UNIVERSITÉ DE NANTES

Master 1 ALMA

Projet de Réseaux et Protocoles Internet

Bomber Unicorn

Étudiants :
Coraline MARIE et Vincent RAVENEAU

Intervenant :
Pierrick PASSARD

21 mars 2014

Table des matières

Introduction

Dans un monde où les connexions sont de plus en plus présentes, il est indispensable, en tant que futurs ingénieurs en informatique, de connaître et de comprendre les différentes structures de réseaux existentes sur lesquelles notre monde évolue.

Dans le cadre du cours *Réseaux et Protocoles Internet*, nous avions pour objectif la création d'une application réseau de type *client/serveur*. Pour ce faire, nous avons recréé le jeu multijoueur de BomberMan avec de nouveaux graphismes : `Bomber Unicorn`.

Ce rapport présente donc la création de `Bomber Unicorn` au travers de nombreuses étapes de travail, ainsi que les différentes fonctionnalités de ce jeu. Pour cela nous verrons d'une part la conception de l'application, et le développement d'autre part.

Conception

Dépendances externes

Avant même de concevoir un logiciel, il faut au préalable le penser, ainsi que définir ses besoins et ses diférentes fonctionnalités. Dans cette optique, notre cahier des charges nous imposait de concevoir notre application en `langage C`. Ce langage ne fût pas choisi pour ses performances dans la conception d'applications réseau, mais plutôt dans une optique pédagogique. En effet, il est moins évident de programmer des échanges entre clients et serveur en `langage C`, plutôt qu'en `Java` ou en `C++`, où tout est mis à disposition pour aider le programmeur.

En ce qui concerne la partie graphique de l'application, nous avons choisi d'utiliser le `CSFML` qui est le binding officiel de la SFML pour le langage C. La `SFML` étant une interface de programmation destinée à construire des jeux vidéo ou des programmes interactifs en `C++`.

Fonctionnalités du logiciel

Pour ce projet, aucune fonctionnalité ne nous a été imposée, excepté le fait qu'il fallait créer une application de type *client/serveur*. Pour ce faire, nous avons décidé d'utiliser le serveur en tant que moteur du jeu et les clients en tant que joueurs. Le serveur est capable de gérer jusqu'à 100 parties en simultané (si la machine qui l'héberge est assez puissante), et chaque partie comprend de 2 à 4 joueurs. Pour des raisons de simplicité, de performance et de fluidité, une partie se déroule en tour par tour (un client doit attendre son tour pour jouer).

