Projet de Réseaux - Master 1 Informatique

Cyrille d'Halluin - Jean-Marie Lagnier - Arthur Marcinkowski

Université d'Artois - Faculté des Sciences Jean Perrin

Introduction

Ce projet a pour but de créer un serveur de jeu proposant à ses utilisateurs de jouer au jeu "Bomber Man". Le projet consiste à développer d'une part le serveur de jeu lui-même et d'autre part le client permettant à des utilisateurs de se connecter au serveur afin de jouer les uns contre les autres.

Informations générales

- Le projet se fera obligatoirement en binôme, la constitution de trinôme sera exceptionnelle et après accord de votre enseignant.
- La partie serveur devra être écrite en C (ou en C++). La partie cliente sera écrite dans un autre langage au choix du binôme
- Il n'est absolument pas obligatoire d'intégrer des graphismes sophistiqués dans la partie cliente. Il est tout à fait possible de développer le jeu avec des graphismes simples voire avec des caractères.
- Le projet sera mis à disposition sous Git. N'oubliez pas d'ajouter vos enseignants de cours et TP en tant que reporter.

Vous présenterez votre projet dans la semaine du **18 au 22 décembre 2023**. Cette présentation d'une quinzaine de minutes sera faite en binôme et suivant un planning qui vous sera communiqué ultérieurement. Il conviendra d'installer quelques minutes avant votre passage le client et le serveur sur l'une des machines de la salle P309. Client et serveur devront évidemment s'exécuter sans problème sur ces machines. A l'issue de cette présentation, votre projet **disponible et accessible sous git** sera considéré comme rendu.

Qu'attendons-nous de vous ?

Nous serons particulièrement attentifs aux points suivants :

- Qualité du code (commentaires, organisation, etc...)
- Choix des protocoles réseaux utilisés en fonction des besoins
- Qualité du protocole de communication (robustesse, fiabilité,...) Par exemple : que se passe-t-il si le serveur tombe ?
- Facilité d'utilisation, compréhension du jeu pour un utilisateur lambda

Présentation du jeu Bomber Student

Dans le jeu "Bomber Student", chaque joueur incarne un poseur de bombes ayant pour mission de faire exploser les adversaires. Le bomber student est un petit personnage très doué pour causer des catastrophes en tout genre et ce qu'il préfère plus que tout, c'est de faire exploser ses

camarades. Heureusement pour eux, les dommages ne sont pas définitifs puisque les adversaires sont juste étourdis. Ainsi le bomber student touché pourra prendre sa revanche ou pas ...

Il n'y a, a priori, rien d'imposer sur le nombre limite de joueurs pouvant jouer ensemble sur une même partie. Vous devez cependant être capable de gérer au minimum 4 joueurs simultanés

Carte du jeu (map)

Une partie se déroule sur une carte (map) composée de cases. Chaque map est caractérisée par sa taille (nombre de cases par ligne et par colonne) et par la nature de chacune de ces cases.

Il existe trois types différents de cases :

- les **cases libres** : cases permettant aux différents joueurs de circuler. Ces cases peuvent également accueillir toutes sortes d'objets (bombes désactivés, bonus ou malus). Enfin, lorsqu'une explosion a lieu sur ou près d'une case libre, celle-ci est "traversée" par l'explosion (dans la limite de la distance d'impact de la bombe)
- les **murs** : cases contenant un mur infranchissable par les joueurs. Sous cette forme, ces cases ne peuvent en aucun cas accueillir d'objet. Lorsqu'elle est touchée par une explosion, une case mur devient une case libre et en prend alors toutes les propriétés. Après destruction par les effet d'une bombe, un mur peut laisser apparaître un objet (bombe désactivée, bonus ou malus).
- les **murs blindées** : un mur blindé est une case infranchissable par les joueurs et incassable par les explosions.

Vous êtes libres de créer autant de maps différentes que vous voulez (il faudra quoi qu'il en soit en créer au moins une sinon le jeu va très vite perdre de son intérêt). Même si vous décidez de ne créer qu'une seule map, votre client et votre serveur devront être capables d'en gérer plusieurs.

