Projet de Programmation en Rust : Un jeu de rôle en mode texte

Master en Génie Logiciel - UE Algorithme et Modélisation Avancée

1. Introduction et présentation du projet

1.1 Contexte du projet

Dans le cadre de l'UE Algorithme et Modélisation Avancée de notre Master en Génie Logiciel, nous avons été amenés à réaliser un projet de programmation en Rust. Ce projet consistait à développer un jeu de rôle en mode texte orienté sur une simulation logique du monde physique. Ce travail nous a permis d'explorer en profondeur les concepts de modélisation et d'algorithmique tout en nous familiarisant avec les spécificités du langage Rust.

Le choix du langage Rust pour ce projet n'est pas anodin. En effet, Rust est un langage de programmation système moderne qui met l'accent sur la sécurité mémoire sans recourir à un ramasse-miettes (garbage collector), offrant ainsi des performances comparables au C++ tout en garantissant une meilleure sécurité. Ses mécanismes de propriété (ownership) et d'emprunt (borrowing) nous ont permis d'explorer des approches de programmation différentes de celles que nous connaissions déjà, enrichissant ainsi notre bagage technique.

L'objectif pédagogique principal était de nous confronter à la modélisation d'un système complexe et évolutif, où les interactions entre les différentes composantes devaient suivre une logique cohérente. Cette approche nous a poussés à réfléchir en termes d'architecture logicielle robuste et extensible, tout en nous familiarisant avec les bonnes pratiques de développement en Rust.

1.2 Description générale du jeu

Notre implémentation du jeu de rôle en mode texte s'articule autour d'une architecture modulaire qui sépare clairement la logique métier de l'interface utilisateur. Le jeu propose une expérience immersive où le joueur évolue dans un univers persistant composé de salles interconnectées, chacune ayant ses propres caractéristiques et contenus.

L'aspect central de notre jeu repose sur un système d'entités complexe qui modélise différents types d'objets interactifs. Les personnages joueurs disposent d'un système de caractéristiques évolutives et d'un inventaire dynamique leur permettant de collecter, utiliser et échanger des objets. L'interaction avec l'environnement se fait à travers un système de commandes textuelles qui déclenchent des actions spécifiques selon le contexte.

Le monde du jeu est peuplé de personnages non-joueurs (PNJ) qui proposent des interactions variées, notamment à travers un système de commerce et de quêtes. Ces PNJ ne sont pas de simples distributeurs de missions, mais constituent des entités autonomes avec leurs propres comportements et dialogues contextuels. Le système de quêtes s'intègre naturellement dans cette logique, proposant des objectifs variés qui encouragent l'exploration et l'interaction.

Un des aspects les plus innovants de notre implémentation est le système de combat dynamique avec génération procédurale de butin. Les ennemis ne sont pas de simples obstacles statiques, mais des entités complexes avec leurs propres caractéristiques et comportements de combat. Le système de loot ajoute une dimension de récompense et de progression qui maintient l'engagement du joueur.

L'interface utilisateur, bien qu'entièrement textuelle, intègre des éléments visuels à travers un système d'affichage d'images associées aux différentes salles. Cette approche hybride permet de maintenir l'immersion tout en conservant la simplicité d'une interface en ligne de commande.

1.3 Objectifs techniques

L'architecture technique de notre projet répond à plusieurs objectifs fondamentaux qui ont guidé nos choix de conception et d'implémentation.

Séparation stricte des données et de la logique : Conformément aux exigences du projet, nous avons implémenté une séparation complète entre le moteur de jeu et les données de contenu. Toutes les informations relatives aux salles, personnages, objets, dialogues, ennemis et quêtes sont stockées dans des fichiers JSON distincts. Cette approche présente plusieurs avantages majeurs : elle permet la modification du contenu sans recompilation, facilite la localisation du jeu, et rend possible la création de différents scénarios par des personnes sans compétences en programmation.

Architecture modulaire et extensible : Notre code est organisé en modules spécialisés qui encapsulent des responsabilités spécifiques. Le module `entities` contient toutes les structures de données représentant les objets du jeu, le module `traits` définit les comportements communs, et les modules fonctionnels comme `combat` et `dialogue`

implémentent les mécaniques spécifiques. Cette organisation facilite la maintenance, les tests, et l'ajout de nouvelles fonctionnalités.

Exploitation des spécificités de Rust : Nous avons tiré parti des mécanismes avancés de Rust, notamment son système de types expressif, ses traits pour définir des comportements polymorphes, et son modèle de propriété pour garantir la sécurité mémoire. L'utilisation de structures comme `HashMap` pour les collections d'entités et `Vec` pour les listes ordonnées nous a permis d'optimiser les performances tout en maintenant la sécurité.

