

Protocoles et Services Internet - Projet

Minehive - Protocole

Démineur : Le jeu	2
Règles et restrictions de base d'une partie	2
Description et règles du jeu solo	2
Nombre de points obtenus par clic	2
Interactions entre joueurs	2
Notions	2
Déroulement d'une partie	3
Architecture multijoueurs	4
Architecture réseau trois tiers	5
Données, dans un fichier XML ranking.xml	5
Applications : Serveur et hôtes	5
Client	5
Implémentation du Démineur	5
Modèle	5
Mise en place du plateau	6
Aperçu exemple de l'arborescence de développement	6
Communications serveur, hôtes, et clients en TCP	7
Spécification de la forme des messages	7
Serveur	7
Réponses aux requêtes de client	8
Vérification de la disponibilité des joueurs connectés	9
Réponses aux requêtes d'hôte	9
Client	10
Messages au Serveur	10
Messages à l'Hôte	11
Hôte	11
Création d'une partie	11
Connexion d'un joueur	12
Vérification de la disponibilité d'un client	12
En cours de partie	13
Fin de partie	13
Récapitulatif des commandes	13
Commentaires et suggestions	13
Crédits : Équipe Moghive	13

Démineur : Le jeu

Règles et restrictions de base d'une partie

Description et règles du jeu solo

[https://fr.wikipedia.org/wiki/D%C3%A9mineur_\(jeu\)](https://fr.wikipedia.org/wiki/D%C3%A9mineur_(jeu))

Plateau : Rectangulaire 2D de dimensions 16*30, 99 mines.

Fin de partie : Quand toutes les cases sans mines ont été découvertes.

Nombre de points obtenus par clic

- Clic sur une case vide : +1 point
- Clic sur une case numérotée n : + n points
- Clic sur une mine : -10 points

Cliquer sur une case vide dévoile récursivement toutes les cases adjacentes (excluant diagonales) vides jusqu'à tomber sur des numéros. Les cases découvertes ainsi automatique rapportent des points à l'initiateur du clic.

Si au moins un autre joueur a rejoint la partie en cours :

- Multiplicateur de points obtenus/perdus au moment d'un clic: x [nombre de participants présents]. Si un joueur : $x1$. Si 5 joueurs, $x5$.
- A déminé le plus de cases : +50 points, indépendamment du multiplicateur.
- Est tombé sur le plus de mines : -50 points, indépendamment du multiplicateur.
- Pas de bonus/malus selon le temps passé sur la partie.

Interactions entre joueurs

Notions

- **Serveur** (Server) : Liste en ligne des joueurs pas en partie, et liste des parties en cours
- **Classement** (Ranking) : Liste stockée côté serveur contenant l'ensemble des joueurs, leur mot de passe, et leur nombre global de points.
- **Partie** (Match) : Partie de démineur avec 1 à 10 joueurs.
- **Hôte** (Host) : Processus intermédiaire produit par le serveur qui gère une partie entre clients, et retransmet en direct les données au serveur.
- **Joueur** (Player) : Client **connecté** au serveur ou assigné à une partie.

Protocoles et Services Internet - Projet

- **Utilisateur** : Entité joueur que le serveur a connu, ou joueur actuellement connecté. Stocké dans les données du serveur. Si le joueur *Adil* se déconnecte et qu'on veut l'énoncer, on parle alors d'un utilisateur. Dénomination pas forcément logique, mais qui aidera à distinguer par la suite.
- **Connecté** : Un client est dit connecté s'il est apte à échanger des données, même s'il ne le fait pas forcément.
- **Activité** : Un utilisateur est dit actif par rapport à une entité s'il est en train d'interagir avec. Les entités ont divers moyens de repérer un utilisateur inactif : habituellement, aucune action de sa part depuis un certain temps.
- **Identification** : Un utilisateur déjà connecté (socket établie) s'identifie auprès d'une identité en fournissant ses informations d'identification : Identifiant et Mot de passe
- **Plateau** (Board) : Tableau de 16 lignes dont chacune contient 30 cases.
- **Case x y** : Dans un tel ordre, x est toujours l'abscisse, et y l'ordonnée.
- **Numéro** (number) d'une case : Numéro, si existant, d'une case, dans [0, 8].
- **Valeur** (value) d'une case : peut valoir soit {mine, number}.
- **Contenu** (content) d'une case : peut valoir soit {hidden, value}.

