Projet de Programmation Orienté Objet

Mathieu Chedas et Benoit Dajoux

25 Avril 2024

Table des matières

1 Architecture générale **2**

2 Rôle de chaque classe **2**

2.1 Color 2
2.2 Pawn 2
2.3 Combination 2
2.4 Secret 3
2.5 GameSettings 3
2.6 GameTable 3
2.7 Game 3
2.8 Main 3

3 Interface utilisateur **3**

Introduction

Dans ce rapport, nous abordons la conception et l'implémentation du jeu "Mastermind". Le but de ce jeu est de permettre aux joueurs de deviner une combinaison de pions de couleur secrète, choisie aléatoirement selon certains paramètres, tels que le nombre de pions, ou le nombre de couleurs, en un minimum de tentatives.

Nous structurerons notre projet en plusieurs classes métier, chacune dédiée à des aspects spécifiques du jeu, tels que la gestion des pions, la représentation des tentatives et du secret à découvrir.

1 Architecture générale

L'architecture générale du projet est conçue autour de plusieurs classes interagissant ensemble pour implémenter le jeu Mastermind. Ces classes sont organisées de manière à séparer les tâches en fonction de leur niveau. Les tâches les plus bas niveaux, de création d'objets concrets sont géré dans les classes `Color`, `Pawn`, `Combination`, et `Secret`. Les tâches plus haut niveau, qui vont utiliser des objets plus abstraits et plus complexes sont géré dans les classes `GameSettings`, `GameTable`, `Game`, et `Main`.

On retrouvera en annexe un diagramme UML représentant les différentes classes, leurs attributs, et leurs méthodes.

2 Rôle de chaque classe

2.1 Color

La classe `Color` est une énumération qui définit une série de couleurs utilisées dans le jeu. Chaque couleur est associée à un code de couleur ANSI utilisé pour afficher du texte coloré dans la console. Les couleurs disponibles sont : noir, rouge, vert, jaune, bleu, violet, cyan et blanc. Chaque couleur est également associée à un identifiant unique, et a son propre code de couleur.

La classe fournit des méthodes pour accéder à la valeur numérique et au code de couleur d'une couleur, ainsi qu'une méthode pour obtenir une couleur à partir de son indice numérique. De plus, la méthode `toString` est redéfinie pour afficher un point correspondant à un pion (avec la bonne couleur correpondante).

2.2 Pawn

La classe `Pawn` représente un pion. Chaque pion est caractérisé par une couleur et une position. Le constructeur permet de créer un pion avec une couleur et une position spécifiées. La classe possède aussi des accesseurs. La méthode `comparePawn` dans la classe compare deux pions et retourne `true` si leurs couleurs sont identiques. De plus, la méthode `toString` est redéfinie pour retourner le code de couleur ANSI du pion.

2.3 Combination

La classe `Combination` implémente une combinaison de pions. Chaque combinaison consiste en une liste de pions et prend également en attribut les paramètres de la partie. Elle fournit un constructeur pour créer une combinaison à partir des paramètres de la partie, et d'une liste d'entiers représentant les identifiants des couleurs des pions. La classe fournit essentiellement des accesseurs et redéfini la méthode `toString` pour afficher la combinaison sous forme de chaîne de caractères, en utilisant les codes de couleur des pions.

2.4 Secret

La classe `Secret` représente la combinaison secrète de la partie, celle que le joueur doit deviner. Cette classe hérite de la classe `Combination` et ajoute des fonctionnalités spécifiques à la génération de la combinaison secrète et à sa comparaison avec une combinaison donnée. Elle fournit un constructeur pour créer une combinaison secrète en utilisant les paramètres de jeu spécifiés et une liste d'entiers représentant les identifiants des couleurs des pions. De plus, elle fournit des méthodes pour générer une liste aléatoire d'entiers correspondants aux identifiants des couleurs des pions, ainsi qu'une méthode pour comparer la combinaison secrète avec une combinaison donnée, qui retourne un tableau d'entiers représentant les résultats de la comparaison.

2.5 GameSettings

La classe `GameSettings` gère les paramètres de la partie. Il est possible de choisir une difficulté par défaut ou des paramètres personnalisés, tel que le nombre d'essais, la taille de la combinaison, le nombre de couleurs possibles par pion, etc. Elle fournit deux constructeur pour initialiser les paramètres de jeu. Le premier permet à l'utilisateur de définir les paramètres de jeu via une série d'input. Le second permet de recharger les paramètres d'une partie enregistrée. La classe fournit également des méthodes pour obtenir une représentation de la table des couleurs utilisées dans le jeu, pour générer la table des couleurs en fonction du nombre de couleurs spécifié, et redéfini la méthode `toString` afin de retourner une chaîne de caractères décrivant les paramètres de jeu sélectionnés et affichant la table des couleurs utilisées.

2.6 GameTable

La classe `GameTable` s'occupe de la gestion des combinaisons, des essais, et des comparaisons avec la combinaison secrète. Elle fournit deux constructeurs. Le premier initialise la table de jeu en générant une nouvelle combinaison secrète aléatoire, tandis que le deuxième constructeur est utilisé pour charger une partie sauvegardée. De plus, elle fournit des méthodes pour ajouter une nouvelle combinaison proposée par le joueur, passer à l'essai suivant en comparant la combinaison proposée avec la combinaison secrète, pour afficher le résultat de son précedent essai à l'utilisateur, et redéfini la méthode `toString` afin d'afficher la table des couleurs utilisées et les combinaisons données précedement.

