Concevoir une expérience d'apprentissage interactive à la programmation avec PLX

Contexte

Ce travail de Bachelor vise à développer le projet PLX (voir plx.rs), Terminal User Interface (TUI) écrite en Rust, permettant de faciliter la pratique intense sur des exercices de programmation en retirant un maximum de friction. PLX vise également à apporter le plus vite possible un feedback automatique et riche, dans l'idée d'appliquer les principes de la pratique délibérée à l'informatique. PLX peut à terme aider de nombreux cours à la HEIG-VD (tels que PRG1, PRG2, PCO, SYE, …) à transformer les longs moments de théorie en session d'entrainement dynamique, et redéfinir l'expérience des étudiants sur ces exercices ainsi que les laboratoires. L'ambition est qu'à terme cela génère un apprentissage plus profond de modèles mentaux solides, pour que les étudiants aient moins de difficultés avec ces cours.

Problème

Inspiré de Rustlings (TUI pour apprendre le Rust), permettant de s'habituer aux erreurs du compilateur Rust et de prendre en main la syntaxe. PLX fournit actuellement une expérience locale similaire pour le C et C++. Les étudiants clonent un repos Git et travaillent localement sur des exercices afin de faire passer des checks automatisés, à chaque sauvegarde le programme est compilé et les checks sont lancés. Cependant, faire passer les checks n'est que la 1ère étape, faire du code qualitatif, modulaire, lisible et performant demande des retours humains pour pouvoir progresser. De plus, les exercices existants étant stockés dans des PDF ou des fichiers Markdown, cela nécessite de les migrer à PLX.

Défis

Ce TB aimerait pousser l'expérience en classe plus loin pour permettre aux étudiants de recevoir des feedbacks sur leur réponse en live, sur des sessions hautement interactives. Cela aide aussi les enseignants à mesurer l'état de compréhension et les compétences des étudiants tout au long du semestre, et adapter leur cours en fonction des incompréhensions et lacunes.

Pour faciliter l'adoption de ces outils et la rapidité de création/transcription d'exercices, on souhaiterait avoir une syntaxe épurée, humainement lisible + éditable, facilement versionnable dans Git: la syntaxe DY. Cette syntaxe sera adaptée pour PLX, pour remplacer le format TOML actuel.

Ces 2 défis impliquent

1. Une partie serveur de PLX, gérant des connexions persistantes pour chaque étudiant et enseignant connecté, permettant de recevoir les réponses des étudiants et de les renvoyer à l'enseignant. Une partie client est responsable d'envoyer le code modifié et les résultats après chaque lancement des checks.
2. Le but est de définir une syntaxe et de réécrire le parseur en Rust en s'aidant d'outils adaptés (TreeSitter, Chumsky, Winnow, …).

Le projet, les documents et les contributions de ce TB, seront publiés sous licence libre.

Objectifs et livrables

1. Livrables standards: Rapport intermédiaire + rapport final + résumé + poster
2. Un serveur en Rust lancé via le CLI plx permettant de gérer des sessions live
3. Une librairie en Rust de parsing de la syntaxe DY
4. Une intégration de cette librairie dans PLX

Objectifs fonctionnels

Les objectifs fonctionnels posent l'hypothèse du cas d'utilisation où un professeur lance une session live pour plusieurs étudiants. Il n'y a donc pas de rôle spécifique attribuée au professeur par rapport aux étudiants, il y a seulement une distinction des permissions entre le créateur de la session et ceux qui rejoignent.