Simulation logique et cohérence : L'environnement du jeu évolue selon des règles logiques cohérentes. Les systèmes de combat, de commerce, et d'interaction sont conçus pour créer des expériences émergentes où les actions du joueur ont des conséquences prévisibles mais variées. Cette approche simulation contribue à créer un monde crédible et engageant.

2. Approche technique et défis de développement

2.1 Défis techniques et solutions générales

Le développement de ce projet de jeu de rôle en Rust nous a confrontés à plusieurs défis techniques fondamentaux qui dépassent le cadre spécifique de notre implémentation et touchent aux enjeux généraux du développement logiciel moderne. Ces défis nous ont permis d'approfondir notre compréhension des principes de conception logicielle et de développer des compétences transférables à d'autres contextes de développement.

Gestion de la complexité architecturale : L'un des défis majeurs de tout projet de développement logiciel complexe réside dans la gestion de l'architecture et des interdépendances entre composants. Dans le contexte d'un jeu de rôle, cette complexité se manifeste par la nécessité de faire interagir de nombreux systèmes différents : gestion des personnages, environnements, objets, interactions sociales, et mécaniques de jeu. Notre approche a consisté à adopter une architecture modulaire stricte, où chaque composant a des responsabilités clairement définies et des interfaces bien délimitées.

Cette approche modulaire présente plusieurs avantages significatifs. Elle facilite la maintenance du code en permettant de modifier un composant sans affecter les autres, elle simplifie les tests en permettant de tester chaque module indépendamment, et elle facilite le travail en équipe en permettant à chaque développeur de se concentrer sur des modules spécifiques. Cependant, cette approche nécessite également une planification rigoureuse et une communication constante entre les membres de l'équipe pour s'assurer que les interfaces entre modules restent cohérentes.

Sécurité et robustesse du code : Le langage Rust impose des contraintes strictes qui, bien qu'initialement déroutantes pour des développeurs habitués à d'autres langages, contribuent significativement à la robustesse du code final. Le système de propriété et d'emprunt de Rust élimine de nombreuses classes d'erreurs courantes comme les fuites mémoire, les accès à des pointeurs invalides, ou les conditions de course dans les programmes concurrents.

L'adaptation à ces contraintes a nécessité une réflexion approfondie sur la conception des structures de données et des algorithmes. Plutôt que de subir ces contraintes, nous avons appris à les utiliser comme guide pour concevoir des architectures plus robustes et plus prévisibles. Cette expérience nous a sensibilisés à l'importance de la sécurité dès la conception, un principe fondamental du développement logiciel moderne.

Séparation des préoccupations et maintenabilité : Un défi récurrent dans le développement logiciel est la séparation efficace entre la logique métier, les données, et l'interface utilisateur. Notre projet nous a confrontés à ce défi de manière concrète, notamment à travers l'exigence de séparer le moteur de jeu des données de contenu. Cette séparation, bien qu'initialement complexe à mettre en œuvre, s'est révélée extrêmement bénéfique pour la maintenabilité et l'extensibilité du projet.

L'implémentation de cette séparation a nécessité une maîtrise des mécanismes de sérialisation et de désérialisation, ainsi qu'une réflexion approfondie sur la validation des données externes. Ces compétences sont directement transférables à de nombreux autres contextes de développement, notamment dans le développement d'applications web ou de services distribués.

Optimisation et performance : Le développement d'un jeu, même simple, soulève des questions de performance qui ne se posent pas nécessairement dans d'autres types d'applications. La nécessité de maintenir une expérience utilisateur fluide nous a poussés à réfléchir aux implications de nos choix de structures de données et d'algorithmes. Cette réflexion nous a sensibilisés à l'importance de l'analyse de complexité et de l'optimisation précoce des parties critiques du code.

L'expérience nous a également appris l'importance de mesurer les performances réelles plutôt que de se fier aux intuitions. Cette approche empirique de l'optimisation est une compétence essentielle dans le développement logiciel professionnel, où les contraintes de performance peuvent être critiques pour le succès d'un projet.

Gestion de l'état et cohérence des données : Un jeu de rôle implique la gestion d'un état complexe qui évolue constamment en réponse aux actions du joueur. Maintenir la cohérence de cet état tout en permettant des interactions flexibles représente un défi technique significatif. Ce défi nous a confrontés aux problématiques générales de gestion d'état dans les applications interactives, problématiques que l'on retrouve dans

de nombreux autres contextes comme les interfaces utilisateur complexes ou les systèmes distribués.

Notre approche a consisté à centraliser la gestion de l'état tout en définissant des interfaces claires pour les modifications. Cette approche, inspirée des patterns architecturaux modernes, nous a permis de maintenir la cohérence tout en préservant la flexibilité nécessaire pour implémenter des mécaniques de jeu variées.

