Équipe 304

Protocole de communication

Version 1.5

Historique des révisions

Date	Version	Description	Auteur
2023-03-18	1.0	Réponse à la partie 2 avec les fonctionnalités du sprint 1 et 2	Junaid Qureshi
2023-03-18	1.1	Réponse à la partie 3 avec les fonctionnalités du sprint 1 et 2	Junaid Qureshi
2023-03-19	1.2	Ajout des éléments du sprint 3 et ajout de l'introduction	Junaid Qureshi
2023-03-20	1.3	Révision du contenu	Pierre Tran
2023-04-19	1.4	Correction après sprint 2	Junaid Qureshi
2023-04-19	1.5	Ajout des requêtes manquantes	Junaid Qureshi

Table des matières

Protocole de communication

1. Introduction

Durant la conception d'une architecture client-serveur, les développeurs doivent tenir de plusieurs aspects fondamentaux, dont les communications entre plusieurs postes. Ces communications doivent être correctement implémentées pour assurer le fonctionnement de l'architecture. Voilà pourquoi, il est important de bien définir à partir de ce document. Dans l'ensemble, le contenu fournit une vue d'ensemble des protocoles de communication et des paquets de données utilisés dans l'architecture client-serveur. Ce contenu est divisé en deux sections.

La première section explique l'utilisation des requêtes HTTP et des sockets web dans une communication client-serveur. Elle décrit entre autres les fonctionnalités qui utilisent les requêtes HTTP telles que le chargement des niveaux, la mise à jour des constantes et la réinitialisation des scores. Elle explique également les fonctionnalités qui utilisent les sockets web, telles que la mise à jour de l'interface utilisateur en temps réel, la messagerie, la vérification des clics et la participation ou la création d'un jeu.

La deuxième section décrit les différents paquets utilisés dans le protocole WebSocket et le protocole HTTP. D'une part, pour le WS, cette section énumère le nom de l'événement, la direction du flux de données et les données envoyées ou reçues par le serveur et le client. D'autre part, en ce qui concerne les requêtes HTTP, elle énumère la méthode, la route, le corps, les paramètres et les données de retour.

2. Communication client-serveur

Les requêtes HTTP sont utilisées pour les fonctionnalités suivantes :

- Avant de charger la page de sélection ou la page de configuration, tous les niveaux de la base de données sont chargés dans la page. (GET)
- Avant de commencer un jeu, les informations relatives au niveau du serveur sont chargés (GET)
- La création du jeu nécessite également cette technologie, car nous devons envoyer les informations du jeu au serveur. (POST)
- Recevoir les constantes des jeux (GET)
- Mise à jour des constantes globales (PATCH)
- La remise à zéro du classement d'un niveau spécifique (PATCH)
- La réinitialisation des constantes globales (PATCH)
- Recevoir l'historique des jeux (GET)
- Supprimer l'historique des jeux (DELETE)

J'ai utilisé les requêtes HTTP pour ces fonctionnalités car elles impliquent principalement la transmission d'informations entre le client et le serveur. Les requêtes HTTP permettent une communication simple et efficace entre les deux, ce qui permet de charger les données nécessaires à l'affichage de la page, d'envoyer les informations du jeu au serveur pour la création d'une partie, de récupérer les constantes et l'historique des jeux, ainsi que de mettre à jour les données côté serveur telles que les constantes globales et le classement des niveaux.

De plus, ces fonctionnalités ne nécessitent pas de communication bidirectionnelle constante entre le client et le serveur, car elles n'affectent pas directement les autres joueurs. En effet, on ne doit pas mettre à jour l'interface des autres pour ces fonctionnalités. Par conséquent, l'utilisation des requêtes HTTP est suffisante pour ces cas d'utilisation et ne nécessite pas d'autres protocoles de communication plus complexes.

Nous avons utilisé des sockets web pour les fonctionnalités suivantes :

- Le chronomètre du jeu utilise des sockets web pour mettre à jour activement (à chaque seconde) l'interface utilisateur des joueurs dans un match.
- Le système de messagerie les utilise pour mettre à jour la boîte de dialogue lorsqu'un message est envoyé au serveur ou lorsque le serveur doit envoyer un message spécial.

