Rapport: Application de conception
- Développer un jeu quixo -

réalisé par: Ibrahim Jallouli (ILSEN)

Sommaire:

Introduction:

1. Analyse du problème

L'analyse du problème commence par la réalisation d'un jeu Quixo, un défi qui implique une compréhension approfondie des règles spécifiques du jeu. Le développement doit incorporer différents modes de jeu, tels que le joueur contre joueur **(human vs human)** et le joueur contre intelligence artificielle **(human vs AI)**, ajoutant ainsi une complexité supplémentaire à la mise en œuvre. Pour garantir une expérience utilisateur optimale, l'accent sera mis sur un affichage clair et convivial, favorisant ainsi une interaction intuitive avec le jeu.

2. Choix Technologique et Bibliothèques Utilisées

Pour répondre aux exigences du projet, le choix du langage de programmation s'est orienté vers Python, une première expérience pour moi. Cette décision repose sur la diversité des **bibliothèques** disponibles dans l'écosystème Python, offrant une flexibilité et une efficacité accrues dans le développement du jeu Quixo. De plus, la **gestion native des threads** en Python facilite l'implémentation du multithreading, un aspect crucial pour optimiser les performances de l'algorithme Min-Max.

Le fait que Python ne contraigne pas à **l'héritage multiple**, contrairement à certains autres langages comme **Java**, simplifie considérablement la mise en œuvre des design patterns nécessaires à la conception orientée objet du projet. Cela a grandement contribué à la fluidité du processus de développement et à la qualité du code produit.

Tkinter: choisie en raison de sa simplicité et de son adéquation aux besoins minimalistes du projet. Étant donné que le jeu se contente d'éléments graphiques de base tels que des labels et des boutons.

Concurrent.futures: J'ai choisi d'utiliser la bibliothèque concurrent.futures en raison de son implémentation optimale de la programmation multithread. Cette bibliothèque offre une interface de haut niveau pour l'exécution concurrente de tâches, facilitant ainsi la gestion des threads de manière efficace.

Fonctionnalités Attendues:

1. Human vs Human

Les fonctionnalités attendues pour le jeu Quixo incluent un mode "Human vs Human" permettant à deux joueurs humains de s'affronter. Dans ce mode, les utilisateurs peuvent **prendre alternativement le contrôle du jeu**, sélectionnant et **déplaçant les pièces** selon les **règles** spécifiques du Quixo.

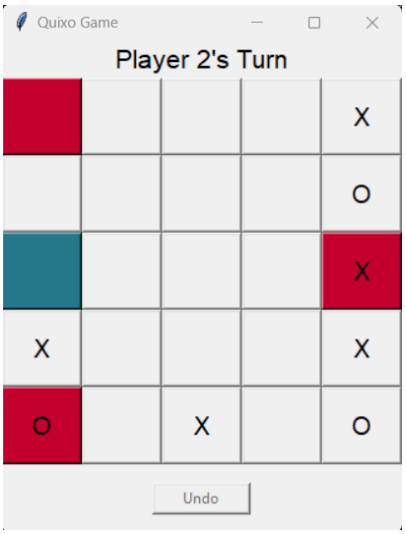
Les fonctionnalités attendues pour ce mode comprennent **la détection des mouvements légaux**, **la mise à jour en temps réel de l'état du plateau**, **et la gestion appropriée des conditions de victoire ou de défaite.**

2. Intelligence artificielle basée sur Min-Max

Une autre fonctionnalité clé attendue est l'intégration d'une intelligence artificielle (IA) basée sur l'algorithme Min-Max. Cette composante permettra aux joueurs de s'engager dans des parties "Human vs AI", offrant ainsi une expérience de jeu solo compétitive. L'IA, utilisant l'algorithme Min-Max, évaluera les différentes configurations du plateau pour choisir le meilleur coup possible à chaque tour. Cette fonctionnalité exigera une implémentation soigneuse de l'algorithme Min-Max, ainsi qu'une évaluation précise de la position actuelle du jeu.

3. Interface graphique

L'interface graphique (GUI) du jeu Quixo sera claire et intuitive, avec un affichage épuré du plateau de jeu et des couleurs indicatives simples. L'objectif est de rendre le jeu accessible et attractif, que ce soit en mode "Human vs Human" ou "Human vs AI".



Conception et Implémentation

1. Mise en place du Design Pattern

- MVC: Le design pattern Modèle-Vue-Contrôleur (MVC) sépare les composants logiques, d'interface utilisateur et de contrôle dans un système logiciel.

  Model: Player.py - PlayerAI.py - QuixoBoard - BoardMemento - QuixoPiece

  View: QuixoGUI.py

  Controller: GameController - GameControllerAI

- Memento: Le Memento dans le contexte du jeu Quixo permet de sauvegarder et de restaurer l'état du plateau de jeu.

