

**Fais-moi un dessin**  
**Protocole de communication**

**Version 1.2**

# Historique des révisions

| Date       | Version | Description   | Auteur                  |
|------------|---------|---|-------------------------|
| 2020-09-10 | 1.0     | Version initiale  | Rostyslav Myhovich      |
| 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 |

# Table des matières

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Introduction</b>  | <b>4</b> |
| <b>2</b> | <b>Communication client-serveur</b>  | <b>4</b> |
| <b>3</b> | <b>Description des paquets</b>   | <b>4</b> |
| 3.1      | Paquets REST   | 5        |
| 3.1.1    | Création du profil d'utilisateur   | 5        |
| 3.1.2    | Authentification et création de sessions d'utilisation                             | 6        |
| 3.1.3    | Déconnexion de la session d'utilisation  | 6        |
| 3.1.4    | Changement de thème d'utilisateur  | 7        |
| 3.1.5    | Demande d'information du profil d'un utilisateur                                   | 8        |
| 3.1.6    | Création d'une paire mot-image   | 9        |
| 3.1.7    | Demander l'historique des messages complet d'un canal de discussion                | 10       |
| 3.1.8    | Demander la liste de tous les canaux de discussions disponibles pour l'utilisateur | 11       |
| 3.1.9    | Demander la liste de tous les groupes de joueurs                                   | 11       |
| 3.2      | Paquets Socket IO  | 12       |
| 3.2.1    | Création d'un nouveau canal de discussion  | 12       |
| 3.2.2    | Modification d'un canal de discussion existant                                     | 13       |
| 3.2.3    | Suppression du canal de discussion   | 13       |
| 3.2.4    | Nouveau message  | 14       |
| 3.2.5    | Nouveau groupe de joueurs  | 14       |
| 3.2.6    | Modification d'un membre du groupe de joueurs                                      | 15       |
| 3.2.7    | Suppression du groupe de joueurs   | 15       |
| 3.2.8    | Commencer une partie   | 16       |
| 3.2.9    | Terminer une partie  | 16       |
| 3.2.10   | Signaler l'information sur le tour de jeu qui suit                                 | 17       |
| 3.2.11   | Envoyer le mot choisi par le dessinateur   | 17       |
| 3.2.12   | Signaler le début du tour  | 18       |
| 3.2.13   | Signaler la fin du tour  | 18       |
| 3.2.14   | Deviner le mot   | 18       |
| 3.2.15   | Modifier la couleur de l'arrière-plan du dessin                                    | 19       |
| 3.2.16   | Ajouter un nouvel élément au dessin  | 19       |
| 3.2.17   | Ajouter des coordonnées à un élément du dessin                                     | 20       |
| 3.2.18   | Supprimer un élément du dessin   | 20       |
| 3.2.19   | Annuler la suppression de l'élément  | 20       |

# 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 bibliothèques 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

|                     |  |  |
|---------------------|--|--|
| <b>Query Params</b> |  |  |
| <b>Headers</b>      | -  |  |
| <b>Data</b>         | {<br>"username": str,<br>"password": str,<br>"firstName": str,<br>"lastName": str,<br>"avatarId": int<br>} | Nom d'utilisateur<br>Mot de passe<br>Prénom<br>Nom de famille<br>Identifiant de l'avatar sélectionné |

Réponses :

Status 200 : Succès de création du compte

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|

Status 400 : Nom d'utilisateur déjà pris

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|

### 3.1.2 Authentification et création de sessions d'utilisation

GET: SERVER\_URL/session

|                     |                                    |                                   |
|---------------------|------------------------------------|-----------------------------------|
| <b>Query Params</b> | -                                  |                                   |
| <b>Headers</b>      | "username": str<br>"password": str | Nom d'utilisateur<br>Mot de passe |
| <b>Data</b>         | -                                  |                                   |

Réponses :

