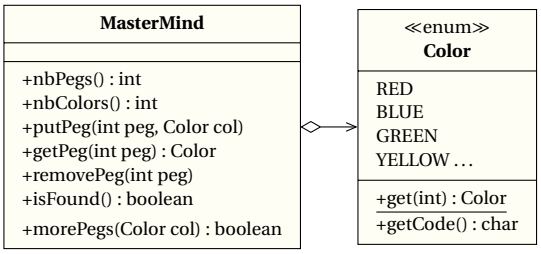
TP4. Encore du backtrack

1 MasterMind

On veut écrire un programme qui joue au mastermind, ce célèbre jeu de Mordecai Meirowitz. Dans ce jeu, une combinaison de couleurs est à retrouver par essais successifs. Cette combinaison est incarnée par un nombre de pions (en anglais *pegs*) de couleurs différentes. Chaque instance vient avec son nombre de pions et son nombre de couleurs.
Notre premier objectif va être de trouver la solution brutalement, sans aucune optimisation. Les couleurs du jeu sont fournies dans la même classe *Color* précédente, qui possède une méthode de classe *get* pour récupérer une couleur par son numéro.



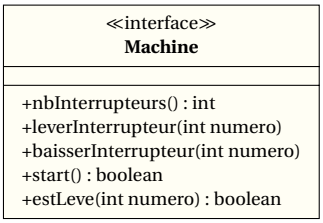
1. Complétez la fonction *checkThemAll(MasterMind m)*, de sorte qu'elle teste toutes les combinaisons de couleurs, naïvement. Vous pouvez librement ajouter des méthodes, et surcharger les méthodes existantes pour ajouter des arguments. Mais les méthodes telles que demandées doivent exister pour être appelé par Caséine.

Nous allons maintenant considérer une autre situation, plus proche du vrai jeu. Admettons que lors d'un tour précédent, vous avez réussi à connaître l'ensemble des couleurs de la combinaison. Vous devez donc écrire un programme qui teste uniquement les permutations de ces couleurs. Le MasterMind intègre une méthode *morePegs(color)* qui vous indique si des pions de cette couleur doivent encore être ajoutés à la combinaison actuelle. Par exemple, si la combinaison est "vert, rouge, rouge, bleu" et que vous avez mis "vert, bleu, null, null", *morePegs* renvoie vrai uniquement pour la couleur rouge.

2. Complétez la fonction *checkWithCows(MasterMind m)* pour qu'elle teste toutes les combinaisons des couleurs demandées et seulement celles-là.

2 Interrupteurs

On s'intéresse à une machine à interrupteurs. Cette machine ne démarre que sur une position précise de ses interrupteurs. On souhaite écrire un programme qui teste toutes les positions jusqu'à ce que la machine démarre.
Chaque interrupteur dispose de deux positions, haut et bas. La classe Machine vous permet d'intéragir avec les interrupteurs via les fonctions *leverInterrupteur(int numéro)* et *baisserInterrupteur(int numéro)*. La fonction *nbInterrupteurs()* retourne le nombre d'interrupteurs de la machine. Enfin, la fonction *start()* tente de démarrer la machine, retournant *true* si cela fonctionne.



3. Completez la fonction *toutTester(Machine m)* pour qu'elle teste toutes les positions pour démarrer la machine. N'oubliez pas d'appuyer sur *start()* pour chaque configuration.

Vous trouvez une note par terre, qui vous indique le nombre d'interrupteurs en position haute de la position optimale. Vous devriez pouvoir améliorer votre recherche!

4. Completez la fonction *testerAvecNote(Machine m, int nbHaut)* pour qu'elle teste toutes les positions à exactement *nbHaut* interrupteurs en position haute pour démarrer la machine. Le but est évidemment d'éviter de construire plein de solutions potentielles qui auraient trop d'interrupteurs levés.

Vous trouvez une deuxième note très utile, qui vous signale que la configuration de démarrage ne comporte pas deux interrupteurs consécutifs en position haute.

5. Completez la fonction *testerNonConsecutifs(Machine m)* pour qu'elle teste toutes les positions d'interrupteurs sans deux interrupteurs levés consécutifs. De la même façon, on cherche à optimiser.
6. Enfin, completez la fonction *testerNonConsecutifsAvecNote(Machine m, int nbHaut)* pour qu'elle teste toutes les positions avec *nbHaut* interrupteurs en position haute sans que deux soient consécutifs.