Équipe 102

Fais-moi un dessin
Protocole de communication

Version 2.0

Historique des révisions

Date	Version	Description	Auteur
2020-09-10	1.0	Version initiale	Rostyslav Myhovych
2020-09-30	1.1	Révision en équipe de la description des différents paquets	Équipe 102
2020-10-01	1.2	Mise en forme et correction du français	Vincent l'Ecuyer-Simard
2020-10-08	1.3	Modification du message du serveur de l'événement Socket IO *channel_edit*	Rostyslav Myhovych
2020-10-20	1.4	Ajout d'un appel REST pour avoir la liste de tous les avatars	Rostyslav Myhovych
2020-11-29	2.0	Révision pour la remise du livrable 2	Xi Chen Shen Vincent L'Ecuyer-Simard

Table des matières

1 Introduction 5

2 Communication client-serveur 5

3 Description des paquets 5

3.1 Paquets REST 6

3.1.1 Création du profil d'utilisateur 6

3.1.2 Authentification et création de sessions d'utilisation 7

3.1.3 Déconnexion de la session d'utilisation 7

3.1.4 Changement de thème d'utilisateur 8

3.1.5 Demande d'information du profil d'un utilisateur 8

3.1.6 Création d'une paire mot-image 10

3.1.7 Faire une recherche d'image en ligne 11

3.1.8 Obtenir une image spécifique 11

3.1.9 Demander l'historique des messages complet d'un canal de discussion 12

3.1.10 Demander la liste de tous les canaux de discussions disponibles pour l'utilisateur 13

3.1.11 Demander la liste de tous les groupes de joueurs 13

3.1.12 Demander la liste de tous les avatars 14

3.2 Paquets Socket IO 15

3.2.1 Création d'un nouveau canal de discussion 15

3.2.2 Ajout d'utilisateur dans un canal de discussion existant 16

3.2.3 Retranchement d'utilisateur d'un canal de discussion existant 16

3.2.4 Suppression du canal de discussion 17

3.2.5 Nouveau message 17

3.2.6 Nouveau groupe de joueurs 18

3.2.7 Ajout d'un membre du groupe de joueurs 18

3.2.8 Retrait d'un membre du groupe de joueurs 19

3.2.9 Suppression du groupe de joueurs 19

3.2.10 Commencer une partie 19

3.2.11 Terminer une partie 20

3.2.12 Signaler l'information sur le tour de jeu qui suit 20

3.2.13 Envoyer le mot choisi par le dessinateur 21

3.2.14 Signaler le début du tour 21

3.2.15 Signaler la fin du tour 21

3.2.16 Réussir à deviner le mot 22

3.2.17 Échouer une tentative de deviner le mot 22

3.2.18 Nouveau score pour un joueur 22
3.2.19 Diminuer le nombre essaie d'une unité 23
3.2.20 Dévoiler un nouvel indice 23
3.2.21 Modifier la couleur de l'arrière-plan du dessin 23
3.2.22 Ajouter un nouvel élément au dessin 24
3.2.23 Ajouter des coordonnées à un élément du dessin 24
3.2.24 Supprimer un élément du dessin 25
3.2.25 Annuler la suppression de l'élément 25

Protocole de communication

1 Introduction

Ce protocole de communication vise à détailler le processus de communication entre le client lourd, le client léger et le serveur. D'abord, dans la section 2, le choix des technologies utilisées dans la communication client-serveur sera justifié. Ensuite, dans la section 3, le contenu des différents types de paquets utilisés au sein de ce protocole de communication sera présenté.

2 Communication client-serveur

Dans le cadre de notre logiciel, la communication réseau se fera uniquement entre un client et un serveur. Il n'y aura donc pas de communications directes entre deux clients ni entre un client et la base de données. Les deux technologies qui seront utilisées dans nos communications client-serveur seront les requêtes REST API et Socket IO.

