WISCART Etienne
CHEVALIER Xavier
SEPTIER Sébastien
LE3

Janvier 2017

Compte-rendu

Projet MCS 2017 – Atomic Killer Reloaded+

Objectif :

Le principe de notre jeu consiste à reproduire un jeu de plateformes où l'objectif est d'éliminer les autres joueurs et d'être le dernier survivant. La scène se déroule dans un environnement fermé avec plusieurs plateformes pour donner une certaine liberté de mouvement aux joueurs. Cette application a donc été développée de manière à pouvoir jouer à distance sur deux pc différents, en suivant le modèle client-serveur.

Bibliothèques et technologies utilisées :

- SDL1.2 pour l'interface graphique
- SDL_image pour pouvoir intégrer tout type d'image
- SDL_ttf pour pouvoir afficher du texte
- Des sockets pour gérer le dialogue entre client et serveur
- Des threads pour gérer l'écoute de connexion/événement

Contenu du livrable :

- Code Client : header.h / main.c / fonctions.c
- Code Serveur : server.h / server_co.c
- Dossiers de ressources (polices et images)
- Documentation doxygen
- Compte-rendu

Fonctionnement du jeu :

Le jeu consiste à faire tourner une boucle (presque) infinie, dans laquelle on affiche différents affichages suivant la valeur de la variable « choix ». A chaque tour de boucle, nous regardons si un évènement se produit à l'aide d'un switch-case, les coordonnées ou l'image d'une surface sont ainsi modifiés en conséquence.

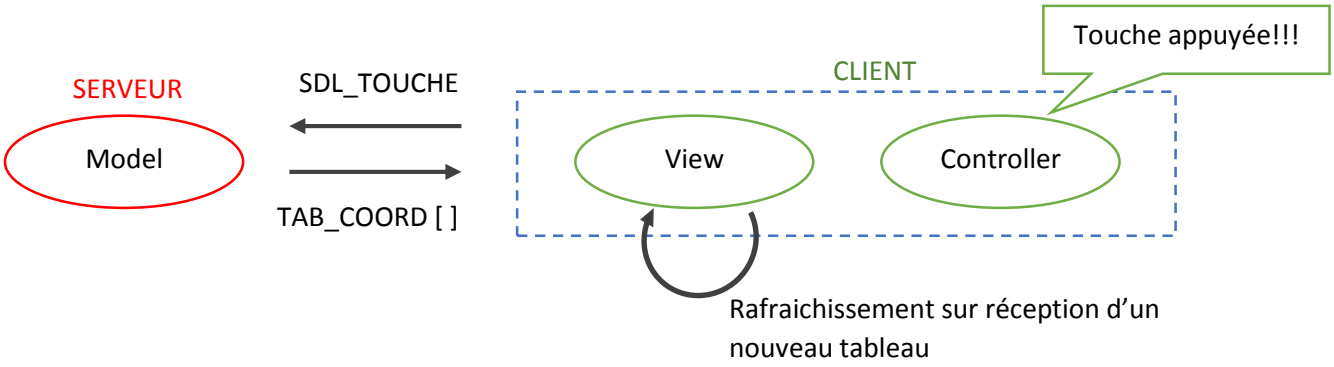
Nous rafraîchissons ensuite l'écran à l'aide d'un SDL_Flit(screen) après avoir chargé la dernière version et la dernière position de chaque surface avec un SDL_BlitSurface(). Pour provoquer certains évènements comme le déplacement des personnages, nous utilisons SDL_GetTicks() qui permet de se référer par rapport au temps.

Nous avons décidé d'organiser notre code de façon à séparer l'interface de la partie calcul de coordonnées. Notre application suit donc le principe MVC (Model View Controller), ce qui permet de synchroniser les clients et de ne pas devoir dupliquer les mêmes fonctions pour chaque client.

La communication Client-Serveur

La communication entre le serveur et ses clients se fait à l'aide de sockets de connexion, dans lesquelles sont renseignées l'adresse ip entrée par l'utilisateur. Chaque client demande une connexion au serveur, puis le serveur répond et enregistre la connexion dans un tableau de sockets pour stocker les connexions. A chaque évènement signalé par un client, le serveur envoie un tableau des coordonnées recalculées des surfaces aux clients.

