Documentation

Sommaire

Fichier `interface.rs`

Le fichier `interface.rs` est le point d'entrée du jeu et gère l'interface utilisateur en utilisant la bibliothèque Cursive.

Fonctions principales

1. **Gestion du menu principal**
    - `main_menu_screen()` : Affiche le menu principal avec les options "Jouer" et "Quitter"
    - `choix_nom()` : Permet au joueur de choisir son nom de personnage
2. **Écran de jeu**

- `game_screen()` : Affiche l'interface principale du jeu avec les informations du joueur et les options disponibles
- `move_screen()` : Gère le déplacement du joueur entre les différents lieux
- `actions_screen()` : Affiche les actions possibles dans le lieu actuel
- `personnage_screen()` : Affiche les informations et options du personnage

3. **Gestion des interactions**
    - `analyse_screen()` : Permet d'analyser la zone actuelle
    - `recolter_screen()` : Gère la récolte de ressources
    - `pnj_screen()` : Affiche les PNJ disponibles pour interagir
    - `combat_screen()` : Gère le système de combat
4. **Gestion des PNJ**
    - `create_dialog_pnj()` : Crée une interface pour interagir avec un PNJ
    - `create_dialog_action_pnj()` : Affiche les actions possibles avec un PNJ
    - `create_dialogue_parler_pnj()` : Gère les dialogues avec un PNJ
    - `create_dialog_commerce_pnj()` : Gère le commerce avec un PNJ
5. **Gestion du commerce**
    - `create_dialog_acheter_pnj()` : Interface d'achat d'objets
    - `create_dialog_vendre_pnj()` : Interface de vente d'objets
    - Fonctions spécifiques pour la vente de différents types d'objets (consommables, ressources, équipements)
6. **Gestion du combat**
    - `combat_attaque_screen()` : Interface des attaques en combat
    - `combat_consommable_screen()` : Interface d'utilisation des consommables en combat
    - `create_dialog_victoire_combat()` : Gère les récompenses de combat
    - `create_dialog_defaite_combat()` : Gère la défaite en combat
7. **Gestion du personnage**

    - `informations_screen()` : Affiche les informations du personnage
    - `statistiques_screen()` : Gère les statistiques du personnage
    - `equipement_screen()` : Gère l'équipement du personnage
    - `attaques_screen()` : Gère les attaques du personnage
8. **Gestion de l'inventaire**
    - `inventaire_screen()` : Interface principale de l'inventaire
    - `inventaire_consommable()` : Gestion des consommables
    - `inventaire_ressources()` : Gestion des ressources
    - `inventaire_equipements()` : Gestion des équipements
    - `inventaire_parchemin()` : Gestion des parchemins
9. **Gestion des quêtes**
    - `quetes_screen()` : Affiche la liste des quêtes
    - `create_quete_dialog()` : Affiche les détails d'une quête
10. **Menu et utilitaires**
    - `menu_screen()` : Affiche le menu du jeu

Fichier `json_manager.rs`

Le fichier `json_manager.rs` gère la sérialisation et la désérialisation des données liées au jeu. Il s'occupe de la lecture et de l'écriture du fichier JSON (masterFile.json), qui contient toutes les données persistantes du jeu : informations sur le joueur, les PNJ, les ennemis, les objets, les quêtes, etc.

Contenu et Structure du fichier

Énumération `Item`

Représente un objet générique du jeu qui peut être de trois types :

- `Ressource(Ressource)`
- `Consommable(Consommable)`
- `Equipement(Equipement)`

Méthodes :

- `get_ressources()` : Récupère les ressources associées à l'objet.
- `get_nom()` : Récupère le nom de l'objet.

Classe principale : `MasterFile`

`MasterFile` est la structure principale de ce fichier et contient toutes les entités du jeu. Elle représente l'état global du jeu à un moment donné. Elle est composée de plusieurs vecteurs, chacune représentant un type d'entité du jeu.

