TP POO C++ : Mastermind

Sommaire

Les TP d'acquisition des fondamentaux visent à construire un socle de connaissances de base, à appréhender un concept, des notions et des modèles qui sont fondamentaux. Ce sont des étapes indispensables pour aborder d'autres apprentissages. Les TP sont conduits de manière fortement guidée pour vous placer le plus souvent dans une situation de découverte et d'apprentissage.

Les objectifs de ce tp sont de découvrir la programmation orientée objet en C++ en étant capable :

- **d'instancier un objet en fonction des besoins exprimés à partir d'une classe fournie ;**
- **d'utiliser les services d'une classe afin d'implémenter en C++ un diagramme de séquence fourni.**

Initiation à la programmation orientée objet

Rappels

La **programmation orientée objet** consiste à **définir des objets logiciels et à les faire interagir entre eux**.

Une **classe** représente la **description abstraite d'un ensemble d'objets possédant les mêmes caractéristiques**. On peut parler également de **type**.

Exemples : la classe Voiture, la classe Personne.

Un **objet** est une entité concrète, possédant **une identité et encapsulant un état et un comportement**. Un **objet** est **une instance d'une classe**.

Exemples : Thierry Vaira est un objet instancié de la classe Personne. La voiture de Thierry Vaira est une instance de la classe Voiture.

Un **attribut** représente un **type d'information (une variable)** contenu dans une classe.

Exemples : vitesse courante, cylindrée, numéro d'immatriculation, etc. sont des attributs potentiels d'une classe Voiture.

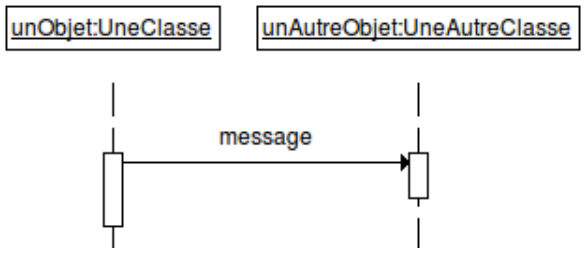
Une **méthode** représente un **élément de comportement (un service)** contenu dans une classe.

Exemples : demarrer et rouler sont des opérations possibles d'une classe Voiture.

Notion de messages

Un objet est une structure de données encapsulées qui répond à un **ensemble de messages**. Cette **structure de données (ses attributs) définit son état** tandis que l'**ensemble des messages (ses méthodes) décrit son comportement**.

L'ensemble des messages forme ce que l'on appelle l'**interface de l'objet**. Les objets interagissent entre eux en **s'échangeant des messages**.



La réponse à la réception d'un message par un objet est appelée **une méthode**. Une **méthode est donc la mise en oeuvre du message** : elle décrit la réponse qui doit être donnée au message.

Mise en situation : le jeu Mastermind

Présentation

Le **Mastermind** ou *Master Mind* est un jeu de société pour deux joueurs dont le but est de trouver un code. C'est un jeu de réflexion, et de déduction, inventé par Mordecai Meirowitz dans les années 1970 alors qu'il travaillait comme expert en télécommunications.

Il se présente généralement sous la forme d'un plateau perforé de 10 rangées de quatre trous pouvant accueillir des pions de couleurs.

Le nombre de pions de couleurs différentes est de 8 et les huit couleurs sont généralement : rouge ; jaune ; vert ; bleu ; orange ; blanc ; violet ; fuchsia.

Il y a également des pions blancs et rouges (ou noirs) utilisés pour donner des indications à chaque étape du jeu.

Principe

Un joueur commence par placer son choix de pions sans qu'ils soient vus de l'autre joueur à l'arrière d'un cache qui les masquera à la vue de celui-ci jusqu'à la fin de la manche.

Le joueur qui n'a pas sélectionné les pions doit trouver quels sont les quatre pions, c'est-à-dire leurs couleurs et positions.

Pour cela, à chaque tour, le joueur doit se servir de pions pour remplir une rangée selon l'idée qu'il se fait des pions dissimulés.

Une fois les pions placés, l'autre joueur indique :

– le nombre de pions de la bonne couleur bien placés en utilisant le même nombre de pions rouges ;
– le nombre de pions de la bonne couleur, mais mal placés, avec les pions blancs.

