Creation d’une nouvelle classe pour une centrale

Dans le cas de la création d’une nouvelle Eolienne

Premièrement il faut savoir que toutes les centrales sont reliées à la classe Centrale. Donc pour connaitre les différentes méthodes déjà utilisables il suffit de parcourir la classe Centrale, et dans le cas ou aucune méthode ne convient il faudra d’abord la crée dans la Classe centrale.

Toutes les centrales ont une déclaration de base qui est similaire.

```csharp

public class Fusion : Centrale //classe pour créer/gérer une source d'énergie

{

public double price;

public Fusion (double max\_prod, double co2, string name, Market market) : base(max\_prod, co2, name)

{

this.price = market.Get\_n\_price(); //récupération du prix KWh du marché

}

public override double Get\_prod() // production

{

return max\_prod;

}

public override double Get\_prix() // prix pour produire l'énergie nucléaire

{

return price \* this.Get\_prod();

}

public override double Get\_co2() // CO2

{

return base.Get\_co2();

}

public override string Get\_name()

{

return base.Get\_name();

}

}

```

Dans le cas d’une centrale nucléaire il n’y a pas besoin de plus, cependant si cette centrale est influencée par la météo, par exemple, il faudra rajouter des méthodes.

1. Le cas où la méthode existe déjà.

Si la méthode est déjà présente dans la classe centrale il suffit simplement de la rajouter dans la nouvelle classe et de rajouter les paramètres qui y sont lier.

Par exemple pour avoir une valeur de vent il faut rajouter la méthode Get\_vent() pour cela il faut rajouter plusieurs chose.

Une variable globale à la classe.

```csharp

Float wind = 0 ;

```

Ensuite dans la déclaration de la classe il faut rajouter l’appel de la classe Meteo car c’est a cette endroit la que l’on récupèrera notre variable de vent. Puis il faut ajouter le methode Get\_wind() de Meteo et l’assigner a notre variable globale crée plus tôt. Cela nous donne.

```csharp

public Fusion (double max\_prod, int co2, string name, Market market , Meteo meteo) : base(max\_prod, co2, name)

{

this.wind = meteo.Get\_wind(); //récupération de la vitesse du vent depuis la classe météo

this.price = market.Get\_x\_price(); //récupération du prix KWh du marché

}

```

Ensuite il nous suffit simplement de rajouter la methode Get\_vent().

```csharp

public override float Get\_vent(Meteo meteo)

{

wind = meteo.Get\_wind();

return wind;

}

```

En plus de cela il faudra modifié la méthode Get\_prod() car la puissance délivré par une éolienne dépend de la vitesse du vent. Donc cela deviendra :

```csharp

public override double Get\_prod() // production

{

coeff = (wind < 50) ? (wind / 100) : 0; //on ne fait pas fonctionner la centrale si le vent est trop fort (+180Km/h ~=50m/s))

return max\_prod \* coeff;

}

```

En n’oubliant pas de le mettre en globale de la classe pour pouvoir le récupérer autre part si besoin.

```csharp

public double price;

```

Si la centrale rajoutée n’existe pas encore, il est très probable qu’il faille aussi rajouter/modifier la méthode Get\_x\_price().

```csharp

this.price = market.Get\_x\_price();

```

--------------------------------------------------------------------------------------------------------------------------------------

# Projet POO 20325 20194

Ce projet est une plateforme développée en C# destinée à des chercheurs qui désirent simuler un réseau électrique à l’échelle désirée.

Ce dernier possède une plateforme facile à modifier.

Il est possible d’ajouter des sources productrices d’énergie, créer de nouveaux types de producteurs énergétiques, ajouter autant de consommateurs que l’on désire, créer de nouveaux types de consommateurs.

Un système de météo est présent qui permet de simuler 3 paramètres : la force du vent, l’ensoleillement ainsi que la température.

Un système de marché permet de gérer les prix unitaires pour plusieurs types d’énergies ainsi que le prix auquel l’électricité est achetée depuis l’étrangers.

## Production

Lors du développement de ce projet, nous avons décidé de créer trois types de producteurs d’énergie sans pour autant limiter ce nombre à l’avenir (nucléaire, éolien, achat à l’étrangers).

Pour créer une centrale d’un type connu, il faut ajouter du code dans la classe Update :

Il faut commencer par déclarer le nom de la centrale (x1) au-dessus du constructeur \*Update()\* :

```csharp

Centrale x1 ;

```

Il faut également ajouter une variable pour stocker la valeur de production :

```csharp

double Xo1;

```

Ensuite, il faut créer une instance au sein du constructeur \*Update()\* :

```csharp

x1 = new Eolien(2000, 1, "x1", market, bruxelles);

```

x1 est le nom utilisé dans le code lors d’appel de méthodes propres à cette instance, ou bien quand on doit récupérer ses attributs. Eolien est le type de centrale crée (il existe pour le moment Nucléaire, Eolien, et Achat) au sein des parenthèses, il faut mettre dans le bon ordre les paramètres de la centrale. On commence par la production maximale théorique exprimée en W, ensuite la quantité de CO2 produite par W. Market est le nom de la plateforme qui gère les prix de production et d’achat dans notre programme (objet de type Market) et dans le cas de l’éolien, nous avons également en paramètre la plateforme de météo (bruxelles) pour pourvoir récupérer la force du vent et ainsi produire de l’énergie en fonction de ce dernier.

