TP3 : Jeu de l'ange et du diable

BUT : (*Durée prévue : 2h*)
- Prise en main de l'outil visual studio pour la modélisation ;
- Utilisation de l'héritage et du polymorphisme.

0.1 Règles du jeu

Le jeu de l'Ange et du Diable oppose un ange et un diable sur un damier de taille N constitué de NxN cases. Au départ du jeu, l'ange est situé sur la case centrale du damier et les autres cases sont vides. Les deux joueurs jouent à tour de rôle. À son tour, l'ange se déplace vers une case vide située à une distance inférieure à sa puissance. À son tour, le diable bouche une case vide du damier, qui reste bouchée tout le long de la partie. Le but de l'ange est d'atteindre une case du bord du damier et celui du diable de bloquer l'ange sur une case du damier qui ne soit pas une case du bord.

Vous trouverez les sources du programme C++ permettant de jouer à ce jeu sur le moodle dans le répertoire "Ressources TP3". La version du programme mise à votre disposition n'a pas été écrite en utilisant toute la puissance de l'orientation objet de C++. Ce TP va permettre d'améliorer ce programme. Dans ce but, nous allons analyser le programme et construire son modèle UML. Ensuite, nous améliorerons le modèle. Ainsi les méthodes compliquées du programme C++ existant seront ré-écrites plus simplement.

0.2 Analyse du programme

Dans un premier temps, créer un projet win32 sous Visual Studio 2010. Importer dans ce projet les fichiers .cpp et .h fournis dans le répertoire "Ressources TP3".

Question 1 *Générez automatiquement le diagramme de classes associé au programme du jeu. Faites apparaître les associations pertinentes. Incluez le diagramme de classes à votre compte rendu de TP.*

Notes :
- *Pour afficher le diagramme de classes : cliquez droit sur le projet et sélectionnez « Afficher le diagramme de classes ».*
- *Pour faire apparaitre les associations entre classes : cliquez droit sur le champ est sélectionnez « Afficher en tant qu'association »*

Question 2 *La classe Ange est-elle associée à la classe Case ? La classe Diable est-elle associée à la classe Case ?*

Question 3 *Quelles sont les méthodes redéfinies ?*

Question 4 *Peut-on faire jouer deux joueurs humains l'un contre l'autre ?*

Question 5 *Peut-on faire jouer deux joueurs aléatoires l'un contre l'autre ?*

Question 6 *Peut-on faire jouer un joueur humain contre un joueur aléatoire ?*

Question 7 *Peut-on faire jouer deux anges l'un contre l'autre ?*

Question 8 *Peut-on faire jouer deux diables l'un contre l'autre ?*

Question 9 *Un joueur humain peut-il jouer le diable ? L'ange ?*

Question 10 *Un joueur aléatoire peut-il jouer le diable ? L'ange ?*

Question 11 *En une phrase, que ferait le programme si on enlevait le mot-clé* `virtual` *devant la méthode* `jouer()` *de la classe Joueur ?*

0.3 Modélisation UML

Question 12 *Créez un nouveau projet de modélisation. Dans ce projet créez un diagramme de séquence UML correspondant à l'interaction de l'utilisateur avec ce programme dans le cas évoqué à la question 6.*

0.4 Polymorphisme et héritage

Question 13 *Factorisez les fonctions* `effectuerCoupSurCase(Case * c)` *et* `jouer()` *en les remontant dans les classes pertinentes.*

Pour simplifier les méthodes `Case * choisirUneCase()` *des joueurs humains, la méthode* `bool verifier(int x,int y))` *doit être définie. Cette méthode correspond à la vérification qu'un coup, désigné par les valeurs (x,y) entrées par un joueur humain, est valide (le diable ne peut pas être sur un bord, l'ange ne peut pas dépasser sa puissance).*

0.4.1 Méthode *verifier*

Question 14 *Supposons que l'on définisse une méthode* `bool verifier(int x,int y)` *dans la classe AngeHumain et une autre dans la classe DiableHumain. Quelle serait la différence entre ces deux méthodes ?*

Question 15 *Définissez une méthode* `bool caseInaccessible(Case * c)` *dans la classe Ange. Cette méthode renvoie vrai si l'ange ne peut pas atteindre la case c.*

Question 16 *Écrivez la méthode* `bool verifier(int x,int y)` *dans les classes DiableHumain et AngeHumain. Attention, on souhaite que les méthodes* `verifier` *des classes AngeHumain et DiableHumain soient identiques. Définir la/les méthode(s) nécessaires dans la classe adéquate. Les méthodes* `verifier` *des classes AngeHumain et DiableHumain étant strictement identiques, peut-on les remplacer par une méthode identique plus générale ?*

0.4.2 Méthode *choisirUneCase*

Question 17 *Écrivez les méthodes* `Case * choisirUneCase()` *des classes AngeHumain et DiableHumain en utilisant les méthodes* `verifier` *et* `afficherPrompt`*. Attention, les méthodes* `choisirUneCase` *des classes AngeHumain et DiableHumain sont identiques.*

Question 18 *Écrivez les méthodes* `Case * choisirUneCase()` *des classes AngeAleatoire et DiableAleatoire en utilisant la méthode* `caseInaccessible`*.*

0.4.3 Héritage multiple

Une joueur humain ne peut appeler la méthode `Case * choisirUneCase()` *d'un joueur aléatoire et réciproquement, ce qui est un avantage. Toutefois, l'inconvénient est que la méthode* `Case * choisirUneCase()` *des joueurs humains est écrite deux fois (une fois dans DiableHUmain et une fois dans AngeHumain), ainsi que la méthode* `bool verifier(int x,int y)`*. De la même façon la méthode* `Case * choisirUneCase()` *des joueurs aléatoires est dupliquée.*

Question 19 *Une proposition est de changer l'ordre des discriminations du diagramme de généralisation. C'est-à-dire qu'aulieu d'avoir l'héritage suivant :*

On a le découpage suivant :

Donnez les avantages et les inconvénients de cette proposition.

Question 20 *Une autre proposition est de faire de l'héritage multiple. Quelles sont les deux classes à ajouter ? Réalisez les. Une fois que vous les avez intégrées au programme, générez le diagramme de classes que vous devrez joindre à votre compte rendu de TP.*

0.5 À rendre

Vous devez rendre un .zip contenant :

- *Votre compte rendu de TP contenant vos réponses aux questions posées dans le sujet. De plus vous devrez y inclure les diagrammes de classes et de séquences qui vous sont demandés.*
- *Vos fichiers sources du projet.*