L'utilisation des requêtes REST API est justifiée par leur simplicité d'utilisation et leur popularité. C'est un standard de communication qui est connu et employé dans la vaste majorité des communications client-serveur dans le monde. Il est donc plus simple de l'implémenter, car beaucoup de librairies et de ressources d'aide sont disponibles. De plus, sa nature *stateless* nous permet d'échanger l'information de façon prévisible, ce qui réduit énormément la complexité des échanges. Cette technologie sera utilisée dans les requêtes simples du client vers le serveur. Plus précisément, elle sera utilisée dans les communications entourant la gestion d'un profil utilisateur (création d'un profil, connexion, déconnexion, consultation des informations de profil, etc.) et lors de la création d'une paire mot-image. De plus, les requêtes REST API seront utilisées lorsqu'un client souhaite acquérir l'état actuel d'une situation (liste des groupes de joueurs existants, liste des canaux de discussion existants, historique des messages d'un canal de discussion) qui sera par la suite mise à jour à l'aide de la deuxième technologie, soit Socket IO.

En effet, l'utilisation du Socket IO est justifiée par notre besoin de communication bidirectionnelle en temps réel. Comme notre application doit permettre les échanges d'information entre de nombreux clients, le serveur doit être capable d'envoyer l'information à ces clients sans une requête explicite de leur part. Dans ces circonstances, fonctionner par requêtes REST API ajoute beaucoup de complexité puisqu'un client devrait explicitement et continuellement demander les mises à jour du serveur, ce qui ajoute ce poids à la communication. Cependant, Socket IO n'établit la chaîne de communication qu'une seule fois, c'est-à-dire au début. Le serveur peut donc envoyer par la suite ses paquets continuellement et de manière très simple aux clients. Cette technologie sera utilisée pour tous les événements se produisant en cours de partie (début de partie, nouveau tour, dessin deviné, etc.), tous les événements en lien avec le clavardage (nouveau message, nouveau canal de discussion, etc.) ainsi que les événements de mise à jour au niveau des groupes de joueurs.

Afin de maintenir une structure cohérente du traitement et stockage des données, le format JSON sera utilisé dans tous nos échanges. Que ce soit pour représenter les informations d'un utilisateur ou les messages dans un groupe de clavardage, le JSON nous permettra de bien structurer l'information. Avoir un seul format pour l'échange de l'information simplifiera beaucoup nos services de gestion de l'information. De plus, tout langage de programmation a des façons simples de gérer le JSON. Comme nous travaillerons avec les langages orientés-objet, le JSON est la façon optimale de transformer tout objet en information pouvant être envoyée à travers le réseau, peu importe le langage de source ou de destination.

3 Description des paquets

Toute la communication réseau client-serveur se fera sous le format JSON. Les paquets de la technologie REST API sont définis par un *endpoint*, la route à la fin de l'URL vers le serveur, ainsi qu'une méthode, telle que GET, POST et DELETE. Les paquets de la technologie Socket IO sont définis uniquement par leur nom d'événement.

L'emplacement de notre serveur est l'URL ci-dessous :

SERVER_URL = "https://p3-server.herokuapp.com"

3.1 Paquets REST

Les paquets qui seront envoyés, ainsi que leurs réponses à travers la technologie REST API sont mentionnés ci-dessous. Les requêtes sont caractérisées par leur méthode (POST, GET, PUT, DELETE) et la route du serveur. Cette route peut être paramétrée pour certains types de requêtes. Dans les cas où la connexion ou la déconnexion sont concernées, les messages contiennent également des paramètres d'en-tête supplémentaires pour y spécifier les identifiants de connexion. Tous les corps des messages sont du type JSON, que ce soit pour les requêtes ou les réponses. Les réponses sont accompagnées par un statut indiquant le succès ou l'échec de la requête, avec un corps de messages dans certaines situations.

