Simulation d'un gaz parfait

2.0

Généré par Doxygen 1.8.7

Lundi 26 Mai 2014 23 :22 :57

Table des matières

1 Index hiérarchique **1**

1.1 Hiérarchie des classes 1

2 Index des classes **3**

2.1 Liste des classes 3

3 Index des fichiers **5**

3.1 Liste des fichiers 5

4 Documentation des classes **7**

4.1 Référence de la classe Dessinable 7

4.1.1 Description détaillée 7

4.1.2 Documentation des fonctions membres 7

4.1.2.1 dessine 7

4.2 Référence de la classe Enceinte 7

4.2.1 Description détaillée 8

4.2.2 Documentation des constructeurs et destructeur 8

4.2.2.1 Enceinte 8

4.2.2.2 Enceinte 8

4.2.3 Documentation des fonctions membres 9

4.2.3.1 dessine 9

4.2.3.2 display 9

4.2.3.3 getHauteur 9

4.2.3.4 getLargeur 9

4.2.3.5 getLongueur 9

4.3 Référence de la classe Fenetre 10

4.3.1 Description détaillée 10

4.3.2 Documentation des constructeurs et destructeur 10

4.3.2.1 Fenetre 10

4.3.2.2 ∼Fenetre 11

4.3.3 Documentation des fonctions membres 11

4.3.3.1 OnExit 11

4.3.4 Documentation des données membres . . . 11
4.3.4.1 fogl . . . 11
4.4 Référence de la classe GenerateurAleatoire . . . 11
4.4.1 Description détaillée . . . 11
4.4.2 Documentation des constructeurs et destructeur . . . 12
4.4.2.1 GenerateurAleatoire . . . 12
4.4.2.2 GenerateurAleatoire . . . 12
4.4.3 Documentation des fonctions membres . . . 12
4.4.3.1 gaussienne . . . 12
4.4.3.2 uniforme_entier . . . 12
4.4.3.3 uniforme_reel . . . 12
4.5 Référence de la classe GLArgon . . . 12
4.5.1 Description détaillée . . . 13
4.5.2 Documentation des constructeurs et destructeur . . . 13
4.5.2.1 GLArgon . . . 13
4.5.2.2 GLArgon . . . 13
4.5.2.3 ∼GLArgon . . . 13
4.5.3 Documentation des fonctions membres . . . 13
4.5.3.1 dessine . . . 13
4.5.3.2 evolue . . . 14
4.6 Référence de la classe GLFluor . . . 14
4.6.1 Description détaillée . . . 14
4.6.2 Documentation des constructeurs et destructeur . . . 15
4.6.2.1 GLFluor . . . 15
4.6.2.2 GLFluor . . . 15
4.6.2.3 ∼GLFluor . . . 15
4.6.3 Documentation des fonctions membres . . . 15
4.6.3.1 dessine . . . 15
4.6.3.2 enregistrerCoordonnee . . . 15
4.6.3.3 evolue . . . 16
4.7 Référence de la classe GLHelium . . . 16
4.7.1 Description détaillée . . . 16
4.7.2 Documentation des constructeurs et destructeur . . . 17
4.7.2.1 GLHelium . . . 17
4.7.2.2 GLHelium . . . 17
4.7.2.3 ∼GLHelium . . . 17
4.7.3 Documentation des fonctions membres . . . 17
4.7.3.1 dessine . . . 17
4.7.3.2 evolue . . . 17
4.8 Référence de la classe GLNeon . . . 18

4.8.1 Description détaillée . . . 18
4.8.2 Documentation des constructeurs et destructeur . . . 18
4.8.2.1 GLNeon . . . 18
4.8.2.2 GLNeon . . . 20
4.8.2.3 ∼GLNeon . . . 20
4.8.3 Documentation des fonctions membres . . . 20
4.8.3.1 dessine . . . 20
4.8.3.2 evolue . . . 20
4.9 Référence de la classe GUI . . . 21
4.9.1 Description détaillée . . . 21
4.9.2 Documentation des fonctions membres . . . 21
4.9.2.1 OnInit . . . 21
4.10 Référence de la classe Particule . . . 21
4.10.1 Description détaillée . . . 22
4.10.2 Documentation des constructeurs et destructeur . . . 22
4.10.2.1 Particule . . . 22
4.10.2.2 Particule . . . 22
4.10.2.3 Particule . . . 22
4.10.3 Documentation des fonctions membres . . . 22
4.10.3.1 afficher . . . 22
4.10.3.2 collision . . . 22
4.10.3.3 evolue . . . 23
4.10.3.4 gere_sorties . . . 23
4.10.3.5 get_rayon . . . 23
4.10.3.6 getPosition . . . 23
4.10.3.7 pavageCubique . . . 23
4.10.4 Documentation des données membres . . . 23
4.10.4.1 masse . . . 23
4.10.4.2 position . . . 23
4.10.4.3 rayon . . . 23
4.10.4.4 vitesse . . . 23
4.11 Référence de la classe Systeme . . . 23
4.11.1 Description détaillée . . . 24
4.11.2 Documentation des constructeurs et destructeur . . . 24
4.11.2.1 Systeme . . . 24
4.11.2.2 Systeme . . . 24
4.11.2.3 ∼Systeme . . . 24
4.11.3 Documentation des fonctions membres . . . 25
4.11.3.1 ajouterArgon . . . 25
4.11.3.2 ajouterFluor . . . 25

4.11.3.3 ajouterHelium . . . 25
4.11.3.4 ajouterNeon . . . 25
4.11.3.5 dessine . . . 25
4.11.3.6 evolue . . . 25
4.11.4 Documentation des données membres . . . 25
4.11.4.1 collectionParticules . . . 25
4.11.4.2 enceinte . . . 25
4.11.4.3 epsilon . . . 25
4.11.4.4 temperature . . . 26
4.11.4.5 tirage . . . 26
4.12 Référence de la classe TXTArgon . . . 26
4.12.1 Description détaillée . . . 26
4.12.2 Documentation des constructeurs et destructeur . . . 26
4.12.2.1 TXTArgon . . . 26
4.12.3 Documentation des fonctions membres . . . 26
4.12.3.1 dessine . . . 26
4.13 Référence de la classe TXTHelium . . . 27
4.13.1 Description détaillée . . . 27
4.13.2 Documentation des constructeurs et destructeur . . . 27
4.13.2.1 TXTHelium . . . 27
4.13.3 Documentation des fonctions membres . . . 27
4.13.3.1 dessine . . . 27
4.14 Référence de la classe TXTNeon . . . 28
4.14.1 Description détaillée . . . 28
4.14.2 Documentation des constructeurs et destructeur . . . 28
4.14.2.1 TXTNeon . . . 28
4.14.3 Documentation des fonctions membres . . . 28
4.14.3.1 dessine . . . 28
4.15 Référence de la classe Vecteur . . . 28
4.15.1 Description détaillée . . . 29
4.15.2 Documentation des constructeurs et destructeur . . . 29
4.15.2.1 Vecteur . . . 29
4.15.2.2 Vecteur . . . 29
4.15.3 Documentation des fonctions membres . . . 30
4.15.3.1 afficher . . . 30
4.15.3.2 getX . . . 30
4.15.3.3 getY . . . 30
4.15.3.4 getZ . . . 30
4.15.3.5 operator" != . . . 31
4.15.3.6 operator∗ . . . 31

4.15.3.7 operator*= . . . 31
4.15.3.8 operator+= . . . 31
4.15.3.9 operator- . . . 32
4.15.3.10 operator-= . . . 32
4.15.3.11 operator== . . . 32
4.15.3.12 operator^= . . . 32
4.16 Référence de la classe Vue_OpenGL . . . 34
4.16.1 Description détaillée . . . 34
4.16.2 Documentation des constructeurs et destructeur . . . 35
4.16.2.1 Vue_OpenGL . . . 35
4.16.2.2 ~Vue_OpenGL . . . 35
4.16.3 Documentation des fonctions membres . . . 35
4.16.3.1 deplace . . . 35
4.16.3.2 dessine . . . 35
4.16.3.3 InitOpenGL . . . 35
4.16.3.4 OnEnterWindow . . . 35
4.16.3.5 OnKeyDown . . . 35
4.16.3.6 OnSize . . . 36
4.16.3.7 OnTimer . . . 36
4.16.3.8 RotatePhi . . . 36
4.16.3.9 RotateTheta . . . 36

5 Documentation des fichiers 37

5.1 Référence du fichier Dessinable.h . . . 37
5.1.1 Description détaillée . . . 37
5.2 Référence du fichier Enceinte.cc . . . 37
5.2.1 Description détaillée . . . 37
5.3 Référence du fichier Enceinte.h . . . 38
5.3.1 Description détaillée . . . 38
5.4 Référence du fichier exerciceP12.cc . . . 38
5.4.1 Documentation des fonctions . . . 39
5.4.1.1 main . . . 39
5.5 Référence du fichier exerciceP9.cc . . . 39
5.5.1 Documentation des fonctions . . . 39
5.5.1.1 main . . . 39
5.6 Référence du fichier Fenetre.cc . . . 39
5.6.1 Description détaillée . . . 39
5.7 Référence du fichier Fenetre.h . . . 40
5.7.1 Description détaillée . . . 40
5.8 Référence du fichier GenerateurAleatoire.cc . . . 40

