<Équipe 101>

<Erratum>
Protocole de communication

Version 1.02

Historique des révisions

Date	Version	Description	Auteur
2023-09-20	1.0	Première itération avec les informations du protocole de communication de Projet 2	Etienne Desclaux
2023-09-22	1.01	Ajout des nouvelles fonctionnalités: Observation, authentication, mode Free-for-All. système d'amis, etc.	Etienne Desclaux
2023-09-29	1.02	Itération finale avec tous les requis : Description des paquets et des interfaces	Mohamed Reda Rhanmouni, Nina Lounici, Etienne Desclaux, Nassour Nassour, Ghali Chraibi, Amirmasood Dadkhah

Table des matières

Protocole de communication

1. Introduction

Ce document vise à donner une vue complète des communications utilisées dans notre application Erratum entre le client et le serveur, ainsi que leur fonctionnement.

En première partie, nous présentons les moyens de communication utilisés pour chaque fonctionnalité de notre application, et justifions notre choix entre le protocole HTTP et Web Socket pour les implémenter.

En deuxième partie, nous précisons chacune des opérations réalisées dans l'application et décrivons les interfaces correspondantes. Chaque opération est expliquée en détail, en mettant en évidence les flux de communication entre le client et le serveur. Les paquets envoyés entre le serveur et le client sont décrits, avec leur contenu et leur fonctionnement.

En somme, ce document offre une vue exhaustive des communications utilisées pour Erratum en décrivant les différents protocoles de communication et en montrant pourquoi tel protocole a été choisi pour chaque fonctionnalité.

2. Communication client-serveur

Le protocole WebSocket est utilisé pour les fonctionnalités spécifiques suivantes :

- La création d'une partie mode classique solo
- La création d'une partie mode classique multijoueur
- La création d'une partie mode temps limité solo
- La création d'une partie mode temps limité multijoueur
- La création d'une partie en mode Free-For-All
- Le système d'amis
- La vérification d'un clic dans une partie
- L'initialisation de la minuterie durant les parties
- La reprise vidéo d'une partie
- Les canaux de messagerie
- Le mode observateur
- L'authentification

Le protocole Web Socket est utilisé pour la logique des parties de jeu et les mises à jour en temps réel de la vue de jeu (mise à jour de l'affichage d'une partie lorsqu'elle a été créé ou supprimée). Le protocole Web Socket est le mieux adapté à la tâche de la mise à jour des données en temps réel comme le client et le serveur qui peuvent s'envoyer des données l'un vers l'autre de manière bidirectionnelle. De plus, le protocole Web Socket permet une connexion constante entre le client et le serveur, ce qui permet une mise à jour dynamique de la vue.
Durant les parties de jeu, le client et le serveur doivent envoyer un message vers l'autre indépendamment de l'autre partie (la vérification d'un clic dans une partie, l'initialisation de la minuterie durant les parties, l'envoi d'un message de partie global, la reprise vidéo d'une partie). Le protocole http ne serait pas approprié pour cette tâche puisque seul le client peut initier une requête, alors dans le protocole Web Socket, le client et le serveur peuvent initier un échange d'informations après la première connexion du client bien entendu.
De plus, le protocole Web Socket permet de créer des parties multijoueur, une fonctionnalité qui nous est utile pour les parties classique à 4, et les nouveaux modes que nous souhaitons implémenter dans nos exigences souhaitables.

Le protocole HTTP est utilisé pour les fonctionnalités spécifiques suivantes :

- La création d'un jeu
- La suppression d'un jeu
- Le chargement des fiches de jeu dans la vue de configuration et dans la vue de sélection
- Le chargement des images dans une partie
- Le chargement de l'historiques des parties jouées
- Le chargement des scores des utilisateurs
- L'envoie des meilleurs scores pour pouvoir attribuer des niveaux aux joueurs
- La réinitialisation des meilleurs scores et de l'historique des parties jouées
- La modification et réinitialisation des constantes de jeu
- La persistance de la reprise vidéo
- Les statistiques des joueurs
- L'évaluation d'une partie.
- La modification de l'avatar
- L'historique des canaux de messagerie

- La création d'un compte utilisateur
- L'ajout de sons personnalisables dans l'application
- Les défis et les badges qu'un utilisateur peut obtenir
- Le classement des joueurs selon leur elo

Dans le projet, le protocole HTTP est utilisé pour interagir avec la base de données pour obtenir, modifier ou supprimer les ressources d'une base de données. Avec les fonctionnalités mentionnées ci-dessus, le protocole HTTP n'est pas utilisé pour avoir des données en temps réel. Les fonctionnalités qui utilisent le protocole http ne demandent pas de maintenir une connexion constante avec le serveur. Puisque les fonctionnalités ci-dessus ne demandent pas de mettre à jour constamment la vue et demande plutôt le traitement de scénario et d'erreur comme des demandes pour des ressources qui n'existent plus, le protocole http est approprié.