3.1.1 Création du profil d'utilisateur

POST : SERVER_URL/profile

Query Params		
Headers	-	

Data	{
	"username": str,
	"password": str,
	"firstName": str,
	"lastName": str,
	"avatarName": string
	}
		
		Nom d'utilisateur
		Mot de passe
		Prénom
		Nom de famille
		Identifiant de l'avatar sélectionné

Réponses :

Status 200 : Succès de création du compte

Data	-	

Status 400 : Nom d'utilisateur déjà pris

Data	-	

3.1.2 Authentification et création de sessions d'utilisation

GET: SERVER_URL/session

Query Params	-	
Headers	"username": str "password": str	Nom d'utilisateur Mot de passe
Data	-	

Réponses :

Status 200 : Succès d'authentification

Data	{
	"userId": int
	}
		Identifiant de l'utilisateur connecté

Status 401 : Nom d'utilisateur / mot de passe incorrect

Data	-	

Status 400 : Utilisateur déjà connecté

Data	-	

3.1.3 Déconnexion de la session d'utilisation

DELETE: SERVER_URL/session

Query Params	-	
Headers	"username": str	Identifiant de l'utilisateur
Data	-	

Réponses :

Status 200 : Succès de déconnexion

Data	-	

Status 404 : Identifiant de l'utilisateur n'existe pas

Data	-	

3.1.4 Changement de thème d'utilisateur

PUT: SERVER_URL/profile/theme

Query Params	-	
Headers	-	

Data	{
	"username":str
	"theme": int
	}
		
		Identifiant du profil
		Identifiant du thème

Réponses :

Status 200 : Succès de changement de thème

Data	-	

Status 404 : theme n'est pas valid

Data	-	

3.1.5 Demande d'information du profil d'un utilisateur

GET: SERVER_URL/profile/{username}

Query Params	"username": string	Identifiant du profil
Headers	-	
Data	-	

Réponses :

Status 200 : Succès de demande d'informations du profil

Data	{
	"username": str,
	"firstName": str,
	"lastName": str,
	"avatarName": string
	"theme": int,
	}
		
		Nom d'utilisateur
		Prénom
		Nom de famille
		Identifiant de l'avatar sélectionné
		Identifiant du thème

Status 404 : username n'existe pas

Data	-	

GET: SERVER_URL/profile/{username}/stats-history

Query Params	"username": string	Identifiant du profil
Headers	-	
Data	-	

Réponses :

Status 200 : Succès de demande d'informations du profil

Data	{	
	"username": str,	Nom d'utilisateur
	"history": {	Historique détaillé
	"sessions": [	Historique des ouvertures / fermetures des sessions
	{	
	"type": str,	Type (ouverture / fermeture)
	"timestamp": long	Date d'événement
	},	
	…	
	],	
	"games": [	Historique des parties jouées
	{	
	"gameMode": int,	Mode de jeu
	"scores": Map<str, float>	Liste des utilisateurs et leur pointage
	"startTime": long	Heure du début
	"endTime": long	Heure de la fin
	},	
	...	
	]	
	},	
	"statistics": {	Statistique des parties
	"totalGamePlayed": int,	Nombre de parties jouées
	"totalFFAPlayed": int,	Nombre de mêlées générales jouées
	"totalSprintPlayed": int,	Nombre de parties sprints joués
	"totalFFAWin": int,	Nombre de victoires en mêlée générale
	"FFAWinRatio": float,	Pourcentage de victoires en mêlée générale
	"totalGameTime": int,	Temps de jeu cumulatif en secondes
	"timePerGame": float,	Temps moyen par partie en secondes
	"bestSoloScore": int,	Meilleur pointage dans une partie en mode sprint solo
	}	
	}	

Status 404 : username n'existe pas

Data	-	

3.1.6 Création d'une paire mot-image

POST: SERVER_URL/pair

Query Params	-	
Headers	-	

Data	{
	"word": str,
	"hints": list[str],
	"difficulty": int,
	"delay": int,
	
	"isRandom": bool,
	"drawing": {
	
	"background": str,
	"backgroundOpacity": float
	"elements": [
	{
	"id":int,
	"color": str,
	"opacity": float,
	"strokeWidth": int
	},
	…
	]
	"coordinates": [
	{
	"id": int,
	"parts": list[str]
	},
	…
	]
	