Status 200 : Succès d'authentification

|             |                         |                                       |
|-------------|-------------------------|---------------------------------------|
| <b>Data</b> | {<br>"userId": int<br>} | Identifiant de l'utilisateur connecté |
|-------------|-------------------------|---------------------------------------|

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

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|

Status 400 : Utilisateur déjà connecté

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|

### 3.1.3 Déconnexion de la session d'utilisation

DELETE: SERVER\_URL/session

|                     |                 |                              |
|---------------------|-----------------|------------------------------|
| <b>Query Params</b> | -               |                              |
| <b>Headers</b>      | "username": str | Identifiant de l'utilisateur |
| <b>Data</b>         | -               |                              |

Réponses :

Status 200 : Succès de déconnexion

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|

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

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|

### 3.1.4 Changement de thème d'utilisateur

PUT: SERVER\_URL/theme

|                     |                          |                      |
|---------------------|--------------------------|----------------------|
| <b>Query Params</b> | -                        |                      |
| <b>Headers</b>      | -                        |                      |
| <b>Data</b>         | {<br>"themeId": int<br>} | Identifiant du thème |

Réponses :

Status 200 : Succès de changement de thème

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|

Status 404 : themeId n'existe pas

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|

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

GET: SERVER\_URL/profile/{profileId}

|                     |                  |                       |
|---------------------|------------------|-----------------------|
| <b>Query Params</b> | "profileId": int | Identifiant du profil |
| <b>Headers</b>      | -                |                       |
| <b>Data</b>         | -                |                       |

Réponses :

Status 200 : Succès de demande d'informations du profil

|             |  |  |
|-------------|--|--|
| <b>Data</b> | <pre>{   "username": str,   "firstName": str,   "lastName": str,   "avatarId": int,   "themeId": id,    "history": {     "sessions": [       {         "type": int,         "timestamp": long       },       ...     ],     "games": [       {         "gamemode": int,         "scores": {           "userId1": int,           "userId2": int,           ...         },         ...       },       ...     ]   },   "statistics": {     "numberOfGames": int,     "winRatio": float,     "averageGameTime": long,     "fullGameTime": long,     "bestSoloScore": int,   } }</pre> | <p>Nom d'utilisateur<br/>Prénom (si soi-même)<br/>Nom de famille (si soi-même)<br/>Identifiant de l'avatar sélectionné<br/>Identifiant du thème choisi (si soi-même)<br/>Historique détaillé (si soi-même)<br/>Historique des ouvertures /<br/>fermetures des sessions<br/>Type (ouverture / fermeture)<br/>Date d'événement</p> <p>Historique des parties jouées</p> <p>Mode de jeu<br/>Liste des utilisateurs et leur<br/>pointage</p> <p>Statistique des parties (si soi-même)<br/>Nombre des parties jouées<br/>Pourcentage de victoires<br/>Temps de jeu moyen<br/>Temps de jeu cumulatif<br/>Meilleur pointage dans une partie en<br/>mode sprint solo</p> |
|-------------|--|--|

Status 404 : profileId n'existe pas

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|



### 3.1.6 Création d'une paire mot-image

POST: SERVER\_URL/pair

|                     |   |  |
|---------------------|---|--|
| <b>Query Params</b> | -   |  |
| <b>Headers</b>      | -   |  |
| <b>Data</b>         | <pre>{   "word": str,   "hints": list[str],   "difficulty": int,   "delay": int,    "isRandom": bool,   "drawing": {     "background": str,     "elements": [       {         "color": str,         "strokeWidth": float,         "parts": list[str]       },       ...     ]   } }</pre> | <p>Mot de la paire mot-image<br/>         Liste d'indices<br/>         Niveau de difficulté<br/>         Délai d'attente avant de dessiner<br/>         entre chaque élément<br/>         Est-ce un mode de dessin aléatoire<br/>         Contenu du dessin</p> <p>Couleur de l'arrière-plan<br/>         Liste d'éléments (traits de crayon)</p> <p>Couleur de l'élément<br/>         Épaisseur du trait l'élément<br/>         Liste des morceaux du <i>SVG path</i> de<br/>         l'élément</p> |