Déroulement d'une partie

Sur un serveur, un joueur peut lancer une partie, hébergée par un hôte *Host*, qui aura pour nom : Partie_1, ..., Partie_X selon un compteur.

Après création, le joueur doit rejoindre la partie : le serveur retire alors le joueur de sa liste de joueurs disponibles *available*.

D'autres joueurs peuvent se joindre à la partie en cours, ou créer une autre partie. Quoi qu'il arrive, le nombre de points est propre à un joueur.

La visibilité du plateau est commune avec pour indications Mine, Numéro, [Vide] : Lorsqu'un joueur démine une case, tous les autres joueurs sont informés de son contenu et ne peuvent plus cliquer dessus.

À chaque clic d'un joueur sur le plateau, l'hôte :

- met à jour ses données : points obtenus/perdus, nombre de clics justes, de mines trouvées par joueur
- informe tous les joueurs des coordonnées et du contenu de la case

S'il a trouvé une case vide, alors les cases adjacentes (excluant diagonales) sont récursivement fouillées jusqu'à tomber sur un numéro.

Protocoles et Services Internet - Projet

En fin de partie, l'hôte :

- Envoie un message de fin de partie à tous les joueurs
- Envoie à tous les joueurs les résultats de la partie : nomJoueur, nbPoints, nbClicsJustes, nbMines.
- Envoie au serveur une liste de joueurs et les points que chacun a gagnés ou perdus.

Architecture multijoueurs

Architecture réseau trois tiers

Fortement inspirée de : https://fr.wikipedia.org/wiki/Architecture_trois_tiers.

Données, dans un fichier XML ranking.xml (optionnel)

[Optionnel] Présent sur la machine où le serveur est lancé, stocke toutes les données du jeu. Il contient tous les joueurs et leur score global au jeu. Voici un [XMLinitial](#).

Ce fichier doit se trouver dans un dossier **res/** (*resources*).

Applications : Serveur et hôtes

Un serveur fait office d'application. Il liste les utilisateurs connectés à celui-ci et prêts à jouer. Sur demande des utilisateurs, il peut créer une partie, gérée par un hôte.

Client

Pour vraiment pouvoir opérer en tant qu'utilisateur, un client doit fournir un identifiant et un mot de passe.

Implémentation du Démineur

Modèle

Une case **Square** contient un entier **content** décrivant son contenu :

- -1 : mine
- 0 : vide
- 1 à 8 : nombre de mines à côté

Elle a aussi une variable booléenne **discovered** initialement false, qui indique si les joueurs *Player* connaissent son contenu.

Protocoles et Services Internet - Projet

Un plateau **Board**, de dimensions **WIDTH**=30 et **HEIGHT**=16, contient un tableau *Square*[16][30] **squares**. Le tableau est imposé comme structure en raison du temps d'accès constant $O(1)$, à condition de connaître l'indice de la case voulue. Il a un attribut **MINES**=99 qui indique le nombre de mines. Il a de plus un entier **totalDiscovered** qui indique le nombre total de cases découvertes, en incluant les mines.

Un joueur **Player** a un nom **username**, un mot de passe **password**, un nombre total de points **points**. Il est géré par le serveur.

Un joueur en partie **InGamePlayer** hérite du **Player** et a en plus le nombre de points **inGamePoints** obtenus durant la partie, le nombre de cases correctement déminées **safeSquares**, le nombre de mines trouvées **foundMines**. Il est géré par un hôte de partie. Il a un attribut **active**=true par défaut, que l'hôte met à false en cas d'inactivité.

[Server](#), [Client](#), et [Host](#) sont également expliqués dans la section [Communications](#).

Un **HostData** regroupe **name**, **IP**, **port** d'un *Host*.

