205

Protocole de communication

Version 2.3

Historique des révisions

Date	Version	Description	Auteur
2023-03-12	1.0	Rédaction de l'introduction	Lahbabi Ryan
2023-03-14	1.1	Description paquet get cards	Alexandre Plante
2023-03-14	1.2	Description des paquets de création de jeu	Zarine Ardekani
2023-03-15	1.3	Description des paquets de la vue de jeu et vue de sélection	Astir Tadrous
2023-03-19	1.4	Ajout et modification paquets de triche	Alexandre Plante
2023-03-21	1.5	Finir la description des requêtes HTTP	Zarine Ardekani
2023-03-21	1.6	Ajout requêtes pour meilleurs temps et message global	Alexandre Plante
2023-04-17	2.1	Introduction mise à jour	Lahbabi Ryan
2023-04-18	2.2	Description de la communication client-serveur finie	Lahbabi Ryan
2023-04-20	2.3	Mise à jour de la description des paquets	Zarine Ardekani

Table des matières

Protocole de communication

1. Introduction

Ce document décrit le protocole de communication de notre projet d'application Web du jeu des 7 différences que nous avons développé lors du semestre d'Hiver 2023 dans le cadre du cours LOG2990. Ce document est organisé en deux sections. La première section décrit le choix de moyen de communication entre les clients et le serveur, ainsi que les raisons d'utilisation des protocoles de communication sélectionnés pour le projet. La deuxième section décrit en détail les différents types de paquets utilisés dans le protocole de communication, en précisant leur contenu et leur présentation en fonction du protocole utilisé. En outre, les explications que vous trouverez dans ce document incluent toutes complétés dans les sprints 1, 2 et 3. Après avoir prédit à la fin du sprint 2 la façon dont nous allions implémenter les fonctionnalités du sprint 3, nous pouvons enfin vous montrer ce que nous avons finalement fait, maintenant que ce dernier sprint touche à sa fin.

2. Communication client-serveur

Dans le cadre d'une application web de jeu, la communication client-serveur est un élément crucial pour fournir une expérience utilisateur fluide et interactive. Dans ce contexte, il est important de choisir la technologie de communication la plus appropriée. Nous avons choisi d'utiliser exclusivement les WebSockets pour notre communication client-serveur pour plusieurs raisons.

Tout d'abord, les WebSockets sont une technologie de communication en temps réel qui permet une communication bidirectionnelle entre le client et le serveur. Cela signifie que les données peuvent être envoyées de manière asynchrone dans les deux sens sans que le client ne doive attendre une réponse du serveur avant de pouvoir envoyer des données supplémentaires. Cette approche est particulièrement adaptée pour les jeux en ligne, où la latence peut avoir un impact significatif sur la qualité de l'expérience utilisateur. Surtout que cela a l'avantage de mettre à jour constamment l'ensemble des clients de notre application lors de l'utilisation du jeu. De plus, les WebSockets sont conçus pour être utilisés avec des applications web modernes. Contrairement aux méthodes de communication traditionnelles telles que les requêtes HTTP, les WebSockets offrent une connexion persistante entre le client et le serveur, ce qui permet une utilisation efficace des ressources et une réduction de la charge du serveur.

En outre, WebSocket a amélioré les performances de notre application en réduisant la latence et en évitant les frais généraux de la création de nouvelles connexions pour chaque requête. WebSocket nous a permis également d'établir une communication de bas niveau, c'est-à-dire nous permettant de personnaliser davantage la façon dont les données sont envoyées et reçues entre le client et le serveur.

Dans le cadre de la création d'une partie, pour des parties multijoueur, un minimum de deux joueurs doivent communiquer ensemble. Pour que cette communication soit possible, il faut absolument utiliser Web Socket. Le protocole HTTP ne peut être utile que pour des parties solo.
Par souci d'homogénéité nous avons jugé qu'il serait plus optimal, de prendre le même protocole pour les parties multijoueurs et les parties solo (au lieu d'utiliser HTTP pour deux modes de jeux sur 4, d'autant plus qu'on utilise Websocket partout ailleurs). Vous allez alors remarquer dans les tableaux ci-dessous que nous avons utilisé pour toutes nos fonctionnalités websocket à la fin du sprint 3 même celles qui étaient dans le sprint 2 en HTTP.

