Zero To Hero – Manuel d'utilisation

Bienvenue dans Zero To Hero, un jeu de rôle en mode console écrit en Rust. Ce manuel d'utilisation sert aussi de README du projet ; il explique comment installer, compiler, jouer, tester et étendre le jeu.

Présentation

Zero To Hero est un projet universitaire (M1 ILSEN – UCE *Algorithme & Modélisation Avancée*) dont l'objectif est de démontrer :

- La modélisation logique d'un monde **orienté simulation** ;
- L'usage des **traits**, de l'**emprunt** et du **typage strict** de Rust ;
- La pratique du **développement piloté par les tests** (TDD) ;
- La séparation des **données (XML)** et du **moteur (Rust)** pour faciliter la création de contenu.

Le jeu repose sur une boucle principale qui interroge le joueur par texte, met à jour le monde et exécute des événements programmés.

Fonctionnalités

- **Création de personnage** : nom, âge, sexe, attributs (santé, faim, force, aura, argent…)
- **Monde persistant** : gestion du temps, lieux connectés, PNJ, objets.
- **Menus** : déplacement, inventaire, statut détaillé, scénarios, historique.
- **Scénarios XML** : choix multiples affectant le personnage et son environnement.
- **Événements logiques** : rencontres aléatoires, raids de police, réunions planifiées.
- **Tests unitaires complets** sur chaque module.

Installation

Prérequis

Outil	Version minimum	Commentaire
Rust	1.75	Installez via `rustup`
Cargo	Inclus	Gestionnaire de paquets & build

Compilation & Exécution

```
# Compilation en mode debug
$ cargo build

# Exécution
$ cargo run
```

La première exécution télécharge automatiquement les dépendances déclarées dans `Cargo.toml` (notamment **quick_xml** pour le parsing des scénarios).

Paramètres de compilation utiles

- `--release` Compile avec optimisations :

```
$ cargo build --release
```

- `RUST_BACKTRACE=1` Affiche la pile en cas de panic :

```
$ RUST_BACKTRACE=1 cargo run
```

Lancement des tests

Le projet comprend plus de 80 tests unitaires couvrant la quasi-totalité du code.

```
$ cargo test
```

Tous les tests doivent passer avant de valider une MR.

Prise en main

Création du personnage

Au démarrage, le jeu vous propose :

```
Entrez le nom de votre personnage [Jean] :
Entrez l'âge de votre personnage [18] :
Choisissez votre sexe :
1) Homme  2) Femme  3) Autre
```

Les crochets `[]` indiquent la valeur par défaut si vous appuyez simplement sur **Entrée**.

Interface & commandes

Une fois la partie lancée, l'écran est actualisé comme suit :

```
+-------------------------------------------------------------------------
--------+
|     O         | Nom : Jean   | Force :  50/100
|
|    /|\        | Sexe : Homme| Aura   :  10/100
|
|    / \        | Age :  18    | Argent : 150$
|
|               | Santé :  75/100| Exp : 1 xp
|
|               | Faim  : 050/100| Level : 1
|
+-------------------------------------------------------------------------
--------+
| Jour 0 – 00:00            Zone : « Home »
+-------------------------------------------------------------------------
--------+
1) Se déplacer
2) Interagir avec le lieu
3) Inventaire
4) Statut (détails)
5) Quitter
```

- **Se déplacer** — Affiche la liste des lieux adjacents.
- **Interagir avec le lieu** — Charge le scénario XML associé à la zone.
- **Inventaire** — Liste, utilise ou jette vos objets.
- **Statut** — Vue détaillée des attributs et de la progression.

👉 Entrez simplement le **numéro** puis **Entrée** pour valider un choix.

Commandes clavier rapides

Touche	Effet
`0`	Retour au menu précédent
`Ctrl+C`	Quitter immédiatement (panic)

Système de scénarios

Chaque lieu référence un fichier XML dans le dossier `scenarios/`. Exemple pour le **centre-ville** :

```
match self.player.current_place {
    1 => "scenarios/city.xml",
    // …
}
```

Le moteur lit le `<scenario>` courant, affiche sa `<description>` puis propose les `<choice>` imbriqués dans `<possible_scenario_id>`.

À la sélection, les **effets** (`<effect>`) sont appliqués : gain/perte d'argent, santé, faim, etc.

Ajouter du contenu

Ajouter un nouveau scénario

1. Créez `scenarios/nom_du_lieu.xml` basé sur la structure :

```
<scenarios>
    <scenario>
        <_id>identifiant_unique</_id>
        <description>Texte affiché…</description>
        <effect>
            <e>health - 10</e>
            <e>money + 50</e>
        </effect>
        <action>
            <possible_scenario_id>
                <choice>
                    <id>autre_scenario</id>
                    <text>Aller ailleurs</text>
                </choice>
            </possible_scenario_id>
        </action>
    </scenario>
</scenarios>
```

2. Référencez-le dans `MoteurDeJeu::get_scenario_file()`.

3. Ajoutez éventuellement des tests dans `tests/scenario_test.rs`.

Ajouter un lieu / PNJ / objet

- **Lieu** — Créez une instance `Lieu::new` dans `main.rs`, ajoutez-la au vecteur `places` et configurez `connected_places`.
- **PNJ** — Ajoutez la structure `PNJ` au vecteur `npcs` du `Monde`.
- **Objet** — Déclarez-le dans un scénario ou attribuez-le à un PNJ / Lieu.

Structure du dépôt

```
├── Cargo.toml              # Dépendances et configuration
├── src/
│   ├── characters/         # Joueur, PNJ et logique partagée
│   ├── game_engine/        # Boucle principale et menus
│   ├── world/              # Gestion du monde & du temps
│   ├── items/              # Inventaire & objets utilisables
│   ├── scenario/           # Loader XML & gestion des effets
│   └── utils/              # Types et énumérations communes
├── scenarios/              # Contenu XML
└── tests/                  # Tests unitaires (cargo test)
```

> 🗄️ **target/** est ignoré par Git ; il contient les binaires compilés.

Licence

Ce projet est distribué sous licence **CERI**

> *Bon jeu, et n'hésitez pas à contribuer !*