Rapport du Projet : Jeu de Tactique Tour par Tour

I .Introduction

Le projet consiste à développer un jeu vidéo de tactique tour par tour en 2D en Python, en utilisant la bibliothèque Pygame pour la gestion des éléments graphiques et des interactions.

II. Fonctionnalités Implémentées

a. Fonctionnalités de base

1. **Unités Variées :**
   Trois types d'unités (Archer, Mage, Chevalier) avec des statistiques différentes (PV, attaque, défense, vitesse, esquive).
   Chaque type possède des compétences spécifiques :
   - *Archer* : Tir simple, Tir puissant, Attaque rapide.
   - *Mage* : Boule de feu, Soin, Explosion magique.
   - *Chevalier* : Coup d'épée, Bouclier divin, Coup puissant.
2. **Terrain Varié :**
   Intégration de cases avec différentes propriétés (arbre, boue, obstacles comme rochers ou eau).
   Les unités interagissent avec le terrain pour modifier leur vitesse ou leur esquive.
3. **Gestion des Déplacements :**
   Les unités se déplacent sur une grille, avec des portées de déplacement variables selon leur type et les conditions du terrain.
4. **Combat et Compétences :**
   Les unités peuvent attaquer des cibles à portée, avec des calculs de dégâts basés sur leurs statistiques.
   Implémentation de la précision, de l'esquive et des coups critiques.

b. Fonctionnalités supplémentaires

1. **Sélection des Royaumes :**
   Trois royaumes jouables (« Empire Adrastien », « Saint Royaume de Faerghus », « Alliance de Leicester »).
   Chaque royaume gère dynamiquement la création d'armées avec des probabilités personnalisées de types d'unités.
2. **Interface Utilisateur Améliorée :**
   Menu principal pour démarrer ou quitter le jeu.
   Écran de sélection des royaumes avec des visuels associés.
   Bandeau d'informations affichant les statistiques des unités et les actions possibles.
3. **Effets de Terrain :**
   Les unités subissent des modifications (ralentissement, augmentation d'esquive) selon les cases qu'elles occupent.
4. **Gestion de la Victoire :**
   Le jeu détermine dynamiquement le gagnant lorsque toutes les unités d'un royaume sont éliminées.
1. **Retour au menu après fin de jeu** :

Une fois la partie terminée, le joueur a la possibilité de revenir au menu principal pour recommencer une nouvelle partie ou quitter le jeu.

5. **Musique et Sons :**
   Intégration de musiques différentes pour les écrans de menu et de jeu.

III. Architecture du Code

a. Diagramme UML

. Un diagramme UML détaillé a été conçu pour organiser le projet :voici les relations principales

1. **Classes abstraites :**
   Royaume : Classe abstraite définissant la structure des royaumes.
   Personnage : Classe abstraite pour les unités jouables.
2. **Relations d'héritage :**
   Royaume est étendue par Empire_Adrastien,
   Saint_Royaume_de_Faerghus, et Alliance_de_Leicester.
   Personnage est étendue par Archer, Mage, et Chevalier.
3. **Associations et compositions :**
   Jeu utilise Grille, Carte, et une liste de Personnage.
   Sélection gère les interactions entre les joueurs et les royaumes.

b. organisations des Modules

1. Fichier constante.py :
   Définit les constantes globales (taille de la grille, résolution, couleurs, chemins des images, etc.).
2. Classes Principales : Archer, Chevalier, Mage : Définissent les comportements spécifiques des unités.
3. Royaume, Saint Royaume de Faerghus ,Alliance_de_Leicester, Empire_Adrastien : Gèrent les armées des joueurs.
4. Carte, Grille : Représentent le terrain et les cases.
5. Jeu : Contient la logique principale du jeu.
6. Main : Point d'entrée du projet, gère les états (menu, sélection, jeu).

IV. Choix de Conception

1. **Héritage et Abstraction :**

   La classe abstraite `Personnage` définit les comportements génériques des unités (éviter les redondances).Les sous-classes `Archer`, `Chevalier`, `Mage` spécifient les comportements propres à chaque unité .Les classes Personnage et Royaume sont abstraites, définissant une interface commune pour les sous-classes.

2. **Séparation des Responsabilités :**
   Jeu gère la logique de déroulement (tours, actions, victoires).
   Carte gère les coordonnées et obstacles.
   Menu et Sélection s'occupent des interactions préalables au jeu.
3. **Utilisation de Pygame :**
   La bibliothèque permet une gestion efficace de l'affichage graphique, des animations, et des événements utilisateur.
4. **Gestion Dynamique des Armées :**

Chaque royaume crée ses unités à partir de probabilités, ce qui augmente la rejouabilité.

5. **choix de plusieurs fichiers:**

Les composants sont séparés dans différents fichiers pour une meilleure lisibilité et maintenance.

V . Collaboration et Répartitions des Taches

Le projet a été réalisé en équipe en utilisant **GitHub** pour la gestion du code.

• Répartitions des taches :

- **Melaaz** : Gestion de la grille, de la carte et des obstacles ainsi que du diagramme UML.
- **Mamadou** : Gestion des compétences et des combats entre les unités ainsi que du rapport.
- **Adam** : Assemblage des différents composants.
-

VI . Résultats Obtenus

Un jeu fonctionnel et stable, respectant les consignes du cahier des charges.

Points Forts :

Bonne interface graphique avec gestion des états.

Jeu enrichi par des fonctionnalités supplémentaire