Mise en place du plateau

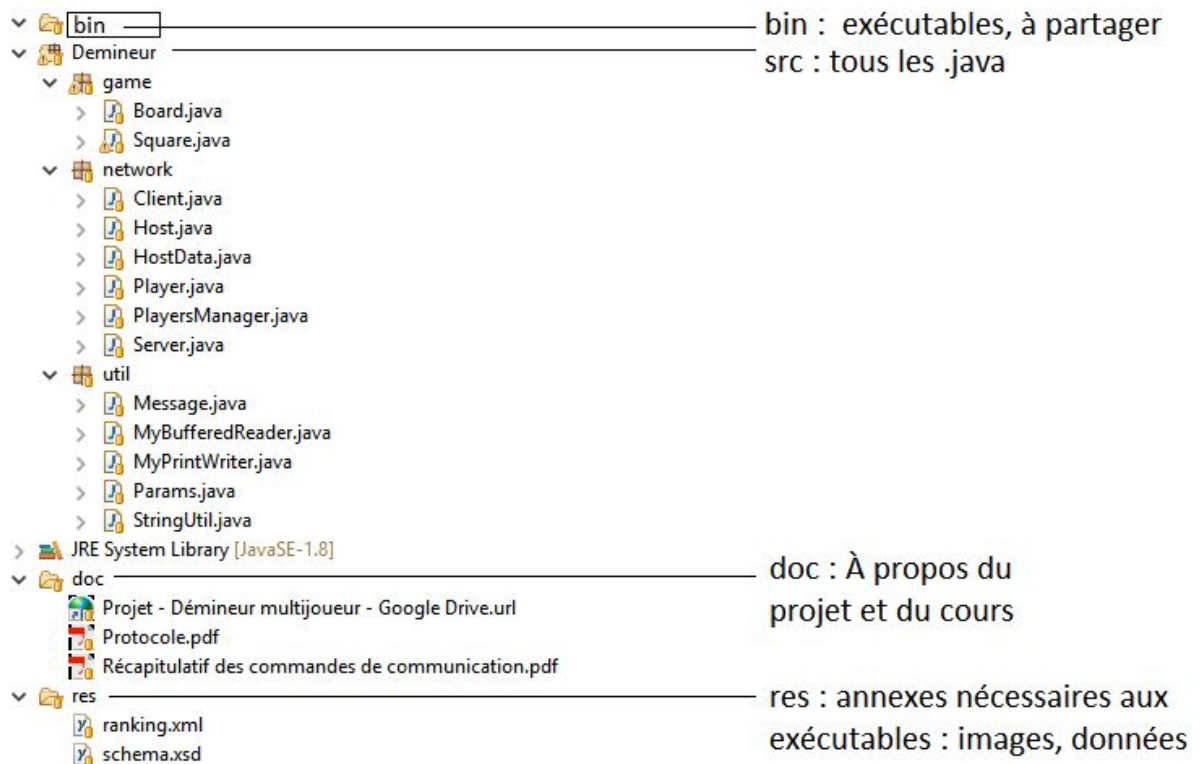
Pour initialiser les mines sur le plateau, il faut un générateur aléatoire qui indique les coordonnées de la prochaine mine à placer. S'il y a déjà une mine à la case indiquée, recommencer.

Pour de meilleures performances, il est conseillé de déterminer le numéro d'une case [directement après la mise en place d'une mine](#).

Première découverte positive : Le premier joueur à cliquer sur une case (**Board.totalDiscovered** == 0) ne doit jamais tomber sur une mine. Si c'est le cas, avant d'envoyer des données, la mine est retirée et placée autre part aléatoirement.

Aperçu exemple de l'arborescence de développement

Protocoles et Services Internet - Projet



Communications serveur, hôtes, et clients en TCP

Spécification de la forme des messages

La réception des messages se fera avec `BufferedReader.readLine()`, mais lors de l'envoi, on ne perdra guère de temps à gérer les sauts de ligne ou les `.flush()` : il suffira d'initialiser un `PrintWriter` (`Writer out, boolean autoFlush`) et d'appeler sa méthode `PrintWriter#println()`. Aucun message envoyé ne doit donc contenir de sauts de ligne (`\r` ou `\n`), ces derniers pouvant provoquer des décalages lors de la réception avec `BufferedReader#readLine()`.

Le caractère séparateur entre les mots d'un message est le dièse (#).

Les quatre premiers octets d'un message serviront à encoder dans une chaîne de caractères son **type**.

