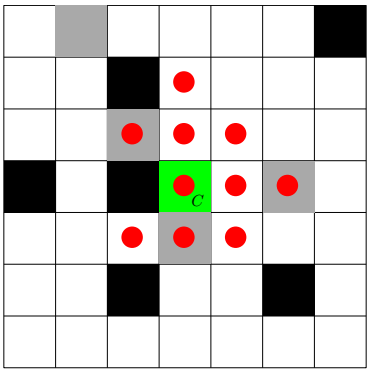
PR6 – Programmation réseaux
Projet Bomberman

Le but de ce projet est d'implémenter un serveur et un client pour un jeu en réseau de *Bomberman* décrit ci-après.

I) Le protocole *Bomberman*

Une partie de *Bomberman* se joue à quatre joueurs sur une grille. La grille comporte des cases libres sur lesquelles les joueurs peuvent se déplacer et des murs infranchissables. Chaque joueur peut à tout moment poser une bombe sur une case libre et se déplacer dans une des quatre directions cardinales.
Il y a deux types de murs, les murs qui peuvent être détruits par une bombe et ceux qui sont indestructibles. Lorsqu'une bombe est posée, elle explose 3 secondes plus tard en éliminant les murs destructibles et les joueurs qui se trouvent à proximité. Pour une case C de coordonnées (i, j), les cases à proximité sont les cases de coordonnées $(i, j \mp k)$, $(i \mp k, j)$, $(i \mp 1, j \mp 1)$, $0 \leqslant k \leqslant 2$, sans mur sur la ligne entre C et la case. Dans la figure ci-dessous, les murs indestructibles sont en noir, ceux destructibles en gris et les cases touchées par l'explosion d'une bombe située dans la case C, sont marquées d'un point rouge.



Notez, qu'une bombe élimine indifféremment et donc que le poseur de bombe doit se protéger au plus vite après avoir déposé sa bombe.
La partie peut être composée soit de quatre adversaires, soit de deux équipes adverses, formée chacune de 2 joueurs.
Au début de la partie, les quatre joueurs sont disposés chacun à un angle de la grille. Si le jeu se joue en équipe, alors les joueurs opposés diagonalement forment une équipe.
Le but du jeu est d'éliminer ses adversaires.
Le serveur devra pouvoir gérer plusieurs parties en parallèle.
Dans un premier temps, les échanges entre les clients et le serveur sont décrits de façon globale. Le format exact des messages du client et du serveur est ensuite décrit précisément.

a) Les échanges client/serveur

Le client représente un joueur. Quand un message comporte un nombre, celui-ci doit toujours être au format *big endian*.

intégrer et démarrer une partie Pour intégrer une partie un joueur se connecte au serveur en mode TCP, puis les échanges entre le client et le serveur sont les suivants :

1. le client commence par dire au serveur s'il souhaite intégrer une partie en mode *4 adversaires* ou en mode *équipes*,
2. le serveur répond en envoyant plusieurs données au joueur.

   Il affecte un identifiant au joueur et lui attribut un numéro d'équipe (0 ou 1) le cas échéant.

   Si aucune partie dans le mode demandé n'est en attente, il crée une nouvelle partie avec une adresse IPv6 et un numéro de port de multidiffusion auxquels les joueurs devront s'abonner, ainsi qu'un numéro de port sur lequel le serveur attendra les messages UDP des joueurs de la partie.

   Si une partie dans le mode demandé est en attente de joueurs, il intègre le joueur à cette partie.

   Il envoie ensuite toutes ces données au joueur.
3. après avoir reçu ces données, le joueur s'abonne à l'adresse de multidiffusion, puis dit au serveur qu'il est prêt à jouer,
4. le serveur lance le début de la partie lorsque 4 joueurs ont intégré la partie et se sont déclarés prêts à jouer. Pour cela, il multidiffuse la grille de jeu initiale, celle où les 4 joueurs sont disposés aux 4 coins de la grille.

La connexion TCP est gardée ouverte pendant toute la partie afin de permettre le tchat et le signalement de la fin de la partie.

déroulement d'une partie À tout moment pendant la partie, le joueur envoie en UDP au serveur la prochaine action qu'il souhaite faire. Cette action peut être :

- le déplacement d'une case vers le nord, le sud, l'est ou l'ouest relativement à la case courante du joueur,
- le dépôt d'une bombe sur la case courante du joueur.