	}
	}
		
		Mot de la paire mot-image
		Liste d'indices
		Niveau de difficulté
		Délai d'attente avant de dessiner entre chaque élément (ms)
		Si c'est un mode de dessin aléatoire
		Contenu du dessin
		
		Couleur de l'arrière-plan
		L'opacité de l'arrière-plan
		Liste d'éléments (traits de crayon)
		
		Identificateur unique du trait
		Couleur de l'élément
		Opacité du trait
		Épaisseur du trait l'élément
		
		Coordonnées à ajouter aux traits
		
		Identificateur unique du trait
		Liste des morceaux du type *path SVG* ajoutés au trait

Réponses :

Status 200 : Succès de création de la paire mot-image

Data	-	

Status 400 : Contenu du message invalide

Data	Error Message	Message spécifiant l'élément du JSON invalide

3.1.7 Faire une recherche d'image en ligne

GET: SERVER_URL/pair/online-search/{query}/{page}

Query Params	"query": string "page": int	Mot ou expression à rechercher Page de résultat demandée
Headers		
Data	-	

Réponses :

Status 200 : Résultats trouvés pour les critères spécifiés

Data	{
	"imageLinks": list[str]
	}
		Liste des URL vers les images

Status 502 : L'API de recherche tiers a retourné une erreur ou aucun résultat

Data	Error Message	Message spécifiant la raison de l'erreur

3.1.8 Obtenir une image spécifique

GET: SERVER_URL/pair/online-image

Query Params	-	
Headers	"path": str	URL de l'image désirée
Data	-	

Réponses :

Status 200 : Image valide trouvée

Data	{
	"base64Image": str
	}
		Image désirée en format base64

Status 502 : L'URL ne menait pas vers une image valide

Data	Error Message	Message spécifiant la raison de l'erreur

3.1.9 Demander l'historique des messages complet d'un canal de discussion

GET: SERVER_URL/chat/{channelName}

Query Params	"channelName": str	Nom du canal de discussion
Headers	-	
Data	-	

Réponses :

Status 200 : Succès de demande

Data	{
	"messages": [
	{
	"channelName": str,
	"username": str,
	"content": str,
	"timestamp": long
	},
	...
	]
	}
		Liste de tous messages chronologiques du canal de discussion
		Nom du canal de discussion
		Nom de l'utilisateur
		Contenu du message
		Temps de l'envoi du message

Status 404 : Identifiant de canal de discussion inexistant

Data	-	

3.1.10 Demander la liste de tous les canaux de discussions disponibles pour l'utilisateur

GET: SERVER_URL/channels

Query Params	-	
Headers	-	
Data	-	

Réponses :

Status 200 : Succès de demande

Data	{
	"channels": [
	{
	"isGameChannel": bool,
	"channelName": str,
	"users": list[str]
	},
	…
	]
	}
		Liste de tous les canaux de discussions
		Est un canal de discussion de jeu
		Nom du canal de discussion
		Liste des identifiants d'utilisateurs faisant partie du canal de discussion

3.1.11 Demander la liste de tous les groupes de joueurs

GET: SERVER_URL/gamerooms

Query Params	-	
Headers	-	
Data	-	

Réponses :

Status 200 : Succès de demande

Data	{
	"gamerooms": [
	{
	"gameroomName": str,
	"gameMode" : int,
	"difficulty": int,
	"users": list[str],
	"isInGame": bool,
	},
	...
	]
	}
		Liste de tous les groupes de joueurs
		Identifiant du groupe de joueurs
		Mode de jeu du groupe
		Difficulté des parties du groupe
		Liste d'identifiants des utilisateurs dans le groupe de jeu
		État du groupe