Les bombes

Il existe trois types de bombes :

- **Classic bomb** : bombe explosant simultanément dans les quatre directions (haut, bas, gauche et droite) avec une distance (par défaut) de deux cases dans chaque sens. Elle explose 4 secondes après avoir été posée (ce délai peut éventuellement être paramétrable dans la configuration du jeu). La gestion du temps de la bombe n'est pas prioritaire. Il n'y a donc pour le moment pas de protocole pour le modifier.
- **Remote bomb** : bombe pouvant être actionnée à distance lorsque le joueur le souhaite ; l'explosion engendrée est identique à celle provoquée par une classic bomb.
- **Mine** : bombe qui, une fois posée, est invisible pour les autres joueurs. Dès qu'un joueur (éventuellement celui qui l'a posée) marche dessus, cette dernière explose. Elle peut également exploser si elle se trouve dans le champs d'explosion d'une classic bomb ou remote bomb.

Notons qu'un mur blindé n'est jamais détruit par une bombe et qu'il bloque les effet de l'explosion. La destruction d'un mur classique par une bombe compte pour 1 au niveau du calcul de la distance d'impact.

Lorsqu'une bombe touche un joueur (bomber student) celui-ci est figé pendant quelques secondes (à définir par vos soins) et perd quelques points de vie (voir ci-dessous pour le calcul des pertes de points de vie).

Bomber Student

Les Bomber students sont les personnages incarnés par les joueurs. Le but de chacun de ceux-ci est d'éliminer les adversaires en réduisant à néant leurs points de vie. Un bomber student dont le nombre de points de vie est de 0 est mort ; il disparaît alors du jeu.

Chaque bomber student est caractérisé par les éléments listé dans le tableau suivant. Nous indiquons également pour chaque élément sa valeur initiale à l'arrivée du joueur :

Caractéristique	**Valeur initiale**
Points de vie	100
Vitesse	75 pixel/s
Distance d'impact	2 cases
Nombre de classic bombs disponibles	2
Nombre de remote bomb disponibles	1
Nombre de mines disponibles	0
Invincible	faux

Chacune de ces valeurs peut évoluer au cours du jeu, par exemple en utilisant les bombes disponibles ou encore en ramassant des objets comme des bombes désactivées ou des bonus-malus.

Note : les classic bombs et les remote bombs ne sont pas traversables par un joueur (elles agissent comme un mur).

Bonus et malus

Il existe un certain nombre de bonus et malus ayant des effets divers et variés sur les caractéristiques de nos personnages ou sur les bombes que ceux-ci manipulent.

- **Impact up** : bonus permettant d'augmenter d'une case la distance d'explosion des bombes posées (avec une distance maximale de 4 cases),
- **Impact down** : malus ayant pour effet de diminuer d'une case la distance d'explosion des bombes posées (avec une distance minimale de 0 cases, c'est-à-dire que la bombe n'a d'effet que sur la case qui la contient),
- **Speed up** : bonus permettant d'augmenter la vitesse de déplacement du bomber student (à vous de définir dans quelle proportion),
- **Speed down** : malus ayant pour effet de diminuer la vitesse de déplacement du bomber student (à vous de définir dans quelle proportion),
- **Invincible** : permet au bomber student d'être invincible pendant une dizaine de secondes,
- **Life max** : réinitialiser le nombre de points de vie à son maximum

Récupérer des bombes, des bonus ou des malus

Pour récupérer une bombe (quelque soit son type), un bonus (ou un malus), il suffit au joueur de passer sur une case contenant l'un de ces objets. Ces objets apparaissent de manière "aléatoire" à la suite de l'explosion d'un mur cassable. Vous êtes libres de provoquer l'apparition de ces objets et bombes comme vous le souhaitez en attribuant par exemple une plus forte probabilité d'apparition à certains éléments. **Pensez surtout à la jouabilité du jeu de manière à bien doser l'apparition de ces éléments.**

Lancement du jeu

Votre projet s'appuie sur une architecture client/serveur, c'est-à-dire que chaque client se connecte à un serveur BomberStudent et échange avec lui suivant un protocole bien défini. Ce protocole décrit le déroulement d'une partie ainsi que les échanges entre client et serveur. Chaque serveur peut évidemment accueillir plusieurs clients. Il est tout à fait possible que plusieurs serveurs de jeu tournent simultanément : chaque client a alors le choix de se connecter au serveur qu'il désire. Lors de son lancement, le client va dans un premier temps rechercher les serveurs BomberStudent disponibles. L'utilisateur choisira alors le serveur sur lequel il veut jouer. Une fois connecté à un serveur, le client demande alors la liste et la constitution des maps existantes.

