Compte Rendu du Projet ChronoTactics

Introduction

ChronoTactics est un jeu de stratégie au tour par tour où les joueurs peuvent déplacer leurs unités, attaquer ou utiliser une compétence spécifique à chaque personnage. Les trois personnages principaux viennent de trois époques distinctes : l'Homme de Cromagnon, l'Homme Moderne et l'Homme du Futur, chacun ayant des compétences et des statistiques uniques. Ce concept basé sur le temps pousse les joueurs à adapter leurs stratégies selon les forces et faiblesses des unités.

Fonctionnalités du Jeu

Personnages et Compétences

Chaque personnage a une compétence unique : l'Homme de Cromagnon se soigne, l'Homme Moderne utilise une grenade pour des dégâts de zone, et l'Homme du Futur attaque à distance avec un fusil. Ces compétences sont caractérisées par leur puissance, portée et précision, ajoutant une dimension tactique variée.

Terrain et Stratégie

Le terrain comporte trois types de cases : les cases classiques pour les déplacements simples, les cases portails pour téléporter aléatoirement une unité, et les cases anomalies qui réduisent les dṕlacements. Cette configuration rend chaque partie unique et stratégique.

Menus et Navigation

Le jeu offre plusieurs menus intuitifs : un **menu principal** pour choisir le mode de jeu, un **menu options** pour régler le volume et les pages, et un **menu de sélection** pour choisir les unités.

Contrôles et Interface

Les commandes incluent les flèches pour les dṕlacements, **C** pour les compétences, **ESPACE** pour les attaques, **N** pour finir le tour, et **ECHAP** pour quitter. Les barres de vie vertes et rouges rendent l'état des unités clair et lisible.

Structure des Classes

Le projet repose sur plusieurs classes bien définies pour garantir une structure claire :

- **Unit** : Gére les propriétés de base des unités telles que la vie, l'attaque et les déplacements.
- **Homme_Cromagnon, Homme_Moderne, Homme_Futur** : Classes dérivées de **Unit**, chacune possède une compétence unique et des statistiques spécifiques.
- **Game** : Gére la logique du jeu, y compris les déplacements, les tours, et les interactions avec le terrain.
- **Menu et GraphicsHandler** : Contrôle les menus de navigation, les régles et l'affichage des images et barres de vie.

Cette organisation modulaire facilite les modifications et ajouts tout en assurant un code propre et maintenable.

Répartition des Tâches

Houssameddine

Houssameddine a eu un rôle essentiel dans la conception des systèmes de jeu et la gestion des menus. Il a développé le fichier **chronotactics.py**, gérant les déplacements, tours de jeu et interactions avec les cases du terrain. Il a **conçu le menu secondaire (menu_secondaire.py)** pour inclure les options de volume sonore et les choix d'unités. Houssameddine a aussi **rédigé les règles du jeu** et optimisé les tailles des fenêtres, tout en ajoutant les éléments visuels des barres de vie et animations de tirs.

Rayane

Rayane a joué un rôle central dans la création des mécaniques avancées. Il a développé la **mécanique des portails** et des **cases anomalies**, renforçant la stratégie et les choix tactiques. Il a amélioré l'immersion visuelle avec **des images pour les portails et anomalies**, et a réfactorisé la classe **compétences** pour en faire une structure plus modulaire. Il a conçu un **troisième menu de sélection** pour personnaliser les unités et ajouté un **système de précision et d'esquive** pour les combats.

Anis

Anis a joué un rôle central dans mécaniques interactives et aux graphique du jeu. Il a ajouté et développé le **menu principal (menu_chronotactics.py)** et ajouté la **visualisation dynamique des portées des compétences**. Il a implémenté un **système de coups critiques** pour ajouter des attaques aléatoires plus puissantes et a optimisé les effets de la **compétence grenade**. Il a aussi créé la classe **GraphicsHandler** pour la gestion des images et corrigé le **bug de la méthode draw**, en résolvant les erreurs pour une meilleure structure visuelle.

Conclusion

ChronoTactics est un jeu stratégique au gameplay riche et innovant, combinant des personnages uniques, un terrain interactif et des compétences variées. Ce projet réalise une synthèse efficace entre conception technique, design graphique et expérience utilisateur, aboutissant à un jeu engageant et équilibré.

1 Diagramme UML

nous avons structuré les classes et leurs interactions en utilisant des relations d'**héritage**, de **composition**, d'**agrégation**, et d'**association**.

