Table des matières

 © 2015 tv - rev.93

Les activités en 2°année

Ces activités permettent de développer le travail en équipe (de 2 à 4 étudiants) et un apprentissage de la démarche de projet. C'est aussi une préparation à l'initiative et l'autonomie de réflexion et d'action, à la communication écrite ou orale, dans le cadre d'activités professionnelles.

Les activités apportent une dimension professionnelle la plus complète possible car ils intègrent différentes étapes et facettes d'un projet logiciel. Ils permettent aussi d'aborder les aspects de méthodologie de développement, de validation de comportement, de mise en œuvre de test.

Présentation

Il s'agit de réaliser un jeu simple de dés en programmation Orientée Objet.

Il sera développé en équipe de 2 étudiants **afin de mettre en oeuvre l'utilisation d'un atelier de génie logiciel (bouml)**.

Expression du besoin

Le client désire jouer une partie de jeu de dés. Le joueur demande à lancer les dés. Le système affiche le résultat. Si le total est égal ou supérieur à 7, le joueur a gagné. Dans tous les autres cas, il a perdu.

Moyens préliminaires disponibles et contraintes de réalisation

Spécifications

Le système est composé de 2 dés à 6 faces. Un dé sera modélisé par une classe du même nom.

La classe De possédera une méthode `lancer()` qui aura pour rôle de déterminer de façon pseudo-aléatoire (cf. la fonction `rand()` et `srand()`) la valeur du dé (une valeur comprise entre 1 et 6, cad le nombre de faces que comporte le dé).

La classe De doit respecter le **principe de séparation Commande-Requête**. c'est un principe de conception Orienté Objet classique pour les méthodes. Ce principe énonce que chaque méthode doit appartenir à l'une des deux catégories suivantes:

- une **commande** est une méthode qui effectue une action. Elle a souvent des effets de bords comme une modification de l'état d'un objet et n'a pas de valeur de retour (sauf pour indiquer si l'action a réussi ou a échoué) ;
- une **requête** est une méthode qui retourne des données à l'appelant et n'a pas d'effets de bord. Elle ne doit pas modifier de façon permanente l'état d'un objet.

La classe De possédera une méthode `lancer()` qui est une commande : elle a pour effet de modifier la **valeur** du dé. En conséquence, elle ne doit pas également

retourner cette nouvelle valeur sinon elle violerait la règle selon laquelle elle ne doit pas appartenir aux deux catégories. Pour obtenir la valeur du dé lancé, on utilisera une méthode `getValeur()` qui est une requête.

C'est un ***pattern*** simple : il permet de raisonner plus facilement sur l'état d'un programme et de rendre les conceptions plus faciles à comprendre. Il est donc agréable de pouvoir lui faire confiance.

*Ce qu'il faut retenir : Un **pattern** (ou motif de conception) est un document qui décrit une solution générale à un problème qui revient souvent.*

Dans le monde de l'orienté-objet, les design patterns se présentent comme un catalogue de méthodes de résolution de problèmes récurrents.

La classe `De` possédera un constructeur par défaut (qui fixera un nombre de faces égal à 6). Pour des soucis de réutilisation, elle disposera aussi d'un constructeur auquel on pourra passer le nombre de face désiré pour ce dé.

Remarque : il n'est pas demandé de modéliser une classe `Joueur`. On se limitera donc pour l'instant à une seule classe pour l'application.

Contrainte de développement

C'est le programme `main.cpp` qui symbolisera le joueur. On y créera les deux dés et on jouera au moins une partie.