5.8.1 Description détaillée . . . 40
5.9 Référence du fichier GenerateurAleatoire.h . . . 41
5.9.1 Description détaillée . . . 41
5.10 Référence du fichier GLArgon.cc . . . 41
5.10.1 Description détaillée . . . 41
5.11 Référence du fichier GLArgon.h . . . 42
5.11.1 Description détaillée . . . 42
5.12 Référence du fichier GLFluor.cc . . . 42
5.12.1 Description détaillée . . . 42
5.13 Référence du fichier GLFluor.h . . . 43
5.13.1 Description détaillée . . . 43
5.14 Référence du fichier GLHelium.cc . . . 43
5.14.1 Description détaillée . . . 44
5.15 Référence du fichier GLHelium.h . . . 44
5.15.1 Description détaillée . . . 44
5.16 Référence du fichier GLNeon.cc . . . 44
5.16.1 Description détaillée . . . 45
5.17 Référence du fichier GLNeon.h . . . 45
5.17.1 Description détaillée . . . 45
5.18 Référence du fichier GUI.cc . . . 45
5.18.1 Description détaillée . . . 46
5.19 Référence du fichier GUI.h . . . 46
5.19.1 Description détaillée . . . 46
5.20 Référence du fichier Particule.cc . . . 46
5.20.1 Description détaillée . . . 47
5.20.2 Documentation des fonctions . . . 47
5.20.2.1 operator<< . . . 47
5.21 Référence du fichier Particule.h . . . 47
5.21.1 Description détaillée . . . 47
5.21.2 Documentation des fonctions . . . 48
5.21.2.1 operator<< . . . 48
5.22 Référence du fichier Systeme.cc . . . 48
5.22.1 Description détaillée . . . 48
5.23 Référence du fichier Systeme.h . . . 48
5.23.1 Description détaillée . . . 48
5.24 Référence du fichier testParticule.cc . . . 49
5.24.1 Documentation des fonctions . . . 49
5.24.1.1 main . . . 49
5.25 Référence du fichier testSysteme.cc . . . 49
5.25.1 Documentation des fonctions . . . 49

5.25.1.1 main . . . 49
5.26 Référence du fichier testTXTArgon.cc . . . 49
5.26.1 Documentation des fonctions . . . 50
5.26.1.1 main . . . 50
5.27 Référence du fichier testTXTNeon.cc . . . 50
5.27.1 Documentation des fonctions . . . 50
5.27.1.1 main . . . 50
5.28 Référence du fichier testVecteur.cc . . . 50
5.28.1 Documentation des fonctions . . . 50
5.28.1.1 main . . . 50
5.29 Référence du fichier TXTArgon.cc . . . 50
5.29.1 Description détaillée . . . 51
5.30 Référence du fichier TXTArgon.h . . . 51
5.30.1 Description détaillée . . . 51
5.31 Référence du fichier TXTHelium.cc . . . 51
5.31.1 Description détaillée . . . 52
5.32 Référence du fichier TXTHelium.h . . . 52
5.32.1 Description détaillée . . . 52
5.33 Référence du fichier TXTNeon.cc . . . 52
5.33.1 Description détaillée . . . 53
5.34 Référence du fichier TXTNeon.h . . . 53
5.35 Référence du fichier Vecteur.cc . . . 53
5.35.1 Description détaillée . . . 53
5.35.2 Documentation des fonctions . . . 54
5.35.2.1 operator∗ . . . 54
5.35.2.2 operator∗ . . . 54
5.35.2.3 operator+ . . . 54
5.35.2.4 operator- . . . 55
5.35.2.5 operator<< . . . 55
5.35.2.6 operator^ . . . 55
5.36 Référence du fichier Vecteur.h . . . 55
5.36.1 Description détaillée . . . 56
5.36.2 Documentation des fonctions . . . 56
5.36.2.1 operator∗ . . . 56
5.36.2.2 operator∗ . . . 56
5.36.2.3 operator+ . . . 57
5.36.2.4 operator- . . . 57
5.36.2.5 operator<< . . . 57
5.36.2.6 operator^ . . . 57
5.37 Référence du fichier Vue_OpenGL.cc . . . 58

5.38 Référence du fichier Vue_OpenGL.h 58

Chapitre 1

Index hiérarchique

1.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

Dessinable 8
  Canon 7
  Enceinte 9
  Particule 22
    GLArgon 13
    GLFluor 15
    GLHelium 17
    GLNeon 19
    TXTArgon 26
    TXTHelium 27
    TXTNeon 28
  Systeme 24
GenerateurAleatoire 13
TXTArgon : :h . . . . . **??**
TXTNeon : :h . . . . . **??**
TXTHelium : :h . . . . . **??**
Vecteur 29
wxApp
  GUI 21
wxFrame
  Fenetre 11
wxGLCanvas
  Vue_OpenGL 33

Chapitre 2

Index des classes

2.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

Canon
Prototype de la classe **Canon** (p. 7) . . . 7
Dessinable . . . 8
Enceinte . . . 9
Fenetre . . . 11
GenerateurAleatoire
Représente des objets capables de générer des nombres aléatoires . . . 13
GLArgon
Prototype de la classe **GLArgon** (p. 13) . . . 13
GLFluor
Prototype de la classe GLluor . . . 15
GLHelium
Prototype de la classe **GLHelium** (p. 17) . . . 17
GLNeon
Prototype de la classe **GLNeon** (p. 19) . . . 19
GUI
Application principale . . . 21
TXTArgon : :h
Représente des atomes d'Argon (version texte) . . . **??**
TXTNeon : :h
Représente des atomes de Neon (version texte) . . . **??**
TXTHelium : :h
Représente des atomes d'Helium (version texte) . . . **??**
Particule
Classe mère dont hérite toutes les classes de type : TXTNom et GLNom Représentation d'une particule (ici version graphique) . . . 22
Systeme
Permet de créer, de dessiner et de faire évoluer des objets de type **Systeme** (p. 24) formés d'une enceinte et de particules . . . 24
TXTArgon . . . 26
TXTHelium . . . 27
TXTNeon . . . 28
Vecteur
Prototype de la classe **Vecteur** (p. 29) . . . 29
Vue_OpenGL
Prototype de la classe **Vue_OpenGL** (p. 33) . . . 33

Chapitre 3

Index des fichiers

3.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Canon.cc**
Fichier permettant la définition de la classe **Canon** (p. 7) 37
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Canon.h**
Est la classe qui contient l'objet enceinte qui est la boîte où seront nos particules 38
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Dessinable.h**
Est la super-classe avec une méthode dessine qui permet d'avoir une spécialisation pour chaque type d'objet 39
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Enceinte.cc**
Fichier permettant la définition de la classe enceinte 40
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Enceinte.h**
Est la classe qui contient l'objet enceinte qui est la boîte où seront nos particules 42
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Fenetre.cc**
Est la définition de la classe fenêtre en OpenGL 44
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Fenetre.h**
Est le prototype de la classe fenetre qui permettra de créer une fentre contenant notre application 45
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Generateur↩ Aleatoire.cc**
Définition de la classe qui permet de générer des nombres aléatoires 46
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Generateur↩ Aleatoire.h**
Prototype de la classe qui permet de générer des nombres aléatoires 47
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLArgon.cc**
Est la définition de la classe de la particule argon en OpenGL 48
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLArgon.h**
Est le prototype de la classe de la particule argon en OpenGL 49
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLFluor.cc**
Est la définition de la classe de la particule fluor en OpenGL qui a en plus une mémorisation et affichage de la trajectoire 50
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLFluor.h**
Est le prototype de la classe de la particule fluor en OpenGL qui a en plus une mémorisation et affichage de la trajectoire 51
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLHelium.cc**
Est la définition de la classe de la particule Helium en OpenGL 53
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLHelium.h**
Est le prototype de la classe de la particule Helium en OpenGL 54
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLNeon.cc**
Est la définition de la classe de la particule Néon en OpenGL 55

/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLNeon.h**
Est le prototype de la classe de la particule néon en OpenGL 56
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GUI.cc**
Est la définition de l'application princpal qui lance tout le programme 57
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GUI.h**
Est le prototype de l'application princpal qui lance tout le programme 58
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Particule.cc**
Définition de la classe mère **Particule** (p. 22) 58
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Particule.h**
Prototype de la classe mère **Particule** (p. 22) 61
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Systeme.cc**
Définition de la classe **Systeme** (p. 24) (objet formé d'une enceinte et de particules) 62
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Systeme.h**
Prototype de la classe de la classe **Systeme** (p. 24) (objet formé d'une enceinte et de particules) 66
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTArgon.cc**
Définition de la classe de la particule Argon en version texte 71
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTArgon.h**
Prototype de la classe de la particule Argon en version texte 71
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTHelium.cc**
Définition de la classe de la particule Helium en version texte 72
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTHelium.h**
Prototype de la classe de la particule Helium en version texte 73
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTNeon.cc**
Définition de la classe de la particule Neon en version texte 73
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTNeon.h** 74
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Vecteur.cc**
Est la définition de la classe qui nous pourmet de gérer la position et la vitesse de nos particules mais aussi de tous l'espace qui les entours 75
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Vecteur.h**
Est le prototype de la classe qui nous pourmet de gérer la position et la vitesse de nos particules mais aussi de tous l'espace qui les entours 79
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Vue_OpenGL.cc** . . 82
/Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Vue_OpenGL.h** . . . 84

Chapitre 4

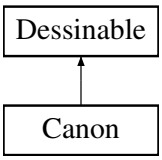
Documentation des classes

4.1 Référence de la classe Canon

Prototype de la classe **Canon** (p. 7).

