MODULE C# **BotFactory**

Visual Studio 2015 :
- Raccourcis
- Configuration

C# 6 :
- Classes
- Interfaces
- Abstraction
- Polymorphisme
- Génériques
- Réflexion
- Héritage
- Programmation asynchrone
- Délégués
- Bibliothèques

.NET 4.6 :
- Tasks
- Threading
- Events
- Collections

Les robots doivent :

- Etre constructibles en série.

- Etre mobiles (pouvoir se déplacer d'eux mêmes)

- Le déplacement des robots doit se faire en temps réel (simulation du déplacement)

- Les robots doivent pouvoir rapporter leurs actions pour vérification par des contrôleurs

- Pouvoir leur dire quand lancer le travail et quand l'arrêter pour aller se recharger

L'usine doit :

- Pouvoir construire les robots constructibles

- La construction des robots doit se faire en temps réel (simulation du temps de construction)

- Pouvoir mettre en queue la construction des robots

- Pouvoir entreposer les robots construits

- Avoir une limite pour la queue

- Avoir une limite pour l'entrepôt

Les taches :

Jour 1

Etape 1 : Création de la solution

On vous demande de créer une solution du nom de BotFactory dans le dossier F:\Projects\BotFactory. Le dossier F:\Projects\ allant servir pour de futurs projets, il est conseillé de configurer Visual Studio pour avoir ce dossier comme dossier par défaut des projets.

Etape 2 : Création de la bibliothèque

Les robots doivent pouvoir être chargés dans différents projets, il faut donc ajouter un projet bibliothèque à la solution. Le nom de la bibliothèque sera Models.

Etape 3 : Création de la première classe

Le client souhaite que l'usine fabrique uniquement les robots constructibles.

- Créer une classe abstraite BuildableUnit dans le namespace BotFactory.Models.

- Ajouter une propriété publique BuildTime du type double

- Créer un constructeur prenant en paramètre le temps de construction.

Le client pense que 5 secondes est un bon temps des construction par défaut.

Etape 4 : Mise en place de la mobilité du robot

Le client souhaite que tous les robots soient mobiles.

- Créer une classe abstraite BaseUnit héritant de BuildableUnit.

- La classe doit se trouver dans le même namespace que sa classe parente.

- Créer un constructeur prenant en paramètre le nom et la vitesse du robot.

- Créer une méthode asynchrone publique Move, prenant en paramètre des coordonnées sur un plan (X, Y).

- Ajouter une propriété publique Name renvoyant le nom du robot.

Le client souhaite que la vitesse par défaut des robots soit de 1.

Pour calculer le chemin a prendre quand la méthode Move est appelée, il faut deux informations :

1. La position actuelle du robot
2. La position souhaitée du robot

- Ajouter une propriété publique CurrentPos renvoyant la position actuelle de l'unité
La position (0, 0) sera mise comme position courante par défaut.

- Créer une classe Coordinates

- Ajouter une propriété publique X du type double

- Ajouter une propriété publique Y du type double

- Surcharger la méthode Equals pour comparer les coordonnées proprement

Pour calculer le temp de déplacement du robot, il faut calculer le chemin qui sera exprimé en vecteurs.

- Créer une classe Vector

- Ajouter une propriété publique X du type double

- Ajouter une propriété publique Y du type double

- Ajouter une méthode publique statique FromCoordinates prenant en paramètre begin et end du type Coordinates et renvoyant Vector. Elle doit calculer le vecteur à partir des points de départ et arrivée du vecteur.

- Ajouter une méthode publique Length renvoyant la longueur du vecteur

Les deux classes Coordinates et Vector seront utilisées ailleurs que dans la bibliothèque Models.

- Créer une nouvelle bibliothèque appelée Common

- Déplacer les deux classes dans la bibliothèque Common

- Changer le namespace des classes en BotFactory.Common.Tools

- Ajouter la référence de la bibliothèque Common dans la bibliothèque Models

Maintenant que les nouvelles classes requises ont été créées :

- Modifier la méthode Move pour qu'elle calcule le temps de parcours

- Ajouter un délai d'exécution du déplacement dans la méthode Move pour simuler le déplacement du robot

- Des taches autres que le déplacement doivent pouvoir s'exécuter pendant que le déplacement est en cours d'exécution. Rendre la méthode Move asynchrone pour permettre cela.