Il y a différentes utilisations des WebSocket en fonction du besoin => ***.emit() - .broadcast() - .on()***:

- Dans le contexte d'une partie, les événements qui pourraient potentiellement être engendrés par l'utilisateur sont constamment en écoute au niveau du serveur, comme le clique d'une différence trouvée ou d'une erreur, l'envoie des message, la création d'une partie etc. On utilise ***.emit()*** en provenance du client et ***.on()*** au niveau du serveur
- Les événement reçus par l'adversaire qu'il écoute et reçoit par l'intermédiaire du serveur qui reçoit l'information et la broadcast comme les cliques ennemies, les messages textes ou d'événements de l'adversaire. On utilise ***.on()*** au niveau du client et ***.broadcast()*** au niveau du serveur
- Les événements demandés par l'utilisateur à l'aide de ***.emit()*** comme la demande d'indice de jeu, les

demandes de constantes de jeu, la demande de lancement du mode triches, l'historique des parties, la réinitialisation des meilleurs temps etc. Pour faire cela, le client ***.emit()*** au serveur, ensuite ***.on()*** au niveau du serveur puis il répond

instantanément avec un ***.emit()*** et le client qui attend la réponse du serveur avec un ***.on()***

Protocole WebSocket	Protocole HTTP
Messages de parties (événements, messages textes)	Aller chercher les anciennes cartes de jeu
Supprimer un jeu spécifique	Supprimer tous les jeux
Aller chercher les cartes de jeu	Supprimer un jeu spécifique
Création de nouvelles carte de jeu	
Mise à jour d'une carte de jeu qui vient d'être créé	
Récupérer les images de triches	
Changer le statut d'un jeu en "joignable"	
Afficher le meilleurs temps des jeux	
Effacer les images de triches côté client	

Figure 1 - Tableau des protocoles de communications entre client et serveur au sprint 2

Protocole WebSocket	Protocole HTTP
Messages de parties (événements, messages textes et messages globaux)	
Historique de parties	
Constantes de jeux	
Création de nouvelles carte de jeu	
Mise à jour d'une carte de jeu qui vient d'être créé	
Réinitialiser les meilleurs temps de tous les jeux/ d'un jeu	
Effacer les images de triches côté client	
Afficher le meilleurs temps des jeux	
Passer à la carte de jeu suivante en temps limité	
Changer le statut d'un jeu en "joignable"	
Indice de jeux	
Récupérer les images de triches	
Détection des cliques ennemis (erreur/différence trouvée)	
Supprimer tous les jeux/ un jeu spécifique	
Aller chercher les cartes de jeu	

Figure 2 - Tableau des protocoles de communications entre client et serveur au sprint 3

3. Description des paquets

Table des paquets WS

Nom de l'événement	**Source**	**Contenu**	**Autre informations**
joinable_game_cards	Client	-	
joinable_game_cards	Serveur	string[]	Liste des id de jeux avec partie en attente
get_cheats	Client	string	id de la partie/du salon
cheat	Serveur	string[]	Liste des images de flash en base 64. Réponse à l'événement get_cheats envoyé par le client
cheat_index	Serveur	number	Index de l'image de triche à retirer parce que la différence a été trouvée

send_card_validation_request	Client	{
		originalImage: string;
		modifiedImage: string;
		range: number,
		}
			Buffers de l'image originale et de l'image modifiée et rayon d'élargissement des différences

card_validation	Serveur	{
		valid: boolean;
		differenceNbr: number;
		difficulty?: Difficulty;
		differenceImage?: string;
		}
			Réponse à l'évènement send_card_validation_request

send_card_creation_request	Client	{
		originalImage: string;
		modifiedImage: string;
		range: number;
		name: string;
		}
			Buffers de l'image originale et de l'image modifiée, rayon d'élargissement des différences et nom du jeu

card_creation	Serveur	{
		valid: boolean;
		}
			Booléen indiquant si le jeu est valide.
			Réponse à l'évènement send_card_creation_request

click_personal	Serveur	{
		valid: boolean;
		playerName?: string;
		penaltyTime?: number;
		differenceNaturalOverlay?: string;
		differenceFlashOverlay?: string;
		}
			Réponse à l'événement send_click. Cet événement indique la validité du clique du joueur et donne accès aux flashOverlay et naturalOverlay si le clic est valide

send_click	Client	{
		gameId : string
		x: number
		y: number
		}
			Envoie des coordonnées du clic et de l'id du jeu en cours. Cet événement avertit le serveur lorsqu'un joueur clique sur une des deux images (soit l'image originale ou l'image modifiée)