3. Description des paquets

Dans cette section, nous allons dans un premier temps décrire les différentes requêtes ainsi que les interfaces utilisées pour chacune de ces requêtes. Nous allons ensuite aborder les web sockets utilisés. Finalement, nous allons donner une description de chacune des interfaces qu'on retrouve dans l'application.

1- Protocole HTTP

Route	**Requête**	**Corps de la requête**	**Description**	**Corps de la réponse**	**Code**	**Requiert jeton d'auth**
/api/chat	GET/	undefined	Dans cette requête, une liste de cette interface est renvoyée correspondant à l'information de chacun des canaux de clavardage présentement créés dont l'utilisateur fait partie.	200: *ChatInfo*[] 500: undefined	200 OK 500 INTERNAL SERVER ERROR	Oui
	DELETE/	undefined	Supprimer tous les canaux de clavardage créés par l'utilisateur.	204: undefined 500: undefined	Code 204 NO CONTENT Code 500 INTERNAL SERVER ERROR	Oui
	GET/:ChatId	*undefined* Dans cette requête, l'utilisateur reçoit les informations du canal correspondant au chatId initialement envoyé.		200: ChatInfo 500: undefined	Code 200 OK Code 404 NOT FOUND. Code 500 INTERNAL SERVER ERROR	Oui
	DELETE/:chat Id	ChatId: string	C'est un chatID qui est envoyé pour spécifier au serveur quel canal de clavardage supprimer.	204: undefined 500: undefined	Code 204 NO CONTENT Code 500 INTERNAL SERVER ERROR	Oui

	POST/	*ChatCreationInfo*	Cette requête sert à envoyer les informations d'un nouveau canal de clavardage pour sa création.	201: undefined 400: undefined 500: undefined	Code 201 CREATED Code 400 BAD REQUEST Code 500 INTERNAL SERVER ERROR.	Oui
/api/user	POST/	*UserInfo*	Cette requête sert à envoyer les informations de création de compte.		Code 201 CREATED Code 400 BAD REQUEST Code 500 INTERNAL SERVER ERROR	Oui
	GET/:username		Dans cette requête, cette interface est renvoyée et correspond à l'information de l'utilisateur correspondant à son pseudonyme initialement envoyé.	200: *UserInfo* 500: undefined	Code 200 OK Code 404 NOT FOUND Code 500 INTERNAL SERVER ERROR.	Oui
	PUT/avatar/:username	Avatar:uri	Dans cette requête, l'utilisateur envoie une image qui sera son nouvel avatar.	200: undefined 500: undefined	Code 200 OK Code 404 NOT FOUND Code 500 INTERNAL SERVER ERROR.	Oui

	PUT/gameStat/:username	*GameStat:* nbDiffFound:number GameTime:number	Dans cette requête, cette interface est renvoyée et correspond aux statistiques de Jeu de l'utilisateur correspondant à son pseudonyme initialement envoyé.	200: undefined 500: undefined	Code 200 OK Code 404 NOT FOUND Code 500 INTERNAL SERVER ERROR.	Oui
	PUT/gameSong/:username	*NewSong:Binary Data*	Dans cette requête, cette interface est renvoyée et correspond à des sons personnalisés.	200: undefined 404: undefined 500: undefined	Code 200 OK Code 404 NOT FOUND Code 500 INTERNAL SERVER ERROR	Oui
	PUT/Config/:username	*Config*	Dans cette requête, cette interface est renvoyée et correspond aux configurations de l'utilisateur.	200: undefined 500: undefined	Code 200 OK . Code 404 NOT FOUND Code 500 INTERNAL SERVER ERROR.	Oui
/api/games	GET/		Dans cette requête, une liste de cette interface est renvoyée correspondant à l'information de chacune des parties qui est sur	200:*GameInfo[]*	200 OK 500 INTERNAL SERVER ERROR.	Non

