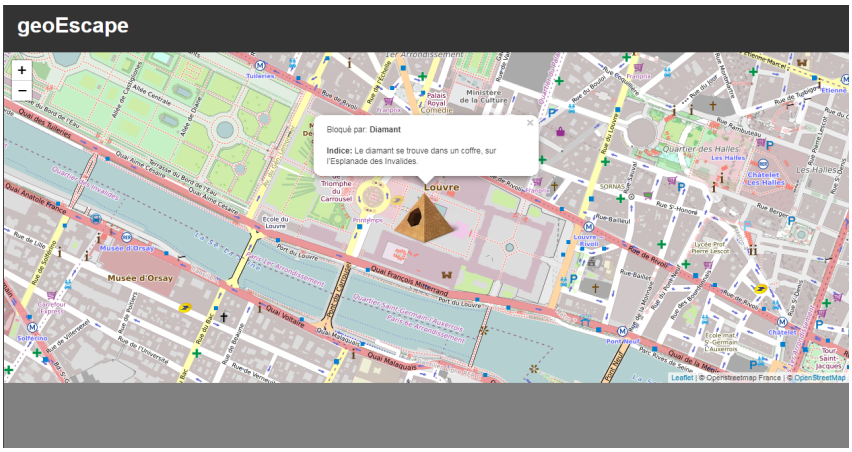
« Escape game » géographique !

L'objectif de ce TP est de créer un « escape game », notamment en résolvant des énigmes et en trouvant des objets. Le tout, sur une carte web.



Le design[1] de l'application est totalement libre. À vous donc de faire les bons choix en matière d'ergonomie, de rendu graphique, d'interaction, etc. Pensez cette application pour le « grand public », elle ne s'adresse pas à des professionnels. Demandez-vous comment vous aimeriez qu'elle fonctionne. Trouvez lui un nom.

Vous n'êtes pas contraints pour la technologie serveur (PHP, Node.js, etc.) ni par le système de base de données (MySQL, PostgreSQL, etc.).

Enfin, cette application doit être développée dans le cadre de la validation du cours. N'en faites pas plus que demandé. Ne restez pas bloqués trop longtemps sur un problème, venez-nous voir en L308, ou envoyez-nous vos questions par mail.

Rendu attendu

- Tous les fichiers client et serveur
  - HTML/CSS/JavaScript
  - PHP/Node.js
  - Images, Vidéos, Sons, etc.
- Export SQL de la base de données
- README pour les consignes « d'installation » des différents composants
- Solutions des énigmes

Soit par mail (attention, pas de format .zip ni .js sur des adresses ENSG/IGN), sur clé USB en L308 ou sur un dépôt Git (GitHub, GitLab, etc.)

Plus ce sera simple à re-déployer sur nos machines personnelles, plus on sera contents. ☺

Note : attention au copiés-collés ! Bien que ce soit un travail en groupe, cela ne signifie pas que vos fichiers doivent être identiques entre les groupes, ni même avec les années précédentes. Il est même difficile que ça puisse être le cas. Et vous devez vous douter qu'il est très facile de faire des recherches de similitudes. À bon entendeur, bon travail ! ;)

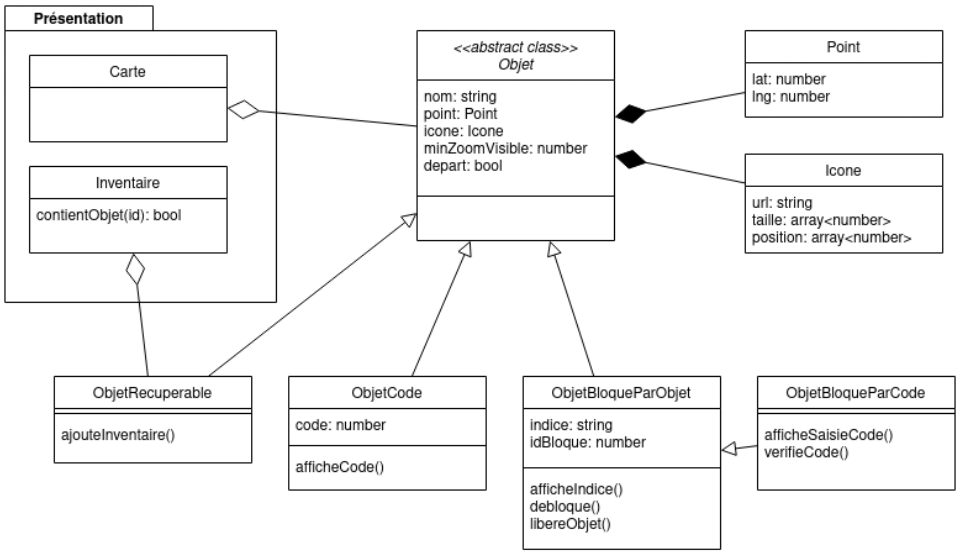
1 Le design au sens large, tel qu'utilisé en anglais, que l'on pourrait traduire par conception