#### Réponses :

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

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|

Status 400 : Contenu du message invalide

|             |               |   |
|-------------|---------------|---|
| <b>Data</b> | Error Message | Message spécifiant l'élément du JSON invalide |
|-------------|---------------|---|

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

GET: SERVER\_URL/chat/{chatroomId}

|                     |                   |                                    |
|---------------------|-------------------|------------------------------------|
| <b>Query Params</b> | "chatroomId": int | Identifiant du canal de discussion |
| <b>Headers</b>      | -                 |                                    |
| <b>Data</b>         | -                 |                                    |

#### Réponses :

Status 200 : Succès de demande

|             |   |   |
|-------------|---|---|
| <b>Data</b> | <pre>{   "messages": [     {       "chatroomId": int,       "userId": int,       "content": str,       "timestamp": long     },     ...   ] }</pre> | Liste de tous messages<br>chronologiques du canal<br>de discussion (plus ancien<br>au plus récent)<br><br>Identifiant du canal de discussion<br>Identifiant de l'utilisateur<br>Contenu du message<br>Temps de l'envoi du message |
|-------------|---|---|

Status 404 : Identifiant de canal de discussion inexistant

|             |   |  |
|-------------|---|--|
| <b>Data</b> | - |  |
|-------------|---|--|

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

GET: SERVER\_URL/chatrooms

|                     |   |  |
|---------------------|---|--|
| <b>Query Params</b> | - |  |
| <b>Headers</b>      | - |  |
| <b>Data</b>         | - |  |

Réponses :

Status 200 : Succès de demande

|             |   |  |
|-------------|---|--|
| <b>Data</b> | <pre>{   "chatrooms": [     {       "id": int,       "isIngame": bool,       "name": str,       "users": list[int]     },     ...   ] }</pre> | <p>Liste de tous les canaux de discussions visibles</p> <p>Identifiant du canal de discussion<br/>Est un canal de discussion de jeu<br/>Nom du canal de discussion<br/>Liste des identifiants d'utilisateurs faisant partie du canal de discussion</p> |
|-------------|---|--|

### 3.1.9 Demander la liste de tous les groupes de joueurs

GET: SERVER\_URL/gamerooms

|                     |   |  |
|---------------------|---|--|
| <b>Query Params</b> | - |  |
| <b>Headers</b>      | - |  |
| <b>Data</b>         | - |  |

Réponses :

Status 200 : Succès de demande

|             |   |   |
|-------------|---|---|
| <b>Data</b> | <pre>{   "gamerooms": [     {       "id": int,       "name": str,       "gamemode": int,       "difficulty": int,       "users": list[int]     },     ...   ] }</pre> | <p>Liste de tous les groupes de joueurs</p> <p>Identifiant du groupe de joueurs<br/>Nom du groupe de joueurs<br/>Mode de jeu du groupe<br/>Difficulté des parties du groupe<br/>Liste d'identifiants des utilisateurs dans le groupe de jeu</p> |
|-------------|---|---|

## 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 : chatroom\_new

|                    |   |   |
|--------------------|---|---|
| <b>Data Client</b> | {<br>"name": str,<br>"users": list[int]<br>}                                    | Nom du canal de discussion<br>Liste des identifiants d'utilisateurs<br>faisant partie du canal  |
| <b>Data Server</b> | {<br>"id": int,<br>"isIngame": bool,<br>"name": str,<br>"users": list[int]<br>} | Identifiant du canal de discussion<br>Est un canal de discussion de jeu<br>Nom du canal de discussion<br>Liste des identifiants d'utilisateurs<br>faisant partie du canal |

### 3.2.2 Modification d'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 avec une nouvelle liste des utilisateurs du groupe.