Il arrive donc surtout en début de partie qu'il ne fasse rien concrètement et qu'il n'ait à dire qu'aucun pion ne correspond, en couleur ou en couleur et position.

La tactique du joueur actif consiste à sélectionner en fonction des coups précédents, couleurs et positions, de manière à obtenir le maximum d'informations de la réponse du partenaire puisque le nombre de propositions est limité par le nombre de rangées de trous du jeu.

Variantes

Il existe de nombreuses variantes suivant le nombre de couleurs, de rangées ou de trous : fr.wikipedia.org/wiki/Mastermind.

Analyse orientée objet

Spécifications

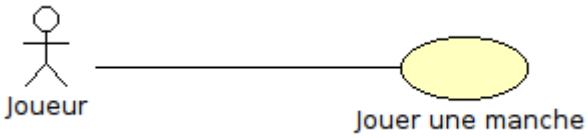
Il s'agit de réaliser une version logicielle du jeu **Mastermind** en respectant un développement orienté objet. Le logiciel sera réalisé sous GNU Linux avec une IHM (Interface Homme-Machine) basique de type "console". Les couleurs seront remplacées par des chiffres de 1 à 8.

Dans ce jeu de société pour deux joueurs, on distinguera les rôles :

– un joueur "humain" qui devra deviner la combinaison secrète (couleur et position des pions) ;
– un joueur "machine" qui assurera logiciellement le déroulement d'une manche en établissant la combinaison secrète puis en déterminant le nombre de pions de la bonne couleur bien placés et mal placés par rapport à la proposition indiquée par le joueur "humain" à chaque tour.

Cas d'utilisation

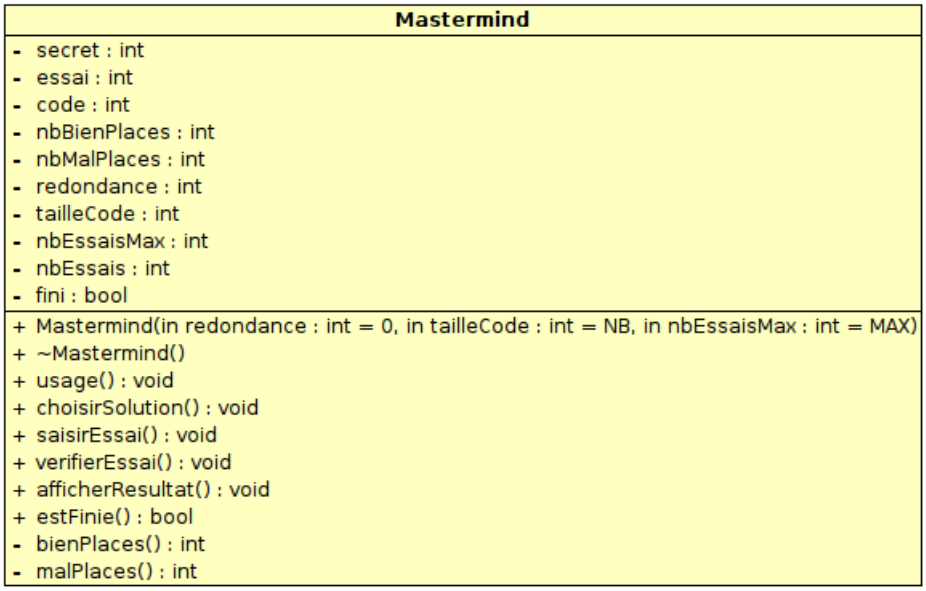
Le diagramme des cas d'utilisation (*Use Case*) du système est le suivant :



Un diagramme des cas d'utilisation décrit les fonctionnalités attendues du système du point de vue d'un utilisateur. Les Cas d'Utilisation (CU) recentrent l'expression des besoins sur les utilisateurs. Les cas d'utilisation sont donc très utiles pour représenter ce que doit faire un système par rapport à son environnement.

La classe `Mastermind`

On vous fournit la classe `Mastermind` qui implémente le jeu :



La déclaration de cette classe se trouve dans le fichier en-tête (*header*) `Mastermind.h`. Cette classe respecte le principe d'encapsulation.

