Projet PSE - SLAM RELOADED - Rapport

Lucas VINCENT
lucas.vincent@etu.emse.fr

Promotion EI23 - Groupe IV
Mai-Juin 2024

Baptiste RIQUIER
baptiste.riquier@etu.emse.fr

Fig. 1. – Logo du jeu télévisé SLAM depuis 2017

Table des matières

1. Objectif du projet

Notre projet consiste à reproduire la première manche du jeu télévisé **SLAM**[1]. SLAM est un jeu de lettres, dont l'objectif est de répondre à des questions de français et de remplir une grille de mots-croisés. Notre projet permettra de voir s'affronter jusqu'à 4 candidats, sur ce jeu dont les règles sont décrites dans le Chapitre 2.

2. Rêgles du jeu

SLAM est un jeu de lettres semblable aux mots-croisés. La manche se découpe en round. Chaque round est composé de trois phases :

1. Question : le présentateur interroge les candidats sur une question de français, dont la réponse est forcément une lettre. Le candidat qui répond le plus rapidement, joue la phase suivante, l'autre candidat doit attendre. Avant de passer à la phase suivante, le présentateur rajoute dans la grille toutes les lettres identiques à celle de la question, comme pour un pendu.
2. Remplissage de la grille : La grille est une grille de mot-croisés classique, comme sur la Fig. 2. Le candidat qui a gagné la phase précédente, choisi un numéro. Le présentateur lui donne une définition, souvent complexe, du mot à trouver. Si le candidat trouve le bon mot, alors il remporte autant de points que le nombre de lettres du mot, sinon il ne gagne rien. Le candidat doit néanmoins répondre en moins de 15 secondes.
3. (facultatif) SLAM : Les candidats peuvent décider de faire un SLAM. Le SLAM consiste à remplir toutes les lignes et colonnes de la grille, sans aucune définition. Si il réussit à trouver tous les mots, il gagne le nombre de points correspondant. Souvent, le SLAM est utilisé pour rattraper un adversaire et passer devant lui.

Lorsque la grille est complète, le gagnant est celui avec le plus de points.

Exemple d'une manche :

Héloïse, Simon, Marie-josé s'affrontent pour gagner la cagnotte. Le présentateur commence par la **question** : *Quelle lettre touche une seule voyelle une fois à bord d'un canoë comme d'une péniche ?* Simon répond N, mais c'est une mauvaise réponse. Héloïse répond C, c'est la bonne réponse. Ainsi, on rajoute tous les C de la grille, donc deux. Héloïse choisit ensuite de **remplir la ligne 7**. Le présentateur donne la **définition** : *Il arrive à table avec les crêpes.* Héloïse répond

[1]Site officiel de FranceTV avec le jeu SLAM : https://www.france.tv/france-3/slam/

le CIDRE. C'est une bonne réponse, elle gagne donc **5 points**. Pas de SLAM, on recommence les phases.

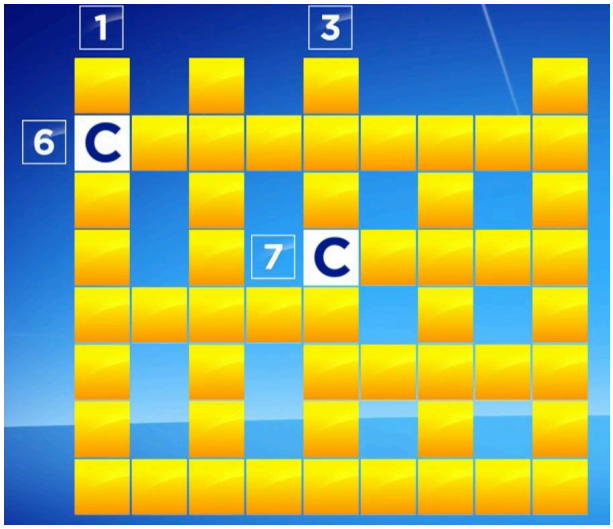

Fig. 2. – Exemple de grille pour le jeu SLAM.

3. Architecture du projet

3.1. Décomposition de l'archive

```
├── 1_Code
│   ├── CLIENT
│   │   ├── client.c
│   │   ├── client.h
│   │   ├── Client_start
│   │   ├── jeu_client.c
│   │   ├── jeu_client.h
│   │   └── makefile
│   ├── LIB
│   │   ├── all_lib.h
│   │   ├── utilities.c
│   │   └── utilities.h
│   ├── README.md
│   ├── SERVEUR
│   │   ├── jeu_serveur.c
│   │   ├── jeu_serveur.h
│   │   ├── JEU_txt
│   │   │   ├── def_1.txt
│   │   │   ├── devinettes_pour_lettres.txt
│   │   │   └── grille_1.txt
│   │   ├── makefile
│   │   ├── serveur.c
│   │   ├── serveur.h
│   │   └── Serveur_start
│   └── SourceMe.txt
└── README.md
```

Le projet est composé de trois dossiers :

- CLIENT : Il contient tous les fichiers nécessaires au bon fonctionnement du client.