Événement : chatroom\_edit

|                    |  |   |
|--------------------|--|---|
| <b>Data Client</b> | {<br>"id": int,<br>"editType": int<br>"user": int<br>} | Identifiant du canal de discussion<br>Type de modification<br>(ajout/suppression)<br>Identifiant de l'utilisateur à<br>modifier |
| <b>Data Server</b> | {<br>"id": int,<br>"users": list[int]<br>}             | Identifiant du canal de discussion<br>Liste des identifiants d'utilisateurs<br>faisant partie du canal                          |

### 3.2.3 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 : chatroom\_delete

|                    |                     |                                    |
|--------------------|---------------------|------------------------------------|
| <b>Data Client</b> | {<br>"id": int<br>} | Identifiant du canal de discussion |
| <b>Data Server</b> | {<br>"id": int<br>} | Identifiant du canal de discussion |

### 3.2.4 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

|                    |  |   |
|--------------------|--|---|
| <b>Data Client</b> | {<br>"chatroomId": int,<br>"userId": int,<br>"content": str,<br>}                      | Identifiant du canal de discussion<br>Identifiant de l'utilisateur<br>Contenu du message                                |
| <b>Data Server</b> | {<br>"chatroomId": int,<br>"userId": int,<br>"content": str,<br>"timestamp": long<br>} | Identifiant du canal de discussion<br>Identifiant de l'utilisateur<br>Contenu du message<br>Temps de l'envoi du message |

### 3.2.5 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

|                    |   |  |
|--------------------|---|--|
| <b>Data Client</b> | {<br>"name": str,<br>"gamemode" : int,<br>"difficulty": int,<br>"users": list[int]<br>}               | Nom du groupe de joueurs<br>Mode de jeu<br>Difficulté du jeu<br>Liste des identifiants d'utilisateur                                     |
| <b>Data Server</b> | {<br>"id": int,<br>"name": str,<br>"gamemode" : int,<br>"difficulty": int,<br>"users": list[int]<br>} | Identifiant du groupe de joueurs<br>Nom du groupe de joueurs<br>Mode de jeu<br>Difficulté du jeu<br>Liste des identifiants d'utilisateur |

### 3.2.6 Modification d'un membre du groupe de joueurs

Appelé par un client lorsqu'il rejoint ou 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\_edit

|                    |   |   |
|--------------------|---|---|
| <b>Data Client</b> | {<br>"id": int,<br>"editType": int,<br>"user": int<br>} | Identifiant du groupe de joueurs<br>Type de modification<br>(ajout/suppression)<br>Utilisateur à modifier |
| <b>Data Server</b> | {<br>"id": int,<br>"users": list[int]<br>}              | Identifiant du groupe de joueurs<br>Liste des identifiants d'utilisateur                                  |

### 3.2.7 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\_edit

|                    |                     |                                  |
|--------------------|---------------------|----------------------------------|
| <b>Data Client</b> | -                   |                                  |
| <b>Data Server</b> | {<br>"id": int<br>} | Identifiant du groupe de joueurs |

### 3.2.8 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

|                    |                       |                                  |
|--------------------|-----------------------|----------------------------------|
| <b>Data Client</b> | {<br>" id ": int<br>} | Identifiant du groupe de joueurs |
| <b>Data Server</b> | {<br>" id ": int<br>} | Identifiant du groupe de joueurs |

### 3.2.9 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

|                    |                       |                                  |
|--------------------|-----------------------|----------------------------------|
| <b>Data Client</b> | -                     |                                  |
| <b>Data Server</b> | {<br>" id ": int<br>} | Identifiant du groupe de joueurs |