Collaboration et intégration continue : Le développement en équipe soulève des défis spécifiques liés à l'intégration du travail de plusieurs développeurs. L'utilisation d'un système de contrôle de version comme Git, bien que familière en théorie, s'est révélée plus complexe en pratique lorsqu'il s'agit de gérer des modifications simultanées sur des parties interdépendantes du code.

Cette expérience nous a sensibilisés à l'importance des bonnes pratiques de développement collaboratif : commits atomiques, messages de commit descriptifs, revues de code systématiques, et tests automatisés. Ces pratiques, essentielles dans le développement logiciel professionnel, ont été mises en œuvre de manière concrète dans le cadre de notre projet.

L'ensemble de ces défis et des solutions que nous avons développées constitue un apprentissage précieux qui dépasse largement le cadre spécifique de notre jeu de rôle. Les compétences acquises en matière d'architecture logicielle, de sécurité, de performance, et de collaboration sont directement applicables à de nombreux autres contextes de développement et constituent une base solide pour notre future carrière d'ingénieurs en génie logiciel.

3. Organisation et répartition des rôles

3.1 Présentation de l'équipe et apprentissage de Rust

Notre équipe de développement était composée de quatre étudiants en Master Génie Logiciel, tous débutants en programmation Rust au commencement de ce projet. Cette situation initiale, loin d'être un obstacle, s'est révélée être une opportunité d'apprentissage collective particulièrement enrichissante qui a renforcé la cohésion de l'équipe et favorisé le partage de connaissances.

Contexte d'apprentissage : Aucun membre de l'équipe n'avait d'expérience préalable significative avec le langage Rust avant le début de cette UE. Notre apprentissage s'est entièrement appuyé sur les cours dispensés par l'enseignant dans le cadre de l'UE Algorithme et Modélisation Avancée. Ces cours nous ont progressivement introduits aux concepts fondamentaux de Rust : le système de propriété (ownership), les mécanismes

d'emprunt (borrowing), la gestion de la mémoire sans garbage collector, et les spécificités syntaxiques du langage.

L'enseignant nous a particulièrement sensibilisés aux concepts de traits en Rust, qui constituent une alternative élégante à l'héritage traditionnel des langages orientés objet. Cette approche nous a permis de comprendre comment Rust favorise la composition plutôt que l'héritage, offrant une flexibilité et une sécurité accrues dans la conception de nos structures de données et de nos algorithmes. Les exemples pratiques fournis en cours nous ont aidés à saisir comment implémenter des comportements polymorphes sans recourir aux mécanismes d'héritage classiques.

Méthodologie d'apprentissage collaborative : Face à la nouveauté du langage pour tous les membres de l'équipe, nous avons adopté une approche d'apprentissage collaborative qui s'est révélée particulièrement efficace. Nous avons organisé des sessions de travail en groupe où chaque découverte ou difficulté rencontrée par un membre était immédiatement partagée avec les autres. Cette approche a permis d'accélérer l'apprentissage collectif et d'éviter que chaque membre reproduise individuellement les mêmes erreurs.

Les concepts les plus complexes de Rust, notamment la gestion des durées de vie (lifetimes) et les règles d'emprunt, ont fait l'objet de discussions approfondies au sein de l'équipe. Ces échanges nous ont permis de développer une compréhension commune des bonnes pratiques et d'établir des conventions de codage cohérentes pour l'ensemble du projet.

Adaptation aux spécificités de Rust : L'apprentissage de Rust nous a confrontés à un changement de paradigme significatif par rapport aux langages que nous connaissions précédemment. Le système de types strict de Rust et ses mécanismes de sécurité mémoire nous ont obligés à repenser notre approche de la programmation. Cette adaptation, bien qu'initialement challengeante, s'est révélée extrêmement formatrice et nous a sensibilisés à l'importance de la sécurité et de la robustesse dans le développement logiciel.

L'utilisation des traits pour définir des comportements communs nous a particulièrement marqués. Cette approche nous a permis de comprendre concrètement les avantages de la composition par rapport à l'héritage, et d'apprécier la flexibilité offerte par ce paradigme pour la conception d'architectures modulaires et extensibles.

3.2 Rôles spécifiques et contributions

Bien que tous les membres de l'équipe soient partis du même niveau de débutant en Rust, chacun a progressivement développé des spécialisations et assumé des responsabilités spécifiques qui ont contribué au succès du projet.

Olivier Fabre - Développeur Rust et Responsable Git : En tant que membre de l'équipe le plus ancien en âge, Olivier a naturellement pris le rôle de développeur et de responsable du dépôt Git. Ses responsabilités incluaient la mise en place de l'architecture initiale du projet, la définition des standards de codage, et la supervision des merge requests pour maintenir la qualité du code.