Le serveur actualise la grille du jeu en fonction des demandes d'action qu'il reçoit des joueurs. Bien sûr, si une demande est impossible, cette demande est ignorée par le serveur (par exemple si le déplacement est sur une case avec un mur ou fait sortir de la grille).
Vous avez à programmer une application temps réel. Cela signifie que le serveur doit adopter un comportement adapté pour traiter les demandes du joueur. À chaque fois qu'il passe en revue les demandes qu'il a reçues d'un joueur, il ne prend en compte que la dernière demande de déplacement, ainsi qu'une seule demande de dépôt de bombe s'il y en a au moins une. Le client doit donc numéroter ses requêtes et deux requêtes de numéros respectifs n_1 et n_2 se succèdent s'il n'y a pas de requête avec un numéro n tel que $n_1 < n < n_2$ ou $n_2 < n < n_1$. Par exemple, si le serveur traite les demandes d'un joueur suivantes, ordonnées suivant leur ancienneté :
`(1,nord), (2,nord), (4,nord), (7,bombe), (9,bombe), (10, ouest)`
le serveur prend en compte un déplacement vers l'ouest avec dépôt de bombe sur la case d'arrivée.
La fréquence `freq` de l'examen des requêtes des joueurs est un paramètre du serveur.
Il multidiffuse aux joueurs d'une même partie :

- toutes les secondes : la grille complète
- toutes les `freq` ms : le différentiel sur la grille entre le moment présent et la dernière multidiffusion. Il faut donc choisir un entier diviseur de 1000 pour `freq`.

Lorsqu'un joueur reçoit une actualisation de la grille, il doit :

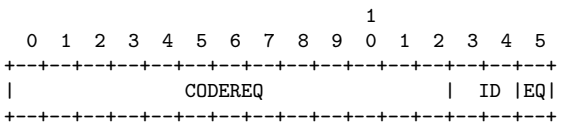
- rafraîchir son affichage du jeu,
- ne plus envoyer d'action au serveur s'il est éliminé. Il doit par contre, pouvoir suivre la fin de la partie. Il peut aussi choisir de quitter la partie.

le tchat Au cours d'une partie, un joueur doit pouvoir envoyer un message aux autres joueurs ou à son co-équipier. Pour cela, il envoie au serveur son message en mode TCP et le serveur le transfert ensuite sur ses connexions TCP à chaque joueur concerné.

la fin de partie Lorsqu'il ne reste plus qu'un joueur ou qu'une équipe, la partie est terminée et le serveur envoie l'identifiant du gagnant à tous les joueurs de la partie sur ses connexions TCP, puis ferme ses connexions liées à la partie.

b) Les messages du client

intégrer et démarrer une partie Le client envoie sa demande pour intégrer une partie ou annonce au serveur qu'il est prêt à jouer en envoyant le message suivant en TCP :



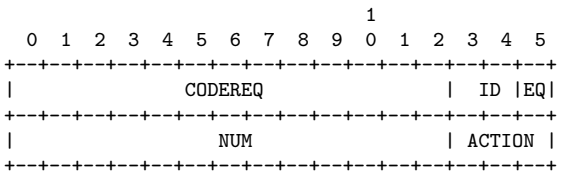
Les deux octets sont au format big endian.
Lorsque le client envoie sa demande pour intégrer une partie :

- `CODEREQ` vaut 1 si la partie est en mode *4 adversaires*, 2 si elle est en mode *équipes*,
- les champs `ID` et `EQ` sont à ce stade sans importance puisqu'ignorés par le serveur et peuvent donc prendre n'importe quelle valeur.

Lorsque le client annonce au serveur qu'il est prêt à jouer :

- `CODEREQ` vaut 3 si la partie est en mode *4 adversaires*, 4 si elle est en mode *équipes*,
- `ID` est une valeur entre 0 et 3 correspondant à l'identifiant du joueur. Les joueurs d'une même partie ont des identifiants distincts,
- `EQ` vaut 0 ou 1 et correspond au numéro de l'équipe du joueur si `CODEREQ` vaut 2. Ce champ est ignoré sinon.

déroulement d'une partie Les messages du client sont en UDP et ont tous la forme suivante :