			le serveur.			
	DELETE/		Supprimer toutes les parties.	204 : undefined 500: undefined	Code 204 NO CONTENT Code 500 INTERNAL SERVER ERROR	Oui
	GET/:gameId		Dans cette requête, cette interface est renvoyée et correspond à l'information de la partie correspondant au gameId initialement envoyé.	200 : *GameInfo* 500: undefined	Code 200 OK Code 404 NOT FOUND Code 500 INTERNAL SERVER ERROR	Non
	DELETE/ :gameId		C'est un gameId qui est envoyé pour spécifier au serveur quelle partie supprimer.	204 : undefined 500: undefined	Code 204 NO CONTENT Code 500 INTERNAL SERVER ERROR	Oui
	POST/	*GameCreationInfo*	Cette requête sert à envoyer les informations d'une nouvelle partie lors de sa création.	201: undefined 400: undefined 500: undefined	Code 201 CREATED : Code 400 BAD REQUEST Code 500 INTERNAL SERVER ERROR.	Non

/api/images	POST/diff	*DifferenceImage*	Cette interface est envoyée lors de la création d'une nouvelle partie, elle contient certaines informations sur la nouvelle partie créée ainsi que la nouvelle image des différences.	200: undefined 400: undefined 500: undefined	Code 200 OK Code 400 BAD REQUEST Code 500 INTERNAL SERVER ERROR	Non
	GET/:gameId		Cette interface est renvoyée en réponse à cette requête. Elle contient les deux images de gauche et de droite, qui permettent de jouer à la partie.	200: *ImageSet* 404: undefined 500: undefined	Code 200 OK Code 404 NOT FOUND Code 500 INTERNAL SERVER ERROR	Non
	GET/diff/:gameId		Un gameId est envoyé et l'image des différences est renvoyée en dataURL.	200: DiffImageUri:string 500: undefined	Code 200 OK Code 404 NOT FOUND Code 500 INTERNAL SERVER ERROR.	Non

	GET/miniature/:gameId		Une gameId est envoyé pour retrouver l'image miniature de la partie correspondante. Les miniatures servent à afficher les images sur le carrousel dans la vue de configuration et de sélection.	200: MiniatureUri:string 500: undefined	Code 200 OK : L'image miniature correspondante est renvoyée. Code 404 NOT FOUND : Si la miniature avec le gameId envoyé n'est pas trouvée, alors le serveur renvoie au client ce code de retour. Code 500 INTERNAL SERVER ERROR.	Non

2- Protocole WebSocket:

Événement	**Source**	**Description**	**Données envoyées**	**Événements potentiellement déclenchés**
sendMessage	Client	Envoi d'un message dans le canal de clavardage global.	*ChatMessage*	*chatMessage*
chatMessage	Server	Lorsqu'un des joueurs écrit un message, cet événement est lancé avec en paramètre le message écrit. Envoyé en broadcast.	ChatMessage	–
sendStartClassic	Client	Cela envoie le signal de création d'une partie Classique pour attendre les autres joueurs.	–	
sendAllDiffFound	Client	Cela envoie le signal de la fin de la partie classique lorsqu'un joueur a trouvé toutes les différences	EndGameInfo	endGame
endGame	Server	Cet événement est lancé lorsqu'un des joueurs termine la partie. Envoyé à tous les utilisateurs dans la même room que l'émetteur.	victoryMessage:string	stopClicks
sendStartLimited	Client	Cela envoie le signal de création d'une partie temps limité pour attendre les autres joueurs.	–	??
timer	Serveur	C'est envoyé depuis le serveur. Le temps du serveur est passé en paramètre pour mettre à jour le temps du côté du client.	TimerInfo	–