Après que le serveur lui ait envoyé cette liste de maps, le client demande la liste des parties disponibles sur celui-ci. A cette requête, le serveur retournera un message contenant un code de retour, le nombre de parties existantes ainsi qu'une liste détaillant les informations pour chaque partie. Le joueur peut décider d'intégrer une partie existante (on dit qu'il rejoint la partie) ou de créer lui-même une partie (qu'il rejoint alors immédiatement). Dans ce dernier cas, il devra définir un nom de partie et choisir la map sur laquelle va se dérouler la partie.

Données échangées entre client et serveur / Protocole

Remarque générale

Dans cette partie, pour des raisons de clarté, le JSON est formaté correctement avec des retours à la ligne. Ces retours à la ligne ne sont pas nécessaires lors de vos communications. Pour le traitement du JSON, vous pouvez, si vous le souhaitez, utiliser une librairie en C.

Nous listons ci-dessous les données échangées entre client et serveur en fonction des différents besoins. Nous attirons votre attention sur le fait que chaque client devrait normalement être utilisable avec n'importe quel serveur développé par un autre binôme. Il convient pour cela d'appliquer scrupuleusement le protocole défini ici et de traiter convenablement les éventuels messages ou réponses non définis.

Cas d'erreurs

En cas d'erreur de syntaxe (en TCP), le serveur répondra :

```
{
   "statut":"400",
   "message": "Bad request"
}
```

En cas d'erreur inattendue ou non connue du système :

```
{
    "statut":"520",
    "message": "Unknown Error"
}
```

Définition d'une position

Toutes le positions seront traitées et transmises sous la forme d'une couple (x,y) avec x le numéro de colonne et y le numéro de ligne. la case d'origine est considéré être la case située dans le coin inférieur gauche.

Recherche d'un serveur

Message envoyé sur le réseau par le client en quête d'un réseau :

looking for bomberstudent servers

Message retourné par un serveur en réponse à la requête précédente :

hello i'm a bomberstudent server

Si le serveur reçoit un message ne correspondant pas à celui attendu, il ne répondra pas.

Récupération de la liste des maps

Requête envoyée par le client :

GET maps/list

Réponse retournée par le serveur

```
{
    "action":"maps/list", "statut":"200", "message":"ok",
    "nbMapsList": 1,
    "maps":[
        {
          "id":0,
          "width":24,
          "height":8,
          "content:" ************************
                     =----------------------=
                     =----==============----=
                     =----------****--------=
                     =------****------------=
                     =----==============----=
                     =----------------------=
                     ************************"
        }
    ]
}
```

avec :

- **action** : action appelée par le client
- **statut** : code de retour du serveur
- **message** : précisant le retour (ok si tout va bien)
- **nbMapsList** : nombre de maps connues et listées après
- **maps** : liste de toutes les descriptions de maps
- **id** : id de la map concernée
- **width** : largeur de la map en nombre de cases
- **height** : hauteur de la map en nombre de cases
- **content** : descriptif de la map

La map est décrite au moyen des caractères suivants :

= correspond à un mur cassable par une explosion de bombe

* correspond à un mur incassable

- correspond à une case libre

Récupération d'une liste de parties

Requête envoyée par le client :

GET game/list

Réponse retournée par le serveur :

```
{
   "action":"game/list", "statut":"200", "message":"ok", "nbGamesList": 2,
   "games":[ {
          "name":"game1",
          "nbPlayer":2,
          "mapId":0,
    },
    {
        "name":"game2",
        "nbPlayer":2,
        "mapId":1
    }],
}
```

avec :

- **action** : action appelée par le client
- **statut** : code de retour du serveur
- **message** : précisant le retour (ok si tout va bien)
- **nbGamesList** : nombre de parties en cours
- **games** : liste des parties
- **name** : nom de la partie
- **nbPlayer** : nombre de joueurs actuel dans la partie
- **mapId** : identifiant de la map sur laquelle se joue la partie

S'il n'existe pas de partie en cours sur le serveur, celui retourne :

```
{
    "action":"game/list", "statut":"200", "message":"ok",
    "nbGamesList": 0,
}
```

Création d'une partie

Message envoyé par le client

```
POST game/create
{
  "name":"game1",
  "mapId": 1
}
```

Réponse du serveur :

```
{
   "action":"game/create", "statut":"201", "message":"game created",
        "nbPlayers":0,
        "mapId" : 1,
        "startPos":"3,2", "player":{
            "life":100,
            "speed":1,
            "nbClassicBomb":2,
            "nbMine":0,
            "nbRemoteBomb":1,
            "impactDist":2,
            "invincible":false,
      }
}
```