On peut **ajouter du texte après chaque contenu de message + paramètres imposés** : Il est soit ignoré si pas implémenté, soit traité sans communication réseau supplémentaire, mais ne doit en aucun cas faire planter le programme.
Exceptions : nombre de paramètres variables, pour éviter les confusions.

À toute réception de message de type inconnu, l'entité réceptionniste répond

Protocoles et Services Internet - Projet

“**IDK**[Initiale entité]”. Par exemple, si un client envoie “PILE” à un serveur qui ne traite pas ce type de message, ce dernier répond “IDKS”.

Serveur

Un serveur **Server** est créé et lancé sur le **port** 5555 par défaut, avec un `ServerSocket`. Il peut [lancer](#) jusqu’à 10 hôtes, et donc 10 parties, mais ne conserve des hôtes que des données : `List<HostData> hostsData`. Il ne doit surtout pas créer de nouvelle instance de `Host` (`new`). En revanche, il vérifie son activité avec un échange de messages ping-pong, avec une socket d’écoute sur le port 7777.

[Optionnel] Le serveur initialise une liste *users* de joueurs connus depuis un fichier *ranking.xml* : `List<Player> users`. Cette liste *users* n’est pas limitée.

Il peut accueillir un maximum de 110 joueurs (incluant ceux en partie) simultanément connectés : `List<Player> available`, `Map<Player, HostData> inGame`. Un utilisateur déjà en cours de partie ne peut pas rejoindre le serveur ou [une autre partie](#).

Une fois qu’un client s’est enregistré ou connecté avec succès et a été ajouté à la liste *available*, la connexion est maintenue jusqu’à que le client décide de quitter, soit jugé déconnecté ou inactif, rejoigne une partie en cours, ou qu’il y ait une autre connexion avec les mêmes identifiants. Le serveur n’envoie KICK que lorsqu’il choisit d’éjecter le client indépendamment des actions de ce dernier.

Avant d’effectuer toute action (excepté pour LEAV), l’utilisateur doit s’identifier avec REGI.

Réponses aux requêtes de client

Consulter d’abord [Client - Messages au Serveur](#).

- *REGI Username Password* : Connexion acceptée : **IDOK**, **IDNO** sinon.
 - Si le serveur est plein (110 connectés), **IDNO**. Sinon, continuer.
 - Si le message n’est pas de type REGI : **IDKS**.
 - Vérifier la présence de *Username* et *Password* : **IDNO** si invalide. Sinon, continuer.
 - *Username* inconnu (pas dans *users*) ? Créer un *Player*, ajouté à *user* et *available*. [Optionnel] Mettre à jour *ranking.xml*. **IDOK**. Sinon :
 - Mot de passe invalide ? **IDNO**. Sinon :
 - Parmi sa liste *available*, cherche un *Player* qui a *Username*.

Protocoles et Services Internet - Projet

Trouvé ? Oui : **IDOK**, “reconnexion”. Le serveur ferme la socket déjà ouverte pour accepter une nouvelle connexion. Sinon :

- Chercher dans *inGame*. Trouvé ? Oui : **IDIG hostIP hostPort**, où *hostIP* et *hostPort* sont les données du *HostData* correspondant dans la map *inGame*. Puis le serveur ferme immédiatement la socket. L'utilisateur doit d'abord se connecter à l'hôte pour finir sa partie. Sinon :
- Connexion : ajout du *Player* dans *available*. **IDOK**.
- **LSMA** : Envoyer **LMNB nbMatches**.
 - Demander à chaque Host **RQDT Username**.
 - Transférer les données reçues au client correspondant à *Username* sous la forme **MATC IP Port nomMatch Complétion [nomJoueur inGamePoints]...**, où *IP* et *Port* permettent de se connecter à *Host*, *nomMatch* est le nom de l'hôte, *Complétion* un pourcentage égal à $([cases\ déminées]/[total\ cases])*100$, et les noms des joueurs connectés ainsi que leur score dans la partie.
- **LSAV** : Envoyer **LANB nbPlayers**. Envoyer les joueurs un par un sous la forme **AVAI nomPlayer nbPoints**.
- **LSUS** : Envoyer **LUNB nbUsers**. Envoyer les utilisateurs un par un sous la forme **USER nomPlayer nbPoints**.
- **NWMA [nomInvitéN]...** : S'il y a moins de 10 parties en cours :
 - lance un nouveau *Host* nommé *Partie_X*, où *X* est déterminé selon un compteur global, et établit une connexion avec. Le serveur renvoie ensuite **NWOK HostIP HostPort**, à l'initiateur du message, et à chaque Invité dont le nom *nomInvitéN* a été fourni.
 - Sinon, renvoie **FULL**.
 - Pour toute autre raison (exemple : aucun port libre), le serveur peut renvoyer **NWNO**.
- **NWMA ALL** : Pareil que la commande ci-dessus, mais envoie l'invitation à tous les joueurs disponibles *available*.
- **LEAV** : Retire le *Player* correspondant de *available*. Ferme la connexion.