2.7 Game

La classe `Game` représente le moteur de jeu. Elle gère le déroulement des parties en mode solo et en multijoueur, ainsi que la sauvegarde et le chargement de parties. Elle fournit une méthode principale `play` qui lance le jeu et gère les choix des joueurs, la création de parties, les essais et l'affichage des résultats. De plus, elle fournit des méthodes pour calculer le ou les gagnants et les scores des joueurs, ainsi que pour sauvegarder et charger une partie dans un fichier.

2.8 Main

La classe `Main` permet d'encapsuler le lancement du programme Mastermind. Elle crée une instance de la classe `Game`, c'est-à-dire une nouvelle partie. et appelle sa méthode `play` pour démarrer le jeu.

3 Interface utilisateur

L'interface utilisateur du programme est conçue pour être intuitive. Ainsi nous avons décidé de faire entrer tout les choix de l'utilisateur sous forme d'un chiffre afin d'éviter les erreurs du aux majuscules/minuscules/accents et afin de rendre le paramétrage d'une partie plus simple et plus rapide. De plus, nous affichons à chaque essai d'un utilisateur ses précedents essais et leurs résultats pour rendre la réflexion plus simple. Nous affichons également à chaque tour la table des couleurs afin que l'utilisateur n'ai pas besoin de s'en rappeler, ou de remonter constamment dans son terminal afin de vérifier qu'il utilise les bonnes couleurs.

Annexe

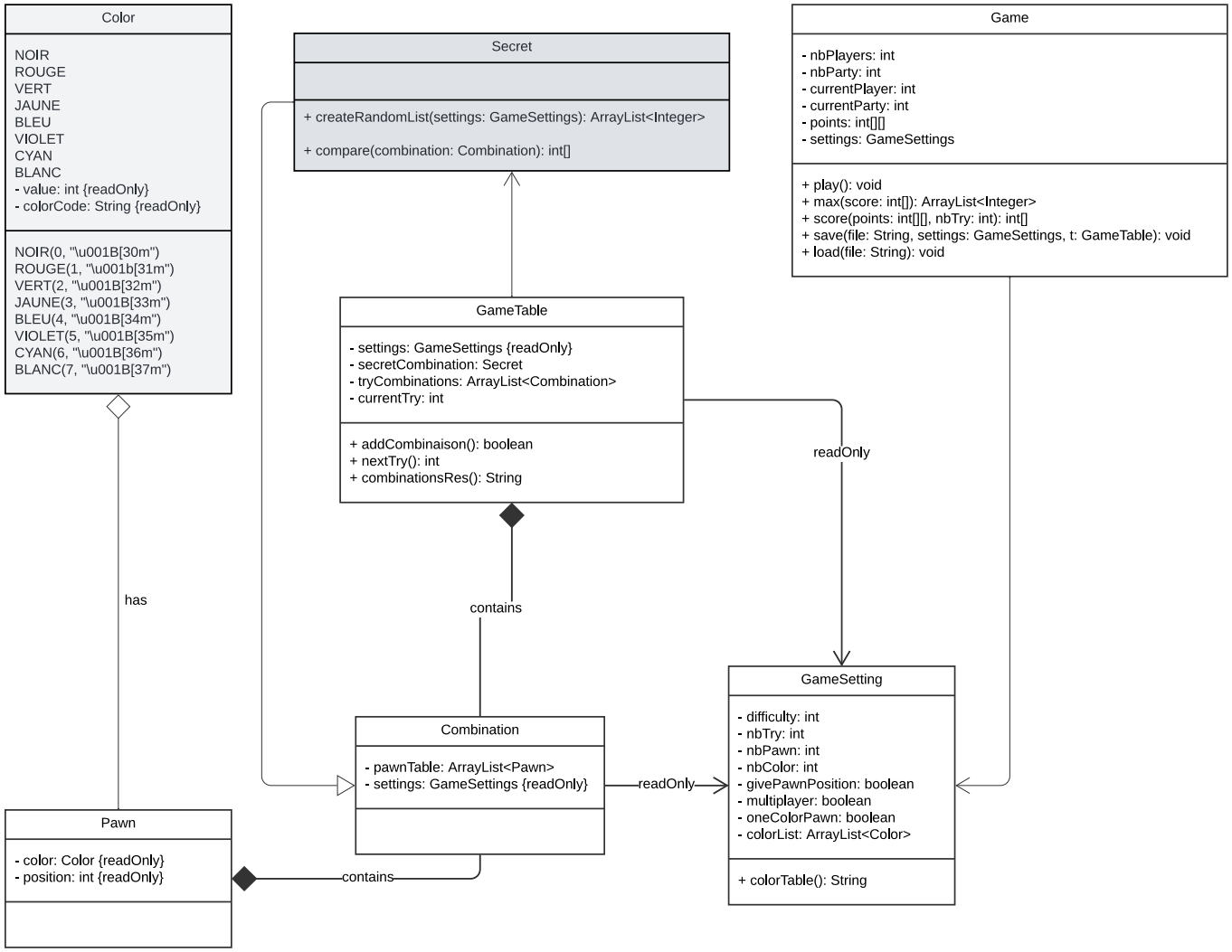

FIGURE 1 – Diagramme UML de l'architecture du projet Mastermind