1. Les professeurs peuvent lancer et stopper une session live via PLX liée au repository actuel, via un serveur défini dans un fichier de configuration présent dans le repository. Il peut exister plusieurs sessions en même temps pour le même repository (afin de supporter plusieurs cours en parallèle dans plusieurs classes). Ils donnent un nom à la session, afin que les étudiants puissent l'identifier parmi les sessions ouvertes. Un code de vérification unique est généré par session permettant de distinguer 2 sessions du même nom dans le même repos.
2. En tant qu'étudiant, une fois le repository cloné, il est possible de lancer PLX, lister les sessions ouvertes et rejoindre une session en cours en s'assurant du code de vérification. Un numéro unique incrémentale est attribué à chaque étudiant pour la session.
3. Le professeur peut choisir une série d'exercices parmi ceux affichés par PLX, lancer un exercice et gérer le rythme d'avancement de la classe. Cet exercice sera affiché directement chez les étudiants ayant rejoint.
4. Une vue globale permet au professeur d'avoir un aperçu général de l'état des checks sur tous les exercices. En sélectionnant un exercice, il est possible de voir la dernière version du code édité ainsi que les résultats des checks pour ce code, pour chaque étudiant.
5. L'intégration de la librairie `dy` dans PLX permet de décrire les informations d'un cours, des compétences et des exercices. Elle détecte les erreurs spécifiques à PLX.
6. L'intégration dans PLX permet d'utiliser uniquement des fichiers `.dy` pour décrire le contenu. Elle doit aussi afficher les erreurs dans une liste sur une commande dédiée (par ex. `plx check`)

Objectifs non fonctionnels

1. Une session live doit supporter des déconnexions temporaires, le professeur pourra continuer à voir la dernière version du code envoyé, et le client PLX essaiera automatiquement de se reconnecter. Le serveur doit pouvoir supporter plusieurs sessions live incluant au total 300 connexions persistantes simultanées.
2. Une session live s'arrête automatiquement après 30 minutes après déconnexion du professeur, cela ne coupe pas l'affichage de l'exercice en cours aux étudiants
3. Pour des raisons de sécurité, aucun code externe ne doit être exécuté automatiquement par PLX. Seul une exécution volontaire par une action dédiée peut le faire.
4. Le temps entre la fin de l'exécution des checks et la visibilité des modifications par l'enseignant ne doit pas dépasser 3s.
5. Le code doit être le plus possible couvert par des tests automatisés, notamment par des tests end-to-end avec de multiples clients PLX.
6. Le parseur DY doit être assez capable de parser 200 exercices en < 1s.
7. Retranscrire à la main un exercice existant du Markdown en PLX DY ne devrait pas prendre plus d'une minute.

Objectif nice to have

1. La librairie `dy` permettrait d'intégrer le parseur et les erreurs spécifiques à un language server permettant une expérience complète d'édition dans VSCode et Neovim
2. La librairie `dy` serait également capable de générer des définitions TreeSitter pour supporter le syntax highlighting via ce système

Calendrier du projet

En se basant sur le calendrier des travaux de Bachelor, voici un aperçu du découpage du projet pour les différents rendus.

Rendu 1 - 10 avril 2025 - Cahier des charges

- Rédaction du cahier des charges
- Analyse de l'état de l'art des parsers, du syntax highlighting et des languages servers
- Analyse de l'état de l'art des protocoles bi-directionnel temps réel (websockets, gRPC, …) et des formats de sérialisation (JSON, protobuf, …)
- Prototype avec les librairies disponibles de parsing et de language servers en Rust, choix du niveau d'abstraction espéré et réutilisation possibles

Rendu 2 - 23 mai 2025 - Rapport intermédiaire

- Rédaction du rapport intermédiaire
- Définition de la syntaxe DY à parser, des préfixes et flags liés à PLX, et la liste des vérifications et erreurs associées
- Définition d'un protocole de synchronisation du code entre les participants d'une session
- Prototype d'implémentation de cette synchronisation
- Prototype des tests automatisés sur le serveur PLX
- Définition du protocole entre les clients PLX et le serveur pour les entrainements live

Moitié des 6 semaines à temps plein - 4 juillet 2025

- Ecriture des tests de validation du protocole et de gestion des erreurs
- Développement du serveur PLX
- Rédaction du rapport final par rapport aux développements effectués

Rendu 3 - 24 juillet 2025 - Rapport final

- Développement d'une librairie `dy`
- Intégration de cette librairie à PLX
- Rédaction de l'affiche et du résumé publiable
- Rédaction du rapport final