AJAX - Enoncé 4

D.Moreaux

13 octobre 2024

Résumé

Lister des éléments reçus du serveur dans un objet JSON. Afficher la liste des valeurs présentes dans les éditeurs. Début de gestion de l'aventure.

1 Editeur

Ajouter le listing des données présentes pour les différents éditeurs (les fonctionnalités d'édition et d'effacement seront effectuées plus tard, après avoir vu les closures).

pièces les pièces encodées seront affichées avec leur numéro dans un tableau par ordre de numéros croissants. Deux icônes seront prévues pour modifier et effacer une entrée.

messages les messages seront affichées avec leurs numéros, dans un tableau et par ordre de numéros croissants. Des icônes seront également prévues pour modifier et effacer.

messages système l'interface est similaire à l'affichage des messages mais sans possibilité d'ajout ou d'effacement.

déplacements Un tableau affichera les numéros des pièces de départs et disposera de une ou plusieurs lignes pour les destinations (sous la forme mot de vocabulaire et pièce destination plus une option pour effacer l'entrée). Les ajouts seront programmés plus tard.

Objets Un tableau listera les objets par numéros croissants. une liste déroulante affichera les valeurs *non créé*, *transporté*, *porté* ainsi que la liste des pièces existantes. Pour le moment les changements ne seront pas sauvés (closures). Une colonne disposera d'une entrée *vide* (par exemple 5 tirets) et de la liste des mots disponibles dans une liste déroulante. On prévoira également les boutons d'édition et d'effacement

Vocabulaire Le tableau listera les mots par numéros croissants. chaque numéro peut proposer plusieurs mots.

Les mots de 1 à 12 sont destinés aux directions (nord, sud, ...) et on peut prévoir une indication visuelle (couleur, petite icone,...).

Les mots dont le numéro est supérieur à 1000 indiquent que le mot peut désigner un objet que l'on peut porter (vêtement, armure, ...) et pourra également être indiqué.

Chaque mot disposera d'un bouton qui permetta de l'effacer (closure).

Un bouton en haut de page permettra d'ajouter un mot en précisant son numéro et le mot.

événements et status Cet éditeur sera programmé plus tard (après avoir vu la sérialisation).

2 Jeu

Un début de traitement des commandes sera mis en place. L'ordinogramme se trouve aux pages 34 et 35 du manuel de The Quill (voir descriptif du projet).

2.1 Etat de la partie

La partie contiendra une série de valeurs qui seront conservées dans un tableau situé dans la session. Normalement, ce tableau devrait être sauvé lors de la sortie de session et chargé lorsque l'on rentre en session. La session contiendra d'autres ressources telles que l'id de l'utilisateur connecté qui se trouveront hors de ce tableau d'état.

Pour les nouveaux comptes ou en cas d'aventure terminée, le champ dans la DB sera vide. Dans ce cas, il faudra l'initialiser. On se contentera d'initialiser l'état de la machine d'état, le reste le sera par l'état *initialisation* de cette machine d'état.

L'état de la partie contiendra

— un tableau de flags
— un tableau indiquant les positions des objets
— la pièce courante
— l'état du jeu [1]. On y trouvera l'état de la machine d'état, les mots entrés, la table en cours, l'entrée dans la table, la ligne d'action,...
— l'état de l'affichage depuis la dernière interaction avec le joueur [2]

1. le bouton logout peut être enfoncé à n'importe quel moment, par exemple alors qu'on faisait une pause entre deux messages. Il faut donc pouvoir restaurer l'état de jeu quand on se reconnecte.

2. tout comme l'état du jeu, cela permet de restaurer une partie suspendue.

Le tableau de flags sera initialement un tableau de 33 éléments nuls[3]. Le flag 1 sera ensuite initialisé au nombre d'objets transportés.

Le tableau indiquant les positions des objets sera initialisé à partir de la DB

La pièce courante sera initialisée à 0.

L'état de l'affichage sera initialisé à une chaîne vide.

2.2 Côté client : les actions

Le point central de l'interface sera une gestion d'actions. Cela permet au client d'être contrôlé par le serveur.

