Rapport Projet Rust

Jeu de Rôle

Mohamed

Ibrahim

Nassim

Mouhamadou

Sommaire

➢ 1. Description du Projet

➢ Contexte pédagogique et objectifs :

Le jeu de rôle textuel développé dans ce projet est un terrain d'apprentissage concret pour les bases avancées de la programmation en Rust. Il simule un environnement de jeu logique, modélisé à partir d'entités interdépendantes : joueurs, zones, PNJ, objets, monstres, quêtes. Le joueur progresse dans un monde évolutif, influencé par ses actions et les règles internes prédéfinies.

→ Modélisation logique du jeu :

Chaque entité du jeu est représentée par une structure Rust dédiée, construite selon des principes d'encapsulation et de typage fort. Par exemple :

- Un `Player` contient son nom, sa classe, ses statistiques, son inventaire, sa position.
- Une `Zone` contient sa description, les directions possibles, les objets, PNJ ou monstres qu'elle contient.
- Les `NPC`, `Monster`, `Item`, `Quest` suivent des schémas similaires, tous centralisés par des modules individuels dans `src/`.

Les interactions entre entités sont implémentées via des `traits`, permettant d'uniformiser les comportements tout en conservant une logique propre à chaque type (par exemple, `Combatant` pour tout acteur impliqué dans un combat).

→ Mécanique de simulation autonome :

Le jeu n'est pas uniquement réactif : il intègre une logique d'évolution indépendante du joueur :

- Certains monstres apparaissent ou réapparaissent selon un cycle logique.
- Des objets peuvent être générés ou consommés par des actions de PNJ.
- Des zones deviennent accessibles ou modifiées une fois certaines conditions atteintes.

Cela simule un monde vivant, en cohérence avec l'objectif de "simulation logique du monde physique" du projet.

→ Approche algorithmique :

Des structures de données efficaces (vecteurs, hashmaps) ont été utilisées pour stocker les entités. L'algorithme de déplacement est basé sur un graphe de connexions entre zones, et le système de combat utilise un modèle de boucle d'attaque/défense avec priorités selon l'agilité.

Les conditions de validation des quêtes sont exprimées sous forme de prédicats appliqués sur l'état du joueur (inventaire, position, dialogue accompli). Des expressions `match` permettent de diriger le flux logique des événements selon des entrées utilisateur ou des états internes.

→ Simulation / Interaction / Extensibilité :

Le moteur Rust fonctionne sur des fichiers JSON définissant le monde. Il est donc possible de remplacer tout ou partie de l'univers du jeu (nouveaux objets, nouvelles zones, nouveaux PNJ) sans modifier le code. Ceci fait du moteur une base adaptable à tout type de jeu de rôle textuel.

Enfin, une attention particulière a été portée à la lisibilité du code, à la réutilisabilité des composants, et à la clarté des messages utilisateurs à l'écran, offrant une expérience à la fois stable, modulaire et enrichissante.

→ Objectif Général :

Ce projet a pour objectif de créer un jeu de rôle textuel en Rust, axé sur la simulation logique d'un monde cohérent et interactif. Il permet de consolider les compétences en modélisation, en conception algorithmique, et de maîtriser les paradigmes sécuritaires du langage Rust : gestion de la mémoire sans garbage collector, typage statique, fiabilité des exécutions.

→ Structure du Projet :

L'organisation suit la structure standard d'un projet Rust :

- `Cargo.toml` : décrit les dépendances et les métadonnées du projet (serde, serde_json).
- `src/` : contient les fichiers de code source modulaire.
- `data/` : contient les fichiers JSON décrivant le monde du jeu.
- `diagramme_relations.png` : schéma relationnel entre les entités (joueur, PNJ, objets, quêtes, zones).

→ Modules Rust et leur rôle :

- `main.rs` : point d'entrée principal, qui orchestre le chargement des données et le lancement du jeu.
- `data_loader.rs` : chargement générique des fichiers JSON en structures Rust.
- `game.rs` : moteur principal qui traite les interactions, transitions d'états et l'interface terminal.
- `player.rs` : définit les attributs et actions du joueur (déplacement, inventaire, interaction).
- `npc.rs` : gère les personnages non-joueurs, leurs dialogues et quêtes associées.
- `monster.rs` : modèle les monstres et intègre les mécaniques de combat.
- `item.rs` : structure les objets du jeu et leurs effets (soin, attaque, défense, etc.).
- `quest.rs` : modélise les objectifs secondaires et leur avancement logique.
- `zone.rs` : structure chaque lieu du jeu et les entités qu'il contient.