Contrainte de l'environnement

Système d'exploitation : **Linux**
Environnement de développement : **GNU C++**
Atelier de génie logiciel : **bouml**

Contrainte économique

Aucune

Documents et moyens technologiques mis à disposition

Documents :

- Les documentations suivantes :

Ref.	**Description**
exemple-pratique-bouml-1-creation-de-classe.pdf	Tutoriel présentant la création de classe sous bouml
exemple-pratique-bouml-2-generation-de-code.pdf	Tutoriel présentant la génération automatique de code sous bouml
cours-c-c++-programmation-modulaire.pdf	Cours sur la fabrication modulaire en C/C++ présentant notamment l'utilisation de l'outil make et son fichier Makefile

Liens :

- génération de code avec bouml : http://bpages.developpez.com/tutoriels/bouml/classes-generation/

Moyens technologiques :

- Accès Internet

- vidéo-projecteur avec écran interactif et rétro-projecteur

Exigences qualité à respecter

Exigences qualité sur le produit à réaliser

Le produit à réaliser doit répondre aux facteurs de qualité suivants :

- maniable : il sera facile d'emploi avec une interface homme-machine simple et conviviale
- robuste : il conservera un fonctionnement conforme aux spécifications après un arrêt normal ou d'urgence; garantira la validité des informations échangées.
- maintenable : il offrira une bonne facilité de localisation et correction des problèmes résiduels.
- adaptabilité : il facilitera la suppression, l'évolution de fonctionnalités existantes ou l'ajout de nouvelles fonctionnalités
- portabilité : il minimisera les répercussions d'un changement d'environnement logiciel et matériel

Exigences qualité sur le développement

En ce qui concerne les exigences qualité du développement :

- L'architecture du logiciel sera Orientée objet.
- Le codage doit respecter le standard C/C++ en cours dans la section
- Un utilitaire de compilation automatisé de type « make » sera utilisé

Exigences qualité sur la documentation à produire

Les exigences qualité à respecter, relativement aux documents, sont :

- sur leur forme : respect de normes et de standards de représentation, homogénéité, lisibilité, maintenabilité ;
- sur leur fond : complétude, cohérence, précision.

Exigences qualité sur la livraison

Les produits à mettre à disposition du client sont :

- **la modélisation UML (sous bouml) ;**
- **les codes sources de l'application de la dernière version livrable ainsi que le fichier de type Makefile.**

Ces produits seront livrés sous forme informatique regroupés dans une archive au format `tar.gz` ou `zip`. Le nom de l'archive sera formaté de la manière suivante : **`mp0-teamN-vX.Y.tar.gz`** ou **`mp0-teamN-vX.Y.zip`** où :

- N représente le numéro de l'équipe de développement (numéro donné par l'enseignant)
- X le numéro de version majeur de l'application
- Y le numéro de version mineur de l'application (0 par défaut)

Exigences qualité sur l'environnement d'exploitation

Aucune

 © 2015 tv - rev.93

Exploitation pédagogique

Plus spécifiquement dans cette activité :

Activités professionnelles	Codage et réalisation
Capacité	REALISER
Compétences terminales susceptibles d'être abordées et évaluées	
Compétence terminale	Critères d'évaluation
Coder un module logiciel	- Qualité de la traduction en code de la spécification logicielle. Respect de la syntaxe du langage utilisé. - Degré de maîtrise de la manipulation des pré-processeurs, compilateurs, interpréteurs, assembleurs, lieurs, ... - Utilisation correcte d'un système de fichiers. - Qualité du document. Pertinence des commentaires. Clarté de la présentation des points d'entrée du module.

Activités professionnelles	Installation et exploitation
Capacité	INSTALLER
Compétences terminales susceptibles d'être abordées et évaluées	
Compétence terminale	Critères d'évaluation
Mettre en oeuvre un environnement de programmation - Installer et configurer un environnement de développement. - **Utiliser les assistants de l'environnement pour créer les squelettes de l'application, des classes, des fonctions et des messages.**	- Fonctionnement de l'environnement de développement. - Propriétés du squelette de chacun des éléments.

Planification des tâches

Ref.	Description	L	M	M	J	V	S	D	L	M	M	J	V
T2.1	Interprétation des spécifications logicielles et identification des fonctions principales												
T2.8	Utilisation des diagrammes de classe pour produire une maquette de l'application												
T3.3	Codage et assemblage des modules logiciels (dans le respect des standards entreprise)												
T3.4	Fabrication de modules logiciels réutilisables et réalisation de la documentation associée												
T3.7	Élaboration de documents de suivi de réalisation et de codage												
T8.1	Intégration et travail dans une organisation par projet												
T8.5	Renseignement des indicateurs permettant le suivi d'un projet												

