemple :



- Ici le bouton « Commencer » permet de commencer une partie.
- Modifier le fichier HTML pour qu'il affiche vos noms (au lieu de *Auteur 1* et *Auteur 2*)

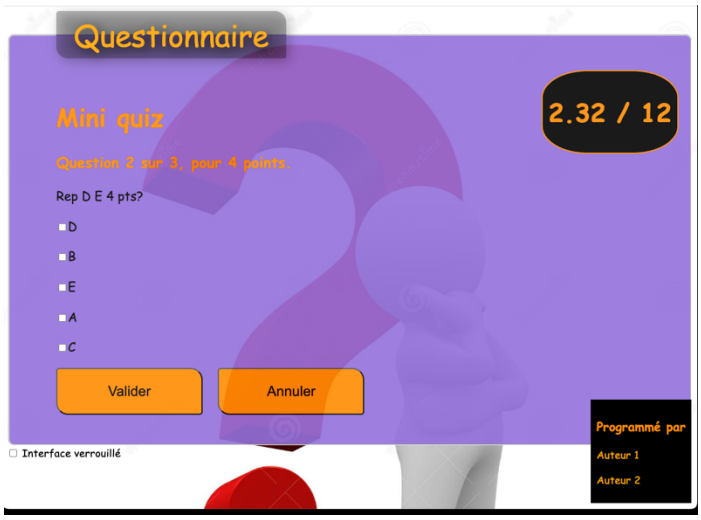
4. Choix des questions du questionnaire (quiz)

   Si le bouton « Jouer » est cliqué, il faut choisir aléatoirement, parmi l'ensemble des questions du tableau d'objets « Question », 5 questions pour fabriquer un objet « Quiz ». Il ne faut pas que les questions se répètent dans le même questionnaire (quiz).

   Après la sélection des 5 questions pour le quiz, le tableau d'objets « Question » principal ne doit pas être altéré.

5. Présentation d'une question

Maintenant que l'on a un objet « Quiz », il faut poser les questions de ce « Quiz ». Chaque question est présentée une à la fois à l'utilisateur. On doit voir le numéro de question à laquelle on est rendu, le nombre de questions total et le nombre de points accordés pour la question courante. Il doit y avoir 2 boutons (*Valider* et *Annuler*). La navigation doit être simple et claire. On ne peut pas revenir en arrière. Voici un exemple de présentation :



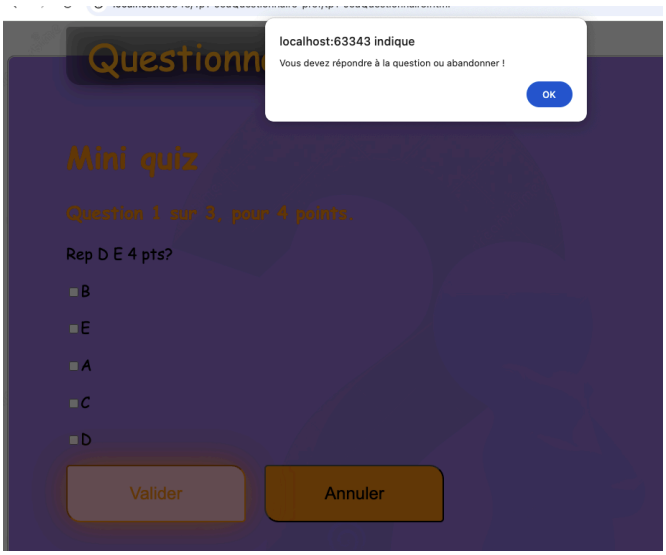
Ici le bouton « *Valider* » permet de montrer à l'utilisateur la correction de la question. Le bouton « *Annuler*» termine et retourne aussitôt à l'écran d'introduction. Cette dernière éventualité sera présentée plus loin. Notez que tous les boutons doivent être placés de façon permanente dans le fichier HTML. Pour afficher ou masquer un bouton, utilisez la propriété :

```
bouton.style.display = "inline"; // pour afficher
bouton.style.display = "none"; // pour le masquer complètement
```

Il faut utiliser des checkbox pour que l'utilisateur puisse choisir toutes les réponses qui lui conviennent. Les textes des questions seront placés dans les *labels* des *checkbox*es.

6. Bouton « Vérifier réponse »

Lorsque ce bouton est cliqué, la correction de la question est présentée. Mais avant, il faut vérifier si l'utilisateur a saisi au moins une réponse. S'il a appuyé sur Valider sans avoir saisi de réponse, il faut lui afficher un dialogue pour l'obliger à saisir une réponse :



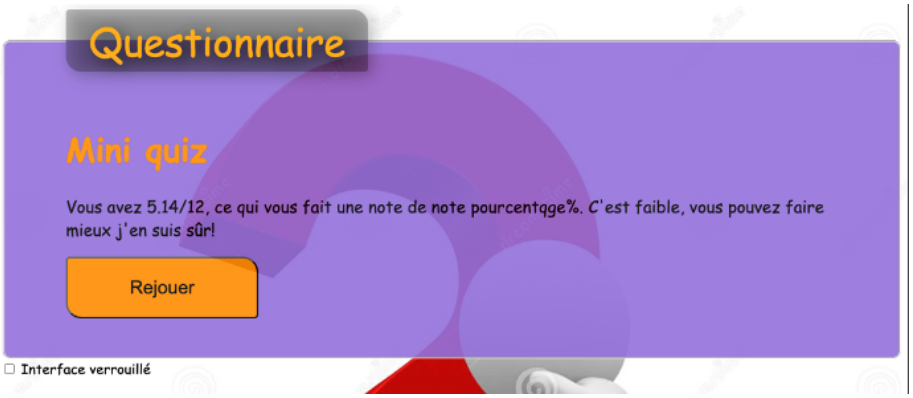
La correction doit ressembler à ceci :