À présent, nous pouvons assigner la centrale à une ligne électrique. Si vous souhaitez raccorder la centrale à une ligne existante, il suffit d’ajouter le bloc suivant dans la partie:

« ASIGNATION DES LIGNES & GESTION ERREURS SURCHARGES COTE PRODUCTION »

Dans le cas de l’éolien nous pouvons choisir de désactiver sa production en pleine simulation avec cette condition que nous détaillerons plus tard.

```csharp

if (status\_x1 == true){

```

Il faut commencer par placer dans la variable la valeur de production en appelant la méthode Get\_prod() de l’objet x1 :

```csharp

double Xo1 = x1.Get\_prod();

```

Nous assignons la ligne l1 avec la méthode ligne\_in() et nous mettons en paramètres la production de la centrale :

```csharp

Xo1 = l1.Ligne\_in(Xo1);

```

Il est également nécessaire d’ajouter cette production à la production totale se trouvant dans la variable prod\_tot :

```csharp

prod\_tot += Eo1;

```

Cette partie gère l’affichage de message quand il y a une surcharge au niveau de la ligne véhiculant la production de x1 en précisant l’endroit et le moment ou la surcharge s’est passée :

```csharp

if (Xo1 == 0)

{

erreurs += DateTime.Now.ToString();

erreurs += errors.Line\_Overload(l1);

erreurs += "\n";

}

```

Par défaut, il existe déjà une centrale éolienne donc dans les ligne suivante nous devons incrémenter les valeurs de Co2 émis, du cout de production ainsi que la production de l’ensemble des centrales éoliennes:

```csharp

prix\_eolien += x1.Get\_prix();

co2\_eolien += x1.Get\_co2();

prod\_eolien += Xo1;

```

Dans cette condition nous affichons un message lorsque la centrale est désactivée durant la simulation :

```csharp

else

{

erreurs += DateTime.Now.ToString();

erreurs += errors.Deactivate(x1);

erreurs += "\n";

}

```

Si x1 est un type de centrale déjà existant, il n’y a plus de manipulations s à faire.

Cependant, s’il s’agit d’un nouveau type de centrale, il faudra créer quelques variables en plus avant de passer dans la classe view\_graphe.

Il faut créer une variables associées au cout de production, au CO2 ainsi qu’à la production au-dessus du constructeur public Update() :

```csharp

public double prix\_fusion;

public double co2\_fusion;

public double prod\_fusion;

```

Dans le cas où cette centrale doit pouvoir être arrêtée durant la simulation, il faut ajouter une variable booléenne dont nous changerons l’état dans la classe view\_graphe :

```csharp

public bool status\_x1 = true;

```

Il faut également ajouter la production de ce type de centrale dans la variable totale à la fin de la classe :

```csharp

total += prod\_fusion ;

```

À présent nous pouvons passer à la classe view\_graphe.

Dans la partie AFFICHAGE, nous devons créer un nouveau type de centrale et y afficher ses infos. Nous utilisons un richTextBox rtbMessage pour faire cela :

```csharp

rtbMessage.AppendText("Fusion: "

+ up.prix\_fusion.ToString() + " € " //prix\_x1

+ up.co2\_fusion.ToString() + " g " //co2\_x1

+ up.prod\_fusion.ToString() + " W " //prod-x1

+ "\n");

```

Pour gérer l’état d’une centrale, il suffit de créer un checkbox qui sera nommé comme la centrale. Il faut créer une méthode qui sera semblable à celle-ci :

```csharp

public void cb\_x1\_CheckedChanged(object sender, EventArgs e)

{

if (cb\_x1.Checked)

{

up.status\_x1 = true;

}

else

{

up.status\_x1 = false;

}

}

```

## Nouveau type de centrale

## Ajout consommateur

Si l’utilisateur souhaite ajouter un nouveau consommateur, il y a deux possibilités, soit il est d’un type déjà existant, soit il sera d’un nouveau type.

Nous allons procéder à l’ajout d’un consommateur de type aléatoire nommé lambda à titre d’exemple.