Création de l’énigme

Tout d’abord, voici les différents types d’objets que nous souhaitons :

- Objet récupérable
  - Un objet que l’on peut récupérer et conserver dans notre inventaire
- Objet code
  - Un objet qui affiche un code à 4 chiffres
- Objet bloqué par un autre objet
  - Un objet qui nécessite d’avoir le bon objet dans son inventaire pour le débloquer
  - Quand débloqué, libère un autre objet
- Objet bloqué par un code
  - Un objet qui nécessite un code pour le débloquer
  - Quand débloqué, libère un autre objet

Un objet peut être chargé au démarrage du jeu ou non. Un objet doit également avoir un nom, une position géographique, un niveau de zoom minimum à partir duquel il est visible, une icône (image, taille, position de l’ancre). Un objet bloqué peut également avoir un indice pour aider le joueur. Voilà un diagramme de classe représentant le modèle conceptuel des données :



Commencez par créer le déroulement de votre jeu (enchaînements des objets, histoire, etc.). Il doit y avoir au minimum un objet de chaque type, et au maximum 16 objets. Il peut être assez simple au départ, il sera ensuite facile de l’enrichir, sans pour autant modifier la structure de votre code. C’est même l’objectif principal.

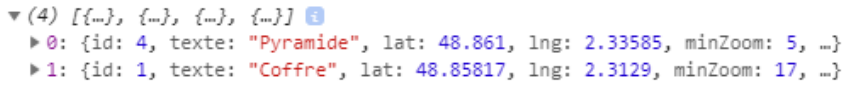
Base de données

- En partant du modèle proposé, implémentez une structure de base de données pour vos objets (cours UML)
- Remplissez cette base avec vos données

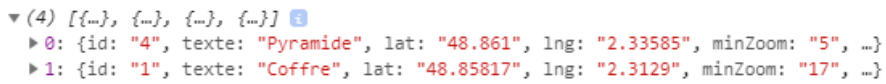
API

- Créez votre service web. Par exemple :
  - `objets.php` renvoie tous les objets qui permettent de commencer le jeu
  - `objets.php?id=N` renvoie l'objet dont l'identifiant est N
  - Mieux, si vous utilisez le framework FlightPHP (tutoriel sur le site du cours), vous pouvez créer votre propre API REST et gérer des URLs de type
    - `GET api/objets`
    - `GET api/objets/2`
  - A noter que PHP n'est pas indispensable et que l'API peut être réalisée en Python et ou Node.js (mais il faut aussi penser à servir les fichiers HTML, CSS et JS)
- Votre service web/API doit renvoyer les résultats en JSON
  - Vérifier l'encodage des nombres
- Si vous avez plusieurs tables, faites les bonnes jointures et faites attention aux noms de champs identiques

Bon retour JSON



Moins bon retour JSON



Interface web

La partie cliente doit contenir une carte (Leaflet, Google Maps, etc.), éventuellement avec le provider de tuiles que vous souhaitez (OpenStreetMap, Mapbox, etc.) et un inventaire qui permet de stocker les objets récupérés.

Lors du début du jeu, appelez votre API en AJAX pour charger les objets utiles au départ du jeu (retour JSON). Pour chaque objet, créez un marqueur sur la carte (à la bonne position, avec son icône, et seulement visible en fonction du zoom)

Pour chaque objet, il faut ensuite gérer l'évènement `click` et exécuter les bonnes actions en fonction de son type :

- L'objet est récupérable :
  - On le déplace dans l'inventaire
  - L'objet n'est plus visible sur la carte
- L'objet est un code :
  - On affiche le code
- L'objet est bloqué par un autre objet :
  - On a déjà l'objet qui débloque (l'objet est bien sélectionné dans mon inventaire)
    - L'objet n'est plus bloqué, et libère un nouvel objet
      - On appelle l'API pour récupérer les données de ce nouvel objet
      - On crée le nouvel objet (marqueur) comme précédemment
  - On n'a pas encore l'objet qui débloque
    - On appelle l'API en AJAX pour connaître l'objet bloquant et l'indice associé
- L'objet est bloqué par un code :

    - On appelle l’API en AJAX pour connaître l’objet code et affichez l’indice
    - On crée un formulaire pour saisir le code
    - On vérifie les données envoyées lors de la validation
    - On valide ou non
    - L’objet libère un nouvel objet (idem précédemment)

Scores

- À la fin du jeu, sauvegardez le score du joueur dans la base de données (vous aurez besoin de son nom/pseudo). Il faudra bien entendu établir une méthode de calcul d’un score.
- Ajoutez sur votre page d’accueil le « Hall of fame » (le top 10 des meilleurs scores par ex) pour tous les joueurs, généré dès le chargement de la page

Bon travail !