Elle commence par demander l'action courante au serveur et réagira aux situations suivantes :

— CMD : demande d'envoyer une commande, elle sera transmise au serveur. Dans les autres situations, la ligne de commande sera soit désactivée, soit aura une utilisation particulière (point 9 *Get Command* dans l'ordinogramme).
— QUIT : demande de confirmation de fin de partie. Cela se fera soit par une boîte de dialogue, soit par l'entrée de oui ou non sur la ligne de commande. Lorsque cet état est envoyé, il sera accompagné des messages système 12 (texte de la question), 30 (oui) et 31 (non).
— END : demande si on désire recommencer une partie. Similaire à QUIT mais avec les messages 13, 30 et 31.
— ANYKEY : affiche le message 16 (pressez une touche) et attends que l'on appuie sur une touche ou le bouton d'envoi de commande
— SAVE : sera programmé plus tard, permettra de choisir le slot de sauvegarde et d'effectuer cette dernière
— LOAD : idem SAVE mais en permettant de choisir le slot.
— PAUSE : attend le délai avant de continuer
— TEXT : affiche un texte passé en paramètre (description, message, ...). Un paramètre indique s'il faut effacer l'écran avant d'afficher le message.
— RESET : Nouvelle partie (remettre l'écran de jeu en situation de début de partie)
— LOGOUT : Fermeture de session (partie non sauvée côté serveur)
— NOP : cet état est un état bidon qui sera utilisé en sortie de certains fichiers PHP pour que le traitement revienne à la normale

D'autres commandes pourraient être ajoutées plus tard pour des choses telles que changer la couleur de l'écran et d'écriture, jouer un son ou afficher une image. Mais ces fonctionnalités ne sont pas prévues pour le moment.

Le fonctionnement s'orientera comme suit côté navigateur.

3. mais, contrairement au moteur Quill original, il sera possible d'avoir plus de flags

On commence (après un login) comme si on avait reçu une action NOP.

On envoie au serveur l'action qu'on vient de traiter et les paramètres éventuels. Le serveur enverra en retour un objet JSON qui contiendra l'action suivante ainsi que les éventuels paramètres.

La fonction de traitement fera un switch sur l'action reçue. Soit elle a une action ponctuelle (TEXT par exemple) et on retourne à la fonction qui envoie l'action qui vient d'être traitée, soit elle demande une action de l'utilisateur, on *prépare* le terrain (activer des gestionnaires d'événements, mettre des valeurs à certaines variables globales, ...) avant de laisser la main au navigateur.

Pour les fonctions qui demandent une action de l'utilisateur, lorsque l'événement correspondant se déclenche (click, keydown, timer,...), la fonction traitera l'événement et se chargera d'envoyer la réponse au serveur (sauf exception, en utilisant la même fonction de traitement que la fonction *de base*)

Pour le moment, on s'intéressera uniquement aux actions CMD, TEXT et NOP.

Quelques cas particuliers :

— l'action LOGOUT forcera un LOGOUT (utilisée après avoir répondu non à l'état END)
— l'action RESET ramènera le jeu à la situation d'initialisation (comme après un login sur une nouvelle partie)
— Les actions LOAD et SAVE enverront sur des fichiers PHP séparés qui effectueront la sauvegarde avant de retourner un état NOP

2.3 La machine d'état

Au niveau serveur, on s'articulera autour de quelques fichiers PHP : les fichiers load.php et save.php qui seront utilisés pour compléter les sauvegarde et le fichier principal qui traitera tout le reste.

Le fichier principal utilisera l'état du système pour savoir ce qu'il doit faire. Nous ne gérerons dans cet exercice qu'une partie de l'ordinogramme.

initialisation : atteint après un END où la réponse est Oui ou après un login si la partie sauvée est vide. L'état suivant sera description. Envoie une action RESET

description : Envoie une action TEXT avec en contenu la description de la pièce courante. L'état suivant est temporairement getcommand.

getcommand : envoie une action CMD, l'état suivant est decodecommand.

decodecommand : recherche les mots de la commande dans le vocabulaire.Une fois que deux mots ont été trouvés, arrête la recherche. Les numéros des mots seront sauvés dans deux variables d'état.

Si aucun mot n'est reconnu, on envoie une action texte avec le message système 6 et on retourne (temporairement) à l'état getcommand.

Sinon, on poursuit (case 10 dans l'ordinogramme) en cherchant dans la table de déplacement le premier mot entré pour la pièce en cours.

Si on le trouve, on change la pièce courante en la destination correspondante et on envoie une action NOP avec comme état suivant description.

Si on ne le trouve pas, on envoie (temporairement) une action TEXT avec le message système 7 si le premier mot a un numéro inférieur à 13 ou le message système 8 sinon et on retourne à getcommand.

Les états peuvent être sauvés sous forme de chaîne ou de valeurs numériques (avec des define() ou des commentaires dans le code).

Dans l'état actuel, le programme permettra de se déplacer d'une salle à l'autre mais ne reconnaîtra aucune autre commande.

3 La DB

Le fichier `demo.sql` permet d'insérer une aventure de test dans la DB. Cette aventure est celle présentée dans l'ouvrage *beginner's guide* reprise dans l'intro du projet.