`#include <Canon.h>`

Graphe d'héritage de Canon :



Fonctions membres publiques

- **Canon** ()
  Constructeur sans paramètre.
- ∼**Canon** ()
  *Destructeur de **Canon** (p. 7) qui détruit chacune des parties du canons.*
- virtual void **dessine** () const
  Prototype de la méthode dessine en OpenGL.

4.1.1 Description détaillée

Prototype de la classe **Canon** (p. 7).

Cette classe est celle qui permet de créer et de dessiner un canon à particule. Nous avons dessidé de le faire car nous voulions ajouté des particules au systeme.

Elle hérite de dessinable car il fallait avoir une méthode dessine et cela respecte le bonne programmation car on utilise l'héritage.

Définition à la ligne **29** du fichier **Canon.h**.

4.1.2 Documentation des constructeurs et destructeur

4.1.2.1 Canon : :Canon ( )

Constructeur sans paramètre.

Constructeur de la classe du **Canon** (p. 7) qui initialise la longueur et le rayon et les quadrics qui permette d'avoir chacune des composants de notre canon cylindre : structure principale Boule : permet d'avoir un arrondi sur le bout du canon roue1 et roue2 sont les roues du canon enjoliveurs sont les caches de roues

Définition à la ligne **32** du fichier **Canon.cc**.

4.1.2.2 Canon : :∼Canon ( ) [inline]

Destructeur de **Canon** (p. 7) qui détruit chacune des parties du canons.

Définition à la ligne **53** du fichier **Canon.h**.

4.1.3 Documentation des fonctions membres

4.1.3.1 void Canon : :dessine ( ) const [virtual]

Prototype de la méthode dessine en OpenGL.

elle va prendre prendre une couleur noire, et dessiner le canon qui a plusieurs composants dans les différents endroits grae aux matrices de translations, rotations dessin du cylindre du canon

dessin du fond du canon

dessin de la roue 1

dessin de la roue deux

Implémente **Dessinable** (p. 9).

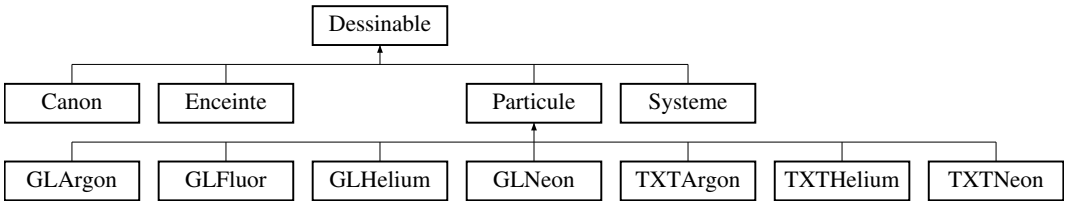
Définition à la ligne **46** du fichier **Canon.cc**.

La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Canon.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Canon.cc**

4.2 Référence de la classe Dessinable

`#include <Dessinable.h>`

Graphe d'héritage de Dessinable :



Fonctions membres publiques

- virtual void **dessine** () const =0
  une méthode virtuelle pure car on veut qu'elle soit redéfinie dans toutes les sous-classes

4.2.1 Description détaillée

Définition à la ligne **12** du fichier **Dessinable.h**.

4.2.2 Documentation des fonctions membres

4.2.2.1 virtual void Dessinable : :dessine ( ) const [pure virtual]

une méthode virtuelle pure car on veut qu'elle soit redéfinie dans toutes les sous-classes

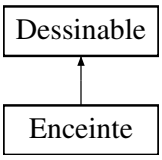
Implémenté dans **Systeme** (p. 25), **Canon** (p. 8), **Enceinte** (p. 10), **GLFluor** (p. 17), **GLArgon** (p. 15), **GLNeon** (p. 20), **GLHelium** (p. 19), **TXTArgon** (p. 27), **TXTHelium** (p. 28), et **TXTNeon** (p. 29).

La documentation de cette classe a été générée à partir du fichier suivant :
— /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Dessinable.h**

4.3 Référence de la classe Enceinte

`#include <Enceinte.h>`

Graphe d'héritage de Enceinte :



Fonctions membres publiques

— **Enceinte** ()
Constructeur sans paramètre.
— **Enceinte** (bool reglage)
Constructeur avec paramètre bool pour l'interface utilisateur.
— **Enceinte** (double largeur, double longueur, double hauteur)
Constructeur avec paramètres largeur, longueur et hauteur de l'enceinte.
— double **getLargeur** () const
Prototype du getters de la largeur de l'enceinte.
— double **getLongueur** () const
Prototype du getters de la longueur de l'enceinte.
— double **getHauteur** () const
Prototype du getters de la hauteur de l'enceinte.
— void **display** () const
Prototype de la méthode qui écrit dans le terminal la largeur hauteur et longueur de l'enceinte.
— virtual void **dessine** () const
Prototype de la méthode dessine en OpenGL.

4.3.1 Description détaillée

Cette classe est celle qui permet de créer une enceinte, de la dessiner grae au fait qu'elle hérite de dessinable donc nous avons pu utiliser le polymophisme pour appeler la méthode dessine.

Définition à la ligne **19** du fichier **Enceinte.h**.

4.3.2 Documentation des constructeurs et destructeur

4.3.2.1 Enceinte : :Enceinte ( )

Constructeur sans paramètre.

Constructeur de la classe de l'enceinte mais qui initialise à largeur = 100, longueur =100, hauteur = 100

Définition à la ligne **27** du fichier **Enceinte.cc**.

4.3.2.2 Enceinte : :Enceinte (bool *reglage*)

Constructeur avec paramètre bool pour l'interface utilisateur.

Constructeur de la classe de l'enceinte mais qui initialise aux valeurs prises comme paramètre

Paramètres

bool	reglage : permet en fonction de l'utilisateur souhaite d'avoir un appel différents du constructeur

Définition à la ligne **38** du fichier **Enceinte.cc**.

4.3.2.3 Enceinte : :Enceinte (double *largeur,* double *longueur,* double *hauteur*)

Constructeur avec paramètres largeur, longueur et hauteur de l'enceinte.

Constructeur de la classe de l'enceinte mais qui initialise aux valeurs prises comme paramètre

Paramètres

double	largeur
double	longueur
double	hauteur

Définition à la ligne **64** du fichier **Enceinte.cc**.

4.3.3 Documentation des fonctions membres

4.3.3.1 void Enceinte : :dessine ( ) const [virtual]

Prototype de la méthode dessine en OpenGL.

définition de la méthode dessine de l'enceinte en OpenGL

elle va prendre prendre une couleur noire, et dessiner des traits qui vont faire le tours de l'enceinte

Implémente **Dessinable** (p. 9).

Définition à la ligne **113** du fichier **Enceinte.cc**.

4.3.3.2 void Enceinte : :display ( ) const

Prototype de la méthode qui écrit dans le terminal la largeur hauteur et longueur de l'enceinte.

méthode pour afficher dans le terminal la dimension de l'enceinte

Définition à la ligne **101** du fichier **Enceinte.cc**.

4.3.3.3 double Enceinte : :getHauteur ( ) const

Prototype du getters de la hauteur de l'enceinte.

Methode qui permet de retourner les attributs privés de la hauteur

Renvoie

la hauteur

Définition à la ligne **94** du fichier **Enceinte.cc**.

4.3.3.4 double Enceinte : :getLargeur ( ) const

Prototype du getters de la largeur de l'enceinte.

Methode qui permet de retourner les attributs privés de la largeur

Renvoie

la largeur

Définition à la ligne **76** du fichier **Enceinte.cc**.

4.3.3.5 double Enceinte : :getLongueur ( ) const

Prototype du getters de la longueur de l'enceinte.

Methode qui permet de retourner les attributs privés de la longueur

Renvoie

la longueur

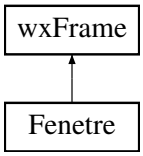
Définition à la ligne **85** du fichier **Enceinte.cc**.

La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Enceinte.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Enceinte.cc**

4.4 Référence de la classe Fenetre

`#include <Fenetre.h>`

Graphe d'héritage de Fenetre :



Fonctions membres publiques

- **Fenetre** (wxString const &titre, wxSize const &taille=wxDefaultSize, wxPoint const &position=wxDefault↩ Position, long style=wxDEFAULT_FRAME_STYLE)
  Constructeur avec arguments.
- **Fenetre** (wxString const &titre, bool reglage, wxSize const &taille=wxDefaultSize, wxPoint const &position=wxDefaultPosition, long style=wxDEFAULT_FRAME_STYLE)
  Constructeur avec arguments et reglage pour l'interface utilisateur.
- virtual ∼**Fenetre** ()
  destructeur de fenetre

Fonctions membres protégées

- void **OnExit** (wxCommandEvent &event)
  méthode pour pouvoir refermer la fenetre

Attributs protégés

- **Vue_OpenGL** ∗ **fogl**

4.4.1 Description détaillée

Prototype de la classe **Fenetre** (p. 11) pour l'OpenGL et permet de créer une fenetre avec les caractéristiques que l'on souhaite.