On crée un thread d'écoute qui permet de réceptionner les messages des clients. Chaque réponse du serveur contient un tableau d'entiers rassemblant les coordonnées et les états de la surface du client. Ainsi, chaque client reçoit en même temps les coordonnées de chaque surface. On peut modéliser cela par le schéma suivant :



Pour chaque touche, on envoie le tableau de coordonnées que de la surface concernée :

TAB_COORD [ ] :
posmario.x
posmario.y
orientationMario
…

TAB_COORD [ ] :
posluigi.x
posluigi.y
orientationLuigi
…

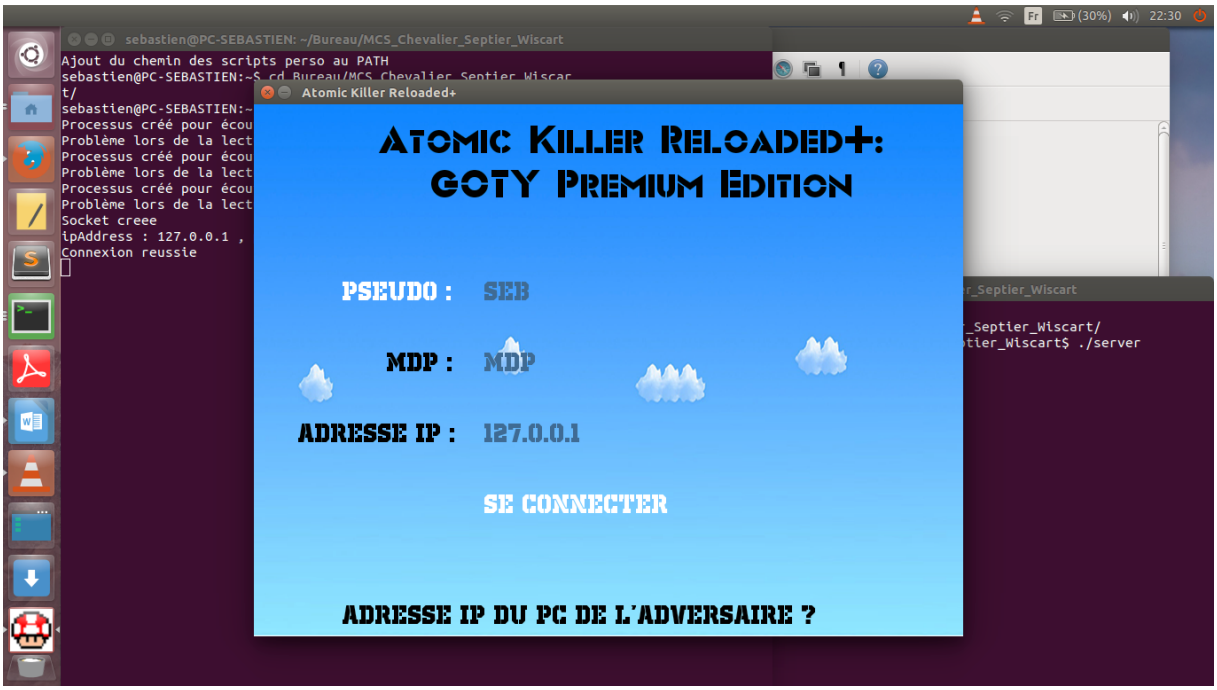
Le résultat de l'application :

Pour générer l'application, un fichier makefile a été créé, il est donc possible de taper make, make clean, make server, et make main.

Lorsque l'on lance le serveur, celui-ci se met à écouter les possibles connexions client :

```
sebastien@PC-SEBASTIEN:~/Bureau/MCS_Chevalier_Septier_Wiscart$ ./server
Socket creee
bind reussi
Attente d'arrivee des connexions...
```

On peut alors lancer les deux clients en lançant l'exécutable 'main' sur deux terminaux différents :



L'interface demande alors à l'utilisateur de rentrer son pseudo, son mot de passe, ainsi que l'adresse ip du serveur (**127.0.0.1**).

