Programmation Orientée Objets

Projet :

Gomoku

Etudiants :
Syrine BEN HASSINE
Jean-baptiste LARGERON
Karam ELNASORY
Licence 2 info (S4)

1 Architecture et principales classes

— **Color.java** : Enumère les couleurs disponibles pour les joueurs, chaque couleur étant associée à un code ANSI pour l'affichage console. . Prise en charge de l'affichage adapté à Windows/Mac/Linux.

— **Coordonates.java** : Représente un point (x, y) sur la grille. Fournit des méthodes d'accès et une représentation textuelle des coordonnées.

— **User.java** (abstraite) : Représente un joueur (nom, score, couleur). IL gére le score, les jetons, et le choix du placement.

— **Human.java** / **Computer.java** : Spécialisations de `User` pour les joueurs humains et l'ordinateur. `Human` gère l'interaction utilisateur (choix de couleur, placement). `Computer` implémente une IA de choix de coup (aléatoire ou avec stratégie).

— **Token.java** : Représente un jeton de couleur posé sur la grille.

— **Cell.java** : Représente une case de la grille, stocke le jeton courant et ses voisins pour la détection des alignements.

— **Matrix.java** : Représente la grille de jeu, gère la pose de jetons, l'exten-

sion de la grille, la vérification des conditions de victoire, et l'affichage de la grille.

— **Game.java** : Gère une partie complète : initialisation des joueurs, de la grille, gestion du tour, sauvegarde/chargement, et conditions de victoire.

— **Launcher.java** : Point d'entrée du programme, gère les menus (principal, démarrage, en jeu), l'instanciation des parties et la navigation utilisateur.

1.1 Aparthé sur le code

Nous avions fait le choix d'implémenter une IA basé sur le théorème de l'arbre de Min-max (aussi appelé arbre minimax) de Von Neumann, mais nous avons eu des complications concernant l'implémentation et nous avons choisit d'implémenter un système plus simple dans l'optique de facilement pouvoir le remplacer dans le futur.