Voici une brève description de ses méthodes publiques :

- `usage()` : affiche un texte informatif sur le jeu ;
- `choisirSolution()` : détermine aléatoirement la combinaison secrète et initialise une nouvelle manche ;
- `saisirEssai()` : assure la saisie d'une combinaison proposée par le joueur ;
- `verifierEssai()` : vérifie si la combinaison proposée par le joueur correspond à la combinaison secrète et détermine le nombre de pions de la bonne couleur bien placés et mal placés ;
- `afficherResultat()` : affiche l'état du tour (le numéro de tour, le nombre de tours restant et le nombre de pions de la bonne couleur bien placés et mal placés) et de la manche si elle est finie (en dévoilant la combinaison secrète en cas de défaite du joueur) ;
- `estFinie()` : retourne vrai (`true`) si la manche est fini sinon faux (`false`).

Les méthodes `bienPlaces()` et `malPlaces()` sont privées.

La classe `Mastermind` possède un **constructeur** qui permet de paramètrer le type de manche à jouer en indiquant :

- si la redondance des couleurs est admise dans le code ;
- le nombre de pions dans le code ;
- le nombre d'essais maximum pour deviner la combinaison secrète en une manche.

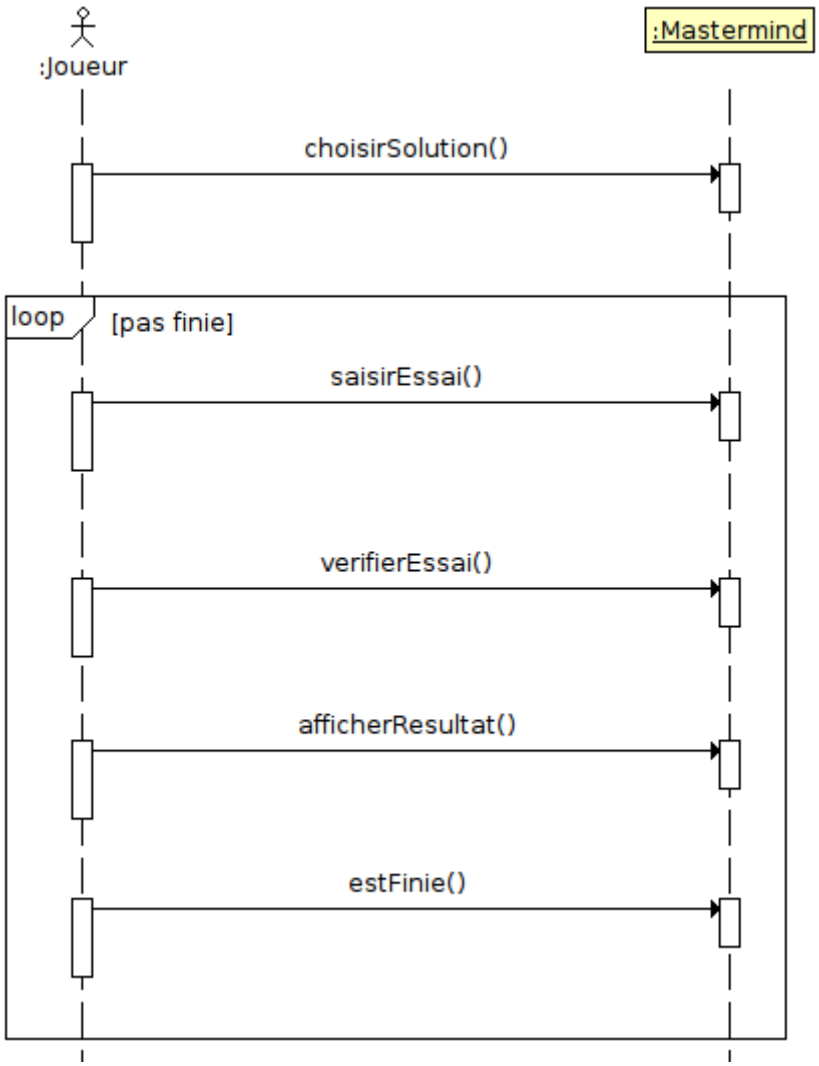
```
// Valeurs par défaut
#define NB 4   // nombre de pions dans le code secret
#define MAX 10 // nombre d'essais maximum
#define REDONDANCE 1 // la redondance des couleurs dans le code est admise

Mastermind(int redondance=0, int tailleCode=NB, int nbEssaisMax=MAX);
```

Déclaration du constructeur de la classe Mastermind

Le constructeur par défaut initialisera une manche avec 4 pions de couleurs uniques à deviner en 10 tours maximum.