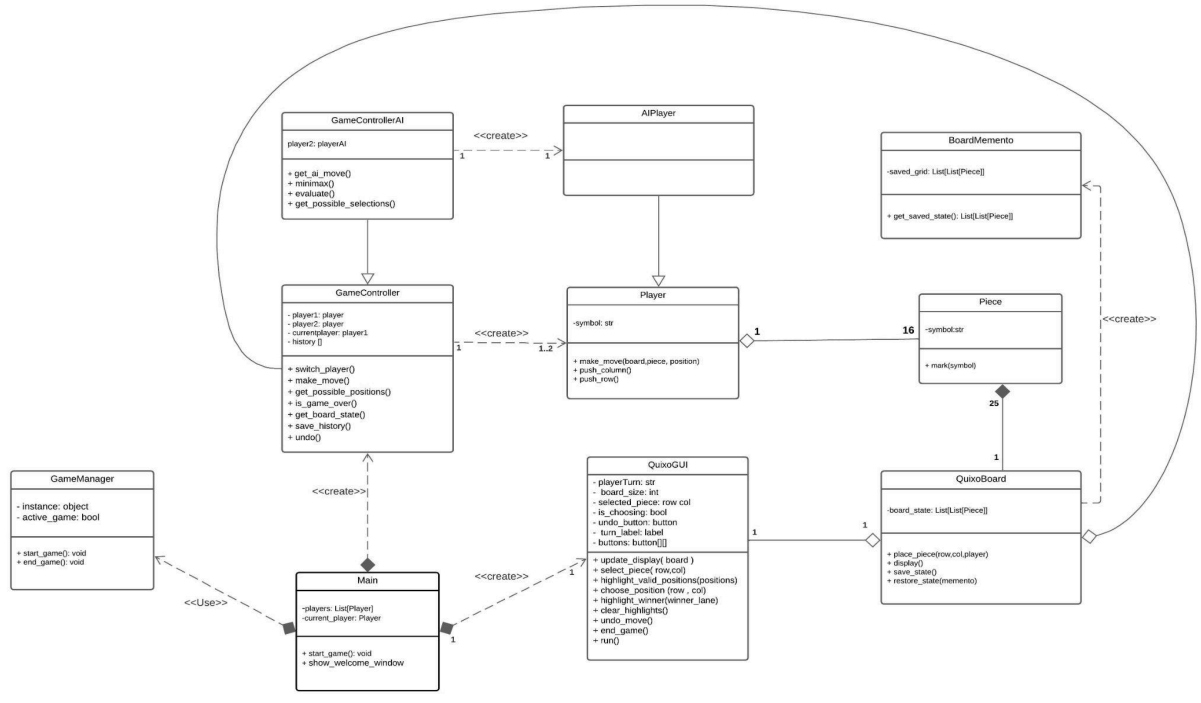
  Originator : Dans votre cas, il s'agirait de la classe **QuixoBoard**, qui peut produire des **instantanés** (snapshots).

  Memento : Cela est représenté par la classe **BoardMemento**.

  Caretaker : c'est le **GameController**.

- Singleton: Le **GameManager** a été créé en réponse à une **faiblesse** potentielle du Singleton standard, qui aurait pu être problématique en présence d'une **hiérarchie** entre les contrôleurs. **L'héritage** aurait pu conduire à la création de multiples instances de contrôleurs, compromettant ainsi la singularité du Singleton. Pour pallier cela, le GameManager a été introduit avec une implémentation personnalisée du Singleton, garantissant qu'une seule instance de la classe GameManager existe pour la gestion du jeu en cours, indépendamment de toute hiérarchie d'héritage entre les contrôleurs.

2. Présentation des classes UML



3. Fonctions clés

switch_player: Permet de passer au joueur suivant à chaque tour.

get_possible_selections: Identifie les sélections possibles pour le joueur en cours.

get_possible_positions: Détermine les positions possibles pour déplacer un pion en fonction de sa position actuelle.

minimax: Implémente l'algorithme récursif Min-Max avec **depth = 2** pour déterminer le meilleur coup possible en évaluant différentes configurations du plateau à une profondeur donnée.

evaluate: Attribue des scores statiques et dynamiques aux positions sur le plateau du jeu Quixo en fonction de différents critères. Chaque critère est associé à un score statique, et la fonction somme ces scores en prenant en compte la position des pièces. De plus, la fonction évalue également des configurations dynamiques, telles que trois pièces alignées horizontalement, verticalement ou en diagonale, en attribuant des scores spécifiques pour ces motifs particuliers.

is_game_over: Vérifie si la partie est terminée en analysant les lignes, colonnes et diagonales du plateau du jeu Quixo, et retourne des informations sur la victoire le cas échéant.

Critique:

La fonction d'évaluation actuelle présente plusieurs aspects pouvant être optimisés pour rendre le système plus efficace. Certains problèmes potentiels comprennent :

- **Simplicité de l'évaluation :** La fonction actuelle repose sur des scores statiques et dynamiques relativement simples. Cela pourrait entraîner des évaluations moins précises, car elle ne capture peut-être pas toutes les subtilités du jeu Quixo.

- **Manque de spécificité :** Les critères d'évaluation actuels sont génériques et pourraient ne pas prendre suffisamment en compte les situations spécifiques sur le plateau, ne s'adaptant pas à l'évolution du jeu. Une évaluation plus spécifique pourrait améliorer la compréhension fine des positions.

Pour remédier à ces problèmes et rendre l'évaluation plus optimale, une approche basée sur l'apprentissage automatique pourrait être explorée. Cela permettrait à l'IA d'apprendre à évaluer les positions en fonction de nombreux facteurs, conduisant potentiellement à des décisions plus intelligentes et plus adaptatives.