Champs :

- **Joueur** : Représente le joueur actuel.
- **Pnj** : Une liste de PNJ (personnages non-joueurs) présents dans le jeu.
- **Ennemie** : Une liste d'ennemis dans le jeu.
- **Lieu** : Une liste des différents lieux (ou zones) du jeu.
- **Quete** : Une liste des quêtes disponibles.
- **Consommable** : Liste des objets consommables comme des potions, etc...
- **Ressource** : Liste des ressources collectables (clé, minerai, etc...).
- **Equipement** : Liste des équipements que le joueur peut porter ou utiliser (armures, armes, etc.).
- **Parchemin** : Liste des parchemins (objet débloquant des attaques) dans le jeu.
- **Attaque** : Liste des attaques disponibles pour le joueur ou les ennemis.

Fonctions de gestion du fichier :

1. **Gestion de l'instance** :
    - `get_instance()` : Retourne une référence à une instance unique (singleton) de `MasterFile`, en la créant à partir du fichier JSON si nécessaire.
2. **Gestion des données** :
    - `sauvegarder()` : Sérialise l'état actuel de `MasterFile` en JSON et le sauvegarde dans un fichier.
    - `recharger()` : Recharge les données du fichier JSON et met à jour les entités du jeu.
    - `newGame()` : Réinitialise le jeu avec les données du fichier

Game.json.

Fonctions d'accès aux entités :

1. **Lieux** :
    - `prendre_lieu_id()` : Retourne un lieu par son ID.
    - `prendre_lieu_mut_id()` : Retourne une référence mutable à un lieu.
2. **Joueur** :
    - `get_joueur()` : Retourne une copie du joueur.
    - `get_joueur_mut()` : Retourne une référence mutable au joueur.
3. **PNJ** :
    - `prendre_pnj_id()` : Retourne un PNJ par son ID.
    - `prendre_pnj_id_string()` : Retourne un PNJ par son ID (version avec String).
4. **Ennemis** :
    - `prendre_ennemie_id()` : Retourne un ennemi par son ID.
5. **Objets** :
    - `prendre_consommable_id()` : Retourne un consommable par son ID.
    - `prendre_ressource_id()` : Retourne une ressource par son ID.
    - `prendre_equipement_id()` : Retourne un équipement par son ID.
    - `prendre_parchemin_id()` : Retourne un parchemin par son ID.
    - `prendre_attaque_id()` : Retourne une attaque par son ID.
    - `prendre_item_id()` : Retourne un objet générique par son ID.
6. **Quêtes** :
    - `prendre_quete_id()` : Retourne une quête par son ID.
    - `prendre_quete_mut()` : Retourne une référence mutable à une quête.

Fonctions de transaction :

1. **Commerce** :
    - `acheter()` : Permet au joueur d'acheter un objet d'un PNJ.

- `vendre()` : Permet au joueur de vendre un objet.
- `voler()` : Permet au joueur de voler un objet d'un PNJ.

Fichier `structs.rs`

Le fichier `structs.rs` contient les structures fondamentales utilisées dans tout le jeu. Il regroupe les entités de base comme les entités, les personnages et les ressources.

Variables globales

- `CHANCE_CRITIQUE` : Variable statique mutex pour gérer la chance de coup critique dans le jeu.

Entite

Représente une entité générique du jeu. C'est la base commune à tous les objets, personnages, lieux, etc.

Champs

- `id: String` — Identifiant unique.
- `description: String` — Description de l'entité.
- `nom: String` — Nom affiché.

Implémentation

- Implémente `Display` pour un affichage formaté :
  `"Entite : id = {id}, description = {description}, nom = {nom}"`

EquipementType

Enumération représentant les emplacements où un équipement peut être porté.

Variants

- `Arme`
- `Casque`
- `Plastron`
- `Gants`
- `Jambieres`
- `Bottes`

Implémentation

- Implémente `Display` pour renvoyer le nom de chaque variant.

Personnage

Représente un personnage (joueur ou PNJ) dans le jeu, avec ses caractéristiques, ses équipements, son inventaire et ses attaques.

