Justification du projet Hex comme travail final de Programmation Orientée Objet

Jair Vasquez
Santiago Florido
IN204 – Programmation Orientée Objet

***Résumé*—Le jeu de Hex mis en place offre déjà une architecture orientée objet claire et extensible : `Board` encapsule le plateau, `GameState` regroupe la logique de victoire et des coups, `Cube` modélise la géométrie hexagonale. Cette base se prolonge naturellement vers une interface graphique SFML qui continuera d'exploiter et d'enrichir les principes de POO. Ce document présente de façon globale pourquoi ce projet est un candidat pertinent pour le travail final de POO.**

I. Vue d'ensemble orientée objet

L'application console actuelle (`main.cpp`) illustre l'abstraction du domaine Hex en trois classes :

- **Board** : gère la taille, l'état des cases et l'unique point d'écriture (`place`) pour garantir l'intégrité du plateau.
- **GameState** : fournit les représentations linéaires, les coups disponibles et la détection de victoire (BFS sur les connexions), en s'appuyant sur le modèle de données sans se soucier d'affichage.
- **Cube** : porte le système de coordonnées cubiques et les directions, isolant le calcul géométrique.

La responsabilité de chaque classe est nette et rend le projet aisé à maintenir, tester et faire évoluer.

II. Justification pour le projet final

Le code déjà en place démontre les notions fondamentales de POO (encapsulation, séparation des responsabilités, modélisation d'objets du domaine). Il est suffisamment complet pour une partie jouable et assez modulable pour intégrer des fonctionnalités supplémentaires sans remettre en cause le noyau objet. Le passage à une interface SFML enrichira le projet en illustrant l'extension de classes existantes et l'introduction de nouveaux objets dédiés à l'affichage et aux entrées.

III. Apports POO via l'interface SFML

L'intégration d'une interface graphique SFML permet de continuer la démarche objet :

- Création d'une classe `HexView` responsable du dessin du plateau (tuiles, pions, grille), s'appuyant sur `Board` pour lire l'état courant.
- Introduction d'un `InputHandler` SFML pour transformer les clics souris en coordonnées hexagonales et invoquer `Board::place`, conservant l'API existante.
- Possibilité d'un `GameController` qui orchestre `GameState`, `HexView` et les entrées SFML, sans même modifier les classes de logique : la POO est renforcée par la composition d'objets spécialisés.
- Ajout d'animations ou de surlignages en encapsulant ces comportements dans des objets graphiques dédiés, tout en gardant les règles du jeu au sein de `GameState`.

Cette organisation montre comment l'interface visuelle s'appuie sur les abstractions existantes et comment de nouvelles classes prolongent la structure objet.

IV. Apports algorithmiques pour renforcer le modèle

En s'inspirant de projets similaires , qui combine modélisation POO et algorithmes de recherche avancés, il est possible d'enrichir le jeu Hex avec une couche décisionnelle claire et modulaire.

A. Analyse de connectivité : BFS et A*

La détection de victoire utilise déjà un BFS, mais cette logique peut être étendue :

- **Évaluation de positions** : calcul de la distance minimale entre les bords à relier par chaque joueur.
- **Heuristique pour IA** : A* permet d'obtenir une estimation de progression plus informative que BFS.

Une classe `PathEvaluator` peut encapsuler BFS/A* tout en restant indépendante de l'affichage.

B. Prise de décision : Minimax avec élagage α–β

Pour ajouter une IA comparable à celle du projet Quoridor, un module décisionnel peut être intégré :

- **`AIPlayer`** : implémente une interface `IPlayer` et applique Minimax/Negamax.
- **Élagage α–β** : réduit les branches explorées pour maintenir un temps de réponse compatible avec SFML.
- **Heuristique d'évaluation** : repose sur `PathEvaluator` pour estimer la force de chaque position.

Cette architecture respecte la séparation logique : les règles restent dans `GameState` et la décision est déléguée à un module externe.

C. Extensions

Plusieurs éléments peuvent enrichir le projet :

- **Difficulté ajustable** : profondeur de recherche, aléatoire contrôlée, étendue de l'exploration.
- **Replays et logs** : via un `MatchRecorder`.
- **Tests unitaires d'algorithmes** : validation de l'heuristique, vérification des chemins, robustesse du contrôleur.

Ainsi, le modèle objet initial s'étend naturellement vers une architecture hybride logique–algorithmique cohérente.

V. Plan de travail orienté POO pour le jeu complet

La structure actuelle sert de socle pour un ensemble cohérent d'étapes, toutes centrées sur des objets spécialisés :

- **Gestion des joueurs** : interface `IPlayer` avec implémentations Humain et IA ; injection dans `GameController`.
- **Règles configurables** : objet `Ruleset` (taille, couleurs, timer) paramétrant `Board` et `HexView`.
- **Chrono et scores** : classes `Clock` et `Scoreboard` gérées par SFML.
- **Rejeu et sauvegarde** : `MatchRecorder` pour enregistrer et rejouer une partie.
- **Menu et scènes** : `SceneManager` permettant de naviguer entre menu, partie et rejoue.
- **Effets visuels modulaires** : overlays graphiques sans altérer `GameState`.

VI. Conclusion

Le projet Hex combine une base jouable, une architecture orientée objet claire et un potentiel d'extension naturel vers SFML et des modules algorithmiques. Cette continuité du modèle vers la décision et l'affichage justifie pleinement son choix comme travail final.