Protocoles et Services Internet - Projet

Démineur Multijoueur - Protocole

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)

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) : Thread 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.

- **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.

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.

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

Présent impérativement 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 XML initial.

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.

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*.

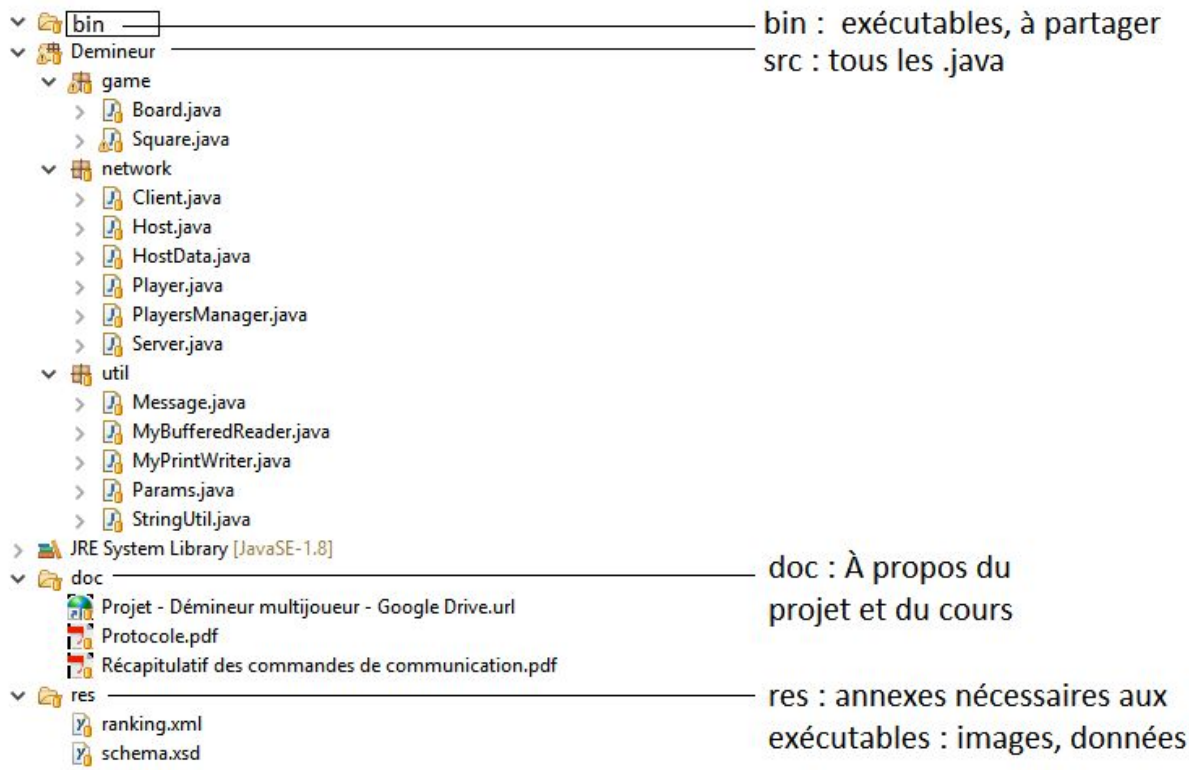
Mise en place du plateau

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



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#print**ln**(). 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 dièze (#).

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.

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).

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 le client décide de quitter (LEAV), soit jugé déconnecté ou inactif (KICK), ou rejoigne une partie en cours (JOIN), ou qu'il y ait une autre connexion avec les mêmes identifiants (KICK).

Avant d'effectuer toute action, même quitter (LEAV), l'utilisateur doit s'identifier avec REGI.

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

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.
  - 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*. Mettre à jour ranking.xml. **IDOK**. Sinon :
  - Mot de passe invalide ? **IDNO**. Sinon :
  - Parmi sa liste *available*, cherche un *Player* qui a *Username*. 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**
    - Transférer les données reçues 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 [nomInvitéX]...** : 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 renvoie **NWOK HostIP HostPort**, à l'initiateur du message, et à chaque Invité dont le nom *nomInvitéX* 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.

