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

Introduction

Nous avons structuré les classes et leurs interactions en utilisant des relations d'**héritage**, de **composition**, d'**agrégation**, et d'**association**. Cette approche a permis de créer une architecture modulaire et robuste, où chaque classe joue un rôle clairement défini.

Architecture des Classes et leurs Interactions

L'architecture du projet repose sur une logique bien développée, s'appuyant sur des classes centrales pour gérer les différents aspects du jeu, comme les mécaniques principales, les compétences, les menus et les affichages graphiques.

Classe `Game`

La classe `Game` est le noyau du projet. Elle gère tous les aspects du jeu, de la logique principale à l'interaction avec les unités et le terrain. Ses principaux attributs incluent la gestion des surfaces graphiques (`screen`), des listes d'unités pour les joueurs et les ennemis, ainsi que les portails et anomalies présents sur le terrain. Ces éléments sont directement composés dans la classe, permettant une interaction fluide et cohérente. Parmi ses méthodes les plus importantes figurent `render()`, qui affiche l'état actuel du jeu, et `teleport_unit()`, qui met en œuvre la téléportation des unités via les portails temporels.

Classe `GraphicsHandler`

Pour s'assurer d'une expérience visuelle immersive, la classe `GraphicsHandler` est responsable de l'affichage graphique. Elle gère l'arrière-plan, les menus, et les images des différentes unités. En plus de cela, elle met à jour dynamiquement les barres de vie et les compétences visibles sur l'écran. La relation d'agrégation avec `Game` reflète une dépendance claire mais flexible, permettant de réutiliser cette classe dans d'autres contextes graphiques si besoin.

Classe `Unit`

La classe `Unit` constitue la base pour toutes les entités du jeu. Ses spécialisations incluent les classes `Homme_Cromagnon`, `Homme_Moderne`, et `Homme_Futur`. Chaque unité possède des statistiques uniques (comme la vie, la défense, et l'attaque) ainsi qu'une compétence spécifique. Par exemple, l'Homme de Cromagnon peut se soigner, tandis que l'Homme du Futur utilise un fusil à longue portée. La composition avec la classe `Competence` renforce cette flexibilité, offrant à chaque unité une compétence adaptée à son style de jeu.

Classe `Competence`

La classe `Competence` regroupe les compétences communes aux différentes unités. Les spécialisations, comme `Soin`, `Grenade` et `Fusil`, permettent d'ajouter des fonctionnalités

sans complexifier la classe parent. Cette architecture par héritage facilite l'intégration de nouvelles compétences tout en préservant une structure claire et maintenable.

Classes de Menu

Les menus sont gérés par plusieurs classes, dont `Menu`, `MenuSecondaire`, et `ClassSelectionMenu`. Ces classes permettent une navigation fluide entre les différentes phases du jeu, de la sélection des unités aux réglages de volume. Leur conception modulaire garantit une interactivité efficace et une expérience utilisateur intuitive.

2 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`