- SERVEUR : Il contient tous les fichiers nécessaires au bon fonctionnement du serveur.
- LIB : Il contient toutes les importations de bibliothèques et les fonctions utiles, communes au client et au serveur.

Pour simplifier le code et sa compréhension, nous avons décider de profiter de la modularité. Ainsi, chaque dossier est décomposé en fichiers .c et .h. Au sein des dossiers CLIENT et SERVEUR, les fichiers *jeu_serveur* et *jeu_client* contiennent les fonctions pour le bon déroulement du jeu et les fichiers client.c et serveur.c contiennent l'aspect communication client/serveur.

Le dossier JEU_txt, contient les grilles de jeux : *grille_1.txt*, les définitions associées *def_1.txt* et les devinettes pour les lettres : *devinettes_pour_lettres.txt*

Les fichiers *grilles.txt* et *questions.txt* contiennent les grilles de jeu et les questions associées.

Enfin, les fichiers sont compilés à l'aide du fichier *SourceMe.txt*, qui contient les commandes nécessaire au démarrage du jeu.

3.2. Bibliothèques utilisées

Pour ne pas surcharger notre projet avec des fonctions et dépendances inutiles, nous avons décidé de ne pas inclure les fichiers d'en-tête du répertoire PSE. Dans ce projet, nous utilisons les bibliothèques suivantes :

```
#include <stdlib.h>    #include <string.h>
#include <time.h>      #include <assert.h>
#include <unistd.h>    #include <signal.h>
#include <sys/types.h> #include <sys/socket.h>
#include <sys/stat.h>  #include <sys/wait.h>
#include <fcntl.h>     #include <netinet/in.h>
#include <arpa/inet.h> #include <netdb.h>
#include <pthread.h>   #include <semaphore.h>
#include <errno.h>     #include <libgen.h>
#include <stdarg.h>    #include <stdio.h>
```

Les bibliothèques les plus notables sont :

- *string.h* qui facilite la gestion des chaînes de caractères.
- *time.h* qui permet d'utiliser la fonction *rand()*, utilisé pour générer des nombres semi-aléatoires.
- *pthread.h* qui permet d'utiliser des threads et des forks.
- *errno.h* pour relever les erreurs d'exécution.
- *stdlib.h* et *stdio.h* pour les fonctions de base du C.

Les arguments **-Wall -Wextra** permettent d'inclure un plus large éventail d'erreurs pour le débogage et donc d'encourager une méthode de coding, plus propre.

3.3. Points clefs

3.3.1. Phases

De manière à simplifier son fonctionnement et à assurer un fonctionnent synchrone des différents clients et threads, le projet est construit selon l'architecture machine d'états et top level. Le main représente le top level. Il gère les entrées et sorties des 2 à 4 threads, qui sont semblable à des machines d'états. Le main dicte la phase en cours, la question, la grille à remplir, etc et reçoit les réponses des clients à travers les threads.

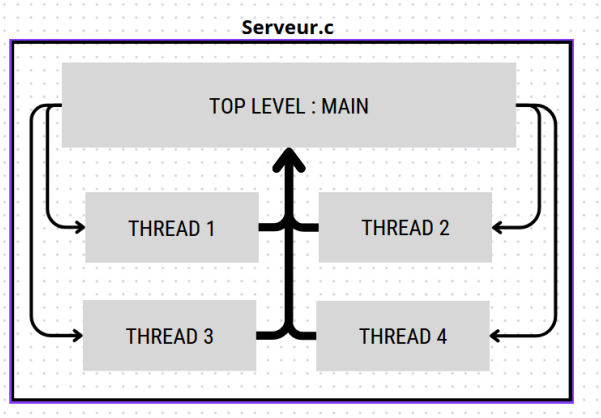

Fig. 3. – Fonctionnement en *top level/ machine d'états.*

Les threads fonctionnent donc comme des machines d'états. Elles sont composées de 4 états, de 0 à 3. L'état actuel est dicté par le main, à travers la variable phase. Chaque état correspond à une phase du jeu : SLAM.

Au début, on est dans l'état 0 : attente. Tous les threads reçoivent alors phase = 1 et une question. Ils transmettent alors la question qu'ils ont reçu vers le client associé, récupèrent sa réponse et l'envoient au main.

Ensuite, le client ayant correctement répondu et en premier, reçoit phase = 2, tandis que tous les autres reçoivent, phase = 0. Le gagnant essaye de remplir une des colonnes/lignes de la grille.

Enfin, tous les clients reçoivent phase = 3, qui leur permet de choisir, si oui ou non ils veulent faire un SLAM.

Et on recommence : phase = 0, etc.