Réponses aux requêtes d'hôte

Consulter d'abord les requêtes possibles d'un [hôte](#).

- **LOGI**: **IDOK** si accepté, **IDNO** si refusé, ferme la connexion.
- **PLIN MatchName Username Password** :
 - Si *Username* n'existe pas dans *users* : **PLNO Username**. Sinon :
 - Si *Password* est invalide : **PLNO Username**. Sinon :
 - S'il est déjà dans *inGame* : **PLNO Username**. Sinon :
 - Renvoie **PLOK Username nbPoints** et ajoute le *Player*

Protocoles et Services Internet - Projet

correspondant à *inGame*. De plus, si le *Player* est dans *available* (et est donc connecté au *Serveur*), le retire et ferme la connexion et le service de gestion associés.

- *SCPS Username totalPoints* : Retire de *inGame* l'entrée (Player/HostData) correspondante. Met à jour les points du Player qui a pour nom Username. [Optionnel] Met à jour le fichier XML.
- *ENDS matchName* : Retire de *hostsData* celui qui contient *matchName* et ferme la connexion avec.

Vérification de la disponibilité des autres entités

Détection de la **connexion** du **client** : Le serveur envoie toutes les **5 secondes RUOK**, ce qui doit amener le client à répondre **IMOK**. S'il ne reçoit aucun message du client pendant **10 secondes**, il le considère comme déconnecté et ferme la socket.

Détection de la **connexion** de l'**hôte** : Pareil qu'avec le client. En cas de déconnexion de l'hôte, le serveur retire les joueurs correspondants de *inGame*.

[Optionnel] Détection de l'**activité** : On vérifie également que l'utilisateur reste disponible sur le serveur. Si le serveur ne reçoit aucun message autre que *IMOK* pendant plus de **300 secondes**, il est éjecté. Dans ce dernier cas, puisqu'il pense que la socket est encore ouverte, il envoie le message **KICK**, ferme la socket correspondante, n'interagira plus avec, et finalement retire le *Player* de sa liste *available*.

Client

Le client est la seule entité où l'utilisateur peut entrer des données, soit avec le clavier, soit avec une interface graphique. Il tentera de se connecter à un serveur *Server*, puis à un hôte de partie *Host*, mais il lui faudra d'abord un nom de joueur *Username*, et le mot de passe *Password* associé.

Pour entrer en communication avec un serveur lancé, un client **Client** doit initialiser une socket à **[ServerIP]/[ServerPort]**. Il peut alors envoyer des messages au serveur.

Lorsqu'un *Client* envoie un **[TYPE] Username Password** à une entité, en règle générale, les cas ci-dessous produisent IDNO :

Paramètres d'envoi	Retour, voire actions
Username ou/et Password non défini	IDNO
Username existant, Password faux	IDNO

Protocoles et Services Internet - Projet

Username identique déjà connecté	IDNO
----------------------------------	------

Des réactions spécifiques liées à certains TYPE sont précisées en détails dans la section correspondante.

Messages au Serveur

- **REGI Username Password** : S'enregistrer ou se connecter à un serveur.
- **LSMA** : Obtenir la liste des parties en cours, avec pour chacune les joueurs présents.
- **LSAV** : Obtenir la liste des joueurs disponibles pour jouer.
- **LSUS** : Obtenir la liste des utilisateurs.
- **NWMA [nomInvitéX]...** : Créer une nouvelle partie en invitant d'autres joueurs
- **NWMA ALL** : Pareil, en invitant tous les joueurs disponibles.
- **LEAV** : Quitter le serveur, sans rejoindre de partie.