Dans la classe update, il faut créer la variable associée à la consommation :

```csharp

public double conso\_lambda;

```

Ensuite il faut déclarer le nom du consommateur avant de créer un objet de type consommateur :

```csharp

Consommateur lambda;

```

À présent nous pouvons créer une instance de lambda. En paramètre, il faut indiquer la consommation maximale théorique (1000W dans ce cas) ainsi que son nom qui sera affiché dans la fenêtre de simulation :

```csharp

lambda = new Consommateur\_random(1000, "lambda");

```

Dans la partie:

ASIGNATION DES LIGNES & GESTION ERREURS SURCHARGES COTE CONSOMMATION,

nous ajoutons ce bloc pour assigner une ligne à ce consommateur lambda :

Il faut commencer par placer dans la variable la valeur de consommation en appelant la méthode Get\_conso() de l’objet lambda :

```csharp

double lam = lambda.Get\_conso();

```

Nous assignons la ligne l4 avec la méthode ligne\_in() et nous mettons en paramètres la consommation de lambda :

```csharp

lam = l4.Ligne\_in(lam);

```

Il est également nécessaire d’ajouter cette production à la production totale se trouvant dans la variable prod\_tot :

```csharp

prod\_tot += Eo1;

```

Cette partie gère l’affichage de message quand il y a une surcharge au niveau de la ligne véhiculant la production de x1 en précisant l’endroit et le moment ou la surcharge s’est passée :

```csharp

if (lam == 0)

{

erreurs += DateTime.Now.ToString();

erreurs += errors.Line\_Overload(l4);

erreurs += "\n";

}

```

Pour terminer, il faut ajouter la consommation de ce lambda :

```csharp

conso\_tot += lam;

conso\_lambda = lam;

```

il nous reste plus qu’a passer à la classe view\_graphe pour afficher le résultat de la consommation de lambda dans la partie AFFICHAGE :

```csharp

rtbMessage.AppendText("lambda: "

+ up.conso\_lambda.ToString() + " W" //conso\_lambda

+ "\n");

```

## Nouveau type de consommateur

## Créer de nouveaux messages (info, erreurs)

Pour afficher les messages, nous avons une classe Errors. Chaque message est affiché en appelant une des méthodes s’y trouvant.

Ensuite nous récupérons ces erreurs et le ajoutons à notre variables erreurs pour les afficher via la classe view\_graphe.

Les méthodes que vous allez créer doivent renvoyer une chaine de caractères pour que cela puisse fonctionner.

## --------------------------------------------------------------------------------------------------

## Diagrame de classe

![diagrame](Documents/diagrame.png)

## Creation d’une nouvelle classe pour une centrale

Dans le cas de la création d’une nouvelle Eolienne

Premièrement il faut savoir que toutes les centrales sont reliées à la classe Centrale. Donc pour connaitre les différentes méthodes déjà utilisables il suffit de parcourir la classe Centrale, et dans le cas ou aucune méthode ne convient il faudra d’abord la crée dans la Classe centrale.

Toutes les centrales ont une déclaration de base qui est similaire.

```csharp

public class Nom\_centrale : Centrale //classe pour créer/gérer une source d'énergie

{

public double price;

public Nom\_centrale (double max\_prod, double co2, string name, Market market) : base(max\_prod, co2, name)

{

this.price = market.Get\_n\_price(); //récupération du prix KWh du marché

}

public override double Get\_prod() // production

{

return max\_prod;

}

public override double Get\_prix() // prix pour produire l'énergie nucléaire

{

return price \* this.Get\_prod();

}

public override double Get\_co2() // CO2

{

return base.Get\_co2();

}

public override string Get\_name()

{

return base.Get\_name();

}

}

```

Dans le cas d’une centrale nucléaire il n’y a pas besoin de plus, cependant si cette centrale est influencée par la météo, par exemple, il faudra rajouter des méthodes.

1. Le cas où la méthode existe déjà.

Si la méthode est déjà présente dans la classe centrale il suffit simplement de la rajouter dans la nouvelle classe et de rajouter les paramètres qui y sont lier.

Par exemple pour avoir une valeur de vent il faut rajouter la méthode Get\_vent() pour cela il faut rajouter plusieurs chose.

Une variable globale à la classe.

```csharp

Float wind = 0 ;

```

Ensuite dans la déclaration de la classe il faut rajouter l’appel de la classe Meteo car c’est a cette endroit la que l’on récupèrera notre variable de vent. Puis il faut ajouter le methode Get\_wind() de Meteo et l’assigner a notre variable globale crée plus tôt. Cela nous donne.

```csharp

public Nom\_centrale (double max\_prod, int co2, string name, Market market , Meteo meteo) : base(max\_prod, co2, name)

{

this.wind = meteo.Get\_wind(); //récupération de la vitesse du vent depuis la classe météo

this.price = market.Get\_e\_price(); //récupération du prix KWh du marché

}

```

Ensuite il nous suffit simplement de rajouter la methode Get\_vent().

```csharp

public override float Get\_vent(Meteo meteo)

{

wind = meteo.Get\_wind();

return wind;

}

```