- La vérification d'un clic est également gérée par des sockets web, car une fois le clic vérifié, l'autre joueur doit également être notifié. Cela permet aux deux joueurs d'être informés du nombre de différences trouvées.
- La suppression d'un niveau doit les utiliser car elle doit vérifier s'il y a actuellement des joueurs qui attendent une partie ou qui sont en jeu dans ce niveau. S'il en trouve, il doit en informer tous les joueurs concernés.
- La fonctionnalité consistant à rejoindre ou à créer un jeu repose aussi fortement sur les sockets web, car il y a une communication constante entre deux joueurs à la recherche d'une partie. Cela inclut l'annulation, le refus ou l'acceptation d'une demande de partie.
- L'abandon d'une partie utilise également des sockets web, car il s'agit de notifier à l'autre joueur que le match est terminé.
- La page de sélection et de configuration en a besoin pour qu'ils se mettent à jour si une page est supprimée.
- Le mode triche les utilisent pour pouvoir changer l'état du jeu et recevoir les données des différences.
- Le système d'indice l'utilise pour savoir quelles différences ont été trouvées.
- La sauvegarde d'un match se fait par des sockets, car c'est la passerelle WS qui détermine la fin d'un match.
- Les messages globaux doivent être gérés par les sockets car tous les joueurs doivent les recevoir.
- Le mode de jeu classique et temps limité sont entièrement géré par les WS.
- Lors de l'ajout ou la suppression d'un niveau, tous les clients doivent avoir les interfaces rafraîchies.

Nous avons utilisé des sockets web pour tout le reste. Ce qui inclut notamment toute la logique du jeu car cette logique doit être répliquée sur plusieurs clients via leur interface de jeu. De plus, toutes les informations de la base de données qui doivent être mises à jour après la fin d'un jeu sont également gérées dans ces sockets car elles doivent savoir ce qui se passe dans le jeu à tout moment.

3. Description des paquets

Protocole WS

Nom de l'événement	**Direction**	**Données**
OnJoinNewGame	Le client vers le serveur	Le serveur reçoit un objet "data" qui contient: - levelId (number): Le numéro identifiant du niveau. - playerName (string): Le nom du joueur.
OnCreateTimedGame	Le client vers le serveur	Le serveur reçoit un objet "gameInformation" qui contient: - levelId (number): Le numéro identifiant du niveau. - playerName (string): Le nom du joueur.

onClick	Le client vers le serveur	position (number): La position du clic sur le canvas.
onGameSelection	Le client vers le serveur	Le serveur reçoit un objet "data" qui contient: - levelId (number): Le numéro identifiant du niveau. - playerName (string): Le nom du joueur.
onGameAccepted	Le client vers le serveur	Aucune
onGameCancelledWhileWaiting	Le client vers le serveur	Aucune
OnTimedGameCanelled	Le client vers le serveur	Aucune
onGameRejected	Le client vers le serveur	Aucune
onDeleteLevel	Le client vers le serveur	levelId (number): Le numéro identifiant du niveau.
onAbandonGame	Le client vers le serveur	Aucune
onDemandForHint	Le client vers le serveur	Aucune
onGameSelection	Le client vers le serveur	Le serveur reçoit un objet "data" qui contient: - levelId (number): Le numéro identifiant du niveau. - playerName (string): Le nom du joueur.
onStartCheatMode	Le client vers le serveur	Aucune
onStopCheatMode	Le client vers le serveur	Aucune
onMessageReception	Le client vers le serveur	message (ChatMessage): Le message envoyé par le client