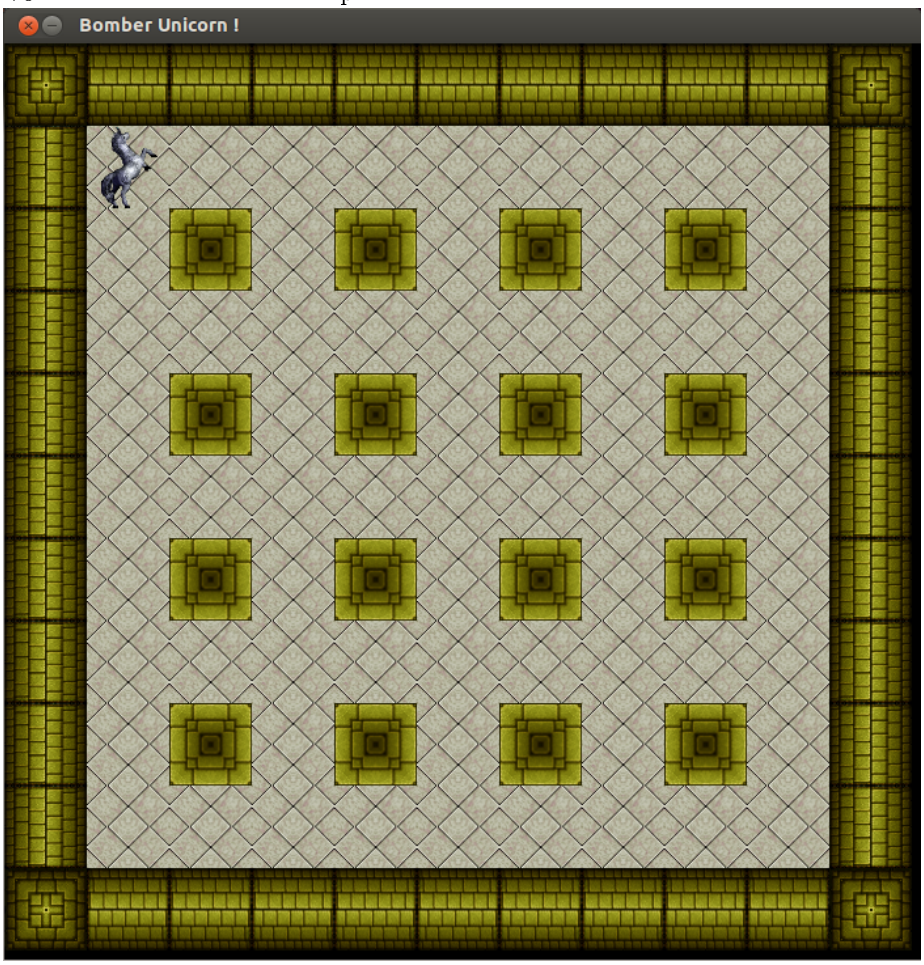
Les règles du jeux sont simple, pour gagner, un joueur doit éliminer tous les autres en réduidant leur point de vie à 0. Pour ce faire, chaque joueur dispose au départ de 3 points de vie, et peut poser jusqu'à 3 bombes en même temps sur la carte. Lorsqu'une bombe est posée, elle change d'état progressivement jusqu'à exploser. Un joueur peut être blessé par ces propres bombes, et lorsque l'une d'elle explose, les joueurs présent autour de la bombe sont également touchés et perdent un point de vie.

Interface graphique

En ce qui concerne l'interface graphique, nous l'avons entièrement construite à l'aide de la `CSFML`. Il s'agit d'un ensemble d'éléments graphiques, appelés *sprite* que l'on place dans une fenêtre. Nous avons créé l'intégralité de nos sprites nous même à l'aide d'une base de donnée libre d'images et d'un logiciel infographique.

Voici quelques exemples de sprites :

Voici la carte créée avec les sprites :



Développement

Les clients et le serveur communiquent en s'envoyant divers messages. Ces messages concerne l'état du jeu ou l'état des joueurs (position, points de vie, nombre de bombes, tour de jeu...). Ces messages sont sous la forme de chaînes de caractères et agissent sur les différentes variables du jeu.

Serveur

Lorsque le serveur est lancé, il reçoit en paramêtre le nombre de parties à gérer, et le nombre de joueurs nécéssaire par partie.

[[[[[[[Partie du rapport à faire par toi si possible]]]]]]]

Client

Avant de lancer les divers clients, il faut s'assurer que le serveur tourne déjà. Lorsque un client est exécuté, il doit avoir en paramêtres : une chaîne de caractères (le nom du joueur) et une adresse (l'adresse ip du serveur). Grâce à ces paramêtres, le client demande une connexion au serveur qui, s'il n'est pas surchargé, répond en lui renvoyant le numéro de la partie qui lui a été attribué et son numéro de joueur. Si le serveur est surchargé, il répond qu'aucune partie n'est disponible.

Déroulement d'une partie

Lorsque le client est connecté au serveur et qu'il possède un numéro de partie et un numéro de joueur, alors la partie peut commencer. Le joueur dispose de deux états, mis à jour par le serveur : l'un lorsque c'est à son tour de jouer, et l'autre quand ça ne l'est pas.