click_enemy	serveur	{
		valid: boolean;
		playerName?: string;
		penaltyTime?: number;
		differenceNaturalOverlay?: string;
		differenceFlashOverlay?: string;
		}
			Cet événement survient seulement lors des modes 1VS1 et coop lorsque le 2e joueur effectue un clic. Cet événement indique la validité du clic du 2e joueur et donne accès aux flashOverlay et naturalOverlay si le clic est valide

endgame	serveur	{
		finalTime?: number;
		newBestTimes?: BestTimes;
		players: PlayerRecord[];
		}
			Contient les informations de fin de partie soit les informations des joueurs, le temps final mis pour compléter la partie si lieu et les nouveaux meilleurs temps de jeu si lieu
			PlayerRecord contient les informations suivantes :
			{
			name: string;
			winner: boolean;
			deserter: boolean;
			}
			BestTimes contient trois objets BestTime qui contiennent chacun les informations suivantes:
			{
			name: string;
			time: TimeConcept;
			}

leave_game	Client	{
		gameId : string
		}
			Contient l'id de la partie courante du jeu.
			Cet événement survient lorsqu'un joueur abandonne une partie en cours ou si un joueur quitte l'attente d'autres joueurs pour une partie 1vs1 (mode classique) ou pour une partie coop (mode temps limité)

validate_player	Client	{
		playerId : string
		gameId : string
		canJoin : boolean
		}
			Contient les informations concernant le 2e joueur voulant rejoindre une partie de jeu classique 1vs1.
			Cet événement est envoyé au serveur lorsque le 1er joueur d'une partie 1vs1 mode classique

			accepte ou rejete un 2e joueur essayant de rejoindre sa partie

request_to_play	Client	{
		gameMode: GameMode
		cardId? : string
		playerName : string
		}
			Informations envoyées au serveur lorsqu'un joueur veut jouer à une partie solo ou multijoueur en mode classique ou en mode temps limité.
			
			cardId est undefined pour le mode temps limité

response_to_play_request	Serveur	{
		responseType: GameConnectionAttemptResponseType;
		gameName: string;
		playerNbr: number;
		startingIn: number;
		originalImage: string;
		modifiedImage: string;
		time: number;
		gameId: string;
		difficulty: Difficulty;
		differenceNbr: number;
		hostName: string;
		gameValues: GameValues;
		}
			Réponse à l'événement response_to_play_request.
			
			Cet événement contient les informations nécessaires pour commencer une partie en mode classique ou en mode temps limité.
			
			GameConnectionAttemptResponseType est un enum dont les valeurs possibles sont starting, pending, cancelled et rejected.

player_status	Serveur	{
		playerConnectionStatus: PlayerConnectionStatus;
		user?: SimpleUser;
		playerNbr?: number;
		}
			Contient Informations du 2e joueur. Cet événement survient seulement lors d'une partie 1 vs 1 en mode classique ou lors d'une partie coop en mode temps limité.
			
			Si playerConnectionStatus est Left et le mode de jeu est temps limité, la vue de jeu se transforme en partie solo en mode temps limité
			
			PlayerConnectionStatus est un enum dont les valeurs possibles sont AttemptingToJoin, Left et Joined.
			
			SimpleUser contient les informations suivantes: {name: string, id: string}

next_card	Serveur	{ name: string; originalImage: string; modifiedImage: string; nbDifferences: number;	Cet événement permet d'obtenir les prochaines images (originales et modifées) pour le mode temps limité

		}	

hint	Client	{
		gameId : string
		}
			id de la partie courante du jeu.
			Cet événement est seulement disponible pour les parties solo en mode classique et temps limité

hint	Serveur	{
		start: Coordinates;
		end: Coordinates;
		}
			Coordonnées des coins du rectangle entourant une différence.
			Coordinates contient un nombre x et un nombre y.
			Réponse au message ‘hint’ du client

time	Serveur	number	Contient le temps en secondes du chronomètre. Cet événement sert à envoyer le temps à afficher pour les parties du mode classique et temps limité. Cet événement est envoyé à chaque 0.25 seconde et tient compte de la pénalité d’utilisation d’indices, ainsi que du bonus de temps obtenu à la suite d’une différence trouvée.

chat_message	Serveur	{
		sender: string;
		message: string;
		}
			Cet événement indique au client l’auteur du message et lui fournit le message à afficher.