On observe que chaque classe a des responsabilités bien définies et que leurs interactions reflètent une architecture robuste et modulaire. Le diagramme met en évidence :

- Les relations hiérarchiques entre les classes (`Unit` et ses sous-classes, `Competence` et ses spécialisations).
- Les relations intégrées via la composition (`Game` avec ses unités, portails et anomalies).
- Les interactions fonctionnelles indépendantes, représentées par des associations (`Menu`, `MenuSecondaire`, `ClassSelectionMenu`).

2 Détails des classes et relations

2.1 Classe `Game`

- **Rôle** : Cette classe est au centre de notre système, où l'on gère la logique principale du jeu, les unités des joueurs, des ennemis, et les portails/anomalies.
- **Attributs** :
  - `screen : Surface` : La surface graphique pour l'affichage.
  - `player_units : list[Unit]` : Les unités contrôlées par les joueurs.
  - `enemy_units : list[Unit]` : Les unités contrôlées par l'IA.
  - `portals : list[tuple[int, int]]` : Liste des portails temporels.
  - `anomalies : list[tuple[int, int]]` : Liste des anomalies sur le terrain.
- **Méthodes** :
  - `handle_player1_turn()` : On gère le tour du joueur 1.
  - `handle_player2_turn()` : On gère le tour du joueur 2.
  - `handle_enemy_turn()` : On gère les tours des unités ennemies.
  - `teleport_unit(unit: Unit)` : On téléporte une unité via un portail.
  - `render()` : On affiche les éléments graphiques du jeu.
  - `main()` : Le point d'entrée principal pour démarrer le jeu.
- **Relations** :
  - **Agrégation** : avec `GraphicsHandler`, car cette classe est utilisée pour gérer les graphismes sans être directement intégrée à `Game`.
  - **Composition** : avec les unités (`Unit`), les portails, et les anomalies, qui sont des parties intégrantes de `Game`.

2.2 Classe `GraphicsHandler`

- **Rôle** : Gère l'affichage graphique du jeu.
- **Attributs** :
  - `screen : Surface` : La surface graphique où les éléments sont affichés.
  - `images : dict[str, Surface]` : Dictionnaire des images utilisées.
- **Méthodes** :
  - `display_background()` : On affiche l'arrière-plan du jeu.
  - `display_title(title_text: str)` : On affiche un titre graphique.
  - `display_menu_options(options: list[str], selected_option: int)` : On gère l'affichage des options de menu.
  - `update_display()` : On met à jour les éléments graphiques affichés.
- **Relations** :
  - **Agrégation** : avec `Game`, car `GraphicsHandler` peut exister indépendamment.

2.3 Classe `Unit`

- **Rôle** : Représente les unités (joueurs et ennemis) dans le jeu.
- **Attributs** :
  - `x, y : int` : Les coordonnées de l'unité sur la carte.
  - `nom : str` : Le nom de l'unité.
  - `health, max_health : int` : Les points de vie actuels et maximum.
  - `attaque, defense : int` : Les statistiques offensives et défensives.
  - `competence : Competence` : La compétence unique associée.
- **Méthodes** :
  - `move(dx: int, dy: int)` : On déplace l'unité sur la grille.
  - `attack(target: Unit)` : On attaque une unité cible.
- **Relations** :
  - **Composition** : avec `Competence`, car chaque unité possède une compétence unique.
  - **Héritage** : `Unit` est la classe parent des sous-classes spécialisées (`Homme_Cromagnon`, `Homme_futur`, `Homme_moderne`).

2.4 Classe `Competence` et ses sous-classes

- **Rôle** : Représente les compétences des unités (`Fusil`, `Grenade`, `Soin`).
- **Relations** :
  - **Héritage** : `Competence` est la classe abstraite parente des compétences spécifiques.

2.5 Classes de Menu

- `Menu` : Menu principal, en association avec `MenuSecondaire`.
- `MenuSecondaire` : Gère les sous-menus, en association avec `ClassSelectionMenu`.
- `ClassSelectionMenu` : Permet la sélection des classes et est associé à `Game`.

3 Relations principales dans le diagramme UML

Relation	**Type UML**	**Exemple dans le projet**
Héritage	Triangle vide	`Unit → Homme_Cromagnon, Competence → Fusil`
Composition	Losange rempli	`Game → Unit, Unit → Competence`
Agrégation	Losange vide	`Game → GraphicsHandler`
Association	Ligne simple	`Menu → MenuSecondaire, ClassSelectionMenu → Game`