Champs

- `entite: Entite`
- `pv_actuel`, `pv_max`, `force`, `dexterite`, `intelligence`, `vitesse`, `esquive`, `chance`, `resistance_physique`, `resistance_magique` : `u16`
- `attaques: Vec<String>`
- `equipement: HashMap<EquipementType, Option<String>>`
- `inventaire: HashMap<String, u32>`

Méthodes

- `str_attaques()`, `str_equipement()`, `str_inventaire()` : chaînes lisibles.
- `get_equipement()` : retourne une copie de l'équipement.
- `get_inventaire()` : retourne une copie de l'inventaire.

Calculs de statistiques

- `calcul_force()`, `calcul_dexterite()`, `calcul_intelligence()`, `calcul_vitesse()`, `calcul_esquive()`, `calcul_chance()`, `calcul_resistance_physique()`, `calcul_resistance_magique()` : intègrent les bonus fixes et en pourcentage de l'équipement.

Mécaniques de combat

- `attaque(&self, attaque: &Attaque) -> Vec<u16>` : calcule les dégâts en fonction du type d'arme et applique les bonus d'équipement et de chance critique.
- `attaque_base(&self) -> Vec<u16>` : attaque simple basée sur l'arme équipée.
- `defense(&self, degats_recus: &Vec<u16>) -> u16` : applique les résistances et l'esquive.

Implémentation

- `Display` affiche toutes les stats, l'équipement et l'inventaire sous forme textuelle.

Rarete

Enumération des niveaux de rareté d'un objet.

Variants

- `Commun` (40%)
- `PeuCommun` (30%)
- `Rare` (20%)
- `TresRare` (10%)
- `Epique` (1%)
- `Legendaire` (0.5%)
- `Mythique` (0.01%)
- `Divin` (0.001%)

Méthodes

- `get_value_rarete() -> f32` : retourne un facteur de rareté correspondant au pourcentage.
- Implémente `Display` pour un affichage simple du nom de la rareté.

Ressource

Structure de base représentant une ressource dans le jeu (matière, ingrédient, objet...).

Champs

- `entite: Entite`
- `prix: u32`
- `ressource: HashMap<String, u32>` — composants nécessaires
- `rarete: Rarete`

Méthodes

- `new(entite, prix, ressource, rarete)` : constructeur.
- `get_id()`, `get_nom()`, `get_description()`, `get_prix()`, `get_ressource()`, `get_rarete()`
- `str_ressource()` : renvoie une chaîne `"clé: quantité, ..."`
- `get_value_rarete()` : retourne la valeur de rareté sous forme de f32.

Implémentation

- `Display` affiche le contenu de la ressource de façon complète.

Fichier `consommable.rs`

Le fichier `consommable.rs` définit la structure des objets consommables dans le jeu, comme les potions ou autres objets à usage unique.

Structure `Consommable`

La structure `Consommable` représente un objet qui peut être consommé par le joueur pour obtenir des effets.

Attributs

- `ressource` : Une instance de `Ressource` qui contient les informations de base de l'objet (nom, description, prix, etc.)
- `effets` : Un vecteur de `u16` contenant 9 valeurs représentant les différents effets de l'objet

Méthodes principales

1. **Constructeur**
    - `new(ressource: Ressource, effets: Vec<u16>)` : Crée un nouveau consommable. Vérifie que le nombre d'effets est exactement 9.
2. **Getters basiques**
    - `get_id()` : Retourne l'ID du consommable
    - `get_description()` : Retourne la description du consommable
    - `get_nom()` : Retourne le nom du consommable
    - `get_prix()` : Retourne le prix du consommable
    - `get_rarete()` : Retourne la rareté du consommable
    - `get_ressource()` : Retourne la ressource associée au consommable
    - `get_ressources()` : Retourne les ressources associées au consommable
    - `get_effets()` : Retourne le vecteur des effets du consommable
3. **Méthodes spéciales**
    - `get_value_rarete()` : Retourne une valeur flottante représentant la rareté de l'objet selon une échelle prédéfinie
    - `str_effets()` : Convertit le vecteur d'effets en une chaîne de caractères formatée

Implémentation de Display

La structure implémente le trait `Display` pour permettre l'affichage formaté d'un consommable, incluant sa ressource et ses effets.