→ `Fichiers de Données :`

- `zones.json`, `npcs.json`, `monsters.json`, `items.json`, `quests.json`, `attributes.json` : ils décrivent l'univers du jeu indépendamment du code. Cela permet une extensibilité narrative sans recompilation.

→ `Fonctionnalités du Jeu :`

- Création d'un personnage à partir d'un profil prédéfini avec des attributs initiaux.
- Exploration de zones interconnectées, chacune avec ses descriptions, PNJ et monstres.
- Interactions avec les PNJ et objets, affectant les états du joueur.
- Système de quêtes évolutives dépendant du contexte.
- Mécanique de combat tour par tour avec gain de loot et expérience.
- Simulation logique (apparition de monstres, condition d'accès aux zones ou objets, etc.).

→ `Aspects techniques Rust mis en valeur :`

- Utilisation de `serde` pour désérialiser les données JSON.
- Traits pour modéliser les comportements communs (ex : `Interactable`, `Fighter`).
- Typage strict, sécurité mémoire et gestion des durées de vie.
- Tests unitaires et intégration pour assurer la fiabilité du jeu.

➢ 2. Répartition des Rôles

Notre groupe est composé de quatre membres : Ibrahim, Mohamed, Nassim et Mouhamadou. La répartition des tâches s'est faite selon les compétences et les préférences de chacun, tout en assurant une collaboration continue à travers un dépôt Git partagé.

→ `Ibrahim – Développement des Entités PNJ et Quêtes`

- Responsable du module `npc.rs` : gestion des dialogues et interaction avec le joueur.
- Intégration des quêtes dans `quest.rs` : création des conditions d'activation et de réussite.
- Participation à la rédaction de `npcs.json` et `quests.json`.

→ `Mohamed – Système de Combat et Monstres`

- Mise en place du module `monster.rs` avec statistiques et comportements de combat.
- A développé les règles de combat dans le moteur (`game.rs`).
- Réalisation du fichier `monsters.json` et intégration du loot des objets.

→ Mouhamadou – Conception du Monde et Navigation

- Création du module `zone.rs` pour gérer les zones et leurs connexions.
- Organisation du fichier `zones.json` avec logique de déplacement.
- Intégration de la carte dans le moteur du jeu.

→ Nassim – Intégration Générale et Affichage

- Développement de `main.rs` et orchestration des appels aux modules.
- Gestion de l'interface utilisateur texte, du menu principal et des commandes.
- Chargement dynamique via `data_loader.rs`.

Chaque membre a également participé aux tests, au débogage, et à l'écriture collaborative du rapport et de la documentation technique.

➢ 3. Réalisations personnelles :

→ Réalisations personnelles de Nassim

En tant que développeur principal de l'intégration générale du projet, j'ai eu en charge plusieurs éléments critiques du jeu :

1. Initialisation et orchestration du moteur de jeu

- J'ai développé le fichier `main.rs` qui joue le rôle de point d'entrée principal du jeu.
- Ce module initialise les composants essentiels : chargement des données (via `data_loader.rs`), création du personnage joueur, démarrage du moteur de jeu.
- J'ai conçu la boucle principale de jeu qui gère les actions disponibles, les transitions entre exploration, combat, dialogue et menu.

2. Interface utilisateur textuelle

- Mise en œuvre de l'affichage des choix de l'utilisateur, retours après actions, menus contextuels et progression dans l'aventure.
- Gestion d'une interface fluide avec instructions claires, manipulation d'entrées/sorties standards, et gestion des erreurs d'entrées.

3. Chargement dynamique des données JSON

- Utilisation de la bibliothèque `serde` pour désérialiser les données issues de `zones.json`, `items.json`, `npcs.json`, etc.
- Développement du module `data_loader.rs` pour fournir des fonctions génériques et réutilisables de chargement et de transformation des fichiers en structures Rust.

4. Intégration des modules entre eux

- J’ai connecté les modules indépendants (`player.rs`, `quest.rs`, `zone.rs`, etc.) via des interfaces cohérentes dans le moteur (`game.rs`).
- En assurant une logique fluide entre la lecture des données, l’implémentation fonctionnelle et l’interface utilisateur, j’ai joué un rôle clé dans la stabilité globale du jeu.

5. Résolution de bugs et tests

- J’ai écrit plusieurs tests unitaires pour garantir la validité des entrées utilisateur, la bonne exécution de la logique de quête et la robustesse des changements d’état.
- J’ai aussi effectué des sessions de débogage collaboratif pour corriger des incohérences dans la logique de déplacement et l’apparition des monstres.