- On utilise un code de couleur pour indiquer la validité des réponses :
  - Vert est une bonne réponse qui a été choisie par l'utilisateur.
  - Rouge est une bonne réponse qui a été oubliée par l'utilisateur.
  - Orange est une mauvaise réponse que l'utilisateur n'aurait pas dû cocher.
  - En noir sont les mauvaises réponses qui n'ont pas été choisies par l'utilisateur.
- Après chaque réponse la correction ajoute une flèche et le nombre de points accordé ou perdu pour chaque élément de réponse.
- Dans la boite noire en haut à droite, on retrouve le pointage total du quiz mis à jour à chaque validation de réponse.
- Dans la page de validation, on désactive les *checkboxes* parce que l'utilisateur ne peut alors plus changer ses réponses.
- Les 2 boutons, « *Continuer* » et « *Annuler* », permettent alors respectivement de passer à la question suivante ou d'annuler le quiz et retourner à l'affichage d'introduction.

7. Fin du questionnaire sans abandon

Lorsque vous affichez **la correction** de la **dernière question**, remarquez ici que le bouton « Annuler » a disparu, car, de toute façon, le questionnaire est fini. Maintenant, je peux que cliquer sur le bouton « C'est terminé, voir vos résultats! », pour voir mon résultat.

Voici la présentation des résultats finaux pour un quiz.



Vous affichez le nombre de points sur le nombre total qu'on aurait pu avoir, la note en pourcentage et un commentaire **d'encouragement personnalisé** selon la note en pourcentage. Voici la liste des différents intervalles permettant d'émettre des **messages personnalisés d'encouragement croisant**.

- De 0% à 29.99%
- De 30% à 59.00%
- De 60% à 69.99%
- De 70% à 84.99%
- De 85% à 94.99%
- De 95% à 100%

À vous de composer les messages d'encouragement positifs selon les intervalles.

8. Bouton « Abandonner le questionnaire »

Lorsque ce bouton est cliqué, le jeu retourne immédiatement à l'écran d'accueil.

Si je rejoue, j'ai de nouveau 5 **nouvelles** questions et je recommence le jeu. Assurez-vous que les pointages sont bien réinitialisés à 0 partout !

9. Boni

- Chaque personne peut effectuer un seul des 2 boni.

BONUS 1 possible, si votre code s'adapte dynamiquement aux nombres de messages, la personne qui le fait obtiendra 3% de bonus. Le code devra donc déterminer dynamiquement les plages de note en fonction du nombre de messages dans le tableau de messages. Les plages sont alors uniformes.

Bonus 2 Un second 3% de bonus seront accordés à la personne qui désactivera l'interface au complet lorsque le *checkbox* « *Interface verrouillée* » est coché. Cela doit être fait en gérant les événements avec un seul gestionnaire appliqué à un seul endroit. Le bonus ne sera accordé que si votre solution est élégante, simple et efficace.

Progression suggérée

1. Ajoutez vos noms comme auteurs dans le fichier *html*.
2. Faire les données statiques dans les fichiers de données.
3. Importer les données statiques dans script et les convertir en tableau d'objet Question.
    a. Faire le choix et le mélange des questions et des réponses.
4. Programmez les différentes interfaces dans l'ordre d'exécution du quiz et ajoutez le code nécessaire dans *elementHTML.js* au fur et à mesure.
    a. La page d'introduction
    b. La page de question. Note que c'est l'objet Quiz qui devrait gérer le changement de données entre 2 pages successives de questions.
        i. Programmez la validation des réponses et le calcul des points
        ii. Programmez l'ajustement de l'interface pour montrer le résultat.
    c. La page de validation
        i. Algo de validations
    d. La page de résultats finaux.
5. Corrigez les bogues et ajustez le UI.

Grille de correction

La grille de correction vous sera fournie la semaine prochaine.

Conseils

- Répartir le travail équitablement.
- Fixez-vous des objectifs pour chaque semaine et donnez une date d'échéance à vos objectifs. Si vous ne parvenez pas à l'atteindre avant l'échéance, entendez-vous avec votre coéquipier. Si vo
- Prendre le temps de bien comprendre l'ensemble du problème.
- Dresser une liste détaillée de petites actions ou étapes de développement à faire et ordonner cette dernière.
- Développer une action à la fois et tester correctement son fonctionnement.
- Ne pas essayer de développer des comportements définitfs trop vite, mais passer plutôt par une évolution itérative vers un état final.
- Servez-vous de F12 ou ctrl-shift-i dans le « browser » pour voir et comprendre vos erreurs.
- Ne surestimez pas votre programmation, programmez à petits pas, prenez le temps de tester, de tracer et de déboguer votre code au fur et à mesure que vous avancez.