3.1.12 Demander la liste de tous les avatars

GET: SERVER_URL/avatar/all

Query Params	-	
Headers	-	
Data	-	

Réponses :

Status 200 : Succès de demande

Data	{
	“avatars”: [
	{
	“name”: str,
	“svg” : str
	},
	...
	]
	}
		Liste de tous les avatars
		
		Nom de l’avatar
		Image SVG de l’avatar

3.2 Paquets Socket IO

Cette section présente les paquets envoyés du client au serveur, ainsi que du serveur au client avec la technologie Socket IO. Les messages envoyés avec cette technologie ne reçoivent aucune réponse, contrairement aux messages de la technologie REST API. Il n'existe qu'une seule connexion au serveur Socket IO, qui est maintenue tout le long de la session active. Les messages ne sont distingués que par le nom de leur événement, qui caractérise le type d'événement, et sont formés d'un objet de type JSON, contenant toute l'information sur l'événement. Le JSON est le contenu des messages du client et du serveur. Dans les rares cas, un message vide est envoyé du serveur pour signaler le début ou la fin d'un événement du jeu. Tous les messages Socket IO, ainsi que leur événement et leur contenu sont mentionnés ci-dessous.

URL de connexion au serveur : SERVER_URL

3.2.1 Création d'un nouveau canal de discussion

Appelé par un client lorsqu'il crée un nouveau canal de discussion. Le seul utilisateur dans la liste des utilisateurs est lui-même.

Appelé par le serveur et envoyé à tous utilisateurs si c'est un canal de discussion public (créé explicitement par un autre client) ou à tous les membres d'un groupe de joueurs lorsqu'ils rejoignent celui-ci.

Événement : *channel_new*

Data Client	{
	“channelName”: str,
	“users”: list[str]
	}
		
		Nom du canal de discussion
		Liste des identifiants d'utilisateurs faisant partie du canal

Data Server	{
	“isGameChannel”: bool,
	“channelName”: str,
	“users”: list[str]
	}
		
		Est un canal de discussion de jeu
		Nom du canal de discussion
		Liste des identifiants d'utilisateurs faisant partie du canal

3.2.2 Ajout d'utilisateur dans un canal de discussion existant

Appelé par un client lorsqu'il rejoint ou quitte le canal de discussion. Le type de modification spécifie si le client veut s'ajouter dans le groupe ou bien s'enlever.

Appelé par le serveur pour notifier tous les clients concernés du changement.

Événement : *channel_add_user*

Data Client	{
	"channelName": str,
	"username": str
	}
		Nom du canal de discussion
		Nom de l'utilisateur à ajouter

Data Server	{
	"channelName": str,
	"username": str
	}
		Nom du canal de discussion
		Nom de l'utilisateur à ajouter

3.2.3 Retranchement d'utilisateur d'un canal de discussion existant

Appelé par un client lorsqu'il quitte le canal de discussion.

Appelé par le serveur pour notifier tous les clients concernés du changement.

Événement : *channel_remove_user*

Data Client	{
	"channelName": str,
	"username": str
	}
		Nom du canal de discussion
		Nom de l'utilisateur à enlever

Data Server	{
	"channelName": str,
	"username": str
	}
		Nom du canal de discussion
		Nom de l'utilisateur à enlever

3.2.4 Suppression du canal de discussion

Appelé par le client lorsqu'il supprime un canal de discussion

Appelé par le serveur à tous les clients concernés lorsqu'un client a fait l'appel.

Événement : *channel_delete*

Data Client	{
	"channelName": str
	}
		Nom du canal de discussion

Data Server	{
	"channelName": str
	}
		Nom du canal de discussion

3.2.5 Nouveau message

Appelé par le client lorsqu'il crée un nouveau message

Appelé par le serveur à tous les clients faisant partie du canal de discussion en générant un *timestamp*.