2 Schéma UML

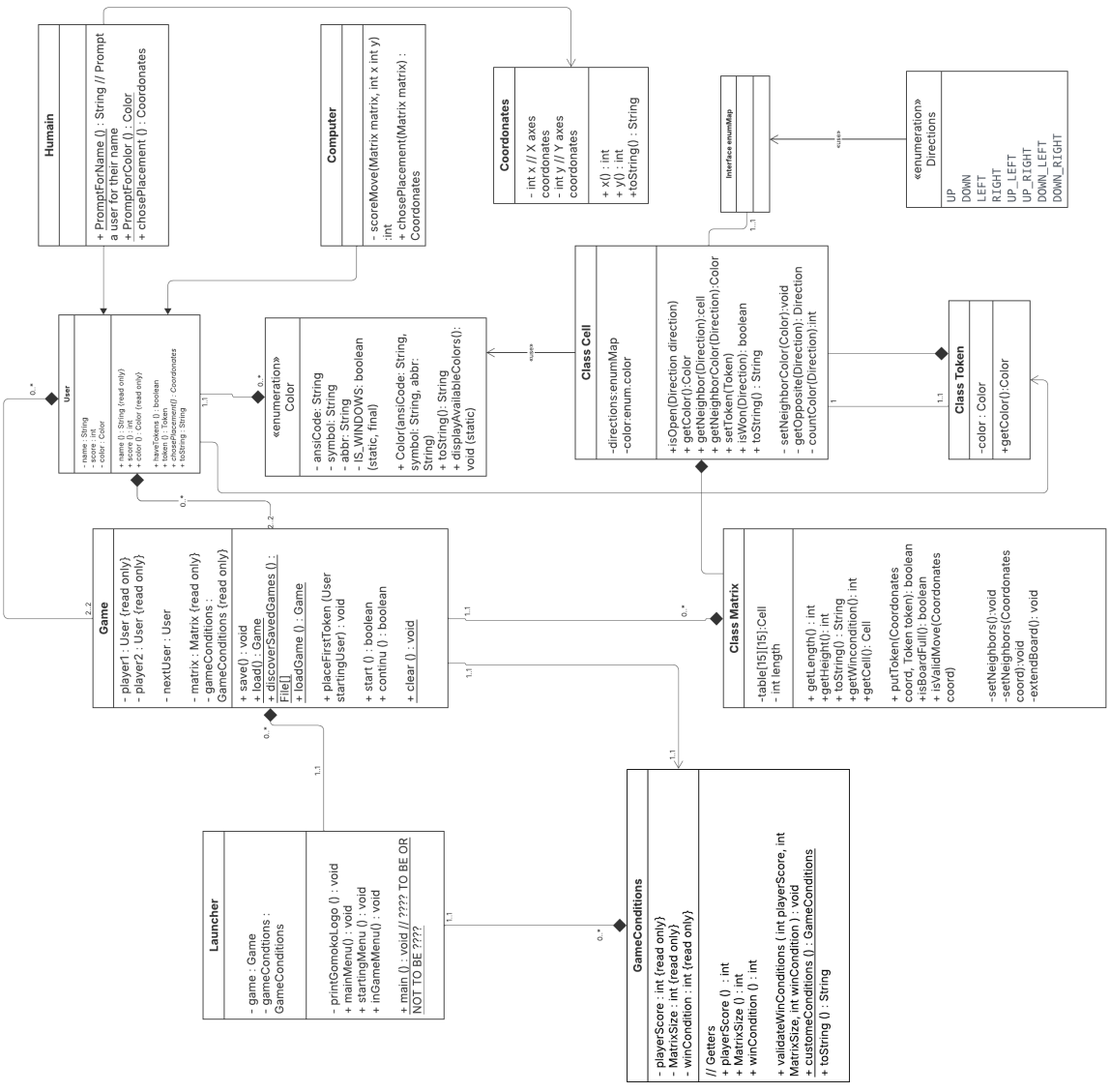

Figure 1 – UML

2.1 Description UML

L'UML que nous avons choisi d'implémenter ci dessus. Nous avons fait le choix d'encapsuler au maximum les attributs et méthodes dans des objets afin que ceux-ci soient au maximum indépendant.

— La classe **Game** centralise la logique de la partie en cours(joueurs,grille, conditions de victoires...)

— La classe **Matrix** représente la grille dynamique et fournit les méthodes nécessaires

— La classe **Cell** et **Token** modélisent les éléments de base du plateau

— Le concept de joueur est modélisée par une classe abstraite **User** puis spécialisée en **Human** pour un joueur ou **Computer** pour l'IA

— Une énumération **Color** permet de gérer les couleurs pour l'affichage dans le terminal

— **Launcher** est le contrôleur global, responsable de la navigation, la gestion de sauvegarde, les menus et le lancement de parties

3 Diagramme de menus

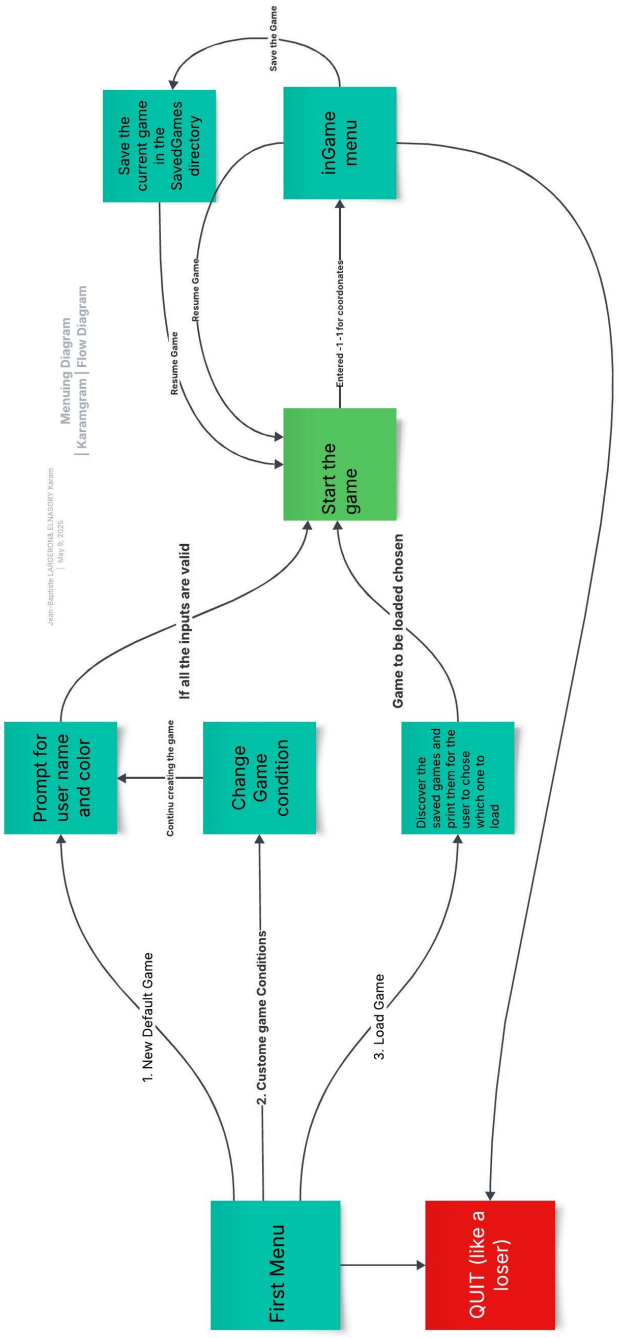

FIGURE 2 – Diagramme de menus

3.1 Description Diagramme de menus

Ce diagramme représente l'enchaînement logique des différents menus disponibles pour le joueur.

Voici les principales étapes :

— Le **First Menu** propose trois options principales : démarrer une partie, charger une partie, quitter le jeu

— Si une nouvelle partie est lancée, le joueur peut soit utiliser les conditions par défaut, soit modifier les paramètres de jeu.

— Le(les) joueur(s) choisit ensuite son nom et sa couleur de jeton

— La partie commence via **Start the game**
— En cours de partie, il est possible d'ouvrir un menu de jeu (**inGame menu**) en saisissant les coordonnées -1 -1 permettant de sauvegarder ou de quitter

— Les sauvegardes sont gérées via un dossier **SavedGame** et peuvent être rechargé via **First Menu**

Cela permet une navigation fluide pour l'utilisateur ainsi qu'une souplesse de configuration et la possibilité d'interrompre/reprendre à tout moment