Vérification de la disponibilité des joueurs connectés

Détection de la **connexion** : Le serveur étant en écoute permanente du client, il est capable de détecter sa déconnexion, et peut de son côté fermer la socket et le retirer de *available*.

Détection de l'**activité** : On vérifie également que l'utilisateur reste disponible sur le serveur. Si le serveur ne reçoit aucun message pendant plus de 300secs, il est éjecté.

Lorsqu'un serveur éjecte un client, il envoie le message **KICK**, ferme la socket correspondante et n'interagira plus avec, et finalement retire le *Player* de sa liste *available*.

Réponses aux requêtes d'hôte

Consulter d'abord les requêtes possibles d'un hôte.

- **PLIN MatchName Username Password** : Si Username et Password

sont valides, que le Player correspondant est dans *available*, et que d'après *inGame* il y a au moins de 10 joueurs pour le HostData correspondant :
    - **PLNO Username**.
    - Sinon, retire *Player* de *available*. Renvoie **PLOK Username nbPoints**. Ferme la connexion correspondante avec le client. Ferme la connexion avec l'hôte.
1. **SCPS Username totalPoints** : Retire de *inGame* l'entrée (Player/HostData) correspondante. Met à jour le Player qui a pour nom Username. Met à jour le fichier XML.
2. **ENDS matchName** : Retire de *hostsData* celui qui contient *matchName*.

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
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 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.

À tout moment, s'il reçoit *RQDT* du serveur, il répond **SDDT 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 **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.

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 *InGamePlayer.active* est vrai, **JNNO** parce qu'il est déjà en train de jouer.
- Autrement : Reconnexion, il renvoie **JNOK nbLignes** au Client, et enchaîne avec **BDIT numLigne Case1 … Case30** où *numLigne* vaut de 0 à 15, et *CaseX* vaut :
  - Si discovered vaut false, 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*. Il envoie de plus à chaque autre InGamePlayer le profil du nouvel arrivant : **CONN Username inGamePoints totalPoints safeSquares foundMines**.
- Autrement, Host envoie alors à Server **PLIN MatchName Username Password**. S'il reçoit :
  - *PLNO*, il renvoie **JNNO** avec éventuellement un message d'erreur reçu du *PLNO* du serveur.
  - *PLOK Username Password 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 d'un hôte, 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*.

L'hôte *Host* doit pouvoir repérer, à sa façon mais sans communication réseau, quand un joueur est inactif, et mettre InGamePlayer.active à false. Il envoie alors à tous les *InGamePlayer* : **DECO Username**. Dans ce cas, le multiplicateur de points par case est décrémenté.

La liste *inGamePlayers* n'est **pas modifiée**.

Si tous les *inGamePlayers* sont marqués inactifs, alors un décompte est lancé, et après 60, 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 :

- La case a déjà été découverte : **LATE**
- Les coordonnées sont invalides : **OORG wrongAbscissa wrongOrdinate**
- 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 [...]* pour chaque case.

La fin de partie survient dès qu'il n'y a plus de bonne case cliquable, soit lorsque [*totalDiscovered - nbMinesDécouvertes]* est égal à [total cases - nbMines].

Fin de partie

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

1. Au serveur : **SCPS Username totalPoints**. Répète pour chaque InGamePlayer.
2. Au serveur : **ENDS matchName**.
3. À tous les *InGamePlayer* : **ENDC nbPlayers**.
4. 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.

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

Duo Tempus Finis

Tomek Lecocq

Kévin Khau

Duo Milites

Aylin Kocoglu

Adil Champion

Protocoles et Services Internet - Projet

Duo Caetuna

Valloris Cylly

Somaya
Ben Khemis

Duo Lorica

Christophe Lam

Al Hassane
Megninta

Héros Machina

Helmi Zegaya