TIC-DVC3 / Bomberman / Bomberman

Modalités

Dépôt	https://rendu-git.etna-alternance.net/module-6426/activity-35936/group-760845
Fichiers requis	*Makefile*, code source
Durée	2 run
Correction	Soutenance finale
Langage autorisé	Langage C
Bibliothèques autorisées	libC, Pthreads, SDL, SDLTTF, SDLImage
Environnement	• Debian Buster • Version du standard C : c17
Flags de compilation	`-Wall -Wextra -Werror`

Objectif

- Aborder les notions de programmation de jeu vidéo et mettre en place une boucle en maîtrisant le taux de rafraîchissement ainsi que les notions de base de communication client/serveur sur le réseau (socket).

Consignes

À ce stade, vous avez un programme qui doit permettre les comportements suivants :

- lancer le programme,
- afficher un menu permettant de : se connecter à un serveur et héberger des clients,
- afficher une interface simple,
- gérer certains événements du clavier,
- permettre la communication entre un ou plusieurs clients et un serveur,
- permettre à plusieurs clients de se connecter au serveur,
- lancer un client,
- lancer un serveur.

Au terme de ce sujet, votre programme doit permettre de :

- mettre en attente un client une fois la connexion au serveur établie. Ce dernier doit attendre un message du lancement de la partie,
- lancer une partie quand au moins 2 clients sont connectés,
- paramétrer des règles de jeu dans un menu (points de vies des joueurs, bonus, bombes, etc.),
- jouer selon les règles de bases du Bomberman :
  - un joueur peut se déplacer de haut en bas et de gauche à droite,
  - un joueur ne peut traverser une bombe ni un bloc,
  - un joueur peut poser une ou plusieurs bombes en fonction de son stock,
  - une bombe explose après un temps donné même si personne n'effectue d'action pendant ce temps,
  - une bombe explose si elle est déclenchée par une autre bombe,
  - un mur est détruit lorsqu'il est atteint par une bombe,
  - un joueur meurt lorsqu'il est touché par une bombe,
  - une partie se finit lorsqu'il ne reste qu'un seul joueur ou que le timeout de la partie est levé.

Ces règles précédemment citées doivent être respectées lors de leur implémentation. Libre à vous d'en rajouter d'autres par la suite.

Makefile

Comme précisé dans les modalités, vous ne devez rendre qu'un fichier *Makefile* et votre code source.

Votre Makefile doit compiler toutes les sources nécessaires au fonctionnement de votre programme et générer un exécutable nommé `bomberman`.

Votre Makefile doit comporter les règles suivantes : `all`, `clean`, `fclean` et `re`.

Attention

Si vous utilisez votre bibliothèque *libmy*, toutes les sources doivent être présentes dans un dossier séparé.
Le Makefile de votre jeu doit également compiler les sources de votre *libmy*.

Protocole

Le serveur et les clients vont échanger en permanence sur l'état de la partie et sur les mouvements désirés par les différents clients.

Voici un exemple de conversation client / serveur :

- Le client se connecte,
- Le serveur lui envoie son numéro de joueur s'il accepte la connexion, sinon il répond que le serveur est plein, ou indisponible puis déconnecte,

Une fois les clients connectés :

- Le serveur envoie l'ordre de commencer la partie.
- Le serveur renvoie les données du jeu, soit :
  - les informations du joueur :
  - la liste des joueurs connectés,
  - le score,
  - la position sur la carte,
  - l'état,
  - le nombre de bombes,
  - le super pouvoir activé,
  - la direction
  - etc.
  - les informations de la carte :
  - position des bombes,
  - position des murs,
  - position des bonus,
  - etc.
  - les informations de la partie :
  - temps restant,
  - information sur les autres joueurs,
  - etc.
- Le client envoie sa volonté d'avancer à droite.
- Le serveur renvoie les données du jeu en réponse.
- Le client envoie sa volonté de poser un bombe.
- Le serveur renvoie à nouveau les données du jeu.
- Etc...

À ce stade, il est souhaitable que chaque client reçoive constamment une copie des informations du jeu à chaque fois qu'une nouvelle action est effectuée, et/ou à chaque intervalle de temps donné.

Exemple de protocole

Cet exemple est basé sur un protocole entièrement binaire (par opposition aux protocoles "textes").
Cela signifie que des structures de données seront directement échangées, plutôt que des commandes textuelles avec paramètres.
Pour communiquer avec le serveur, le client utilisera toujours la même structure de données.

```
typedef struct s_client_request
{
unsigned int magic; /* Un magic number commun entre le client et le serveur, ou l'identifiant d'un type de structure */
int x_pos; /* La position x souhaitée par le client */
int y_pos; /* La position y souhaitée par le client */
int dir; /* La direction souhaitée par le client */
int command; /* Une commande du client (0 : Ne rien faire / 1 : Poser une bombe) */
int speed; /* La vitesse du joueur */
int checksum; /* Un checksum simple */
} t_client_request;
```

Après avoir traité la demande du client, le serveur renvoie toujours la structure de données suivante :

```
typedef struct s_game
{
t_player_infos player_infos[MAX_PLAYERS];
t_map map;
t_other infos;
} t_game;
```

Cette structure est composée tout d'abord des informations sur les joueurs connectés :

```
typedef struct s_player_infos
{
char connected;
char alive;
int x_pos;
int y_pos;
int current_dir;
int current_speed;
int max_speed;
int bombs_left;
int bombs_capacity;
int frags;
} t_player_infos
```

Puis de la carte :

```
typedef char t_map[MAP_SIZE];
```

La carte est représentée par un tableau de char, chaque case étant décomposée de la façon suivante :

```
/*
** Bit 0 : Indique si la case est en flammes (1) ou non (0)
** Bits [1..2] : Indique le type de terrain (00 : Terrain vide, 10 : Brique indestructible, 11 : Brique destructible). Note : Combinaison 01 inutilisée.
** Bit 3 : Présence d'une bombe (0 : Pas de bombe, 1 : Bombe)
** Bit 4 : Présence d'un bonus / malus (0 : Pas de bonus / malus, 1 : Bonus / malus présent)
** Bits [5..7] : Type de bonus / malus.
**
** Pour l'exemple, liste des bonus / malus
** 000 : Bonus portée bombes
** 001 : Malus portés bombes
** 010 : Bonus nombre bombes
** 011 : Malus nombre bombes
** 100 : Bonus vitesse
** 101 : Malus vitesse
** 110 : Rien
** 111 : Rien
*/
```