Plan Détaillé du Projet Rogue-like : Dystopie d'Entreprise

Projet Informatique, Version simplifiée

6 novembre 2025

Contexte et Objectif du Jeu

- **Univers :** Monde futuriste et dystopique où des entreprises (la Corporation) exploitent des humains créés pour le travail jusqu'à l'épuisement.
- **Pitch :** Vous incarnez un "clone de bureau" qui a atteint le point de rupture et se rebelle.
- **Concept :** Un jeu *rogue-like* en ligne de commande (CLI) où le joueur doit **monter** les étages du Building de la Corporation pour atteindre le sommet et affronter le P-DG (Boss Final).
- **Mode de Jeu : Non-infini** (Bonus Section 4 : Boss Final) – La partie se termine à la victoire ou à l'échec.

Phase 0 : Initialisation et Structure du Code (Priorité : Modularité et Robustesse)

0.1 Architecture des fichiers (Python)

Mise en place de la structure pour assurer la modularité (Section 13 du cahier des charges).

```
/projet_dystopie_rogue/
   src/
      game.py          # La boucle principale, gestion de la progression, fin.
      io_cli.py        # Fonctions pour I/O (affichage, saisie robuste).
      models/
         player.py    # Logique du joueur (stats, inventaire, actions).
         enemy.py     # Logique des ennemis (création, capacités).
      systems/
         rng.py       # Encapsule random (gestion de la seed).
         scaling.py   # Fonctions d'échelle de difficulté par étage.
         encounter.py# Logique de résolution du combat (épreuve).
   data/
      enemies.json     # Définition des "employés" et "sécurité".
      items.json       # Objets et implants/passifs.
      skills.json      # Compétences de rébellion.
   documentation.md     # Livrable (document de présentation).
```

0.2 Initialisation des outils système (Exigences minimales)

- `systems/rng.py` : Implémenter la gestion de la **seed** au lancement du jeu (Section 3.7).
- `io_cli.py` : Implémenter la fonction de saisie de choix (`get_player_choice`) pour garantir la **robustesse CLI** (redemander poliment, ne pas crasher) (Section 9).

0.3 Conception des données (Thème Dystopie)

Remplir les fichiers JSON avec des concepts thématiques :

- `enemies.json` : "Agent de Sécurité", "Drone de Surveillance", "Employé Surmené (muté)", "P-DG (Boss Final)".
- `items.json` : "Kit de Survie (Potion de soin)", "Implant de Puissance (Passif ATK)", "Clé de Données (Bonus Score/Info)".
- `skills.json` : "Surcharge Mentale (Dégâts élevés)", "Coup de Poing de la Pause Café (Attaque simple)", "Protocole d'Évasion (Défense + Fuite)".

Phase 1 : Le Cœur du Jeu (Combat et Modèles)

1.1 Modèles de base (`models/`)

- `player.py` : Définir la structure du dictionnaire `player` (PV, PV_max, ATK, Bouclier, Inventaire, Compétences). Créer la fonction d'initialisation.
- `enemy.py` : Créer la fonction pour générer un ennemi aléatoire en utilisant le JSON et en appliquant les ajustements de difficulté.

1.2 Le Combat (`systems/encounter.py`)

Implémenter la fonction `start_combat(player, enemy, depth)` :

- **Tour par Tour (Section 10.3)** : Gérer l'ordre (Joueur $\rightarrow$ Ennemi) et les effets de fin de tour (réinitialisation du Bouclier).
- **Actions Obligatoires (Section 3.2)** : Le joueur doit pouvoir choisir entre au moins trois actions distinctes : **Attaquer**, **Défendre**, **Utiliser Compétence/Objet**.
- **Journalisation (Section 9)** : Afficher les PV/ATK, les dégâts, le blocage.
- **Fin d'Épreuve (Section 3.5)** : Le combat se termine si PV du joueur ou de l'ennemi ≤ 0.

1.3 Difficulté Croissante (`systems/scaling.py`)

- Implémenter la logique d'augmentation des statistiques des ennemis et de la qualité des récompenses en fonction de l'étage/profondeur d (Section 3.4 et 12).
- Définir l'étage du Boss Final (Ex: Étage 20).

Phase 2 : Boucle Principale, Progression et Fin de Jeu

2.1 La Boucle de Jeu (`game.py`)

- Remplacer la boucle infinie par une boucle contrôlée : `while player['pv'] > 0 and depth < ETAGE_BOSS:`
- Appeler `start_combat()` pour chaque étage.

2.2 Système d'Amélioration (Récompenses)

- Après chaque victoire, offrir **au moins 3 options** d'amélioration au choix (Section 3.3) : Stat, Compétence, Objet/Passif.
- Mettre à jour les stats du joueur en conséquence.

2.3 Fin de Partie et Score (Exigences Minimales)

- **Défaite** : Si `player['pv']` tombe à 0. Afficher "Game Over : Vous avez été démantelé(e) pour recyclage."
- **Victoire** : Après avoir vaincu le P-DG (Boss Final). Afficher "FÉLICITATIONS : La Corporation est tombée !"
- **Score (Section 3.6)** : Afficher un score en fin de partie (nombre d'étages franchis + bonus).
- **Seed** : Afficher la seed au résumé final (Section 9).

Phase 3 : Finalisation et Livrables

3.1 Assurance Qualité et Robustesse

- **Qualité de Code (Section 3.8)** : Ajouter des **docstrings** (commentaires pertinents) à toutes les fonctions pour décrire leur utilité et leurs arguments.
- **Bornes** : S'assurer que les PV sont entre 0 et PV max et que les index ne sortent jamais des listes (Section 9).

3.2 Documentation (`documentation.md`)

Rédiger le document de présentation (2-3 pages) en répondant aux questions de la Section 8, en justifiant les choix de conception et d'équilibrage :

- Univers, Stats, Actions, Difficulté (Boss Final), Améliorations, Extensions Implémentées (Mode Boss).
- **Diagramme d'Architecture** (simples fonctions et leurs interactions).

Bonus (Si les fonctionnalités minimales sont garanties)

- **Couleurs ANSI** : Utiliser des codes ANSI (`\x1b[...m`) dans `io_cli.py` pour un affichage plus agréable (Section 4).
- **Classes de Départ** : Permettre de choisir une ”Spécialisation” au début (Ex : Analyste (plus de Compétences), Technicien (plus d’ATK), Manœuvre (plus de PV)) (Section 4).
- **Sauvegarde/Chargement** : Utiliser le module `json` pour sauvegarder/charger la progression dans un fichier (Section 4).