Diagramme de séquence

Le diagramme de séquence du cas d'utilisation "jouer une manche" est le suivant :



Un diagramme de séquence est un diagramme d'interaction. Le but est de décrire comment les objets collaborent au cours du temps et quelles responsabilités ils assument. Il décrit un scénario d'un cas d'utilisation ou le cas d'utilisation lui-même. Un diagramme de séquence représente donc les interactions entre objets, en insistant sur la chronologie des envois de message. C'est un diagramme qui représente la structure dynamique d'un système car il utilise une représentation temporelle. Les objets, intervenant dans l'interaction, sont matérialisés par une « ligne de vie », et les messages échangés au cours du temps sont mentionnés sous une forme textuelle. Les messages sont des méthodes d'une classe et l'envoi d'un message correspond donc à l'appel de cette méthode.

Le diagramme de séquence ci-dessus utilise un **fragment combiné** de type *loop* (boucle) qui permet de répéter le fragment tant que la condition indiquée entre crochets (`[]`) est vérifiée. Évidemment, il existe d'autres type de fragments combinés comme *alt* (alternatif) qui est équivalent à la structure conditionnelle "`SI ... ALORS ... SINON ... FSI`".

Travail demandé

On vous fournit le programme `main.cpp` suivant :

```cpp
// Fichiers d'entete
#include "mastermind.h"
#include <iostream>

using namespace std;

// Programme principal
int main()
{
    // TODO :

    // 1. Instancier un objet de type Mastermind en utilisant le constructeur par défaut

    // Avant de jouer, il est conseillé d'afficher les règles du jeu

    // 2. Implémenter le diagramme de séquence "jouer une manche"

    return 0;
}
```

main.cpp

Question 1. Compléter le programme `main.cpp` ci-dessus en respectant les spécifications exprimées précédemment.

À la fin, vous devez pouvoir "joueur une manche" du **Mastermind** :

```
$ ./mastermind
Le Mastermind est un jeu de société dont le but est de trouver un code secret.
C'est un jeu de réflexion et de déduction, inventé par Mordecai Meirowitz dans les années
    1970.
Le nombre de couleurs différentes est de 8 (de 1 à 8).
Vous avez 10 essais pour deviner un code de 4 pions de couleurs strictement différentes.

Entrez votre essai : 1 2 3 4
Essai 1/10 : bien places 1, mal places 1
Entrez votre essai : 1 5 6 2
Essai 2/10 : bien places 0, mal places 3
Entrez votre essai : 5 2 1 7
Essai 3/10 : bien places 2, mal places 1
Entrez votre essai : 5 2 3 4
Essai 4/10 : bien places 2, mal places 0
Entrez votre essai : 5 2 8 1
Essai 5/10 : bien places 4, mal places 0
Felicitations !
```


Pour fabriquer l'application, il vous suffit de faire :

```
$ make
```

La commande `make` utilise par défaut un fichier `Makefile` se trouvant dans le répertoire courant. Voici son contenu :

```
CXX=g++

GENERIC_CFLAGS :=
GENERIC_LIBS :=

TARGET=mastermind

HEADERFILES    := $(wildcard *.h)
SRCFILES       := $(wildcard *.cpp)
OBJFILES       := $(patsubst %.cpp, %.o, $(SRCFILES))

.PHONY: clean $(TARGET)

all: $(TARGET)

%.o: %.cpp %.h
	$(CXX) -c $< $(GENERIC_CFLAGS) -o $@

$(TARGET): $(OBJFILES)
	$(CXX) $(GENERIC_CFLAGS) $(GENERIC_LIBS) -o $(TARGET) $^

clean:
	rm -f $(TARGET) $(OBJFILES) *~
```

Makefile

Voir fr.wikipedia.org/wiki/Make.

Question 2. Modifier le programme `main.cpp` afin de jouer une manche autorisant la redondance des couleurs pour une combinaison de 5 pions en 12 tours maximum.

Ne pas oublier de compléter correctement l'en-tête du fichier `main.cpp` pour chacune des versions à fournir.