- **action** : action appelée par le client
- **statut** : code de retour du serveur
- **message** : précisant le retour (ok si tout va bien)
- **nbPlayer** : nombre de joueurs actuel dans la partie
- **mapId** : identifiant de la map sur laquelle se joue la partie
- **startPos"** : position initiale d'un nouveau joueur,
- **player** : objet JSON décrivant les caractéristiques d'un nouveau joueur
- **life** : nombre de points de vie du joueur
- **speed** : vitesse du joueur (1 correspond à la vitesse de base)
- **nbClassicBomb** : nombre de classic bombs possédées par le joueur
- **nbMine** : nombre de mines possédées par le joueur
- **nbRemoteBomb** : nombre de remote bombs possédées par le joueur
- **impactDist** : nombre de cases impactées par une explosion (pour une direction donnée)
- **invincible** : indique si le joueur est en mode invincible

En cas de problème le serveur répond :

```
{
    "action":"game/create", "statut":"501", "message":"cannot create game"
}
```

Rejoindre une partie

Pour rejoindre une partie, le client envoie la requête suivante :

```
POST game/join
{
    "name":"game"
}
```

avec **name** définissant le nom de la partie que le joueur souhaite rejoindre.
Le serveur répond :

```
{
    "action":"game/join", "statut":"201", "message":"game joined",
    "nbPlayers":2,
    "mapId" : 1,
    "players":[{"name":"player1","pos":"0,0"},
               {"name":"player2","pos":"0,79"},],
    "startPos":"5,3",
    "player":{
        "life":100,
        "speed":1,
        "nbClassicBomb":1,
        "nbMine":0,
        "nbRemoteBomb":0,
        "impactDist":2,
        "invincible":false,
     }
}
```

Nous ne décrivons pas à nouveau ici l'ensemble des champs puisqu'ils sont identiques à ceux décrits ci-dessus. Notons simplement que **players** liste les noms et positions des autres joueurs et que **startPos** définit la position initiale du nouveau joueur rejoignant la partie. Chaque joueur commence à une position différente décidée aléatoirement par le serveur.

Dans le cas d'un problème pour rejoindre cette partie, le serveur répond :

```
{
    "action":"game/join", "statut":"501", "message":"cannot join the game"
}
```

Déplacement des joueurs

Lorsqu'un joueur se déplace, le client envoie le message suivant au serveur :

```
POST player/move
{
    "move":"up" // down, left, right
}
```

Suite à ce message le serveur vérifie si le joueur n'a pas marché sur une case minée.

Si le déplacement a réussi, le serveur envoie alors le message suivant à tous les clients :

```
POST player/position/update
{
    "player":"player2",
    "dir":"up" }
```

Les clients se chargeront alors de déplacer visuellement les joueurs. Vous pouvez dans un premier temps faire du déplacement case par case. Dans une phase d'amélioration de votre projet, vous pourrez ensuite trouver une solution pour réaliser un déplacement plus fluide.

Poser des bombes

Rappel : les bombes ne peuvent être posées que sur des cases libres.

Lorsqu'un joueur souhaite poser une bombe, le client envoie la requête suivante :

```
POST attack/bomb
{
    "pos":"5,3",
    "type":"classic" //mine, remote
}
```

Le champs **pos** correspond à la position du joueur, le champs **type** correspond au type de bombe placée (classic, remote, mine).

Le serveur vérifiera si l'action est possible :

- la case peut-elle recevoir cette bombe
- le joueur possède-t-il au moins une bombe du type souhaité

En retour de la requête le serveur répondra au client demandeur :

```
{
    "action":"attack/bomb", "statut":"201", "message":"bomb is armed at pos 5,3"
    "player":{
        "life":100,
        "speed":1,
        "nbClassicBomb":1,
        "nbMine":0,
        "nbRemoteBomb":0,
        "impactDist":2,
        "invincible":false,
    }
}
```

Ce message informe le client des nouvelles caractéristiques du joueur (le nombre de bombes disponibles est mis à jour par exemple).

Le serveur envoie également le message suivant à tous les autres clients de manière à leur indiquer le placement d'une bombe.

```
POST attack/newbomb
{
    "pos":"5,3",
    "type":"classic"
}
```

C'est le serveur qui se charge de gérer le temps d'explosion d'une bombe. Ainsi si une bombe a été posée sur la case "5,3", et que celle-ci doit exploser, le serveur envoie à tous les clients :

```
POST attack/explose
{
  "pos":"5,3",
  "type":"classic",
  "impactDist":2,
  "map":"...",
```

Le champs **pos** indique la position de la bombe qui explose, le champs **type** permet de rappeler le type de bombe, le champs **impactDist** indique le nombre de cases concernées par l'explosion (dans chaque direction). Enfin le champs **map** permet de renvoyer la nouvelle configuration de la carte après explosion.