Événement : *chat_message_new*

Data Client	{
	"channelName": str,
	"username": str,
	"content": str,
	}
		
		Nom du canal de discussion
		Nom de l'utilisateur
		Contenu du message

Data Server	{
	"channelName": str,
	"username": str,
	"content": str,
	"timestamp": long
	}
		
		Nom du canal de discussion
		Nom de l'utilisateur
		Contenu du message
		Temps de l'envoi du message

3.2.6 Nouveau groupe de joueurs

Appelé par un client lorsqu'il crée un nouveau groupe de joueurs. Le seul utilisateur dans la liste des utilisateurs est lui-même.

Appelé par le serveur et envoyé à tous utilisateurs.

Événement : ***gameroom_new***

Data Client	{
	"gameroomName": str,
	"gameMode" : int,
	"difficulty": int,
	"users": list[str]
	}
		
		Identifiant du groupe de joueurs
		Mode de jeu
		Difficulté du jeu
		Liste des identifiants d'utilisateur

Data Server	{
	"gameroomName": str,
	"gameMode" : int,
	"difficulty": int,
	"users": list[str],
	"isInGame": bool,
	}
		
		Identifiant du groupe de joueurs
		Mode de jeu
		Difficulté du jeu
		Liste des identifiants d'utilisateur
		État du gameroom

3.2.7 Ajout d'un membre du groupe de joueurs

Appelé par un client lorsqu'il rejoint le groupe de joueurs.

Appelé par le serveur pour notifier tous les clients du changement, pour mettre à jour la liste des utilisateurs du groupe.

Événement : ***gameroom_add_user***

Data Client	{
	"gameroomName": str,
	"username": str
	}
		
		Identifiant du groupe de joueurs
		Utilisateur à ajouter

Data Server	{
	"gameroomName": str,
	"username": str
	}
		
		Identifiant du groupe de joueurs
		Utilisateur à ajouter

3.2.8 Retrait d'un membre du groupe de joueurs

Appelé par un client lorsqu'il quitte le groupe de joueurs.

Appelé par le serveur pour notifier tous les clients du changement, pour mettre à jour la liste des utilisateurs du groupe.

Événement : *gameroom_remove_user*

Data Client	{
	"gameroomName": str,
	"username": str
	}
		Identifiant du groupe de joueurs
		Utilisateur à retirer

Data Server	{
	"gameroomName": str,
	"username": str
	}
		Identifiant du groupe de joueurs
		Utilisateur à retirer

3.2.9 Suppression du groupe de joueurs

Jamais appelé par le client

Appelé par le serveur à tous les clients lorsque la liste d'utilisateurs devient vide.

Événement : *gameroom_delete*

Data Client	-	

Data Server	{
	"gameroomName": str
	}
		Identifiant du groupe de joueurs

3.2.10 Commencer une partie

Appelé par le client pour signaler au serveur que le groupe de joueur désire commencer sa partie.

Appelé par le serveur pour notifier les membres du groupe de joueurs du début du jeu.

Événement : *game_start*

Data Client	{
	"gameroomName": str
	}
		Identifiant du groupe de joueur

Data Server	{
	"gameroomName": str
	}
		Identifiant du groupe de joueur

3.2.11 Terminer une partie

Jamais appelé par le client.

Appelé par le serveur pour notifier les membres du groupe de joueurs de la fin de la partie.

Événement : *game_end*

Data Client	-	

Data Server	{
	“gameroomName”: str
	}
		Identifiant du groupe de joueurs

3.2.12 Signaler l’information sur le tour de jeu qui suit

Jamais envoyé par le client.

Envoyé par le serveur à tous les clients du groupe de joueurs.