Définition à la ligne **23** du fichier **Fenetre.h**.

4.4.2 Documentation des constructeurs et destructeur

4.4.2.1 Fenetre : :Fenetre (wxString const & *titre,* wxSize const & *taille =* `wxDefaultSize`, wxPoint const & *position =* `wxDefaultPosition`, long *style =* `wxDEFAULT_FRAME_STYLE`)

Constructeur avec arguments.

Constructeur de la classe **Vecteur** (p. 29) avec trois arguements de type double

```
\param titre : le nom que portera la fenetre
\param taille : donne la taille de la fenetre
\param position : place la fenetre à l'endroit voulu
\param style : style de la fenêtre
```

Barre de menu

Barre de menu

Barre de menu

Barre de menu

Affiche (montre) la fenetre

Initialise OpenGL

Définition à la ligne **28** du fichier **Fenetre.cc**.

4.4.2.2 Fenetre : :Fenetre (wxString const & *titre,* bool *reglage,* wxSize const & *taille =* `wxDefaultSize`, wxPoint const & *position =* `wxDefaultPosition`, long *style =* `wxDEFAULT_FRAME_STYLE`)

Constructeur avec arguments et reglage pour l'interface utilisateur.

Constructeur de la classe, permet de gérer l'interface utilisateur Nous avons décider de le faire comem cela car ça nous permettait de le faire simplement en respectant une meilleure conception

Paramètres

titre	: le nom que portera la fenetre
reglage	: permet de gérer les souahaits de l'utilisateur
taille	: donne la taille de la fenetre
position	: place la fenetre à l'endroit voulu
style	: style de la fenêtre

Barre de menu

Barre de menu

Barre de menu

Barre de menu

Affiche (montre) la fenetre

Initialise OpenGL

Définition à la ligne **65** du fichier **Fenetre.cc**.

4.4.2.3 virtual Fenetre : :∼Fenetre ( ) [inline], [virtual]

destructeur de fenetre

Définition à la ligne **42** du fichier **Fenetre.h**.

4.4.3 Documentation des fonctions membres

4.4.3.1 void Fenetre : :OnExit (wxCommandEvent & *event*) [inline], [protected]

méthode pour pouvoir refermer la fenetre

Définition à la ligne **46** du fichier **Fenetre.h**.

4.4.4 Documentation des données membres

4.4.4.1 Vue_OpenGL∗ Fenetre : :fogl [protected]

Définition à la ligne **49** du fichier **Fenetre.h**.

La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Fenetre.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Fenetre.cc**

4.5 Référence de la classe GenerateurAleatoire

Représente des objets capables de générer des nombres aléatoires.

`#include <GenerateurAleatoire.h>`

Fonctions membres publiques

- **GenerateurAleatoire** ()
  Constructeur par défaut.
- **GenerateurAleatoire** (unsigned int graine)
  Constructeur.
- int **uniforme_entier** (int min, int max)
  Méthode permettant de tirer uniformément un entier sur l'intervalle [min, max].
- double **uniforme_reel** (double min, double max)
  Méthode permettant de tirer uniformément un réel sur l'intervalle [min, max[.
- double **gaussienne** (double moyenne, double ecart_type)
  Méthode permettant de tirer un réel suivant la loi normale des probablités (dîtes loi de Gauss)

4.5.1 Description détaillée

Représente des objets capables de générer des nombres aléatoires.

Plusieurs méthodes peuvent être utilisées pour réaliser différents types de tirage

Définition à la ligne **20** du fichier **GenerateurAleatoire.h**.

4.5.2 Documentation des constructeurs et destructeur

4.5.2.1 GenerateurAleatoire : :GenerateurAleatoire ( )

Constructeur par défaut.

Constructeur sans paramètre. Permet d'initialiser l'attribut générateur avec un nombre aléatoire

Définition à la ligne **16** du fichier **GenerateurAleatoire.cc**.

4.5.2.2 GenerateurAleatoire : :GenerateurAleatoire (unsigned int *graine*)

Constructeur.

Constructeur prenant un paramètre de type int. Permet d'initialiser l'attribut générateur avec un nombre donné (utile pour debugger)

Définition à la ligne **23** du fichier **GenerateurAleatoire.cc**.

4.5.3 Documentation des fonctions membres

4.5.3.1 double GenerateurAleatoire : :gaussienne (double *moyenne,* double *ecart_type*)

Méthode permettant de tirer un réel suivant la loi normale des probablités (dîtes loi de Gauss)

Paramètres

moyenne	valeur autour de laquelle s'effectue le tirage
ecart_type	valeur représentant l'écart moyen à la moyenne des valeurs du tirage

Renvoie

Retourne un réel tiré selon la loi normale N(moyenne, ecart_type)

Définition à la ligne **50** du fichier **GenerateurAleatoire.cc**.

4.5.3.2 int GenerateurAleatoire : :uniforme_entier (int *min,* int *max*)

Méthode permettant de tirer uniformément un entier sur l'intervalle [min, max].

Paramètres

min	minimum de l'intervalle sur lequel le tirage s'effectue
max	maximum de l'intervalle sur lequel le tirage s'effectue

Renvoie

Retourne un entier tiré uniformément sur [min, max]

Définition à la ligne **32** du fichier **GenerateurAleatoire.cc**.

4.5.3.3 double GenerateurAleatoire : :uniforme_reel (double *min,* double *max*)

Méthode permettant de tirer uniformément un réel sur l'intervalle [min, max[.

Paramètres

min	minimum de l'intervalle sur lequel le tirage s'effectue
max	maximum de l'intervalle sur lequel le tirage s'effectue

Renvoie

Retourne un réel tiré uniformément sur [min, max[

Définition à la ligne **41** du fichier **GenerateurAleatoire.cc**.

La documentation de cette classe a été générée à partir des fichiers suivants :

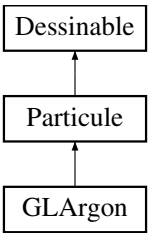
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GenerateurAleatoire.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GenerateurAleatoire.cc**

4.6 Référence de la classe GLArgon

Prototype de la classe **GLArgon** (p. 13).

`#include <GLArgon.h>`

Graphe d'héritage de GLArgon :



Fonctions membres publiques

- **GLArgon** (**Vecteur position**, **Vecteur vitesse**)
  Constructeur avec arguments sans la température du système.
- **GLArgon** (**Enceinte** const &enceinte, **GenerateurAleatoire** &tirage, double temperature)
  Constructeur avec arguments avec la température du système et le tirage aléatoire des vecteurs positions.
- virtual ∼**GLArgon** ()
  Dstructeurs.
- virtual void **evolue** (double dt)
  Prototype de la méthode evolue.
- virtual void **dessine** () const
  Prototype de la méthode evolue.

Membres hérités additionnels

4.6.1 Description détaillée

Prototype de la classe **GLArgon** (p. 13).

Cette classe est celle qui permet de créer une particule d'argon en OpenGL

Définition à la ligne **25** du fichier **GLArgon.h**.

4.6.2 Documentation des constructeurs et destructeur

4.6.2.1 GLArgon : :GLArgon (Vecteur *position,* Vecteur *vitesse*)

Constructeur avec arguments sans la température du système.

sans le paramètre de température, initialise un rayon et une masse à notre particule

Paramètres

position	: est la position de la particule que l'on souhaite
vitesse	: est la vitesse de la particule que l'on souhaite

Définition à la ligne **20** du fichier **GLArgon.cc**.

4.6.2.2 GLArgon : :GLArgon (Enceinte const & *enceinte,* GenerateurAleatoire & *tirage,* double *temperature*)

Constructeur avec arguments avec la température du système et le tirage aléatoire des vecteurs positions.

sans le paramètre de température, initialise un rayon et une masse à notre particule

Paramètres

enceinte	: permet de connaitre les dimmensions de l'emceinte pour calculer les positions
tirage	: donne une position aléatoire de la particule dans l'enceinte
gtempérature	: la température que l'on souhaite pour calculer la vitesse

Définition à la ligne **31** du fichier **GLArgon.cc**.

4.6.2.3 GLArgon : :∼GLArgon ( ) [virtual]

Dstructeurs.

Définition à la ligne **35** du fichier **GLArgon.cc**.

4.6.3 Documentation des fonctions membres

4.6.3.1 void GLArgon : :dessine ( ) const [virtual]

Prototype de la méthode evolue.

dessine la particule de couleur à la position souhaitée par une matrice de translation couleur vert, 100%

Implémente **Dessinable** (p. 9).

Définition à la ligne **56** du fichier **GLArgon.cc**.