Messages à l'Hôte

Là encore, il faut l'IP et le port de l'hôte pour pouvoir interagir avec.

- **JOIN Username Password** : Se connecter à un hôte.
- **CLIC Abscisse Ordonnée** : Fouiller la case aux coordonnées envoyées, sans avoir besoin de s'identifier. Si le plateau est de dimensions 16*30, alors abscisse sera accepté seulement si entre 0 et 15, et ordonnée entre 0 et 29. Ne marche que si la connexion TCP a bien été établie.

Le [serveur doit ensuite réagir](#) à chacun de ces messages.

Hôte

Un hôte **Host** a un nom de partie **name**, initialisé par le serveur avec **Partie_X**, selon un compteur global. Il a une adresse **IP** et un **port** de connexion, en plus de l'adresse IP du serveur **serverIP**, et de son port **serverPort**. Il certifie au serveur son activité en répondant **IMOK** à chaque **RUOK**.

Il gère une liste de *InGamePlayer* **inGamePlayers** initialement vide. Il crée chaque *InGamePlayer* avec des données reçues du *Server*.

Il a et [initialise](#) un plateau **Board** du Démineur.

Il initialise une connexion au serveur à son port 7777, en envoyant **LOGI MatchName**.

Protocoles et Services Internet - Projet

À tout moment, s'il reçoit *RQDT Username* du serveur, il répond **SDDT Username IP Port nomMatch Complétion [nomJoueur inGamePoints]...**, où *IP* et *Port* permettent de se connecter à *Host*, où *nomMatch* est le nom de l'hôte, *Complétion* un pourcentage **entier** égal à $([totalDiscovered]/[total cases])*100$, et les noms des joueurs connectés ainsi que leur score dans la partie.

Création d'une partie

Le serveur se charge de trouver un nom *name*, une *IP* et un *Port* libres pour construire des données d'hôte *HostData*, qu'il garde dans sa liste *hostsData* : on se base sur le réseau et non la POO, donc Server et Host doivent communiquer par sockets.

[*Optionnel* : thread à la place de programme] Le serveur *lance un programme Host* en fournissant en paramètres, dans l'ordre : serverIP serverHost HostData.name HostData.IP HostData.port

Connexion d'un joueur

Quand un client établit une socket de communication TCP à Host, il doit envoyer *JOIN Username Password*.

- S'il figure déjà dans *inGamePlayers* :
 - Si le mot de passe ne correspond pas : **JNNO**.
 - Si *InGamePlayer.active* est vrai, **JNNO** parce qu'il est déjà en train de jouer.
 - Autrement : Reconnexion, *InGamePlayer.active* vaut true, et *Host* renvoie **JNOK nbLignes** au Client, et enchaîne avec **BDIT numLigne Case1 ... Case30** où *numLigne* vaut de 0 à 15, et *CaseColumn* vaut :
 - Si *discovered* vaut false, un String "X".
 - Autrement, le *content* entier du Square, valant de -1 à 8.
 De plus, il envoie **IGNB nbInGamePlayers**, puis **IGPL Username inGamePoints totalPoints safeSquares foundMines** pour chaque *InGamePlayer* dans *inGamePlayers*.
 Pour finir, il envoie à chaque **autre** *InGamePlayer* actif le profil du nouvel arrivant : **CONN Username inGamePoints totalPoints safeSquares foundMines**.
- Si *inGamePlayers* est plein, **JNNO**. Sinon :
- Host envoie alors à Server **PLIN MatchName Username Password** pour identifier l'utilisateur. S'il reçoit :
 - *PLNO Username*, il renvoie **JNNO** avec éventuellement un message d'erreur reçu du *PLNO* du serveur.

Protocoles et Services Internet - Projet

- *PLOK Username nbPoints* : Renvoie **JNOK nbLignes**, puis la suite comme **ci-dessus**. Crée un nouvel *InGamePlayer* et l'ajoute à *inGamePlayers*.