Fichier `parchemin.rs`

Le fichier `parchemin.rs` définit la structure des parchemins dans le jeu, qui sont des objets permettant d'apprendre de nouvelles attaques.

Structure `Parchemin`

La structure `Parchemin` représente un objet qui permet d'apprendre une nouvelle attaque au joueur.

Attributs

- `ressource` : Une instance de `Ressource` qui contient les informations de base de l'objet (nom, description, prix, etc.)
- `attaque` : Une chaîne de caractères représentant l'ID de l'attaque que le parchemin permet d'apprendre

Méthodes principales

1. **Constructeur**
    - `new(ressource: Ressource, attaque: String)` : Crée un nouveau parchemin avec une ressource et une attaque associée
2. **Getters basiques**
    - `get_id()` : Retourne l'ID du parchemin
    - `get_description()` : Retourne la description du parchemin
    - `get_nom()` : Retourne le nom du parchemin
    - `get_prix()` : Retourne le prix du parchemin
    - `get_rarete()` : Retourne la rareté du parchemin
    - `get_ressource()` : Retourne la ressource associée au parchemin
    - `get_ressources()` : Retourne les ressources associées au parchemin
    - `get_attaque()` : Retourne l'ID de l'attaque associée au parchemin
3. **Méthodes spéciales**

- `get_value_rarete()` : Retourne une valeur flottante représentant la rareté de l'objet selon une échelle prédéfinie

Implémentation de Display

La structure implémente le trait `Display` pour permettre l'affichage formaté d'un parchemin, incluant sa ressource et l'attaque associée.

Fichier `equipement.rs`

Le fichier `equipement.rs` définit les structures liées aux équipements dans le jeu, qui sont des objets que le joueur peut porter pour obtenir des bonus de statistiques.

Énumérations

`Categorie`

Représente la catégorie d'un équipement :

- `Arme(Arme)` : Une arme
- `Armure(Armure)` : Une pièce d'armure

`Arme`

Types d'armes disponibles :

- `ArmeMelee` : Arme de mêlée
- `ArmeDistance` : Arme à distance
- `ArmeMagie` : Arme magique

`Armure`

Types d'armures disponibles :

- `Casque` : Protection de la tête
- `Plastron` : Protection du torse

- `Gants` : Protection des mains
- `Jambieres` : Protection des jambes
- `Bottes` : Protection des pieds

Structure `Equipement`

La structure `Equipement` représente un objet que le joueur peut équiper pour obtenir des bonus de statistiques.

Attributs

- `ressource` : Une instance de `Ressource` qui contient les informations de base de l'objet
- `bonus_pv`, `bonus_force`, `bonus_dexterite`, etc. : Bonus de statistiques fixes
- `pourcent_bonus_pv`, `pourcent_bonus_force`, etc. : Bonus de statistiques en pourcentage
- `categorie` : La catégorie de l'équipement (arme ou armure)

Méthodes principales

1. **Getters basiques**
    - `get_id()`, `get_description()`, `get_nom()`, `get_prix()`
    - `get_ressource()`, `get_ressources()`
    - `get_rarete()`, `get_value_rarete()`
2. **Getters des bonus**
    - `get_bonus_pv()`, `get_bonus_force()`, etc.
    - `get_pourcent_bonus_pv()`, `get_pourcent_bonus_force()`, etc.
3. **Getters de catégorie**
    - `get_categorie()` : Retourne la catégorie de l'équipement

Implémentation de Display

La structure implémente le trait `Display` pour permettre l'affichage formaté d'un équipement, incluant sa ressource, tous ses bonus et sa catégorie.

Fichier `lieu.rs`

Le fichier `lieu.rs` définit la structure des lieux dans le jeu, qui sont des zones que le joueur peut explorer.