### 3.2.10 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

|                    |   |   |
|--------------------|---|---|
| <b>Data Client</b> | -   |   |
| <b>Data Server</b> | <pre>{     "drawer": int,     "wordChoices": list[str],     "virtualDrawing": {         "delay": int,         "drawing": {             "background": str,             "elements": [                 {                     "color": str,                     "strokeWidth": float,                     "parts": list[str]                 },                 ...             ]         }     } }</pre> | Identifiant du dessinateur<br>Liste des choix des mots pour le dessinateur<br>Information du dessin virtuel, si le dessinateur est virtuel<br>Temps d'attente (ms) entre chaque partie d'élément<br>Contenu du dessin virtuel<br>Couleur de l'arrière-plan<br>Liste d'éléments du dessin (traits) en ordre<br>Couleur de l'élément<br>Épaisseur du trait de l'élément<br>Liste des coordonnées <i>path SVG</i> en ordre |

### 3.2.11 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

|                    |                                |                                   |
|--------------------|--------------------------------|-----------------------------------|
| <b>Data Client</b> | <pre>{     "word": str }</pre> | Mot choisi parmi les choix donnés |
| <b>Data Server</b> | <pre>{     "word": str }</pre> | Mot choisi parmi les choix donnés |

### 3.2.12 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

|                    |  |  |
|--------------------|--|--|
| <b>Data Client</b> | -  |  |
| <b>Data Server</b> | {<br>“endtime”: long,<br>“nAttempts”: int<br>} | Temps de fin du tour<br>Nombre des tentatives pour deviner |

### 3.2.13 Signaler la fin du tour

Jamais envoyé par le client.

Envoyé par le serveur pour signaler la fin du tour.

Événement : turn\_end

|                    |   |                            |
|--------------------|---|----------------------------|
| <b>Data Client</b> | - |                            |
| <b>Data Server</b> | - | Événement de signal (vide) |

### 3.2.14 Deviner le mot

Envoyé par le client lorsqu’il a deviné le bon mot

Envoyé par le serveur pour signaler le succès d’un joueur de la partie avec son pointage.

Événement : word\_guess

|                    |  |   |
|--------------------|--|---|
| <b>Data Client</b> | -                                      | Événement de signal (vide)                                |
| <b>Data Server</b> | {<br>“user”: int,<br>“score”: int<br>} | Identifiant de l’utilisateur<br>Pointage de l’utilisateur |

### 3.2.15 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

|                    |                                  |                                     |
|--------------------|----------------------------------|-------------------------------------|
| <b>Data Client</b> | {<br>"backgroundColor": str<br>} | Couleur de l'arrière-plan du dessin |
| <b>Data Server</b> | {<br>"backgroundColor": str<br>} | Couleur de l'arrière-plan du dessin |

### 3.2.16 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

|                    |  |   |
|--------------------|--|---|
| <b>Data Client</b> | {<br>"id":int,<br>"color": str,<br>"strokeWidth": int<br>} | Identifiant unique du trait<br>Couleur du trait<br>Épaisseur du trait |
| <b>Data Server</b> | {<br>"id":int,<br>"color": str,<br>"strokeWidth": int<br>} | Identifiant unique du trait<br>Couleur du trait<br>Épaisseur du trait |

### 3.2.17 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

|                    |  |   |
|--------------------|--|---|
| <b>Data Client</b> | {<br>" id ": int,<br>" parts ": list[str]<br>} | Identifiant unique du trait<br>Liste des morceaux du type <i>path</i><br>SVG ajoutés au trait |
| <b>Data Server</b> | {<br>" id ":int,<br>" parts ": list[str]<br>}  | Identifiant unique du trait<br>Liste des morceaux du type <i>path</i><br>SVG ajoutés au trait |

### 3.2.18 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

|                    |                       |                                      |
|--------------------|-----------------------|--------------------------------------|
| <b>Data Client</b> | {<br>" id ": int<br>} | Identifiant unique du trait supprimé |
| <b>Data Server</b> | {<br>" id ":int,<br>} | Identifiant unique du trait supprimé |

### 3.2.19 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

|                    |                       |  |
|--------------------|-----------------------|--|
| <b>Data Client</b> | {<br>" id ": int<br>} | Identifiant unique du trait à réafficher |
| <b>Data Server</b> | {<br>" id ": int<br>} | Identifiant unique du trait à réafficher |