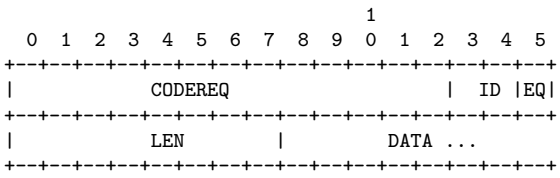
où

- `CODEREQ` vaut 5 si la partie est en mode *4 adversaires*, 6 si elle est en mode *équipes*,
- `ID` est l'identifiant du joueur,
- `EQ` numéro de l'équipe du joueur si `CODEREQ` vaut 6. Ce champs est ignoré sinon.
- `NUM` est le numéro du message modulo 2^{13},
- `ACTION` est un nombre correspondant à l'action souhaitée :
  - 0 pour un déplacement d'une case vers le nord,
  - 1 pour un déplacement d'une case vers l'est,
  - 2 pour un déplacement d'une case vers le sud,
  - 3 pour un déplacement d'une case vers l'ouest,

- 4 pour le dépôt d'une bombe,
- 5 pour annuler la dernière demande de déplacement.

Les deux premiers octets et les 2 suivants sont chacun au format big endian.
Les messages du client sont numérotés à partir de 0 afin que le serveur puisse savoir quel est le message le plus récent reçu du client. Faites attention au passage de 8191 à 0.

le tchat Un joueur peut envoyer des messages à ses adversaires ou co-équipier via le serveur. Les messages sont envoyés en TCP au serveur et ont le format suivant :



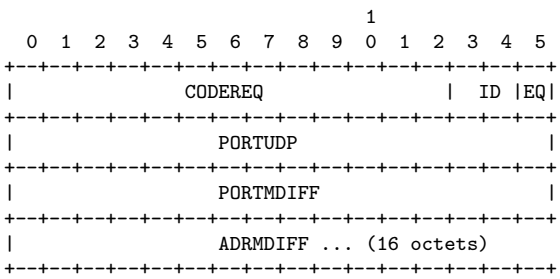
où

- `CODEREQ` vaut 7 si le message est destiné à tous les joueurs de la partie, 8 si c'est uniquement pour le co-équipier (si cela a du sens),
- `ID` est l'identifiant du joueur auteur du message,
- `EQ` est le numéro d'équipe du joueur auteur du message (si cela a du sens),
- `LEN` est un entier représentant le nombre de caractères du texte à transmettre,
- `DATA` est le texte à transmettre.

Les deux premiers octets sont au format big endian.

c) Les messages du serveur

intégrer et démarrer une partie Lorsque le client se connecte au serveur, ce dernier l'intègre à une partie en attente ou crée une nouvelle partie s'il n'y pas de partie en attente. Le message TCP envoyé alors au client a la forme suivante :



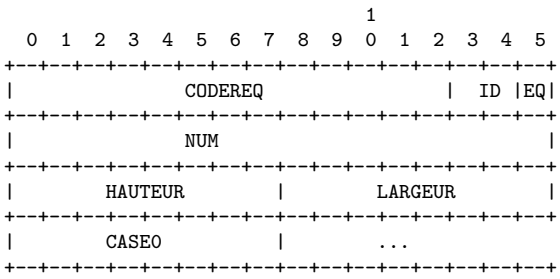
où

- `CODEREQ` vaut 9 si le joueur demande à intégrer une partie en mode *4 adversaires*, 10 en mode *équipes*.
- `ID` est l'identifiant attribué par le serveur au joueur. Cet identifiant vaut entre 0 et 3 et permet de distinguer les joueurs d'une même partie,
- `EQ` est le numéro d'équipe attribué par le serveur au joueur. Ce numéro vaut entre 0 et 1 et permet de distinguer les équipes d'une même partie,
- `PORTUDP` est le numéro de port sur lequel le serveur attend les messages UDP des joueurs de la partie,

- `PORTMDIFF` est le numéro de port sur lequel le serveur multidiffusera ses messages aux joueurs de la partie,
- `ADRMDIFF` est l'adresse IPv6 de multidiffusion sous forme d'entier à laquelle les joueurs doivent s'abonner.

Les deux premiers octets et les champs `PORTUDP`, `PORTMDIFF` et `ADRMDIFF` doivent être chacun au format big endian.
La première multidiffusion de la grille complète à la même forme que les multidiffusions suivantes et est décrite dans le paragraphe suivant.

déroulement d'une partie Le serveur multidiffuse toutes les secondes la grille complète au format suivant :



