Application Multi-Utilisateurs (6h)

Le but de ce travail est de réutiliser les différentes connaissances vues durant l'électif en réalisant un grand projet.

1 Vue globale du projet

Le projet consiste à réaliser une application pouvant gérer de multiples utilisateurs. **Le choix du projet est laissé au choix des élèves**. Nous prendrons pour exemple une application de *Snake* multijoueurs que les élèves peuvent coder si ils le souhaitent.

Le but étant de réutiliser les connaissances du cours l'utilisation de certains composants tels que des *processus*, *threads*, *IPC* ou *fichiers* est **fortement encouragée**.

Par exemple nous présentons une spécification possible d'une application de Snake multijoueurs :

— Affichage dans le terminal
— Jouable jusqu'à quatre joueurs sur le même clavier
— Un processus client pour les interactions joueurs et un serveur pour la logique du jeu. Cela permettra de facilement porter le jeu sur le réseau. Il suffira de mettre un serveur en ligne sur lequel les applications clients pourront se connecter.
— Sauvegarde des statistiques de jeu par nom de joueur dans des fichiers

2 Critères d'Évaluation

Les élèves effectueront le projet en **binôme** ou **trinôme**. L'évaluation du projet long comportera les éléments suivants :

— Une présentation orale
— Une évaluation du projet

Les élèves peuvent librement choisir de développer en **C**, en **Rust** ou les deux.

2.1 Évaluation Orale

2.1.1 Présentation de groupe

Une présentation de **15** minutes sera donnée (**20** minutes pour un trinôme) suivie de quelques questions du jury. La présentation devra notamment comporter :

— La spécification du projet
— Présentation des éléments techniques
— Une démonstration
— Des axes d'amélioration

2.1.2 Entretien individuel

Un entretien **individuel** de 5 minutes suivra la présentation. Les questions posées porteront sur le code du projet et sur les principes cœurs étudiés durant l'électif.

2.2 Évaluation du projet

Après la présentation, le projet sera envoyé au jury qui sera chargé de noter le code. Certaines pratiques de code seront fortement appréciées et valorisées :

— Commentaires informatifs
— Présence de tests
— Découpage modulaire des différentes parties du projet

De plus l'encadrant **valorisera également l'utilisation de notions vues durant l'électif** tels que les *processus*, *threads*, *IPC* ou *fichiers*.

2.3 Remarque sur le recopie de code

De nombreux projets intéressants circulent sur Internet et vous êtes invités à les regarder pour vous inspirer et vous aider. Si des parties de votre application sont **fortement inspirées du code d'autrui**, **vous êtes invités à le mentionner** durant la présentation et dans le code.

Toute détection de copie d'un code extérieur sans aucune mention sera **sévèrement sanctionnée**.

De plus la présence de code suspect fera très certainement l'objet de questions durant l'entretien individuel.

3 Décomposition des étapes du projet

Ce projet étant très ouvert **n'hésitez surtout pas à demander de l'aide** aux encadrants que ce soit pour résoudre un problème ou pour vous aider à démarrer. Pour vous guider durant ce projet, plusieurs étapes vous sont proposées. Ces différentes étapes sont illustrées avec notre exemple de Snake multijoueurs.

3.1 Création d'une version fonctionnelle

Pour commencer, le premier objectif est d'obtenir un prototype fonctionnel. Par exemple, dans notre Snake nous allons d'abord nous focaliser sur la programmation d'un Snake classique avec un seul joueur.

3.1.1 Définition des composants et de leurs interactions

Pour commencer, définissez les différents composants nécessaires à votre application. Pour le Snake nous aurons :

— une structure `Snake` qui contiendra la représentation et la logique d'un serpent
  — un serpent possède un corps et une direction par exemple
  — un serpent a besoin d'une fonction qui le fait "avancer"
  — un serpent a besoin d'une fonction qui le fait "grandir" quand il mange une pomme
— une structure `Game` qui gèrera autres composants et leurs interactions. Par exemple les pommes du serpent, le terrain, l'affichage ...
  — une partie de snake comporte un terrain de jeu (un tableau)
  — un serpent et sa représentation dans le terrain
  — une pomme qui apparaît aléatoirement
  — une fonction d'initialisation `init` qui crée une instance de `Game` correspondant au début de partie
  — ...

Cette décomposition peut aussi vous aider pour organiser votre projet en différents modules/fichiers.

3.1.2 Affichage des composants

Si vous codez un projet avec une interface graphique, il sera vite nécessaire de la développer pour tester votre projet. Un exemple d'interface graphique dans le terminal vous est fourni avec le sujet du projet. La librairie Rust utilisée est `termion` et de la documentation est accessible :
`https://ticki.github.io/blog/making-terminal-applications-in-rust-with-termion/`
`https://docs.rs/termion/1.5.5/termion/index.html`

3.1.3 Récupération des inputs utilisateurs

Votre application interagit avec l'utilisateur, vous aurez besoin de récupérer les entrées de l'utilisateur. En Rust aidez-vous de la documentation sur `stdin` :
`https://doc.rust-lang.org/std/io/struct.Stdin.html`

3.1.4 Codage de la logique

Ensuite il faut coder la logique de l'application, les interactions possibles. Pour un jeu comme Snake il s'agit des interactions : lorsque le serpent mange une pomme, touche un mur...
— codage d'une fonction `should_grow` pour déterminer lorsque le serpent doit grandir
— codage d'une fonction `check_collisions` pour déterminer si le serpent touche un mur

4 Améliorations de l'application

Ensuite à partir de ce prototype de nombreuses améliorations sont possibles. Si vous n'avez pas d'idée, je suis certain que vos encadrants en auront pour vous.

4.1 Fonctionnalités de l'application

Par exemple pour le Snake :
— Passage de l'application en mode multijoueurs avec plusieurs serpents
— Rajout d'un compteur de point
— Codage d'une AI simple pour avoir un adversaire

MultiThreading Utilisation des threads pour améliorer les performances de l'application. Par exemple, pour chaque serpent il serait possible de générer ces mouvements dans des threads différents.

Application Client-Serveur Séparation de l'application en deux morceaux : côté client et côté serveur. Cette infrastructure avait été utilisée auparavant durant le BE de C/Unix.

Gestion de Fichiers Utilisez des fichiers si vous souhaitez garder des informations de manière permanentes. Par exemple l'application Snake pourrait retenir tous les scores, et demander un nom aux joueurs battant des records.