errorClick	Client	Événement envoyé lorsqu'un client clique sur un canvas de la vue du jeu sans trouver de différences. C'est alors une erreur. Un objet avec des informations sur le clic (la position et le canvas sur lequel il a eu lieu) est passé en paramètre.	ClickValidation	mouseClick
mouseClick	Serveur	Lorsqu'un des joueurs clique dans la partie, cet événement est lancé avec un objet *JSON* en paramètre qui contient l'information du clic et le joueur qui l'a fait. Envoyé à tous les utilisateurs dans la même room que l'émetteur.	ClickValidation	–
differenceFoundClick	Client	Événement envoyé lorsqu'un clic a trouvé une différence. Un vecteur (interface Vec2) de la position de la différence est passé en paramètre.	ClickValidation	mouseClick, diffFoundCount
stopClicks	Serveur	Événement envoyé pour désactiver les cliques lorsque la partie doit se terminer (par exemple quand toutes les différences sont trouvées en mode classique).	–	–
diffFoundCount	Serveur	Événement envoyé lorsqu'une différence a été trouvée. Le nouveau compte de différences du joueur est passé en paramètre. Événement envoyé à tous ceux dans la room de l'émetteur, sauf l'émetteur.	totalDiffFound:int	–

nextGameId	Client	Événement envoyé pour donner la prochaine fiche qui devra être affichée dans le mode temps limité.	–	nextGameId
nextGameId	Serveur	Événement envoyé pour donner la prochaine fiche a afficher dans le mode temps limité Envoyé a tous ceux dans la room de l'émetteur.	GameCreationInfo	–
leftGame	Client	Événement pour annoncer l'abandon d'une partie	–	teammateLeft
teammateLeft	Serveur	Événement pour annoncer l'abandon d'une partie d'un coéquipier dans une partie. Envoyé a tous ceux dans la même room de l'émetteur, sauf l'émetteur.	leftPlayerName:string	–
creationInProgress	Client	Évènement envoyé avec un *gameId* en paramètre lorsqu'un joueur crée une partie afin de modifier l'affichage du bouton de la fiche correspondante pour 'Joindre' dans la vue de sélection.	gameId:string	creationInProgress
creationInProgress	Serveur	Évènement envoyé avec un *gameId* en paramètre lorsqu'un joueur crée une partie afin de modifier l'affichage du bouton de la fiche correspondante pour 'Joindre' dans la vue de sélection. Envoyé en broadcast.	gameId:string	–

creationFinished	Client	Évènement envoyé avec un *gameId* en paramètre lorsque la création d'une partie est terminée afin de modifier l'affichage du bouton de la fiche correspondante pour 'Créer' dans la vue de sélection.	gameId:string	creationFinished
creationFinished	Serveur	Évènement envoyé avec un *gameId* en paramètre lorsque la création d'une partie est terminée afin de modifier l'affichage du bouton de la fiche correspondante pour 'Créer' dans la vue de sélection. Envoyé en broadcast.	gameId:string	–
Accepted	Client	Événement envoyé lorsque le créateur de la partie a accepté son adversaire qui était en train d'attendre.	opponentId:string	createMultiGame, joinMultiGame
Declined	Client	Événement envoyé lorsque le créateur de la partie a refusé son adversaire qui était en train d'attendre.	opponentId:string	declineOpponent
closeWaitingDialog	Client	Événement envoyé pour que le joueur sorte de l'état d'attente.	gameId:string	stopWaiting
stopWaiting	Serveur	Événement envoyé pour enlever un joueur en attente de la liste des joueurs en attente.	gameId:string	–

gameDeleted	Server	Événement envoyé lorsqu'une partie a été supprimée.		
gameCreated	Server	Événement envoyé lorsqu'une partie a été créée.	gameId:string	
observer	Client	Événement envoyé lorsqu'un observeur s'ajoute à la partie	GameId:string	sendGameInfo
sendGameInfo	Server	Événement envoyé avec les infos pour l'observateur.	*GameInfo*	
sendNewPassword	Client	Événement envoyé pour réinitialiser le mot de passe.	Email:string Newpassword:string	sendNewPasswordAcce pted
sendNewPasswordAccepted	Server	Événement envoyé lorsque le serveur accepte la reinitialisation de mot de passe		
sendRating	Client	Événement envoyé pour modifier la note d'un jeu précis.	gameId:string, newRating: number	sendNewRating
sendNewRating	Server	Événement envoyé pour la nouvelle note d'une fiche de jeu..	newRating:number	–
startMultiGame	Server	Événement envoyé avec un objet contenant l'id de la partie que le joueur rejont.	gameId, gameType	--
createMultiGame	Server	Événement envoyé avec un objet contenant les informations de l'utilisateur qui commence la	gameId, gameType	closeWaitingDialog()

		partie multijoueur.		
joinMultiGame	Server	Événement envoyé avec un objet contenant les informations de l'utilisateur qui tente de rejoindre une partie multijoueur.	–	–-
declineOpponent	Server	Si le créateur de la partie refuse son opposant, alors cet événement est lancé. Événement lancé à la personne déclinée.	–	–
startMultiGame	Server	Événement envoyé avec un objet contenant les informations de l'utilisateur qui tente de rejoindre une partie multijoueur.	–	–
disconnect	Server	Événement envoyé lorsqu'un des joueurs se déconnecte.		
sendFriendRequest	Server	Événement envoyé lorsque le serveur reçoit la demande d'ami à un utilisateur.	usernameRequested:string	decline(), accept()