Vérification de la disponibilité d'un client

Les contrôles de connexion du client sont similaires [à ceux du serveur](#). En cas de déconnexion, il met *InGamePlayer.active* à false, et envoie à tous les *InGamePlayer* : **DECO Username**. Dans ce cas, le multiplicateur de points par case est décrémenté. La liste *inGamePlayers* n'est **pas modifiée**.

[Optionnel] En cas de déconnexion, un joueur ne peut rejoindre ni le Server, ni un autre Host. Il peut se reconnecter à sa partie en cours avec *JOIN hostIP hostPort*.

[Optionnel] L'hôte *Host* doit pouvoir repérer, à sa façon mais sans communication réseau, quand un joueur est inactif. Il fait alors les mêmes choses que *DECO*, mais avec **AFKP Username**. La socket doit rester **ouverte**. Quand le joueur AFK est de retour, **BACK Username** est envoyé à tous.

[Optionnel] Si tous les *inGamePlayers* sont marqués inactifs, alors un décompte est lancé, et après 60 secondes, si personne ne redevient actif, tous les *inGamePoints* positifs sont mis à 0, les *totalPoints* réajustés, et la fin de partie est forcée sans aucun bonus/malus spécial.

En cours de partie

Quand un client veut déminer une case avec *CLIC Abscisse Ordonnée*, il envoie, si :

- Les coordonnées sont invalides : **OORG wrongAbscissa wrongOrdinate**
- La case a déjà été découverte : **LATE**
- Case valide : *squares[ordonnée][abscisse].discovered* vaut true. Incrémente de 1 *totalDiscovered*. Envoie à tous les *InGamePlayers* : **SQRD abscisse ordonnée content nbPoints Username**, où *content* est un entier au contenu de la case, *nbPoints* le nombre de points reçus ou perdus en considérant le multiplicateur dépendant du **nombre de joueurs actifs**, *Username* celui qui a cliqué sur la case. Met à jour les données de l'*InGamePlayer* qui a cliqué. Si la case est vide (valeur 0), l'hôte explore récursivement les cases adjacentes (excluant diagonales) jusqu'à tomber sur des numéros : Il envoie *SQRD [.f.]* pour chaque case.

La fin de partie survient dès qu'il n'y a plus de bonne case cliquable, soit

Protocoles et Services Internet - Projet

lorsque $[totalDiscovered - nbMinesDécouvertes] == [total cases - nbMines]$.

Fin de partie

Le serveur applique le bonus +50 à celui qui a déminé le plus de bonnes cases, et -50 à celui tombé sur le plus de mines. Il envoie ensuite, dans l'ordre :

1. Au serveur : **SCPS Username totalPoints**. Répète pour chaque InGamePlayer.
2. Au serveur : **ENDS matchName**.
3. À tous les InGamePlayer : **SCPC Username inGamePoints totalPoints safeSquares foundMines [Commentaire]**, où *Commentaire* optionnel est un Mot décrivant un bonus obtenu : BestMinesweeper, ou MostBlownUp. Répète l'envoi pour chaque InGamePlayer. Puis ferme la connexion.
4. À tous les InGamePlayer : **ENDC nbPlayers**.

Récapitulatif des commandes

Voir la [Google Sheets correspondante](#).

Commentaires et suggestions

Chaque client est libre de traiter les données reçues comme il l'entend. De ce fait, au lieu d'envoyer "CLIC 3 10", il peut convertir depuis l'utilisateur un "K4" beaucoup plus instinctif.

Côté client, l'affichage et la mise à jour des données reçues d'un hôte, lors d'une partie, n'est pas précisée ni même obligatoire.

Plus les données reçues seront lisibles et compréhensibles, et plus l'envoi de messages est simplifiée et instinctive, plus l'utilisateur sera à l'aise pour jouer et accumuler efficacement des points.

Crédits



Équipe
Moghive

Protocoles et Services Internet - Projet

Duo Tempus Finis



Kévin Khau

Tomek Lecocq



Duo Milites



Adil Champion

Aylin
Kocoglu



Duo Caetuna



Valloris Cyllly

Somaya
Ben Khemis



Duo Lorica



Christophe Lam

Al Hassane
Megninta



Héros Machina

Protocoles et Services Internet - Projet

Helmi Zegaya