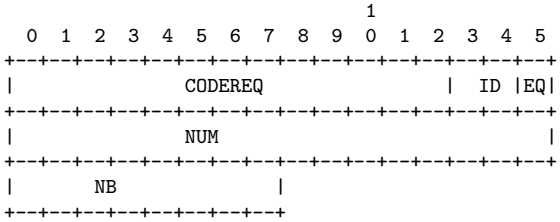
où

- `CODEREQ` vaut 11,
- `ID` et `EQ` valent 0,
- `NUM` est le numéro du message modulo 2^{16}. Il numérote les messages multidiffusé toutes les secondes,
- `HAUTEUR` et `LARGEUR` sont la hauteur et largeur de la grille,
- ensuite chaque case de la grille est encodée sur 1 octet et peut prendre les valeurs suivantes :
  - 0 si la case est vide,
  - 1 si la case est un mur indestructible,
  - 2 si la case est un mur destructible,
  - 3 si la case contient une bombe,
  - 4 si la case est explosée par une bombe,
  - $5 + i$ si la case contient le joueur d'identifiant i, $0 \leqslant i \leqslant 3$.

  Les cases sont listées de gauche à droite, puis de haut en bas.

Les deux premiers octets et le champ `NUM`, sont chacun au format big endian.

Le serveur multidiffuse toutes les `freq` ms les cases modifiées depuis la dernière multidiffusion. Chaque message commence par l'entête suivant :

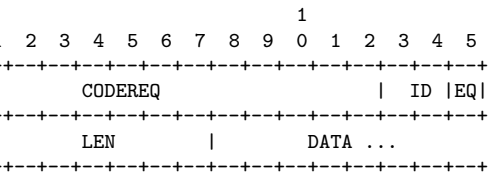


où

- `CODEREQ` vaut 12,
- `ID` et `EQ` valent 0,
- `NUM` est le numéro du message modulo 2^{16}. Il numérote les messages multidiffusés toutes les `freq` ms,
- `NB` est le nombre de cases transmises.

Les deux premiers octets et le champs `NUM`, sont chacun au format big endian.
Puis chaque case transmise est encodée sur 3 octets, le 1er octet correspondant à son numéro de ligne, le 2ème octet à son numéro de colonne et le dernier octet au contenu de la case comme explicité dans le format de message précédent.

le tchat Le serveur doit retransmettre aux joueurs de la partie les messages de tchat reçus d'un des joueurs. Les messages retransmis ont le format suivant :

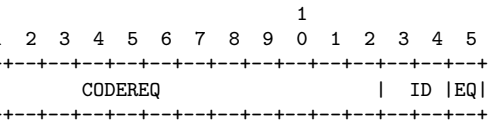


où

- `CODEREQ` vaut 13 si le message est destiné à tous les joueurs et 14 si c'est uniquement pour le co-équipier (si cela a du sens),
- `ID` est l'identifiant du joueur auteur du message,
- `EQ` est le numéro d'équipe du joueur auteur du message (si cela a du sens),
- `LEN` est un entier représentant le nombre de caractères du texte à transmettre,
- `DATA` est le texte à transmettre.

Les deux premiers octets sont au format big endian.

la fin de partie Lorsque le serveur détecte une fin de partie, il envoie à chaque joueur, sur la connexion TCP, le message au format suivant :



où

- `CODEREQ` vaut 15 si la partie est en mode *4 adversaires*, 16 si elle est en mode *équipes*,
- `ID` est l'identifiant du joueur gagnant si `CODEREQ` vaut 15, et ignoré sinon,
- `EQ` est le numéro de l'équipe gagnante si `CODEREQ` vaut 16, et ignoré sinon.

Les deux octets sont au format big endian.

d) Les paramètres du serveur

Le serveur peut décider de la valeur de certains paramètres au démarrage. Cela concerne la taille de la grille, le temps entre le dépôt d'une bombe et son explosion (proposé à 3s par défaut), la fréquence de multidiffusion de la grille complète (proposée à 1s par défaut) et la fréquence de traitement des requêtes (qui est également celle de multidiffusion du différentiel sur la grille).

II) La bibliothèque graphique `ncurses`

La bibliothèque `C ncurses` permet de réaliser des affichages graphiques dans le terminal. Elle est assez simple d'utilisation.
Si vous l'utilisez, cela vous permettra de mieux voir ce qu'il se passe et de résoudre plus rapidement les bugs. Mais la partie graphique ne sera pas notée. Ne passez pas du temps dessus !
Le répertoire `ncurses` contient un exemple d'utilisation que vous avez le droit de plagier.