global_message	Serveur	string	
record_beater	Serveur	string	Cet événement est généré lorsqu’un joueur réussit à battre un meilleur temps, afin qu’il s’affiche dans son message de fin de partie
deserter	Serveur	string	Cet événement est généré lorsqu’une personne quitte un partie coop ou 1v1. L’événement avertit le joueur que le 2e joueur a quitté la partie.

send_chat_message	Client	{
		gameId : string
		message : string
		}
			Envoie le message écrit par le joueur et l’id du jeu.
			Cet événement est généré à chaque fois qu’un joueur écrit un message dans une partie 1v1 ou

			une partie coop pour envoyer le message au serveur
is_playing	Client		Demande au serveur si une partie est en cours
is_playing	Serveur	boolean	Envoie un boolean indiquant si la partie a bien commencée ou non. Si une erreur est survenue, le boolean serait à false et le joueur sera transporté à la page d'accueil. Sinon, le jeu continue normalement.

set_game_values	Client	{
		timerTime: number;
		penaltyTime: number;
		gainedTime: number;
		}
			Modifier les valeurs des constantes de jeu

get_game_values	Client	-	Obtenir les valeurs des constantes de jeu

game_values	Server	{
		timerTime: number;
		penaltyTime: number;
		gainedTime: number;
		}
			Informations sur les constantes de jeu pour la partie qui va commencer. À afficher dans l'interface pendant la partie

all_game_cards	Client	-	Obtenir toutes les cartes de jeu

all_game_cards	Server	{
		id: string;
		name: string;
		difficulty: number;
		classicSoloBestTimes: BestTimes;
		classic1v1BestTimes: BestTimes;
		originalImage: string;
		}[ ]
			BestTimes contient trois objets BestTime qui contiennent chacun les informations suivantes:
			{
			name: string;
			time: TimeConcept;
			}

delete_all_cards	Client	-	Demander à supprimer toutes les fiches de jeu
delete_all_records	Client	-	Demander à supprimer l'historique

delete_card	Client	string	L'id de la carte à supprimer
card_delete_response	Server	boolean	Booléen indiquant si la carte a été supprimée
get_all_records	Client	-	Obtenir l'historique des parties jouées

all_records	Server	{
		startDate: string;
		duration:
		{ seconds: number;
		minutes: number };
		gameMode: GameMode;
		players: PlayerRecord[];
		}[ ]
			PlayerRecord est une interface contenant les informations suivantes:
			{
			name: string;
			winner: boolean;
			deserter: boolean;
			}

reset_best_times	Client	string	Réinitialiser les meilleurs temps de la carte ayant l'id spécifié
reset_all_best_times	Client	-	Réinitialiser les meilleurs temps de toutes les cartes de jeu

all_frontend_card_times	Server	{
		id?: string;
		name: string;
		classicSoloBestTimes: BestTimes;
		classic1v1BestTimes: BestTimes;
		difficulty: Difficulty;
		differenceNbr: number;
		differences: FinalDifference[];
		}[ ]
			FinalDifferences contient les informations suivantes:
			{
			yMin: number;
			yMax: number;
			lines: FinalLinearSet[][];
			hints: Hint[];
			}
			avec FinalLinearSet contient { start: number; end: number } et Hint contient { start: Coordinates; end: Coordinates }

frontend_card_times	Server	{
		id?: string;
		name: string;
		classicSoloBestTimes: BestTimes;
		classic1v1BestTimes: BestTimes;
		difficulty: Difficulty;
		differenceNbr: number;
		differences: FinalDifference[];
		}
			FinalDifferences contient les informations suivantes:
			{
			yMin: number;
			yMax: number;
			lines: FinalLinearSet[][];
			hints: Hint[];
			}
			avec FinalLinearSet contient { start: number; end: number } et

			Hint contient { start: Coordinates; end: Coordinates }

game_card	Server	{
		id: string;
		name: string;
		difficulty: number;
		classicSoloBestTimes: BestTimes;
		classic1v1BestTimes: BestTimes;
		originalImage: string;
		}
			BestTimes contient trois objets BestTime qui contiennent chacun les informations suivantes:
			{
			name: string;
			time: TimeConcept;
			}

history_received	Server	{
		startDate: string;
		duration:{ seconds: number;
		minutes: number };
		gameMode: GameMode;
		players: PlayerRecord[];
		}
			PlayerRecord est une interface contenant les informations suivantes:
			{
			name: string;
			winner: boolean;
			deserter: boolean;
			}

Table des paquets HTTP

Nous n'utilisons plus le protocole HTTP dans la version finale du projet.