4.6.3.2 void GLArgon : :evolue (double *dt*) [virtual]

Prototype de la méthode evolue.

appelle la méthode évolue de la classe particule

Paramètres

dt	: le pas de temps

Réimplémentée à partir de **Particule** (p. 23).

Définition à la ligne **47** du fichier **GLArgon.cc**.

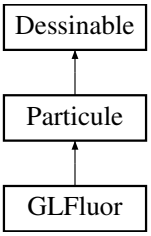
La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLArgon.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLArgon.cc**

4.7 Référence de la classe GLFluor

Prototype de la classe GLluor.

`#include <GLFluor.h>`

Graphe d'héritage de GLFluor :



Fonctions membres publiques

- **GLFluor** (**Vecteur position**, **Vecteur vitesse**)
  Constructeur avec arguments sans la température du système.
- **GLFluor** (**Enceinte** const &enceinte, **GenerateurAleatoire** &tirage, double temperature)
  Constructeur avec arguments avec la température du système et le tirage aléatoire des vecteurs positions.
- virtual ∼**GLFluor** ()
  Dstructeurs.
- virtual void **evolue** (double dt)
  Prototype de la méthode evolue.
- virtual void **dessine** () const
  Prototype de la méthode evolue.
- void **enregistrerCoordonnee** ()
  Prototye de la méthode qui va enregistrer les coordonnées.

Membres hérités additionnels

4.7.1 Description détaillée

Prototype de la classe GLluor.

Cette classe est celle qui permet de créer une particule de fluor en OpenGL et permet aussi de suivre à la trace cette particule grâce à un deque qui enregistre chaque position pendant un certain intervalle. Nous avos pas besoins de toutes les positions d'ou le fait que nous supprimons après 150 pas de temps les anciennes positions. Nous avos choisis un deque car il fallait une faible complexité pour détruire la première ligne.

Définition à la ligne **32** du fichier **GLFluor.h**.

4.7.2 Documentation des constructeurs et destructeur

4.7.2.1 GLFluor : :GLFluor (Vecteur *position,* Vecteur *vitesse*)

Constructeur avec arguments sans la température du système.

Construteur.

sans le paramètre de température, initialise un rayon et une masse à notre particule

Paramètres

position	: est la position de la particule que l'on souhaite
vitesse	: est la vitesse de la particule que l'on souhaite

Définition à la ligne **22** du fichier **GLFluor.cc**.

4.7.2.2 GLFluor : :GLFluor (Enceinte const & *enceinte,* GenerateurAleatoire & *tirage,* double *temperature*)

Constructeur avec arguments avec la température du système et le tirage aléatoire des vecteurs positions.

Construteur.

sans le paramètre de température, initialise un rayon et une masse à notre particule

Paramètres

enceinte	: permet de connaitre les dimmensions de l'emceinte pour calculer les positions
tirage	: donne une position aléatoire de la particule dans l'enceinte
gtempérature	: la température que l'on souhaite pour calculer la vitesse

Définition à la ligne **34** du fichier **GLFluor.cc**.

4.7.2.3 GLFluor : :∼GLFluor ( ) [virtual]

Dstructeurs.

Définition à la ligne **38** du fichier **GLFluor.cc**.

4.7.3 Documentation des fonctions membres

4.7.3.1 void GLFluor : :dessine ( ) const [virtual]

Prototype de la méthode evolue.

dessine les points contenu dans le tableau(deque)

dessine la particule de couleur à la position souhaitée par une matrice de translation permet de parcourir le tableau et de le dssiner

dessine les particlues

couleur mauve

J'ai remis 10 partout

Implémente **Dessinable** (p. 9).

Définition à la ligne **80** du fichier **GLFluor.cc**.

4.7.3.2 void GLFluor : :enregistrerCoordonnee ( )

Prototye de la méthode qui va enregistrer les coordonnées.

méthode qui enregistre la position/coordonnée de la particule et ensuite appelle la méthode évolue de la classe particule supprime la première valeur du tableau après 150 pas de temps (complexité de 1)

ajoute la dernière positions au deque

Définition à la ligne **63** du fichier **GLFluor.cc**.

4.7.3.3 void GLFluor : :evolue (double *dt*) [virtual]

Prototype de la méthode evolue.

spéciale pour le fluor car on enresitre ça position dans un deque (une sorte de tableau) pour pouvoir afficher la trajectoire Nous avons fait appel à evolue car c'est la seul méthode qui nous permettait d'enregistrer a tout instant

Paramètres

dt	: le pas de temps

Réimplémentée à partir de **Particule** (p. 23).

Définition à la ligne **54** du fichier **GLFluor.cc**.

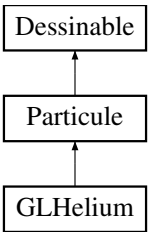
La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLFluor.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLFluor.cc**

4.8 Référence de la classe GLHelium

Prototype de la classe **GLHelium** (p. 17).

`#include <GLHelium.h>`

Graphe d'héritage de GLHelium :



Fonctions membres publiques

- **GLHelium** (**Vecteur position**, **Vecteur vitesse**)
  Constructeur avec arguments sans la température du système.
- **GLHelium** (**Enceinte** const &enceinte, **GenerateurAleatoire** &tirage, double temperature)
  Constructeur avec arguments avec la température du système et le tirage aléatoire des vecteurs positions.
- virtual ~**GLHelium** ()
  Dstructeurs.
- virtual void **evolue** (double dt)
  Prototype de la méthode evolue.
- virtual void **dessine** () const
  Prototype de la méthode evolue.

Membres hérités additionnels

4.8.1 Description détaillée

Prototype de la classe **GLHelium** (p. 17).

Cette classe est celle qui permet de créer une particule d'helium en OpenG

Définition à la ligne **23** du fichier **GLHelium.h**.

4.8.2 Documentation des constructeurs et destructeur

4.8.2.1 GLHelium : :GLHelium (Vecteur *position,* Vecteur *vitesse*)

Constructeur avec arguments sans la température du système.

sans le paramètre de température, initialise un rayon et une masse à notre particule

Paramètres

position	: est la position de la particule que l'on souhaite
vitesse	: est la vitesse de la particule que l'on souhaite

Définition à la ligne **20** du fichier **GLHelium.cc**.

4.8.2.2 GLHelium : :GLHelium (Enceinte const & *enceinte,* GenerateurAleatoire & *tirage,* double *temperature*)

Constructeur avec arguments avec la température du système et le tirage aléatoire des vecteurs positions.

sans le paramètre de température, initialise un rayon et une masse à notre particule

Paramètres

enceinte	: permet de connaitre les dimmensions de l'emceinte pour calculer les positions
tirage	: donne une position aléatoire de la particule dans l'enceinte
gtempérature	: la température que l'on souhaite pour calculer la vitesse

Définition à la ligne **30** du fichier **GLHelium.cc**.

4.8.2.3 GLHelium : :∼GLHelium ( ) [virtual]

Dstructeurs.

Définition à la ligne **34** du fichier **GLHelium.cc**.

4.8.3 Documentation des fonctions membres

4.8.3.1 void GLHelium : :dessine ( ) const [virtual]

Prototype de la méthode evolue.

dessine la particule de couleur à la position souhaitée par une matrice de translation orange

J'ai remis 10 partout

Implémente **Dessinable** (p. 9).

Définition à la ligne **55** du fichier **GLHelium.cc**.

4.8.3.2 void GLHelium : :evolue (double *dt*) [virtual]

Prototype de la méthode evolue.

appelle la méthode évolue de la classe particule

Paramètres

dt	: le pas de temps

Réimplémentée à partir de **Particule** (p. 23).

Définition à la ligne **46** du fichier **GLHelium.cc**.

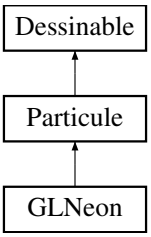
La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLHelium.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLHelium.cc**

4.9 Référence de la classe GLNeon

Prototype de la classe **GLNeon** (p. 19).

`#include <GLNeon.h>`

Graphe d'héritage de GLNeon :



Fonctions membres publiques

- **GLNeon** (**Vecteur position**, **Vecteur vitesse**)
  Constructeur avec arguments sans la température du système.
- **GLNeon** (**Enceinte** const &enceinte, **GenerateurAleatoire** &tirage, double temperature)
  Constructeur avec arguments avec la température du système et le tirage aléatoire des vecteurs positions.

— virtual ∼**GLNeon** ()
Dstructeurs.
— virtual void **evolue** (double dt)
Prototype de la méthode evolue.
— virtual void **dessine** () const
Prototype de la méthode evolue.

Membres hérités additionnels

4.9.1 Description détaillée

Prototype de la classe **GLNeon** (p. 19).

Cette classe est celle qui permet de créer une particule de néon en OpenG

Définition à la ligne **24** du fichier **GLNeon.h**.

4.9.2 Documentation des constructeurs et destructeur

4.9.2.1 GLNeon : :GLNeon (Vecteur *position,* Vecteur *vitesse*)

Constructeur avec arguments sans la température du système.

sans le paramètre de température, initialise un rayon et une masse à notre particule

Paramètres

position	: est la position de la particule que l'on souhaite
vitesse	: est la vitesse de la particule que l'on souhaite

Définition à la ligne **20** du fichier **GLNeon.cc**.