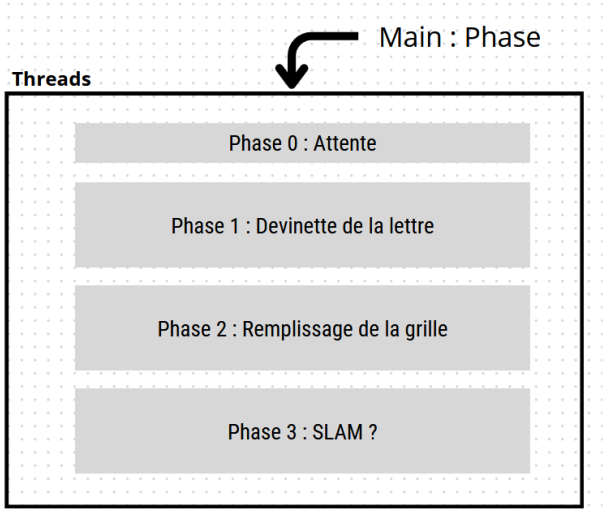

Fig. 4. – Fonctionnement de la *machine d'états*.

[Note] Pour savoir quel joueur a répondu en premier, on utilise une FIFO commune à tous les threads. En effet, la FIFO place les informations dans l'ordre reçu. Ainsi, à partir du main, on peut regarder le contenu de la FIFO, dans l'ordre des réponses et dès qu'on trouve une bonne réponse, on sauvegarde le joueur comme étant le gagnant et on vide la FIFO.

3.3.2. Fonctions

Fonction : Attentes_joueurs()

Cette fonction permet de gérer la salle d'attente. Elle affiche à travers la lecture d'un fichier texte, la liste des joueurs présents. Pour cela, le fichier *salle_attente.txt* est actualisé à chaque arrivé d'un joueur et lu périodiquement, toute les secondes, à l'aide de la commande : **watch -n 1 cat ./SERVEUR/salle_attente.txt**. Pour faire actualiser le fichier et l'afficher en parallèle, on utilise un processus, crée à l'aide de la fonction fork().

Le père écrit dans le fichier et le fils se charge d'afficher le contenu du fichier.

Pourquoi avoir utilisé cat ?

Nous voulions un affichage statique, comme pour un salon de jeu-vidéo. Or si nous avions utilisé la fonction : *printf*, le texte aurait défilé au lieu de garder la même place à l'écran.

Fonction : Session_joueurs()

Cette fonction permet de réaliser l'intermédiaire entre les joueurs et le serveur main(). Elle est appelée par des threads, qui sont chacun associé à un client. Elle fonctionne comme une machine d'état, contrôlé par le main().

Fonction : Decoupe_message()

Cette fonction découpe le message reçue pour retirer la partie fonctionnelle et ne conserver que la partie utile. Cette fonction est lié à notre protocole de communication.

Protocole de communication :

Tous les messages échangés entre le serveur et les clients se doivent d'être de la forme suivante :

1. Une partie fonctionnelle, qui indique le type du message, C : Connexion / R : Réponse ...
2. Une partie utile, qui contient le texte du message : Nom du joueur par exemple ...

Exemple pour la connexion du joueur Basile : *C_Basile*

4. Pourquoi le projet n'est-il pas achevé ?

Le projet que nous avons choisi de réaliser, nous paraissait à première vue intéressant et dans la continuité du cours. Néanmoins, nous avons sous-estimé sa complexité. Là où la majorité de nos camarades s'intéressent à un jeu, ou un seul objectif, nous avons fait l'erreur de travailler sur un projet regroupant trois jeux différents en un seul :

- Un quiz
- Des mots-croisés
- Une interruption du jeu, pour faire un SLAM

Cette complexité, nous a dans un premier temps désemparés, car par quel bout l'aborder en premier ? Puis, a entraîné des problèmes techniques, comment transmettre facilement les questions ? Comment transmettre facilement un tableau au client ? Des problèmes organisationnels, comment répartir le travail au sein du binôme ?[2] Un manque de temps, relatif aux nombreux projets à avancer en simultané.
Enfin, malgré plus d'une vingtaine d'heures de travail, nous sommes confrontés à trop d'erreurs de programmation : fonction qui ne fonctionne pas correctement, Segmentation Fault, etc et la tâche à accomplir pour finir le jeu est encore colossale.
C'est pourquoi, nous avons décidé, lundi 10/06/2024, d'arrêter de coder, pour nous concentrer sur les compte-rendus à produire, laissant le jeu dans un état fonctionnel, mais qui ne reflète pas toutes les réflexions de développement que nous avons eu, ni toutes les parties développé.

Malgré tout cela, nous tirons de ce travail, des enseignements tant sur le plan pratique, à travers l'application des connaissances de cours et nos difficultés techniques, sur le plan organisationnel et sur le plan des connaissances, car nous avons pu enrichir les connaissances acquises en cours, à travers un exercice concret, exigeant, mais instructif.

[2]NOTA BENE : Depuis le projet d'Algorithmique et Programmation I, nous cherchons des solutions pour répartir le travail de coding, une tâche que nous trouvons complexe, au vu de l'interdépendance du code.