Malgré son statut de débutant en Rust, Olivier s'est rapidement approprié les concepts enseignés en cours et a su les appliquer efficacement dans le contexte du projet. Il a participé au codage de nombreuses fonctionnalités, en s'appuyant sur les exemples de traits et les possibilités de composition présentés par l'enseignant. Son rôle dans l'élaboration de la documentation et de la partie commune du rapport a été crucial pour maintenir la cohérence du projet et faciliter la collaboration au sein de l'équipe.

Sa gestion du dépôt Git a permis d'établir un workflow de développement efficace, avec des branches de fonctionnalités clairement définies et des processus de revue de code qui ont contribué à l'apprentissage collectif. Son approche méthodique a été particulièrement appréciée par l'équipe, notamment dans les moments où la complexité du langage Rust créait des difficultés techniques.

Tiago - Product Owner et Développeur Rust : En tant que Product Owner, Tiago a joué un rôle crucial dans la définition de la vision produit et la priorisation des fonctionnalités. Il a été responsable de la conception des mécaniques de jeu et de l'équilibrage du gameplay, s'assurant que chaque fonctionnalité apportait une valeur réelle à l'expérience de jeu.

Son approche de débutant en Rust s'est révélée être un atout pour la conception du jeu, car elle l'a poussé à privilégier des solutions simples et élégantes plutôt que des implémentations complexes. Tiago a particulièrement bien assimilé les concepts de traits enseignés en cours, qu'il a su appliquer de manière créative pour définir les comportements des différentes entités du jeu. Bien sûr, Tiago a aussi participé au développement de nombreuses fonctionnalités, apportant sa perspective de game designer à l'implémentation technique.

Son rôle de Product Owner lui a permis de maintenir une vision d'ensemble du projet tout en contribuant activement au développement. Cette double casquette s'est révélée particulièrement précieuse pour s'assurer que les choix techniques restaient alignés avec les objectifs fonctionnels du jeu.

Ndeye - Scrum Master et Développeur Rust : Ndeye a assumé le rôle de Scrum Master, facilitant l'organisation du travail et la communication au sein de l'équipe. Elle a été responsable de l'animation des cérémonies Scrum et de la résolution des obstacles qui pouvaient ralentir l'équipe, particulièrement ceux liés à l'apprentissage du langage Rust.

Son rôle de Scrum Master s'est révélé particulièrement important dans le contexte d'une équipe de débutants en Rust. Elle a su identifier les moments où l'équipe rencontrait des difficultés techniques liées au langage et organiser des sessions de travail collaboratif pour résoudre ces blocages. Sa capacité à maintenir la motivation de l'équipe face aux défis d'apprentissage a été essentielle pour le succès du projet.

Ndeye a également participé au codage de nombreuses fonctionnalités, en s'appuyant sur les enseignements reçus en cours. Son approche méthodique et sa capacité à documenter les solutions trouvées ont contribué à créer une base de connaissances partagée qui a bénéficié à toute l'équipe.

Amadou Bass - Développeur Rust : Amadou a contribué significativement au projet en se spécialisant dans l'implémentation des statistiques et des inventaires du jeu. Son travail a été essentiel pour créer une expérience de jeu cohérente et engageante.

Malgré son statut de débutant en Rust, Amadou a rapidement développé une expertise dans la gestion des structures de données complexes, en s'appuyant sur les concepts de traits et de composition enseignés en cours. Son approche pragmatique lui a permis de créer des systèmes robustes et extensibles pour la gestion des statistiques des personnages et des mécaniques d'inventaire.

Sa contribution au projet illustre parfaitement comment un apprentissage structuré de Rust peut permettre à des débutants de produire du code de qualité professionnelle. Son travail sur les systèmes de statistiques et d'inventaire a nécessité une compréhension approfondie des mécanismes de propriété et d'emprunt de Rust, qu'il a su maîtriser grâce aux enseignements reçus et à la pratique intensive dans le cadre du projet.

Cette expérience collective d'apprentissage de Rust nous a non seulement permis de mener à bien notre projet, mais nous a également sensibilisés à l'importance de l'apprentissage continu et de la collaboration dans le développement logiciel. Les compétences acquises en Rust, ainsi que les méthodes d'apprentissage collaboratif que nous avons développées, constituent un acquis précieux pour notre future carrière d'ingénieurs en génie logiciel.

Équipe de développement : - Olivier Fabre (upav2104042) - Développeur Rust et Responsable Git - Amadou Bass - Développeur Rust
- Tiago - Product Owner et Développeur Rust - Ndeye - Scrum Master et Développeur Rust

Dépôt GitHub : https://github.com/olfabre/ProjetRustS2

Master en Génie Logiciel - UE Algorithme et Modélisation Avancée