4.9.2.2 GLNeon : :GLNeon (Enceinte const & *enceinte,* GenerateurAleatoire & *tirage,* double *temperature*)

Constructeur avec arguments avec la température du système et le tirage aléatoire des vecteurs positions.

sans le paramètre de température, initialise un rayon et une masse à notre particule

Paramètres

enceinte	: permet de connaitre les dimmensions de l'emceinte pour calculer les positions
tirage	: donne une position aléatoire de la particule dans l'enceinte
gtempérature	: la température que l'on souhaite pour calculer la vitesse

Définition à la ligne **30** du fichier **GLNeon.cc**.

4.9.2.3 GLNeon : :∼GLNeon ( ) [virtual]

Dstructeurs.

Définition à la ligne **34** du fichier **GLNeon.cc**.

4.9.3 Documentation des fonctions membres

4.9.3.1 void GLNeon : :dessine ( ) const [virtual]

Prototype de la méthode evolue.

dessine la particule de couleur à la position souhaitée par une matrice de translation couleur rouge, 100%

J'ai remis 10 partout

Implémente **Dessinable** (p. 9).

Définition à la ligne **55** du fichier **GLNeon.cc**.

4.9.3.2 void GLNeon : :evolue (double *dt*) [virtual]

Prototype de la méthode evolue.

appelle la méthode évolue de la classe particule

Paramètres

dt	: le pas de temps

Réimplémentée à partir de **Particule** (p. 23).

Définition à la ligne **46** du fichier **GLNeon.cc**.

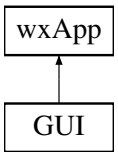
La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLNeon.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GLNeon.cc**

4.10 Référence de la classe GUI

Application principale.

`#include <GUI.h>`

Graphe d'héritage de GUI :



Fonctions membres publiques

- bool **OnInit** ()
  prototype qui lance l'application en appelant les différents constructeurs

4.10.1 Description détaillée

Application principale.

Définition à la ligne **20** du fichier **GUI.h**.

4.10.2 Documentation des fonctions membres

4.10.2.1 bool GUI : :OnInit ( )

prototype qui lance l'application en appelant les différents constructeurs

Nous avons décider de faire une interface utilisateur et la première question est où mettre un tel objet ? Ici dans onInit. C'est de savoir si on veut créer un système aléatoire ou pas et va le transmettre en arguments pour que la donnée transite jusqu'où elle doit aller

`\return true si la fenere à pu être créé`

variable utile pour enregistrer le choix de l'uitlisateur

permet de poser la question à l'utilisateur pour parameter le systeme ou non si il dit oui, le booléens sera true !

Définition à la ligne **24** du fichier **GUI.cc**.

La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GUI.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**GUI.cc**

4.11 Référence de la classe TXTArgon : :h

Représente des atomes d'Argon (version texte)

`#include <TXTArgon.h>`

4.11.1 Description détaillée

Représente des atomes d'Argon (version texte)

Voir également

TestTXTArgon, **TXTHelium** (p. 27), **TXTNeon** (p. 28)

La documentation de cette classe a été générée à partir du fichier suivant :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTArgon.h**

4.12 Référence de la classe TXTNeon : :h

Représente des atomes de Neon (version texte)

`#include <TXTNeon.h>`

4.12.1 Description détaillée

Représente des atomes de Neon (version texte)

Voir également

TestTXTNeon, **TXTArgon** (p. 26), **TXTHelium** (p. 27)

La documentation de cette classe a été générée à partir du fichier suivant :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTNeon.h**

4.13 Référence de la classe TXTHelium : :h

Représente des atomes d'Helium (version texte)

`#include <TXTHelium.h>`

4.13.1 Description détaillée

Représente des atomes d'Helium (version texte)

Voir également

TestTXTHelium, **TXTArgon** (p. 26), **TXTNeon** (p. 28)

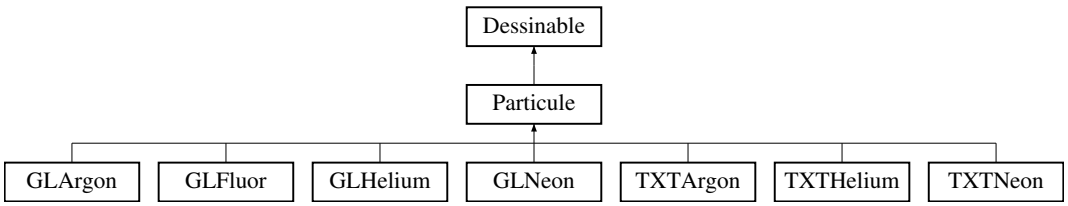
La documentation de cette classe a été générée à partir du fichier suivant :
— /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTHelium.h**

4.14 Référence de la classe Particule

Classe mère dont hérite toutes les classes de type : TXTNom et GLNom Représentation d'une particule (ici version graphique)

`#include <Particule.h>`

Graphe d'héritage de Particule :



Fonctions membres publiques

— **Particule** ()
Constructeur par défaut.
— **Particule** (**Vecteur position**, **Vecteur vitesse**, double **masse**, double **rayon**)
Constructeur.
— **Particule** (**Enceinte** const &enceinte, **GenerateurAleatoire** &tirage, double temperature, double **masse**, double **rayon**)
Constructeur.
— **Vecteur getPosition** () const
Getter de position de la particule.
— std : :ostream & **afficher** (std : :ostream &sortie) const
Permet d'afficher les différents attributs de la particule.
— virtual void **evolue** (double dt)
S'occupe de faire évoluer la particule d'un temps dt.
— void **gere_sorties** (**Enceinte** const &enceinte)
Gére les sorties de l'enceinte.
— **Vecteur pavageCubique** (double espsilon) const
Calcul le pavage cubique autour d'une particule.
— void **collision** (**Particule** &p, **GenerateurAleatoire** &tirage)
Effectue la collision de deux particules.
— double **get_rayon** ()
Getter.

Attributs protégés

— **Vecteur position**
— **Vecteur vitesse**
— double const **masse**
— double const **rayon**

4.14.1 Description détaillée

Classe mère dont hérite toutes les classes de type : TXTNom et GLNom Représentation d'une particule (ici version graphique)

C'est une classe virtuelle pure, on ne peut donc pas créer d'instances de cette classe car il n'est pas logique de créer l'objet générale mais plutôt des classes spécialisées. Elle permet d'avoir une structure générale et des méthodes que chacunes des particules devra redéfinir.

Définition à la ligne **27** du fichier **Particule.h**.

4.14.2 Documentation des constructeurs et destructeur

4.14.2.1 Particule : :Particule ( )

Constructeur par défaut.

Initialise tous les paramètres de type vecteur comme des vecteurs nuls et ceux de type double à 0

Définition à la ligne **18** du fichier **Particule.cc**.

4.14.2.2 Particule : :Particule (Vecteur *position,* Vecteur *vitesse,* double *masse,* double *rayon*)

Constructeur.

Permet d'initialiser une particule en fournissant une position, une vitesse, une masse et un rayon

Paramètres

position	**Vecteur** (p. 29) indiquant la position de la particule
vitesse	**Vecteur** (p. 29) indiquant la vitesse de la particule
masse	Nombre réel indiquant la masse de la particule
rayon	Nombre réel indiquant le rayon de la particule

Définition à la ligne **27** du fichier **Particule.cc**.

4.14.2.3 Particule : :Particule (Enceinte const & *enceinte,* GenerateurAleatoire & *tirage,* double *temperature,* double *masse,* double *rayon*)

Constructeur.

Permet d'initialiser aléatoirement la position et la vitesse de la particule sur une enceinte

Paramètres

enceinte	**Enceinte** (p. 9) dans laquelle doit se trouver la particule

Définition à la ligne **33** du fichier **Particule.cc**.

4.14.3 Documentation des fonctions membres

4.14.3.1 ostream & Particule : :afficher (std : :ostream & *sortie*) const

Permet d'afficher les différents attributs de la particule.

Paramètres

sortie	ostream sur lequel l'affichage s'effectue

Renvoie

ostream sur lequel l'affichage a été fait

Définition à la ligne **57** du fichier **Particule.cc**.

4.14.3.2 void Particule : :collision (Particule & *p,* GenerateurAleatoire & *tirage*)

Effectue la collision de deux particules.

Paramètres

p	**Particule** (p. 22) avec laquelle ∗this entre en collision
tirage	**GenerateurAleatoire** (p. 13) permettant d'effectuer différents tirages de nombres dans la méthode

Définition à la ligne **132** du fichier **Particule.cc**.

4.14.3.3 void Particule : :evolue (double *dt*) [virtual]

S'occupe de faire évoluer la particule d'un temps dt.

Paramètres

dt	Pas de temps

Réimplémentée dans **GLFluor** (p. 17), **GLArgon** (p. 15), **GLNeon** (p. 21), et **GLHelium** (p. 19).

Définition à la ligne **64** du fichier **Particule.cc**.

4.14.3.4 void Particule : :gere_sorties (Enceinte const & *enceinte*)

Gére les sorties de l'enceinte.