onHintRequest	Le client vers le serveur	Aucune
onRefreshLevels	Le client vers le serveur	Aucune
processClick	Le serveur vers le ou les clients d'une partie	Le serveur envoie un objet "GameData" qui contient: - differencePixels (number[]): Le tableau de pixels qui appartiennent à la différence cliquée. - totalDifferences (number): Le nombre de différences de l'image. - amountOfDifferencesFound (number): Le nombre de différences trouvées par le joueur. -amountOfDifferencesFoundSecondPlayer (number, optionnelle): Le nombre de différence trouvées par le deuxième joueur.
victory	Le serveur vers le client	Aucune
defeat	Le serveur vers l'autre client	Aucune
invalidName	Le serveur vers le client	Aucune
sendInvite	Le serveur vers l'autre client	Aucune
updateSelectionPage	Le serveur vers tous les clients	levelId (number): Le numéro identifiant du niveau. canJoin (boolean): Un état qui détermine si le niveau est joignable.
startMultiplayerGame	Le serveur vers un client	levelId: Le numéro identifiant du niveau. playerName: le nom du joueur. secondPlayerName: le nom de l'autre joueur.
rejectGame	Le serveur vers l'autre client	Aucune
shutdown	Le serveur vers un client	Aucune

startCheatMode	Le serveur vers le client	data (number[]): Liste de pixels de différence retrouvés dans l'image
opponentAbandoned	Le serveur vers l'autre client	Aucune
messageSent	Le serveur vers un client	Le serveur renvoie un objet "Message" qui contient:
refreshLevels	Le serveur vers tous les clients	Aucune
hintRequest	Le serveur vers un client	Aucune
startTimedMultiPlayerGame	Le serveur vers un ou deux clients	Aucune
changeLevelTimedGameMode	Le serveur vers un ou deux clients	level (Level): Le niveau de la prochaine fiche affichée.
timeModeFinished	Le serveur vers un ou deux clients	un boolean qui indique si le jeu a été fini à cause d'un manque de temps ou si il y n'y avait plus de fiches de jeu,

Protocole HTTP

Méthode: GET
Route: api/image/allLevels
Description de route:
-image: le contrôleur d'image et de niveaux
-allLevels: les niveaux présents la BD.
Corps: aucun
Paramètre: aucun
Code de retour: OK (200)
Contenu du retour: Une liste de toute l'information de tous les niveaux

Méthode:GET
Route: api/image/:id
Description de route:
-image: le contrôleur d'image et de niveaux
Corps: aucun
Paramètre: id: Le numéro identifiant du niveau
Code de retour: OK (200)
Contenu du retour: Toute l'information sur le niveau particulier

Méthode:POST
Route: api/image/postLevel
Description de route:
-image: le contrôleur d'image et de niveaux
-postLevel: l'ajout d'un niveau
Corps: formData (LevelFormData):
imageOriginal (File): Le fichier représentant l'image originale
imageDiff (File): Le fichier représentant l'image de différence
name (string): Le nom du niveau
isEasy (string): La difficulté du niveau
clusters (string): La de pixels de différence
nbDifferences (string): Le nombre de différence dans le niveau
Paramètre: aucun
Code de retour: Created (201)
Contenu du retour: Un objet Message qui contient soit une confirmation de la création du niveau, sinon un message d'erreur.

Méthode: PATCH
Route: api/database/constants
Description de la route:
-database: le contrôle de la base de données
-constants: les constantes de jeu
Corps: gameConstants (Interface):
countdownTime (number) : temps initiale du compte à rebours
penalityTime (number) : temps perdu à cause d'une pénalité
discoveryTime (number) : temps gagné à cause d'une découverte d'une différence
Paramètre: aucun
Code de retour: OK (200)
Contenu du retour: Rien

Méthode:GET
Route: api/database/constants
Description de la route:
-database: le contrôle de la base de données
-constants: les constantes de jeu
Corps:aucune
Paramètre: aucun
Code de retour: OK (200)
Contenu du retour: Rien

Méthode:PATCHE
Route: api/database/constants/reset
Description de la route:
-database: le contrôle de la base de données
-constants: les constantes de jeu
-reset: réinitialise les constantes de jeu
Corps:aucune
Paramètre: aucun
Code de retour: OK (200)
Contenu du retour: Rien

Méthode:GET
Route: api/database/gameHistories/
Description de la route:
-database: le contrôle de la base de données
-gameHistories: l'historique des jeux
Corps: aucun
Paramètre: aucun
Code de retour: OK (200)
Contenu du retour: Rien

Méthode: DELETE
Route: api/database/gameHistories/
Description de la route:
-database: le contrôle de la base de données
-gameHistories: l'historique des jeux
Corps: aucun
Paramètre: aucun
Code de retour: OK (200)
Contenu du retour: Rien