Événement : *turn_info*

Data Client	-	

Data Server	{
	“drawer”: string,
	
	“virtualDrawing”: {
	
	“word”: str,
	“hints”: list[str],
	“difficulty”: int,
	“delay”: int,
	“isRandom”: bool,
	“drawing”: {
	“background”: str,
	“backgroundOpacity”:float
	“elements”: [
	{
	“color”: str,
	“opacity”: float
	“strokeWidth”: float,
	“parts”: list[str]
	},
	…]
	}
	}
	}
		Identifiant du dessinateur
		
		Information du dessin virtuel, si le dessinateur est virtuel
		Mot de la paire mot-image
		Liste d’indices
		Niveau de difficulté
		Délai d’attente avant de dessiner entre chaque élément (ms)
		Est-ce un mode de dessin aléatoire
		Contenu du dessin virtuel
		Couleur de l’arrière-plan
		L’opacité de l’arrière-plan
		Liste d’éléments du dessin (traits) en ordre
		Couleur de l’élément
		Opacité du trait
		Épaisseur du trait de l’élément
		Liste des coordonnées *path SVG* en ordre

3.2.13 Envoyer le mot choisi par le dessinateur

Envoyé par le dessinateur qui a fait son choix parmi les mots à dessiner.

Envoyé par le serveur à tous les devineurs lorsque le dessinateur choisit son mot.

Événement : *word_choice*

Data Client	{
	“gameroomName”: str
	“words”: str[]
	}
		
		Identifiant du groupe de joueur
		Mots à choisir

Data Server	{
	“gameroomName”: str
	“words”: str[]
	}
		
		Identifiant du groupe de joueur
		Mot choisi parmi les choix donnés (array de taille 1)

3.2.14 Signaler le début du tour

Jamais envoyé par le client.

Envoyé par le serveur pour signaler le début du tour et donner des informations sur le tour.

Événement : *turn_start*

Data Client	-	

Data Server	{
	“endTimestamp”: long,
	“nAttempts”: int
	}
		
		Temps de fin du tour
		Nombre des tentatives pour deviner

3.2.15 Signaler la fin du tour

Jamais envoyé par le client.

Envoyé par le serveur pour signaler la fin du tour.

Événement : *turn_end*

Data Client	-	
Data Server	-	Événement de signal (vide)

3.2.16 Réussir à deviner le mot

Envoyé par le client pour signaler une tentative réussie de deviner le mot.

Envoyé par le serveur pour signaler une tentative réussi de deviner le mot à tout le monde.

Événement : *good_guess*

Data Client	{
	“gameroomName”: str
	“username”: string,
	}
		
		Identifiant du groupe de joueur
		Identifiant de l’utilisateur

Data Server	{
	“gameroomName”: str
	“username”: string,
	}
		
		Identifiant du groupe de joueur
		Identifiant de l’utilisateur

3.2.17 Échouer une tentative de deviner le mot

Envoyé par le client pour signaler une tentative échouée de deviner le mot.

Envoyé par le serveur pour signaler une tentative échouée de deviner le mot aux joueurs de la partie.

Événement : *bad_guess*

Data Client	{
	“gameroomName”: str
	“username”: str,
	“word”: str
	}
		
		Identifiant du groupe de joueur
		Identifiant de l’utilisateur
		Le mot correspondant à la tentative échouée

Data Server	{
	“gameroomName”: str
	“username”: str,
	“word”: str
	}
		
		Identifiant du groupe de joueur
		Identifiant de l’utilisateur
		Le mot correspondant à la tentative échouée

3.2.18 Nouveau score pour un joueur

Jamais appelé par le client.

Envoyé par le serveur pour signaler un nouveau score pour un joueur.

Événement : *new_score*

Data Client		

Data Server	{
	“score”: int
	“username”: string,
	}
		
		Nouveau score de l’utilisateur
		Identifiant de l’utilisateur

3.2.19 Diminuer le nombre essaie d'une unité

Jamais appelé par le client.

Envoyé par le serveur pour signaler qu'on doit diminuer le nombre d'essaies disponibles d'une unité.

Événement : *attempt_consumed*

Data Client		
Data Server		Événement de signal (vide)

3.2.20 Dévoiler un nouvel indice

Envoyé par le client pour signaler qu'un devineur a dévoilé un nouvel indice.