Détermine si la particule sort ou non de l'enceinte. Si elle sort effectivement (c'est à dire le BORD de la particule est sorti de l'enceinte), on change les Vecteurs position et vitesse en conséquence

Paramètres

enceinte	**Enceinte** (p. 9) dans laquelle la particule se trouve et doit rester

Définition à la ligne **73** du fichier **Particule.cc**.

4.14.3.5 double Particule : :get_rayon ( )

Getter.

Utile pour déterminer le pas d'espace epsilon du systeme

Renvoie

Retourne le rayon de la particule

Définition à la ligne **150** du fichier **Particule.cc**.

4.14.3.6 Vecteur Particule : :getPosition ( ) const

Getter de position de la particule.

Utile pour donner la postion aux méthodes de translation dans dessine de chaque particule

Renvoie

Retourne le vecteur position de la particule

Définition à la ligne **51** du fichier **Particule.cc**.

4.14.3.7 Vecteur Particule : :pavageCubique (double *epsilon*) const

Calcul le pavage cubique autour d'une particule.

Paramètres

epsilon	Pas d'espace

Renvoie

Retourne un vecteur qui représente le pavage cubique autour de la particule (et peut-être comparé avec le vecteur de pavage d'une autre particule)

Définition à la ligne **124** du fichier **Particule.cc**.

4.14.4 Documentation des données membres

4.14.4.1 double const Particule : :masse [protected]

Nombre réel représentant la masse de la particule (nécessaire pour le calcul des nouvelles vitesses lors d'une collision)

Définition à la ligne **38** du fichier **Particule.h**.

4.14.4.2 Vecteur Particule : :position [protected]

Vecteur (p. 29) représentant la position de la particule dans l'enceinte

Définition à la ligne **34** du fichier **Particule.h**.

4.14.4.3 double const Particule : :rayon [protected]

Nombre réel représentant le rayon de la particule (nécessaire pour déterminer si des particules entrent en collision)

Définition à la ligne **40** du fichier **Particule.h**.

4.14.4.4 Vecteur Particule : :vitesse [protected]

Vecteur (p. 29) représentant la vitesse de la particule

Définition à la ligne **36** du fichier **Particule.h**.

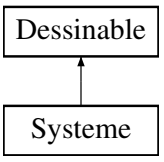
La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Particule.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Particule.cc**

4.15 Référence de la classe Systeme

Permet de créer, de dessiner et de faire évoluer des objets de type **Systeme** (p. 24) formés d'une enceinte et de particules.

`#include <Systeme.h>`

Graphe d'héritage de Systeme :



Fonctions membres publiques

- **Systeme** ()
  Constructeur par défaut.
- **Systeme** (bool reglage)
  Constructeur.
- **Systeme** (double largeur, double longueur, double hauteur, double temperature)
  Constructeur.
- ∼**Systeme** ()
  Destructeur.
- void **ajouterArgon** (unsigned int nbr_argon, bool canon=false)
  Permet l'ajout de nbr_argon particules d'Argon.
- void **ajouterFluor** (unsigned int nbr_fluor, bool canon=false)
  Permet l'ajout de nbr_fluor particules de Fluor.
- void **ajouterHelium** (unsigned int nbr_helium, bool canon=false)
  Permet l'ajout de nbr_helium particules d'Helium.
- void **ajouterNeon** (unsigned int nbr_neon, bool canon=false)
  Permet l'ajout de nbr_neon particules de Neon.
- void **dessine** () const override
  Dessine le systeme (version graphique)
- void **evolue** (double dt)
  Evolue le systeme d'un temps dt.

4.15.1 Description détaillée

Permet de créer, de dessiner et de faire évoluer des objets de type **Systeme** (p. 24) formés d'une enceinte et de particules.

Définition à la ligne **23** du fichier **Systeme.h**.

4.15.2 Documentation des constructeurs et destructeur

4.15.2.1 Systeme : :Systeme ( )

Constructeur par défaut.

Initialise un nombre aléatoire de particules et dans une enceinte par défaut à la température normale

Définition à la ligne **27** du fichier **Systeme.cc**.

4.15.2.2 Systeme : :Systeme (bool *reglage*)

Constructeur.

Ce constructeur permet à l'utilisateur de paramétrer lui même le systeme au lancement du programme grâce à une interface graphique.

Paramètres

reglage	booléen permettant de différencier l'initialisation par défaut et l'initialisation paramétrée

Définition à la ligne **37** du fichier **Systeme.cc**.

4.15.2.3 Systeme : :Systeme (double *largeur,* double *longueur,* double *hauteur,* double *temperature*)

Constructeur.

Permet d'initialiser le systeme en fournissant tous les paramètres avant le lancement du programme. Les particules elles sont initialisées complètement au hasard.

Paramètres

largeur	Largeur de l'enceinte
longueur	Longueur de l'enceinte
hauteur	Hauteur de l'enceinte
temperature	Température du système

Définition à la ligne **79** du fichier **Systeme.cc**.

4.15.2.4 Systeme : :∼Systeme ( )

Destructeur.

j'ai enlevé la valeur par défaut parce que conceptuellement c'est mieux, on initialise rien (hasard total) ou tout (determination totale)

Permet de libérer la mémoire allouée au moment de la création des unique_ptr<Particule>

Définition à la ligne **84** du fichier **Systeme.cc**.

4.15.3 Documentation des fonctions membres

4.15.3.1 void Systeme : :ajouterArgon (unsigned int *nbr_argon,* bool *canon =* `false`)

Permet l'ajout de nbr_argon particules d'Argon.

Paramètres

nbr_argon	Nombre de particules d'Argon à ajouter au système
canon	booléen permettant de déterminer la manière dont l'ajout doit se faire : true -> tire de canon = ajout d'une file de particule allant toutes dans la même direction false -> initialisation aléatoire de la position et de la vitesse de chaque particule

Définition à la ligne **164** du fichier **Systeme.cc**.

4.15.3.2 void Systeme : :ajouterFluor (unsigned int *nbr_fluor,* bool *canon =* `false`)

Permet l'ajout de nbr_fluor particules de Fluor.

Paramètres

nbr_fluor	Nombre de particules de Fluor à ajouter au système
canon	booléen permettant de déterminer la manière dont l'ajout doit se faire : true -> tire de canon = ajout d'une file de particule allant toutes dans la même direction false -> initialisation aléatoire de la position et de la vitesse de chaque particule

Définition à la ligne **185** du fichier **Systeme.cc**.

4.15.3.3 void Systeme : :ajouterHelium (unsigned int *nbr_helium,* bool *canon =* `false`)

Permet l'ajout de nbr_helium particules d'Helium.

Paramètres

nbr_helium	Nombre de particules d'Helium à ajouter au système
canon	booléen permettant de déterminer la manière dont l'ajout doit se faire : true -> tire de canon = ajout d'une file de particule allant toutes dans la même direction false -> initialisation aléatoire de la position et de la vitesse de chaque particule

Définition à la ligne **206** du fichier **Systeme.cc**.

4.15.3.4 void Systeme : :ajouterNeon (unsigned int *nbr_neon,* bool *canon =* `false`)

Permet l'ajout de nbr_neon particules de Neon.

Paramètres

nbr_neon	Nombre de particules de Neon à ajouter au système
canon	booléen permettant de déterminer la manière dont l'ajout doit se faire : true -> tire de canon = ajout d'une file de particule allant toutes dans la même direction false -> initialisation aléatoire de la position et de la vitesse de chaque particule

Définition à la ligne **227** du fichier **Systeme.cc**.

4.15.3.5 void Systeme : :dessine ( ) const `[override],[virtual]`

Dessine le systeme (version graphique)

Dessine l'enceinte, le canon et l'ensemble des particules du systeme

Implémente **Dessinable** (p. 9).

Définition à la ligne **99** du fichier **Systeme.cc**.

4.15.3.6 void Systeme : :evolue (double *dt*)

Evolue le systeme d'un temps dt.

Evolution déterministe sans sauvegarde. Pour chaque particule : on l'évolue, puis on regarde si elle sort de l'enceinte et enfin on regarde si elle entre en collision avec une autre particule

Paramètres

dt	Pas de temps

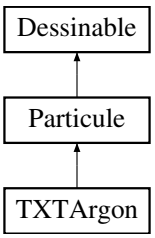
Définition à la ligne **126** du fichier **Systeme.cc**.

La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Systeme.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Systeme.cc**

4.16 Référence de la classe TXTArgon

`#include <TXTArgon.h>`

Graphe d'héritage de TXTArgon :



Classes

- class **h**
  Représente des atomes d'Argon (version texte)

Fonctions membres publiques

- **TXTArgon** (**Vecteur position**, **Vecteur vitesse**)
  Constructeur.
- void **dessine** () const override
  *Redéfinition de la méthode dessine (héritée de **Dessinable** (p. 8))*

Membres hérités additionnels

4.16.1 Description détaillée

Définition à la ligne **20** du fichier **TXTArgon.h**.

4.16.2 Documentation des constructeurs et destructeur

4.16.2.1 TXTArgon : :TXTArgon (Vecteur *position,* Vecteur *vitesse*)

Constructeur.