Cette dernière correspond à la carte avant explosion sur laquelle on apporte les modifications suivantes :

- si l'explosion passe par une case vide, la case reste vide,
- si l'explosion passe par un mur cassable, la case devient vide (et peut éventuellement faire apparaître un bonus, un malus ou une bombe)
- si l'explosion passe par un mur blindé, la case est inchangée.

Remarque importante : nous avons volontairement omis d'expliquer comment le serveur indique la position des bonus, malus ou bombes désamorcées pouvant apparaître suite à l'explosion d'un mur. C'est à vous de proposer une solution et de la mettre en œuvre dans votre projet

Pour les classics bombs, le serveur envoie le message précédent lorsque le timeout de la bombe est atteint. Dans le cas d'une mine, ce message est envoyé lorsqu'un joueur passe sur une case minée. Enfin pour les remotes bombs, le client aura au préalable envoyé le souhait du joueur de faire exploser sa bombe au moyen du message suivant :

```
POST attack/remote/go
```

Remarquons qu'ici la requête ne précise rien d'autre que l'action attendue. Cela est du au fait que les bombers students ne possèdent qu'une seule télécommande avec un gros bouton rouge. Ainsi si plusieurs remote bombs d'un même joueur sont posées, les deux exploseront en même temps. Les télécommandes sont réglées sur des fréquences différentes : il n'y a donc aucun risque de perturbations entre les remote bomb de deux joueurs différents. Si le serveur détecte que dans les cases touchées par l'explosion il y a un joueur, il envoie ce message spécifique au joueur :

```
POST attack/affect
{
    "life":70, // perte de 30 points de vie
    "speed":1,
    "nbClassicBomb":1,
    "nbMine":0,
    "nbRemoteBomb":0,
    "impactDist":2,
    "invincible":false,
}
```

Lorsqu'un joueur est touché par une bombe, il perd des points de vie, sauf dans le cas où il possède le bonus **invincible**.

Les formules pour comptabiliser le nombre de points de vie perdu dépendent des bombes et de la distance (en nombre de cases) entre la case de la bombe et le joueur touché.

- classic bomb : le nombre de points de vie perdus décroît en fonction du nombre de cases séparant le joueur touché et la bombe. Si le joueur se trouve sur la case bombe (distance de 0) il perd 20 points de vie. S'il se trouve à une distance de 1 case, il perd 15 points de vie, 10 points pour 3 cases et enfin 5 points pour 4 cases de séparation. Rappelons que l'impact maximal d'une bombe est de 4 cases.
- remote bomb : identique à classic bomb
- mine : la mine entraîne une explosion localisée et n'a, par conséquent, aucun impact à part sur la case sur laquelle elle se trouve. Elle provoque un dommage de 30 points de vie.

Ramasser des objets

Lorsque le joueur passe sur une case sur laquelle se trouve un objet (bombe ou mine désactivée, bonus ou malus), le client envoie le message suivant :

```
POST object/new
{
   "type":"classicBbomb", // ou remoteBomb, mine, impactUp etc
}
```

avec

- **type** : le type de l'objet parmi les objets possibles : classicBomb, remoteBomb, mine, impactUp, impactDown, speedUp, speedDown, lifeMax et invincible

Le serveur retourne au client un message contenant la mise à jour des caractéristiques de celui-ci :

```
{ "action":"object/new", "statut":201, message="ok"
    "life":70,
    "speed":1,
    "nbClassicBomb":1,
    "nbMine":1, // augmentation du nombre de mine de 1
    "nbRemoteBomb":0,
    "impactDist":2,
    "invincible":true,
}
```

Y'a plus qu'à !

Nous avons décrit de manière assez précise l'ensemble des éléments composant ce projet. Vous avez (ou allez) cependant vous rendre compte qu'il subsiste quelques points non expliqués ici, comme par exemple le traitement de l'apparition des objets récupérables évoqué précédemment ou encore la gestion de l'invincibilité.... C'est à vous de trouver des solution et définir vos propres extensions du protocole afin de traiter l'ensemble de nos attentes. Gardez également en tête que ces choix et leur mise en œvre ne devront pas, quoi qu'il en soit, impacter la compatibilité de vos client/serveur avec ceux des autres binômes... Y'a plus qu'à donc.... et bon courage.