Bilan personnel

Ce projet m’a permis de renforcer ma compréhension de Rust, notamment la gestion de la mémoire et des emprunts, le typage fort, et la sérialisation de données. Travailler sur l’intégration m’a poussé à concevoir une architecture cohérente et modulaire tout en gardant en tête l'expérience utilisateur. Ce fut un excellent exercice de conception logicielle, à la fois formateur et stimulant.

[Section à remplir individuellement par chaque membre, selon ses contributions personnelles détaillées.]

→ `Réalisations personnelles de Mohamed`

En tant que développeur du système de combat et des monstres, j’ai conçu et intégré les éléments de jeu liés à l’affrontement, aux ennemis et aux mécaniques de danger. Voici mes principales réalisations :

1. Conception du module `monster.rs`

- Définition des structures de monstres : nom, santé, force, loot.
- Intégration de comportements comme l’attaque, la gestion de la vie et la disparition après la mort.

2. Mise en œuvre du système de combat

- Intégration du combat dans `game.rs` : détection d’ennemis, choix d’action, priorités selon l’agilité.
- Gestion des tours de combat, des messages dynamiques, de la vérification des morts et du butin.

3. Données JSON – `monsters.json`

- Création du fichier décrivant tous les monstres rencontrables (nom, HP, caractéristiques, récompenses).
- Validation de la cohérence avec les zones et les quêtes.

4. Tests et équilibre du gameplay

- Test des combats contre différents profils de joueurs.
- Ajustement des statistiques pour éviter des combats trop faciles ou impossibles.

Bilan personnel Ce travail m'a appris à concevoir un système d'interaction dynamique et à modéliser la logique d'un combat. J'ai renforcé mes compétences Rust en utilisant les `traits` pour mutualiser les comportements, en structurant des actions conditionnelles et en manipulant des données complexes. L'aspect équilibrage m'a également sensibilisé à l'importance du gameplay dans le ressenti utilisateur.

→ Réalisations personnelles de Ibrahim

J'ai été chargé de la conception des personnages non-joueurs (PNJ) et de l'intégration des quêtes, éléments essentiels de l'immersion et de la progression.

1. Développement du module `npc.rs`

- Création des structures représentant les PNJ, avec noms, descriptions, dialogues, quêtes associées.
- Ajout de la logique d'interaction (déclenchement de quête, dialogue à choix, conditions dynamiques).

2. Gestion des fichiers `npcs.json` et `quests.json`

- Écriture des dialogues dans les PNJ.
- Création de quêtes liées à l'évolution du joueur (ex : ramener un objet, éliminer un monstre).

3. Implémentation des quêtes dans `quest.rs`

- Définition des conditions de démarrage, de progression et de validation.
- Association avec les PNJ dans les zones concernées.

4. Coordination avec les autres modules

- Collaboration avec `player.rs` pour la mise à jour de l'état du joueur après une quête.
- Intégration dans `game.rs` pour afficher les dialogues ou les résultats de quête.

Bilan personnel Ce travail m'a appris à structurer des scénarios interactifs et à les connecter à la logique du monde. J'ai compris comment Rust peut faciliter la gestion de dépendances entre entités via un code structuré. Ce fut aussi un bon exercice de narration interactive.

→ Réalisations personnelles de Mouhamadou

Responsable de la conception du monde et de la navigation, j'ai modélisé la carte du jeu et la logique de déplacement entre les zones.

1. Module `zone.rs`

- Création de structures représentant chaque zone : nom, description, directions accessibles, PNJ et monstres présents.
- Définition de méthodes permettant de lister les actions disponibles dans une zone.

2. Données du monde – `zones.json`

- Organisation d'un monde cohérent avec connexions logiques entre zones (Nord, Sud, etc.).
- Description détaillée des lieux, ambiance, contenu.

3. Intégration dans le moteur de jeu

- Ajout du déplacement dans `game.rs`.
- Affichage des sorties possibles et transitions après action.

4. Cartographie et documentation

- Participation à la création du schéma de relations `diagramme_relations.png`.
- Documentation de la logique de transition entre zones.

Bilan personnel Cette partie m'a permis de pratiquer les liens entre structures Rust, gestion de données externes et logique utilisateur. J'ai découvert l'intérêt de la modularité du code et du typage strict pour éviter les erreurs d'interconnexion. C'était une tâche exigeante mais essentielle pour le réalisme du jeu.