Énumération `Meteo`

Représente les différents types de météo possibles dans un lieu :

- `Soleil` : Temps ensoleillé
- `Pluie` : Temps pluvieux
- `Neige` : Temps neigeux
- `Interieur` : Lieu intérieur (pas de météo)

Structure `Lieu`

La structure `Lieu` représente une zone explorable du jeu.

Attributs

- `entite` : Structure de base contenant l'ID, le nom et la description du lieu
- `destinations` : Liste des IDs des lieux accessibles depuis celui-ci
- `meteo` : Type de météo du lieu
- `contient_ressources` : HashMap des ressources disponibles avec leurs quantités
- `contient_ennemies` : HashMap des ennemis présents avec leurs niveaux possibles
- `contient_pnj` : Liste des IDs des PNJ présents dans le lieu

Méthodes principales

1. **Constructeur**
    - `new()` : Crée un nouveau lieu avec tous ses paramètres
2. **Getters basiques**
    - `get_id()`, `get_description()`, `get_nom()`
    - `get_destinations()`, `get_meteo()`

    - `get_contient_ressources()`, `get_contient_ennemies()`, `get_contient_pnj()`
3. **Gestion des ressources**
    - `remove_contient_ressources()` : Retire des ressources du lieu
    - `recolter_item()` : Permet au joueur de récolter des ressources
4. **Gestion des ennemis**
    - `synchro_ennemie()` : Synchronise les statistiques d'un ennemi en fonction du niveau du joueur
5. **Fonctions utilitaires**
    - `str_destinations()` : Formate la liste des destinations
    - `str_contient_ressources()` : Formate la liste des ressources
    - `str_contient_ennemies()` : Formate la liste des ennemis
    - `str_contient_pnj()` : Formate la liste des PNJ

Implémentation de Display

La structure implémente le trait `Display` pour permettre l'affichage formaté d'un lieu, incluant toutes ses caractéristiques et son contenu.

Fichier `pnj.rs`

Le fichier `pnj.rs` définit la structure des personnages non-joueurs (PNJ) dans le jeu, qui sont des entités avec lesquelles le joueur peut interagir.

Structure `Pnj`

La structure `Pnj` représente un personnage non-joueur dans le jeu.

Attributs

- `personnage` : Structure de base contenant les caractéristiques du PNJ
- `dialogues` : Liste des IDs des dialogues possibles avec ce PNJ
- `commerce_table` : HashMap des items disponibles à l'achat avec leurs prix

Méthodes principales

1. **Getters basiques**
    - `get_id()`, `get_description()`, `get_nom()`
    - `get_inventaire()`, `get_dialogues()`, `get_commerce_table()`
2. **Gestion de l'inventaire**
    - `remove_inventaire()` : Retire des items de l'inventaire du PNJ
3. **Gestion des dialogues**
    - `afficher_dialogue()` : Affiche le texte d'un dialogue
    - `get_dialogue_a_jouer()` : Récupère le prochain dialogue à jouer en fonction des quêtes
    - `terminer_quete_a_enlever()` : Met à jour le statut d'une quête à terminer
4. **Gestion du commerce**
    - `remove_item_commerce_table()` : Retire des items de la table de commerce
5. **Fonctions utilitaires**
    - `str_dialogues()` : Formate la liste des dialogues
    - `str_commerce_table()` : Formate la table de commerce

Implémentation de Display

La structure implémente le trait `Display` pour permettre l'affichage formaté d'un PNJ, incluant ses caractéristiques, dialogues et table de commerce.

Fichier `attaque.rs`

Le fichier `attaque.rs` définit la structure des attaques dans le jeu, qui sont des actions que le joueur ou les ennemis peuvent utiliser en combat.

Structure `Attaque`

La structure `Attaque` représente une attaque qui peut être utilisée en combat.