Jour 2

Etape 5 : Ajout du reporting

Le client souhaite que les robots puissent faire du reporting sur leurs activités pour que des contrôleurs puissent vérifier leur bon fonctionnement :

Pour faire du reporting, il faut que les robots déclenchent des évènements et renvoient des arguments aux observateurs.

- Créer une classe abstraite ReportingUnit

- Ajouter un évènement publique UnitStatusChanged

- Ajouter une méthode surchargeable OnStatusChanged prenant en paramètre les arguments du changement du statut du type EventArgs.

La méthode OnStatusChanged doit faire appel à UnitStatusChanged en lui passant le nouveau statut.

Créer dans le même namespace une classe StatusChangedEventArgs héritant de EventArgs

- Ajouter une propriété NewStatus du type string

- Remplacer le type de paramètre de OnStatusChanged par StatusChangedEventArgs

Le reporting doit pouvoir être fait par n'importe quel robot construit, même s'il n'est pas mobile.

- Ajouter la classe ReportingUnit dans la chaine d'héritage entre BuildableUnit et BaseUnit

Etape 6 : Ajout de la classe de travail

Le client souhaite que les robots puissent se rendre à leur lieu de travail, ou retourner se recharger de façon indépendante.

- Créer une classe abstraite WorkingUnit héritant de BaseUnit

- Ajouter une propriété ParkingPos du type Coordinates

- Ajouter une propriété WorkingPos du type Coordinates

- Ajouter une propriété IsWorking du type bool

- Ajouter une méthode asynchrone surchargeable WorkBegins

- Ajouter une méthode asynchrone surchargeable WorkEnds

La méthode WorkBegins doit s'assurer que le robot aille se mettre à travailler sur son emplacement du travail WorkingPos, et ce quelle que soit sa position au moment ou la méthode est appelée.

La méthode WorkEnds dois s'assurer que le robot aille se recharger à sa position de stationnement ParkingPos, et ce quelle que soit sa position au moment ou la méthode est appelée.

Il est conseillé de découper les différentes taches effectuées par le robot pour aller travailler ou aller se recharger dans le but de garder le code propre et compréhensif. De plus, il faut que les différentes méthodes crées dans ce but soient surchargeables par les classes héritant de la classe WorkingUnit pour pouvoir adapter les taches suivant le modèle du robot.

Etape 7 : Le premier modèle

Implémenter les modèles suivants :

- Classe R2D2, Vitesse 1.5, Temps de construction 5.5

- Classe HAL, Vitesse 0.5, Temps de construction 7

- Classe T-800, Vitesse 3, Temps de construction 10

- Classe Wall-E, Vitesse 2, Temps de construction 4

Le namespace doit être BotFactory.Models

Etape 8 : Création des interfaces

Avant de passer à la création de l'usine, il faut s'assurer de pouvoir manipuler les objets proprement sans avoir faire de conversions de type sur les classes même.

- Créer une interface pour chaque classe contenant uniquement les membres publiques de la classe

Les interfaces doivent être placées dans la bibliothèque Common dans le namespace BotFactory.Interface pour permettre l'utilisation des interfaces tout en permettant la permutation des modèles.

Jour 3

Etape 9 : L'usine

Le client veut être capable d'ajouter de nouvelles usines à l'avenir, voir même les intervertir.

- Créer une nouvelle bibliothèque appelée Factories

- Créer une classe UnitFactory dans le namespace BotFactory.Factories

- Ajouter un constructeur prenant en paramètre la taille de la queue et la taille de l'entrepôt de l'usine. Ces paramètres ne doivent pas être modifiables en dehors du constructeur.

- Ajouter la propriété QueueCapacity du type int

- Ajouter la propriété StorageCapacity du type int

L'usine a besoin de mettre en queue les ordres de construction.

- Créer la classe FactoryQueueElement

- Ajouter la propriété Name du type string

- Ajouter la propriété Model du type Type

- Ajouter la propriété ParkingPos du type Coordinates

- Ajouter la propriété WorkingPos du type Coordinates

- Intégrer le nouveau modèle à l'usine en ajoutant une propriété Queue du type Queue<FactoryQueueElement>

L’usine à aussi besoin d’entreposer les robots construits dans un entrepôt. Ces robots ont aussi besoin d’être prêts à être testes.