III) Le travail demandé

a) Réalisation

Le travail est à réaliser en trinôme. Il y a beaucoup à faire donc un binôme court le risque d'être débordé. Il n'y aura pas de dérogation à cette règle sauf exception qui doit être argumentée. Vous devez envoyer par mail à `anne.micheli@irif.fr` au plus tard le 31 mars 2023, la composition de votre équipe avec les autres membres de l'équipe en copie du mail. L'objet du mail doit être **[PR6] équipe projet**.
Vous devez écrire en C le serveur et le client correspondant au protocole *Bomberman*. Votre serveur devra fonctionner sur IPv6. Il devra, également, accepter les connexions et messages sur IPv4.
Les erreurs devront être gérées, c'est-à-dire qu'il faut réfléchir aux actions en cas de réception d'un message mal formaté ou d'un appel système erroné.
Il faudra également veiller à ce que vos applications ne bloquent pas éternellement. Pensez par exemple au cas du client qui intègre une partie mais ne se déclare jamais prêt. Cela signifie que vos applications doivent décider à un moment, qu'un client ou le serveur n'est plus actif si cela fait trop longtemps qu'elles attendent un message.
La propreté du code, sa lisibilité seront également pris en compte. Pensez à commenter votre code et à un mode verbeux pour la soutenance. Evitez le code spaghetti, mutualisez le code...
Vous pouvez discuter entre vous du fonctionnement du protocole, du format des messages... mais vous ne pouvez pas vous échanger ou montrer du code. Le plagiat est **strictement interdit**.

b) Pour aller plus loin

En plus du projet minimal décrit ci-dessus, toute amélioration utilisant les notions du cours sera prise positivement. Mais attention à tout d'abord terminer la partie obligatoire. Par ailleurs, l'ajout ne doit pas avoir d'impact sur la partie minimale.
Vous pourrez, par exemple, travailler une des extensions suivantes :

- des joueurs polymorphes,
- messages d'erreur ciblés du serveur lorsque la requête du client est mal formatée,
- le serveur envoie périodiquement la quantité de bombes restantes et le nombre de vies restantes (le protocole proposé considère que chaque joueur a une vie et un nombre illimité de bombes).

Cette liste n'est pas exhaustive. Vous pourrez proposer de nouvelles fonctionalités en les décrivant précisément (notamment le format des messages échangés) dans un fichier `extension.md`.

c) Modalités de rendu

Dès que votre trinôme est constitué, vous devez créer un dépôt git **privé** sur le gitlab de l'UFR [1].
Tous les enseignants du cours devront y avoir accès en tant que **Reporter**, c'est-à-dire, Roman

1. https ://moule.informatique.univ-paris-diderot.fr/

Kniazev, Fabien de Montgolfier, Aldric Degorre, Mickaël Laurent et Anne Micheli. Le dépôt devra contenir un fichier `authors.md` contenant la liste des membres de l'équipe (nom, prénom, numéro étudiant et pseudo(s) sur le gitlab).
Votre dépôt final devra contenir vos fichiers source ainsi qu'un `Makefile` pour la compilation et le fichier `extension.md` le cas échéant. La **compilation et l'exécution devront fonctionner sur les machines de l'UFR**.
Ce rendu donnera lieu à une soutenance qui se déroulera en fin de période d'examens. Des précisions seront apportées ultérieurement.
Attention, les soumissions (commits) sur le gitlab rendront compte en partie de votre implication dans le projet. Si, par exemple, nous découvrons le jour de la soutenance en regardant sur le gitlab qu'une personne n'a pas participé au projet, sa note sera 0 au projet.

d) Serveur Discord

Un serveur discord a été mis en place afin que vous puissiez discuter entre vous, nous poser des questions pour clarifier le sujet... Vous trouverez le lien sur Moodle. Seules les réponses données par les enseignants sur cette liste feront foi. Il faut donc que vous posiez les questions concernant le projet sur cette liste.
Vous pourrez également annoncer que votre serveur tourne en précisant son adresse et port, afin que d'autres groupes puissent le tester avec leur client. De plus, vous pourrez préciser les extensions ajoutées avec le format pour que des clients puissent les tester. Il est très important que vos applications interagissent avec d'autres implémentations, donc pensez *a minima* à faire des tests inter-groupes.

e) Soutenances

Les soutenances se dérouleront les semaines du 20 et 27 mai 2024 en 2027.
Il faudra donc que vous veniez avec un **ordinateur**. Si cela n'est pas possible, nous devons absolument être prévenus quelques jours avants.
Vous devrez nous faire une démonstration de votre projet qui devra tourner sur le réseau de l'UFR. Par exemple votre serveur tournera sur `lulu` et vos clients sur d'autres machines de l'UFR. Il est donc impératif que votre ordinateur pour la soutenance puisse **se connecter au réseau de l'UFR**.