Attributs

- `entite` : Structure de base contenant l'ID, le nom et la description de l'attaque
- `degats` : Dégâts de base de l'attaque
- `pourcent_bonus_degats` : Bonus de dégâts en pourcentage
- `categorie` : Type d'arme associé à l'attaque

Méthodes principales

1. **Getters basiques**
    - `get_id()` : Retourne l'ID de l'attaque
    - `get_description()` : Retourne la description de l'attaque
    - `get_nom()` : Retourne le nom de l'attaque
2. **Getters des caractéristiques de combat**
    - `get_degats()` : Retourne les dégâts de base
    - `get_pourcent_bonus_degats()` : Retourne le bonus de dégâts en pourcentage
    - `get_categorie()` : Retourne le type d'arme associé

Implémentation de Display

La structure implémente le trait `Display` pour permettre l'affichage formaté d'une attaque, incluant son entité, ses dégâts, son bonus et sa catégorie.

Fichier `joueur.rs`

Le fichier `joueur.rs` définit la structure du joueur et toutes ses fonctionnalités dans le jeu.

Structure `Joueur`

La structure `Joueur` représente le personnage contrôlé par le joueur, avec toutes ses caractéristiques et capacités.

Attributs

- `personnage` : Structure de base contenant les caractéristiques du personnage
- `position` : ID du lieu actuel du joueur
- `pronom` : Pronom utilisé pour le joueur
- `niveau` : Niveau actuel du joueur
- `temps` : Temps de jeu écoulé
- `reputations` : Liste des réputations du joueur
- `xp` : Expérience actuelle
- `multiplicateur_xp` : Multiplicateur d'expérience
- `points_competence` : Points de compétence disponibles
- `quetes` : Liste des quêtes actives

Méthodes principales

1. **Gestion des statistiques**
    - Getters et setters pour toutes les statistiques (PV, force, dextérité, etc.)
    - `ajout_point_stat()` : Ajoute un point de compétence à une statistique
    - `reset_stats()` : Réinitialise les statistiques
2. **Gestion de l'équipement**
    - `get_equipement()`, `add_equipement()`, `remove_equipement()`
    - `get_categorie_arme()` : Récupère le type d'arme équipée
3. **Gestion de l'inventaire**
    - `get_inventaire()`, `add_inventaire()`, `remove_inventaire()`
    - `recup_ressources()`, `recup_consommables()`, `recup_equipements()`, `recup_parchemins()`
    - `demantelement()` : Permet de démanteler un objet en ressources
4. **Gestion des quêtes**
    - `get_quetes()`, `add_quete()`, `remove_quete()`
    - `ajout_quete_joueur()` : Ajoute une nouvelle quête au joueur
    - `suivi_quete()` : Gère la progression d'une quête
    - `completion_quete()` : Marque une quête comme terminée

5. **Gestion du combat**
    - `degats_recus_net()` : Calcule les dégâts reçus après réduction
    - `application_degats()` : Applique les dégâts au joueur
    - `ajout_recompense_inventaire()` : Ajoute les récompenses de combat à l'inventaire
6. **Gestion des objets**
    - `utiliser_item()` : Utilise un objet consommable
    - `utiliser_parchemin()` : Utilise un parchemin pour apprendre une attaque
    - `appliquer_effets_items()` : Applique les effets d'un objet
7. **Gestion du déplacement**
    - `get_position()`, `set_position()`
    - `deplacement()` : Déplace le joueur vers un nouveau lieu
8. **Gestion de l'expérience et du niveau**
    - `get_xp()`, `add_xp()`, `set_xp()`
    - `get_niveau()`, `add_niveau()`
    - `get_points_competence()`, `set_points_competence()`

Implémentation de Display

La structure implémente le trait `Display` pour permettre l'affichage formaté d'un joueur, incluant toutes ses caractéristiques et son état actuel.

Fichier `ennemie.rs`

Le fichier `ennemie.rs` définit la structure des ennemis dans le jeu, qui sont des entités hostiles que le joueur peut combattre.

Structure `Ennemie`

La structure `Ennemie` représente un ennemi que le joueur peut affronter en combat.