Envoyé par le serveur pour signaler aux autres joueurs qu'un devineur a dévoilé un nouvel indice.

Événement : *hint_asked*

Data Client	{
	"gameroomName": str
	"username": str,
	}
		Identifiant du groupe de joueur
		Identifiant de l'utilisateur

Data Server	{
	"gameroomName": str
	"username": str,
	}
		Identifiant du groupe de joueur
		Identifiant de l'utilisateur

3.2.21 Modifier la couleur de l'arrière-plan du dessin

Envoyé par le client lorsque le dessinateur modifie la couleur de l'arrière-plan du dessin.

Envoyé par le serveur pour signaler la modification de la couleur de l'arrière-plan à tous les devineurs.

Événement : *draw_edit_background*

Data Client	{
	"gameroomName": str
	"color": str,
	"opacity": float,
	}
		Identifiant du groupe de joueurs
		Couleur de l'arrière-plan du dessin
		Opacité de l'arrière-plan du dessin

Data Server	{
	"gameroomName": str
	"color": str,
	"opacity": float,
	}
		Identifiant du groupe de joueurs
		Couleur de l'arrière-plan du dessin
		Opacité de l'arrière-plan du dessin

3.2.22 Ajouter un nouvel élément au dessin

Envoyé par le client dessinateur lorsqu'il ajoute un nouveau trait au dessin.

Envoyé par le serveur à tous les clients devineurs pour signaler un nouveau trait du dessin.

Événement : *draw_new_element*

Data Client	{
	"gameroomName": str
	"id":int,
	"color": str,
	"opacity": float,
	"strokeWidth": int
	}
		
		Identifiant du groupe de joueurs
		Identifiant unique du trait
		Couleur du trait
		Opacité du trait
		Épaisseur du trait

Data Server	{
	"gameroomName": str
	"id":int,
	"color": str,
	"opacity": float,
	"strokeWidth": int
	}
		
		Identifiant unique du trait
		Identifiant unique du trait
		Couleur du trait
		Opacité du trait
		Épaisseur du trait

3.2.23 Ajouter des coordonnées à un élément du dessin

Envoyé par le client dessinateur lorsqu'il ajoute de nouvelles coordonnées à un trait existant.

Envoyé par le serveur à tous les clients devineurs pour signaler de nouvelles coordonnées tracées.

Événement : *draw_new_coords*

Data Client	{
	"gameroomName": str
	"id": int,
	"parts": list[str]
	}
		
		Identifiant du groupe de joueurs
		Identifiant unique du trait
		Liste des morceaux du type *path SVG* ajoutés au trait

Data Server	{
	"gameroomName": str
	"id":int,
	"parts": list[str]
	}
		
		Identifiant du groupe de joueurs
		Identifiant unique du trait
		Liste des morceaux du type *path SVG* ajoutés au trait

3.2.24 Supprimer un élément du dessin

Envoyé par le client dessinateur lorsqu'il efface un trait du dessin.

Envoyé par le serveur à tous les clients devineurs pour signaler la suppression d'un trait.

Événement : *draw_delete_element*

Data Client	{
	"gameroomName": str
	"id": int
	}
		Identifiant du groupe de joueurs
		Identifiant unique du trait supprimé

Data Server	{
	"gameroomName": str
	"id":int,
	}
		Identifiant du groupe de joueurs
		Identifiant unique du trait supprimé

3.2.25 Annuler la suppression de l'élément

Envoyé par le client dessinateur lorsqu'il veut réafficher un trait du dessin.

Envoyé par le serveur à tous les clients devineurs pour signaler le réaffichage d'un trait.

Événement : *draw_undelete_element*

Data Client	{
	"gameroomName": str
	"id": int
	}
		Identifiant du groupe de joueurs
		Identifiant unique du trait à réafficher

Data Server	{
	"gameroomName": str
	"id": int
	}
		Identifiant du groupe de joueurs
		Identifiant unique du trait à réafficher