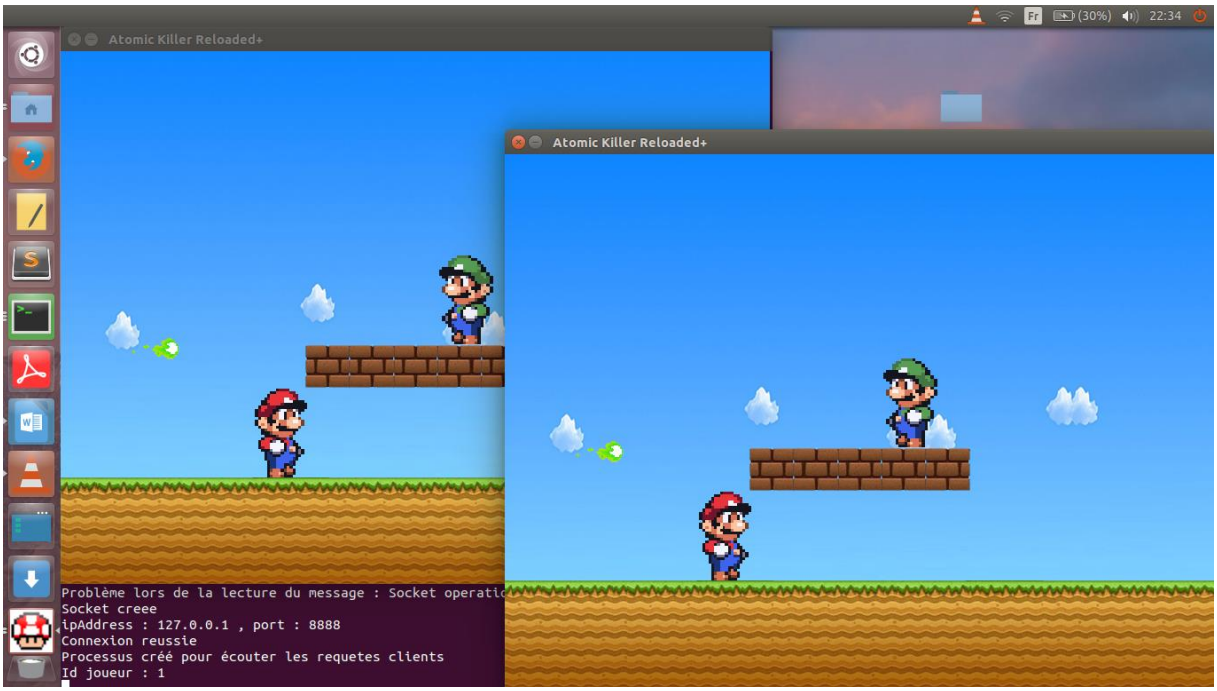
Le serveur reçoit alors les connexions et les enregistre :

```
Attente d'arrivee des connexions...
acceptation reussie
Processus créé pour écouter les requetes clients
acceptation reussie
Processus créé pour écouter les requetes clients
```

Si l'ip rentrée est erronées le client se ferme et renvoie l'information suivante :

```
Socket creee
ipAddress : 127.0.0.0 , port : 8888
Connexion echouee
Error when connecting! Network is unreachable
```

Enfin, lorsque la connexion des deux clients est établie, on peut choisir de créer une partie, et chaque client disposent de la même interface :



Les déplacements se font alors correctement et de façons synchrones.

Conclusion :

Nous avons réussi à faire la majorité des fonctionnalités que l'on souhaitait réaliser. La partie connexion client – serveur est fonctionnelle, et l'interface est terminée.

On pourrait ajouter d'autres fonctionnalités comme :

- Le choix des arènes / gestion des scores
- Des bonus pendant le jeu, et d'autres interactions (bouclier,…)
- La possibilité de tchater avec l'adversaire

Pour conclure, la SDL, utilisée avec les aspects client-serveur et threading permet de faire énormément de choses et nous a permis de voir un autre aspect du langage C. Nos connaissances dans ce langage ont permis une gestion plus simple des choses, comme l'utilisation des sockets et des threads pour instancier chaque client, et partager des ressources communes.