 © 2015 tv - rev.93

Travail demandé

- identifier la fonctionnalité à réaliser pour le client
- spécifier et créer la classe De à partir d'un atelier de génie logiciel
- générer automatiquement le squelette de la classe De à partir d'un atelier de génie logiciel
- coder la fonctionnalité à réaliser ainsi que la classe De à partir d'un environnement de développement intégré
- assurer la synchronisation entre l'AGL (uml) et l'EDI (c++)
- assurer la validation du logiciel
- présenter oralement le diagramme de classe
- assurer une démonstration du logiciel

Grille d'évaluation

Activité :		Livraison finale		
Équipe n°__ :		Version : ____		
Critères		-	0	+
Gestion de projet	Respect du travail en équipe			
	Respect du cahier des charges			
	Existence d'une modélisation UML			
Oral	Qualité de la présentation, précision, rigueur, clarté			
	Utilisation des moyens mis à la disposition et du vidéo-projecteur			
Démonstration	Qualité de la démonstration orale : précision, rigueur, clarté, ...			
Application	L'application livrée respecte les contraintes			
	L'application livrée respecte les exigences qualité			
Entretien	Capacité à répondre avec pertinence, précision et exactitude			
	Capacité à rechercher et à exploiter une documentation			
	Capacité à argumenter et à réagir aux objections			
Bilan	État d'avancement			
Remarques :				

 © 2015 tv - rev.93

Aller plus loin

En appliquant un **cycle de développement itératif et incrémental**, il est possible d'ajouter de nouvelles fonctionnalités pour évoluer l'application.

*Ce qu'il faut retenir : Un développement itératif s'organise en une série de développement très courts de durée fixe nommée itérations. Le résultat de chaque itération est un **système partiel exécutable, testé et intégré** (mais incomplet). Comme le système croît avec le temps de façon incrémentale, cette méthode de développement est nommée **développement itératif et incrémental**.*

On a déjà vu que les dés sont des objets génériques utilisables dans toutes sortes de jeu. L'affectation de la responsabilité au programme principal de les lancer et de les additionner empêche de réutiliser ce service dans d'autres jeux. Il y a un autre problème : il n'est pas possible de demander simplement le total actuel des dés sans devoir les lancer de nouveau.

On va donc introduire un nouveau concept (une nouvelle classe) :

- Il faut toujours essayer d'employer un vocabulaire en rapport avec le domaine. De nombreux jeux de plateau proposent d'utiliser un cornet pour secouer les dés et les lancer sur la table. On propose donc de créer une classe **Cornet** qui contiendra les dés, les lancera et connaîtra leur total.

Vous pouvez donc commencer l'**itération n°2** en modifiant quelque peu les règles du jeu de dés (cf. **Annexe 2**).

Pour couvrir toutes les utilisations possibles d'un atelier de génie logiciel, on vous propose de mettre en œuvre la fonction **Reverse** de bouml. Pour cela, vous devez écrire le squelette de la classe Cornet (les fichiers Cornet.cpp et Cornet.h) afin de pouvoir compiler le programme fourni en Annexe 2. Comme il faut toujours conserver l'ensemble des documents (uml et code source) parfaitement synchronisé, il vous faudra donc importer cette nouvelle classe dans bouml : c'est ici que la fonction Reverse devient intéressante.

 © 2015 tv - rev.93

Annexe 1 : le fichier Makefile

```
CC = g++
RM = rm
MAKEDEP = makedepend

INCLUDES = -I.
CFLAGS = -g $(INCLUDES)
LIBS =

OBJS = main.o de.o
TARGET = jeuDeDes

$(TARGET): $(OBJS)
	$(CC) $(CFLAGS) $(LIBS) $(OBJS) -o $@

.cpp.o:
	$(CC) -c $(CFLAGS) $<

clean:
	$(RM) -f *.o $(TARGET)

cleanall:
	$(RM) -f *~ *.o $(TARGET)

dep:
	$(MAKEDEP) $(CFLAGS) -Y -s "# Dependances by 'make dep'" *.cpp
2> /dev/null
	rm -f Makefile.bak
```