3 - Description des Interfaces

Interface temporaires

Interface	**Corps**	**Description**
GameCreationInfo	gameTitle: string; diffCount: number difficulty: string leftImageUrl: string rghtImageUrl: string diffImageUrl: string	Cette interface contient le titre d'un jeu, son nombre de différence, sa difficulté, l'url pour l'image de gauche, l'url pour l'image de droite et l'url pour l'image droite. Elle est utilisée lors de la création d'une partie dans la vue création. Cette interface sera ensuite envoyée du client vers le serveur et sera enregistrée dans la base de données en incluant des meilleurs temps fictifs et un booléen qui indique si une partie est en cours de création.
ReplayEvent	time: number; type: ClientEvent; arg: any;	Cette interface contient le temps d'un évènement, son type et ses paramètres. Elle permet d'enregistrer les évènements lors d'une partie pour ensuite faire une reprise d'une partie.
ChatCreationInfo	ChatName: String Creator:String Members:String[] IsGameChat:Boolean	Cette interface contient les informations nécessaires pour la modification d'un chat.
GameStats	nbDiffFound:number GameTime:number username: string	Cette information contient les informations nécessaires pour mettre à jour les statistiques du joueur.
Config	Language:String Visual:String username: string	

Interface Base de données

Interface	Corps	Description
ChatInfo	ChatId: String ChatName:String Creator : String Members : String[] HistoryChat : ChatMessage[] IsGameChat: Boolean	Cette interface permet de contenir toutes les informations d'un canal de clavardage précis avec l'id du créateur, les membres du canal et le type de canal (canal d'une partie ou pas).
ChatMessage	userId: string message: string timestamp: string	Cette interface permet de contenir toutes les informations pour afficher les messages d'un chat.
UserInfo	UserInfo: Avatar:uri username:Stirng email:String nbGamePlayed:number nbGamesWon:number AvgDiffFound:number AvgGameTime:number HitsoryActions: Action[] Language: String Visual:String Songs: string[]	Cette interface permet de contenir toutes les informations de l'utilisateur avec ses statistiques de jeu et les configurations de jeu qu'il a choisies.
GameHistory	date: string; startinTime: string; duration: number; gameMode: string; endGameSituation: EndGameSituation; firstRankedPlayer: string secondRankedPlayer: string | undefined thirdRankedPlayer: string | undefined fourthRankedPlayer: string | undefined winningTeam: string[] | undefined losingTeam: string[] | undefined	Cette interface permet de contenir l'historique d'une partie jouée avec la date, l'heure du début de la partie, le mode de jeu, si un joueur a quitté la partie et le nom des autres joueurs s' ils existent. En fonction du mode de jeu, la liste des joueurs de chaque équipe est définie aussi. L'interface est utilisée pour configurer les statistiques des joueurs et définir leur raking pour le mode elo
Score	value: number; playerName:string;	Cette interface permet de contenir le score d'un joueur avec son nom. Il permettra de mettre à jour les statistiques des joueurs.
GameInfo	name: string; diffCount: number;	Cette interface sert

	difficulty: string; bestScoresSolo: Score[]; bestScores1v1: Score[]; creationInProgress: boolean;	

Interface de la communication entre sockets:

Interface	**Corps**	**Description**
ClickValidation	side: string; position: Vec2	Cette interface permet de contenir la position et le côté (gauche ou droite) sur lequel un clic d'une souris a été effectuée. Cette interface est utilisée pour traiter le clic d'un joueur sur un canevas lors d'une partie.
TimerInfo	numeric: number; stringValue: string;	Cette interface permet de contenir le temps en secondes et en string de la minuterie.
PlayingInfo	gameID: string; username:string; otherUsernames: string[]; diffCount: number; diffFoundCount: number; otherDiffFoundCount: number;	Cette interface permet de contenir les informations sur le progrès d'une partie en multijoueur avec l'id de la partie, le nom du premier joueur, le nom du deuxième joueur, le nombre d'indices restants, le nombre de différence, le nombre de différences trouvées et le nombre de différences trouvées par l'adversaire.
UserInfo	username: string; diffFoundCount: number	Cette interface permet de contenir l'information d'un joueur sur son nom et son nombre de différences trouvées.
GameStatus	gameId: string; playerHadQuit: boolean; diiCount: number;	Cette interface permet de contenir l'information sur l'état d'une partie en solo.

		Elle permet de connaître l'id de la partie, si le joueur a abandonné la partie, le nombre d'indice restant et le nombre de différence.
EndGameInfo	message: string; notifyOpponent: boolean;	Cette interface permet de contenir le message de fin de partie et un booléen pour savoir si le serveur doit notifier l'adversaire. Elle est utilisée quand un joueur gagne une partie et que l'adversaire doit être notifié ou quand un joueur abandonne une partie et que son adversaire doit être notifié qu'il a gagné par défaut.
Message	content: string; emitter: string; chatId: string;	Cette interface permet de contenir un message dans le « chat » et de contenir celui qui a émis le message (le serveur, le premier joueur et le deuxième joueur). Elle est utilisée pour envoyer des messages dans le clavardage.
PopUpMessage	content: string; leftRouterLink: string; rightRouterLink: string; leftButtonText: string; rightButtonText: string	Cette interface est utilisée pour afficher un pop-up. Elle contient le message, le lien du routeur pour le bouton de gauche, le lien du routeur pour le bouton de droite, le texte du bouton de gauche et le texte du bouton de droite.

Interfaces des Images

Interface	**Corps**	**Description**
DifferenceImage	diffCount: number; difficulty: string; uri: string;	Cette interface contient le nombre de différences dans une image, son niveau de difficulté et son lien uri. Cette interface est utilisée lors de la création d'une partie dans la page de création lorsqu'il est nécessaire d'afficher l'image de différence.

ImageSet	gameId: string; leftUri: string; rightUri: string;	Cette interface contient l'id d'une partie, le lien pour l'image de gauche et le lien pour l'image de droite. Elle est utilisée pour l'affichage des images de gauche et de droite lors d'une partie de jeu.

Interfaces des Différences

Interface	**Corps**	**Description**
Vec2	x: number; y: number	Cette interface contient les coordonnées d'un pixel.
Direction	x: number; y:number: name: string;	Cette interface contient les coordonnées et la direction d'un pixel. Elle est utilisée lors de la détection d'une différence.
Directions	north: Direction; south: Direction; west: Direction; east: Direction;	Cette interface contient les directions autour d'un pixel. Elle est utilisée pour la visite de pixels adjacents lors de la détection d'une différence.
Difference	start: Vec2	Cette interface permet de contenir les différences trouvées lors de la détection des différences d'une image avec leurs coordonnées.
Images	left: Jimp; right: Jimp; Diff: Jimp	Cette interface permet de contenir l'image de gauche, l'image de droite et l'image de différence lors de l'algorithme de détection de différence.
PixelVisitInfo	position: Vec2; direction: Direction; diffPosition: Vec2;	Cette interface permet de contenir les informations des pixels visités lors de l'algorithme de détection de différence avec la position de la différence, sa direction et la

		position de la différence.

Interfaces des informations sur les sockets

Interface	**Corps**	**Description**
Opponent	username: string; socket: Socket;	Cette interface contient le socket et le nom de l'adversaire. Elle est utilisée pour obtenir des informations sur l'adversaire en multijoueur.
MultiGameCreation	owner: string; opponent: Opponent[]	Cette interface contient le nom du créateur d'une partie ainsi que la liste des adversaires. Elle est utilisée lorsqu'il y a plusieurs joueurs dans la file d'une même partie.
SocketRoomInfo	id: string; waiting: boolean;	Cette interface contient l'id d'une salle créée par un socket avec l'information si la salle est en attente d'autre joueur.
SocketUserInfo	info: UserInfo; socket: Socket;	Cette interface contient l'information d'un joueur et son socket. Elle est utilisée pour identifier et interagir avec un joueur, que cela soit en solo ou en multijoueur.