- Créer une interface ITestingUnit comportant tous les membres des interfaces des robots

- Intégrer la nouvelle interface à l’usine en ajoutant une propriété Storage du type List<ITestingUnit>

Le client souhaite pouvoir ajouter des modèles à construire à la volée dans la queue de construction de l’usine.

- Ajouter une méthode AddWorkableUnitToQueue prenant en paramètre le modèle, le nom du robot, sa position de stationnement, ainsi que sa position de travail.

La méthode doit répondre aux critères suivants :

- L’usine ne peut construire qu’un robot à la fois.

- L’usine ne peut enregistrer plus de commandes si sa queue est pleine

- L’usine ne peut construire plus de robots si son entrepôt est plein

- On peut appeler l’ajout de commande n’importe quand

- La méthode doit retourner false si la commande n’est pas enregistrée

- La construction doit être active tant que la queue n’est pas vide ou l’entrepôt plein

- La construction d’un robot doit être simulée et prendre le temps indiqué par la propriété BuildTime du robot

Etape 10 : Ajout de reporting

Comme pour les robots, l'usine doit faire du reporting pour que les contrôleurs s'assurent qu'elle fonctionne correctement.

- Ajouter un évènement FactoryProgress rapportant si l'usine à commencé ou fini la construction d'un robot en renvoyant l'élément de la queue dans le premier cas, et le robot testable dans le deuxième cas.

Jour 4

Etape 11 : Un peu de clarté

Pour la clarté du contrôleur, ajouter les informations suivantes :

- Propriété QueueFreeSlots

- Propriété StorageFreeSlots

Ces propriétés indiqueront les emplacement libres pour la queue et l'entrepôt respectivement.

Ajouter aussi une propriété Model du type string renvoyant le nom du modèle du robot dans la classe BuildableUnit.

Etape 12 : Le retour des interfaces

L'usine devant être intervertible, il serait sage de créer une interface contenant les membres publiques de l'usine.

Jour 5

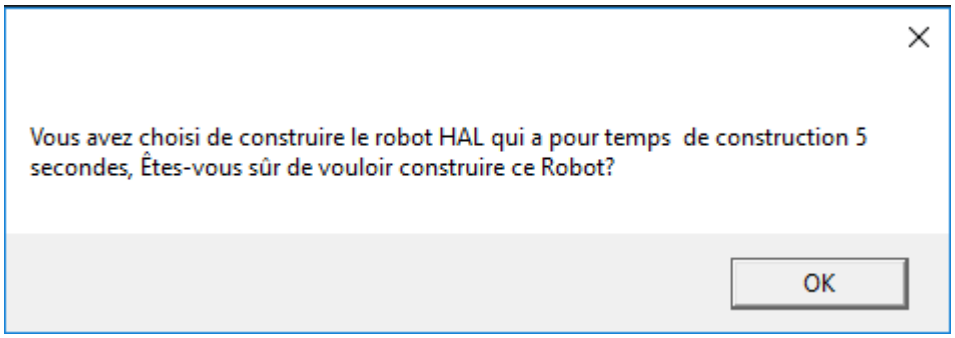
Etape 13 : Vérification visuelle

Ajouter à la solution le projet BotFactory. Compiler et lancer le projet après avoir fait de BotFactory le projet de démarrage. Tester visuellement le bon fonctionnement de l'usine.

Etape 14: Modification de l’interface graphique

Le but de cette étape est de modifier l’interface graphique du projet:

- Il faut remplacer les boutons permettants de rajouter les robots par un DropdownList qui nous permet de sélectionner le nom du robot qu'on souhaite construire.
- En cliquant sur le bouton "Add Unit To Queue", un MessageBox qui nous demande de confirmer l'opération doit être affiché.



Etape Bonus :

Si vous avez fini le projet en avance, prenez le temps de revoir le code, trouver de possibles améliorations et les implémenter.

Pensez à faire une sauvegarde du projet avant de commencer les améliorations. Il est aussi conseillé de prendre des notes des points d'amélioration.