 © 2015 tv - rev.93

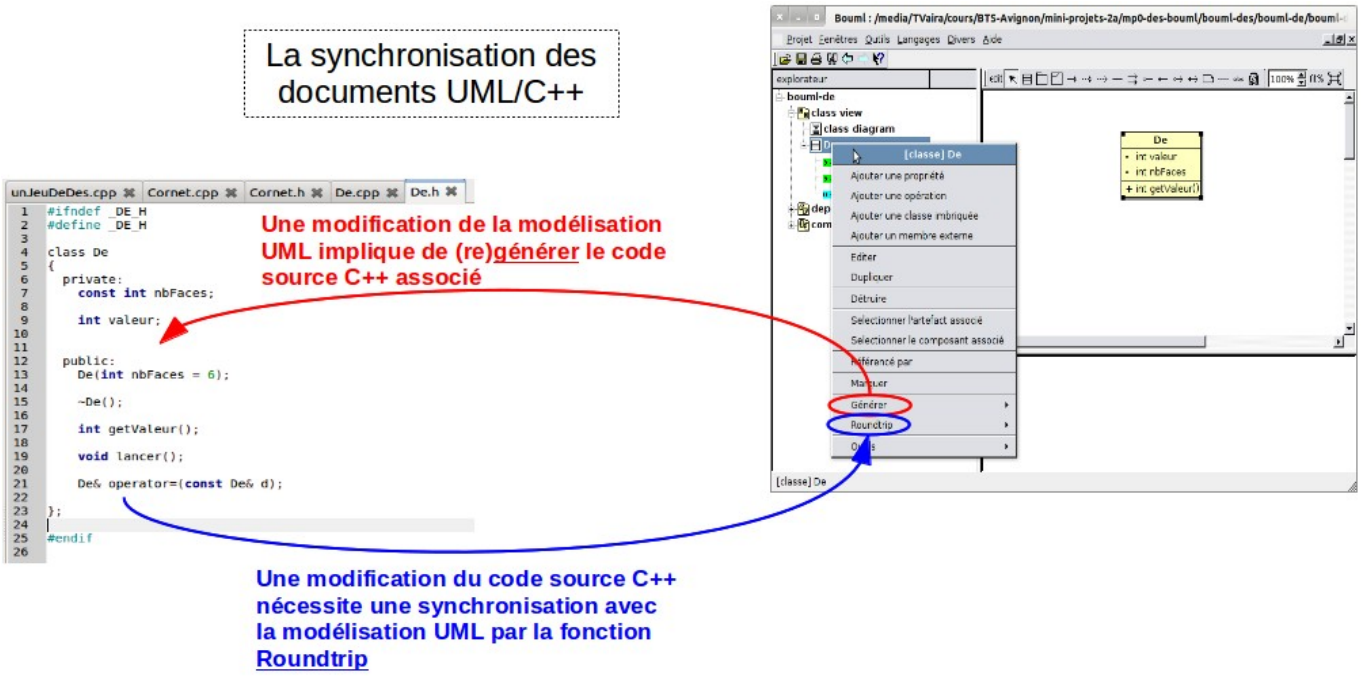
Annexe 2 : un jeu avec un cornet

```
#include <iostream>
#include "Cornet.h"

using namespace std;

int main( int argc, char* argv[])
{
  Cornet cornet;
  int score = 0;

  cornet.lancer();
  score = cornet.getScore();

  if(score == 20)
       cout << "Bravo, vous avez fait un double " << cornet.getValeurDe() << "
et vous avez gagné " << score << " points !" << endl;
  else if(score == 15)
       cout << "Bravo, vous avez fait une suite " << cornet.getValeurDe(1) << "
" << cornet.getValeurDe(2) << " et vous avez gagné " << score << " points !"
<< endl;
  else
       cout << "Vous avez réalisé " << cornet.getValeurDe(1) << " " <<
cornet.getValeurDe(2) << " et vous marquez " << score << " points !" << endl;

  return 0;
}
```

Remarque : n'oubliez pas de mettre à jour votre fichier Makefile !

 © 2015 tv - rev.93

Annexe 3 : la synchronisation UML/C++



 © 2015 tv - rev.93