Attributs

- `personnage` : Structure de base contenant les caractéristiques de l'ennemi
- `mod_pv`, `mod_force`, `mod_dexterite`, etc. : Modificateurs de statistiques
- `xp` : Expérience donnée au joueur lors de la défaite
- `dialogues` : Liste des dialogues possibles de l'ennemi
- `droptable` : Table de loot avec les objets et leurs chances de drop

Méthodes principales

1. **Gestion des statistiques**
    - Getters et setters pour toutes les statistiques de base
    - Getters et setters pour tous les modificateurs de statistiques
2. **Gestion du combat**
    - `degats_recus_net()` : Calcule les dégâts reçus après réduction
    - `application_degats()` : Applique les dégâts à l'ennemi et gère sa mort
    - `combat()` : Gère l'attaque de l'ennemi
    - `attaque()` : Calcule les dégâts d'une attaque
3. **Gestion du loot**
    - `lootable()` : Calcule les objets que l'ennemi peut donner en récompense
    - `get_droptable()` : Récupère la table de loot
4. **Fonctions utilitaires**
    - `str_dialogues()` : Formate les dialogues en chaîne de caractères
    - `str_drop_table()` : Formate la table de loot en chaîne de caractères

Implémentation de Display

La structure implémente le trait `Display` pour permettre l'affichage formaté d'un ennemi, incluant ses caractéristiques, son expérience, ses dialogues et sa table de loot.

Fichier `quete.rs`

Le fichier `quete.rs` définit les structures liées aux quêtes dans le jeu, qui sont des objectifs que le joueur peut accomplir pour obtenir des récompenses.

Énumérations

`StatutQuete`

Représente l'état actuel d'une quête :

- `NonCommencee` : La quête n'a pas encore été commencée
- `EnCours` : La quête est en cours d'accomplissement
- `Terminee` : La quête a été complétée

`FinDeQuete`

Représente les différentes façons dont une quête peut se terminer :

- `Combat(String)` : Terminer un combat contre un ennemi spécifique
- `Dialogue(String)` : Avoir une conversation avec un PNJ spécifique
- `Obtention(String)` : Obtenir un objet spécifique
- `Interaction(String)` : Interagir avec un élément spécifique du jeu

Structure `Quete`

La structure `Quete` représente une quête que le joueur peut accomplir.

Attributs

- `entite` : Structure de base contenant l'ID, le nom et la description de la quête
- `lieu` : ID du lieu où la quête peut être accomplie
- `recompense` : HashMap des récompenses avec leurs quantités
- `quetes_suivantes` : Liste des IDs des quêtes qui seront débloquées après celle-ci
- `quete_joueur` : Indique si la quête est liée au joueur
- `dialogue_a_enlever` : Option contenant l'ID d'un dialogue à retirer après

la quête
- `statut` : État actuel de la quête
- `fin_de_quete` : Condition de fin de la quête

Méthodes principales

1. **Constructeur**
    - `new()` : Crée une nouvelle quête avec tous ses paramètres
2. **Getters et setters basiques**
    - `get_id()`, `get_description()`, `set_description()`
    - `get_nom()`, `set_nom()`
    - `get_lieu()`, `set_lieu()`
    - `get_statut()`, `set_statut()`
3. **Gestion des récompenses**
    - `get_recompense()`, `set_recompense()`
    - `add_recompense()` : Ajoute une récompense à la quête
    - `str_recompense()` : Formate les récompenses en chaîne de caractères
4. **Gestion des quêtes suivantes**
    - `get_quetes_suivantes()`, `set_quetes_suivantes()`
    - `str_quetes_suivantes()` : Formate la liste des quêtes suivantes
5. **Gestion de la fin de quête**
    - `get_fin_de_quete()` : Récupère la condition de fin
    - `str_fin_de_quete()` : Formate la condition de fin en texte lisible
    - `find_fin_de_quete()` : Vérifie si une condition de fin correspond à un ID donné

Implémentation de Display

La structure implémente le trait `Display` pour permettre l'affichage formaté d'une quête, incluant toutes ses caractéristiques et son état actuel.