Si c'est au tour du client de jouer, alors celui-ci récupère une action de l'utilisateur et l'envoie au serveur. Le joueur ne peut faire que deux actions différentes, et une seule par tour : se déplacer ou poser une bombe. Lorsque le serveur à reçu l'action du joueur par le client, il doit alors lui renvoyer un message qui peut être :

- Soit un message disant que l'action de l'utilisateur est réfusée (déplacement incorrect ou impossibilité de poser une bombe). Dans ce cas le client récupère une nouvelle action de l'utilisateur, la renvoie au serveur et reattend un nouveau message du serveur.

- Soit un message avec les nouvelles coordonnées des joueurs et des bombes, qui signifie que l'action à été acceptée, que le client doit mettre à jour l'interface graphique, et que ce n'est plus à son tour de jouer.

Si ce n'est pas au tour du client de jouer, alors celui-ci peut recevoir deux messages différents du serveur :
- Soit un message lui annonçant que tous les joueurs ont fini de jouer et que c'est à son tour.
- Soit un message avec les nouvelles coordonnées des joueurs et des bombes pour que le client mette à jour l'interface graphique.

Mise à jour graphique et fin de partie

A chaque fois que le client reçoit un message du serveur contenant des informations sur les autres joueurs, il met à jour l'interface graphique. Pour cela il fait d'abord des tests sur les données des autres joueurs afin de déterminer :
- la position actuelle de tous les joueurs.
- la direction du regard des joueurs en fonction de leur mouvement.
- l'emplacement des bombes.
- l'état des bombes : juste posée, en décompte ou en train d'exploser.
- les points de vie restant aux joueurs et les joueurs mort.
- le statut de la partie : si il y a un vainqueur ou non.

Ensuite, lorsque toutes les données ont été analysée, le client choisie les sprites correspondantes et les places dans la fenêtre. Cette dernière est ensuite actualisée et affichée sur l'écran du joueur qui a exécuté le client.

Structure du code

Pour simplifier et la création et la lecture du code, nous l'avons divisé en plusieurs parties. Il y a donc des fichiers spécifiques au serveur, d'autres uniquement pour le client et quelques uns communs au client et au serveur.

En ce qui concerne les fichiers communs, il contiennent essentielement des déclarations et des types, utiles à la fois au client et au serveur, comme par exemple les variables d'une partie ou la description d'une bombe.

Pour sa part, le client possède plusieurs fichiers qui lui sont exclusif :
- client.c : fichier principal décrivant le fonctionnement du client.
- clientfunctions.c : fichier secondaire contenant les différentes fonctions indispensables au bon fonctionnement du client.
- graphic.c : fichier contenant les différentes fonctions liées à l'affichage graphique.

Quant au serveur, il dispose également de fichiers particuliers :
- serveur.c : fichier principal décrivant le fonctionnement du serveur.
- serveurfunctions.c : fichier secondaire contenant les différentes fonctions indispensables au bon fonctionnement du serveur.

Conclusion

Nous avons présenté, tout au long de ce rapport, notre application `Bomber Unicorn`. Malgré la simplicité de ce jeu, le projet fut relativement difficile à réaliser. En effet, nous avons utiliser les nouvelles notions de communications réseaux étudiées en cours, dans un langage de programmation plutôt limité, et inadapté à la mise en place d'une interface graphique. De plus, l'utilisation de plusieurs procéssus (multithreading) sur des ordinateurs peu puissant nous a grandement handicapé pour la gestion conception de l'application.

Malgré ces difficultés, nous avons réussi à finaliser le projet, mais il reste cependant quelques points que nous aurions souhaité améliorer, si nous en avions eu le temps. En effet, après avoir fait tester notre jeu à plusieurs personnes, nous nous sommes rendu compte qu'il aurait pu être utile d'ajouter du texte sur l'interface graphique (description des tours de jeu, nombre de points de vie restant, etc...).

`Bomber Unicorn` nous a permis d'utiliser de nouvelles connaissances étudiées en *Réseaux et protocoles Internet*, mais pas seulement. Nous nous sommes servis pour ce projet d'autres notions vues dans d'autres modules d'enseignement.