Rapport du projet PirateGame

1. Structure du Code

Le code est organisé selon une hiérarchie de classes cohérente, orientée objet, pour faciliter la gestion de l'histoire interactive. Voici la justification de la hiérarchie des classes :

- **`Main` :** La classe principale interagit directement avec l'utilisateur pour démarrer, charger ou quitter le jeu. Elle est placée au plus haut niveau, agissant comme le point d'entrée du programme.

- **`Game` :** Cette classe encapsule l'état d'une partie du jeu. Elle est responsable de configurer le jeu, de le démarrer, de gérer le déroulement de l'histoire, et offre des fonctionnalités de sauvegarde et de chargement.

- **`Player` :** Représente le joueur et son personnage associé. Placer cette classe distincte permet d'ajouter d'autres fonctionnalités liées au joueur, si nécessaire.

- **`Event` :** Classe abstraite servant de base pour les différents types d'événements dans le jeu. Elle offre une structure commune pour tous les types d'événements.

  - **`DecisionNode` :** Un type spécifique d'événement où le joueur doit prendre une décision. Il hérite d'`Event` pour partager les propriétés communes et étend ses fonctionnalités.

  - **`InnerNode` :** Représente un événement sans décision du joueur mais avec un texte associé. Il hérite également de `Event`.

  - **`CrewNode` :** Introduit pour gérer les scénarios où le joueur peut former une équipe avec un autre personnage.

  - **`CombatNode` :** Pour les événements déclenchant des combats. Il s'appuie sur `Event` pour une gestion cohérente.

  - **`TerminalNode` :** Marque la fin de l'histoire. Hérite d'`Event` mais indique clairement la conclusion de l'histoire.

  - **`SaveNode` :** Extension introduite pour permettre la sauvegarde du jeu. Il hérite de `DecisionNode` car il inclut une décision du joueur.

  - **`NodeDecorator` :** Une classe abstraite servant de base pour les décorateurs. Elle étend `Event` et contient une référence à un autre `Event`.

- **`ImageNode` :** Un décorateur concret qui ajoute la fonctionnalité d'afficher une image avant le contenu du nœud décoré.

- **`SoundNode` :** Un autre décorateur concret qui pourrait être ajouté pour introduire des effets sonores à certains événements.

Cette hiérarchie de classes est conçue pour offrir une flexibilité maximale dans le développement de l'histoire interactive. Chaque classe a une responsabilité spécifique, contribuant ainsi à une conception propre et extensible.

2. Extensions

Le projet a été étendu pour inclure des décorateurs (`ImageNode` et potentiellement `SoundNode`) pour ajouter des fonctionnalités visuelles et sonores à certains événements. Ces extensions permettent d'améliorer l'expérience utilisateur en introduisant des éléments multimédias.

3. Difficultés Rencontrées

La mise en œuvre des décorateurs a été l'aspect le plus complexe du projet. Il a fallu gérer soigneusement l'ordre dans lequel les décorateurs sont appliqués pour garantir le bon affichage des éléments multimédias. La coordination entre les classes a été cruciale pour garantir une intégration fluide de ces extensions dans le système existant. La gestion des sauvegardes était également compliquée car elle impliquait la notion complexe de sérialisation. Il a fallu se documenter pour bien comprendre le fonctionnement de cette interface. Enfin, trouver un scénario aura été ce qui a pris le plus de temps.

Le projet dans son ensemble a été un défi stimulant, mais il a permis de mettre en pratique de nombreux concepts de programmation orientée objet et de renforcer la compréhension de la gestion des états dans un contexte interactif.