Contructeur prenant deux paramètres de type **Vecteur** (p. 29)

Paramètres

position	**Vecteur** (p. 29) correspondant à la position de la particule
vitesse	**Vecteur** (p. 29) correspondant à la vitesse de la particule

Définition à la ligne **17** du fichier **TXTArgon.cc**.

4.16.3 Documentation des fonctions membres

4.16.3.1 void TXTArgon : :dessine ( ) const [override], [virtual]

Redéfinition de la méthode dessine (héritée de **Dessinable** (p. 8))

Methode permettant de dessiner la version texte d'une particule d'Argon

Implémente **Dessinable** (p. 9).

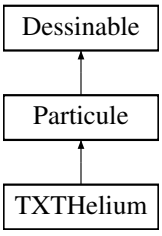
Définition à la ligne **22** du fichier **TXTArgon.cc**.

La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTArgon.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTArgon.cc**

4.17 Référence de la classe TXTHelium

`#include <TXTHelium.h>`

Graphe d'héritage de TXTHelium :



Classes

— class **h**
Représente des atomes d'Helium (version texte)

Fonctions membres publiques

— **TXTHelium** (**Vecteur position**, **Vecteur vitesse**)
Constructeur.
— void **dessine** () const override
*Redéfinition de la méthode dessine (héritée de **Dessinable** (p. 8))*

Membres hérités additionnels

4.17.1 Description détaillée

Définition à la ligne **18** du fichier **TXTHelium.h**.

4.17.2 Documentation des constructeurs et destructeur

4.17.2.1 TXTHelium : :TXTHelium (Vecteur *position,* Vecteur *vitesse*)

Constructeur.

Contructeur prenant deux paramètres de type **Vecteur** (p. 29)

Paramètres

position	**Vecteur** (p. 29) correspondant à la position de la particule
vitesse	**Vecteur** (p. 29) correspondant à la vitesse de la particule

Définition à la ligne **17** du fichier **TXTHelium.cc**.

4.17.3 Documentation des fonctions membres

4.17.3.1 void TXTHelium : :dessine ( ) const [override],[virtual]

Redéfinition de la méthode dessine (héritée de **Dessinable** (p. 8))

Methode permettant de dessiner la version texte d'une particule d'Helium

Implémente **Dessinable** (p. 9).

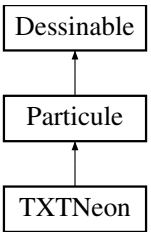
Définition à la ligne **22** du fichier **TXTHelium.cc**.

La documentation de cette classe a été générée à partir des fichiers suivants :
— /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTHelium.h**
— /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTHelium.cc**

4.18 Référence de la classe TXTNeon

`#include <TXTNeon.h>`

Graphe d'héritage de TXTNeon :



Classes

— class **h**
Représente des atomes de Neon (version texte)

Fonctions membres publiques

— **TXTNeon** (**Vecteur position**, **Vecteur vitesse**)
Constructeur.
— void **dessine** () const override
*Redéfinition de la méthode dessine (héritée de **Dessinable** (p. 8))*

Membres hérités additionnels

4.18.1 Description détaillée

Définition à la ligne **18** du fichier **TXTNeon.h**.

4.18.2 Documentation des constructeurs et destructeur

4.18.2.1 TXTNeon : :TXTNeon (Vecteur *position,* Vecteur *vitesse*)

Constructeur.

Constructeur prenant deux paramètres de type **Vecteur** (p. 29)

Paramètres

position	**Vecteur** (p. 29) correspondant à la position de la particule
vitesse	**Vecteur** (p. 29) correspondant à la vitesse de la particule

Définition à la ligne **17** du fichier **TXTNeon.cc**.

4.18.3 Documentation des fonctions membres

4.18.3.1 void TXTNeon : :dessine ( ) const [override], [virtual]

Redéfinition de la méthode dessine (héritée de **Dessinable** (p. 8))

Methode permettant de dessiner la version texte d'une particule de Neon

Implémente **Dessinable** (p. 9).

Définition à la ligne **22** du fichier **TXTNeon.cc**.

La documentation de cette classe a été générée à partir des fichiers suivants :
— /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTNeon.h**
— /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**TXTNeon.cc**

4.19 Référence de la classe Vecteur

Prototype de la classe **Vecteur** (p. 29).

`#include <Vecteur.h>`

Fonctions membres publiques

- **Vecteur** ()
  Constructeur sans arguments.
- **Vecteur** (double x, double y, double z)
  Constructeur avec arguments qui prends les coordonnées du vecteurs.
- double **getX** () const
  Prototype du getters de la première coordonnée.
- double **getY** () const
  Prototype du getters de la deuxième coordonnée.
- double **getZ** () const
  Prototype du getters de la deuxième coordonnée.
- bool **operator==** (**Vecteur** const &) const
  Prototype comparateurs ==.
- bool **operator !=** (**Vecteur** const &v) const
  Prototype comparateurs !=.
- **Vecteur** & **operator+=** (**Vecteur** const &v1)
  Prototype méthodes +=.
- **Vecteur** & **operator-=** (**Vecteur** const &v1)
  Prototype méthodes -=.
- **Vecteur** & **operator∗=** (double const &scalaire)
  Prototype multiplication par un scalaire.
- double **operator∗** (const **Vecteur** &v1) const
  Prototype produit scalaire.
- **Vecteur** & **operator^=** (**Vecteur** const &v1)
  Prototype produit vectoriel.
- const **Vecteur operator-** ()
  Prototype vecteur opposé
- std : :ostream & **afficher** (std : :ostream &sortie) const
  Prototype méthode afficher.

4.19.1 Description détaillée

Prototype de la classe **Vecteur** (p. 29).

Cette classe est celle qui permet de créer un vecteur, de gérer le calcul vectoriel de notre programme.

Définition à la ligne **23** du fichier **Vecteur.h**.

4.19.2 Documentation des constructeurs et destructeur

4.19.2.1 Vecteur : :Vecteur ( )

Constructeur sans arguments.

Constructeur de la classe **Vecteur** (p. 29) sans arguments

Définition à la ligne **20** du fichier **Vecteur.cc**.

4.19.2.2 Vecteur : :Vecteur (double *x*, double *y*, double *z*)

Constructeur avec arguments qui prends les coordonnées du vecteurs.

Constructeur de la classe **Vecteur** (p. 29) avec trois arguements de type double

Paramètres

x	: première coordonnée x
y	: deuxième coordonnée y
z	: troisième coordonnée z

Définition à la ligne **30** du fichier **Vecteur.cc**.

4.19.3 Documentation des fonctions membres

4.19.3.1 ostream & Vecteur : :afficher (std : :ostream & *sortie*) const

Prototype méthode afficher.

Paramètres

ostream&	sortie : permet de choisir le flot de sortie

Renvoie

un ostream& pour pouvoir faire des "appels multiples"

Définition à la ligne **172** du fichier **Vecteur.cc**.

4.19.3.2 double Vecteur : :getX ( ) const

Prototype du getters de la première coordonnée.

Methode qui permet de retourner les attributs privés du **Vecteur** (p. 29)

Renvoie

la coordonée en x

Définition à la ligne **41** du fichier **Vecteur.cc**.

4.19.3.3 double Vecteur : :getY ( ) const

Prototype du getters de la deuxième coordonnée.

Methode qui permet de retourner les attributs privés du **Vecteur** (p. 29)

Renvoie

la coordonée en y

Définition à la ligne **50** du fichier **Vecteur.cc**.

4.19.3.4 double Vecteur : :getZ ( ) const

Prototype du getters de la deuxième coordonnée.

Methode qui permet de retourner les attributs privés du **Vecteur** (p. 29)

Renvoie

la coordonée en z

Définition à la ligne **59** du fichier **Vecteur.cc**.

4.19.3.5 bool Vecteur : :operator != (Vecteur const & *v*) const

Prototype comparateurs !=.

Paramètres

Vecteur *(p. 29)*	const& v : permet de comparer la différence au deuxième vecteur

Renvoie

true si le vecteur est différents de v sinon il retourne false

Définition à la ligne **84** du fichier **Vecteur.cc**.

4.19.3.6 double Vecteur : :operator∗ (const Vecteur & *v1*) const

Prototype produit scalaire.

fait le produit scalaire de deux vecteurs

Paramètres

Vecteur *(p. 29)*	const& v1 : permet de calculer le produit scalaire

Renvoie

un double (nombre réel) qui est le produit scalaire

Définition à la ligne **163** du fichier **Vecteur.cc**.

4.19.3.7 Vecteur & Vecteur : :operator∗= (double const & *scalaire*)

Prototype multiplication par un scalaire.

permet de multiplier le vecteur à un nombre scalaire et de l'enregistrer dans le vecteur appelant

Paramètres

double	const& scalaire : le nombre par lequel on va multiplier

Renvoie

le vecteur appelant

Définition à la ligne **133** du fichier **Vecteur.cc**.

4.19.3.8 Vecteur & Vecteur : :operator+= (Vecteur const & *v1*)

Prototype méthodes +=.

aditionne le vecteur appelant au vecteur v en paramètre et l'enregistre dans l'objet appelant

Paramètres

Vecteur *(p. 29)*	const& v : permet d'additionner le deuxième vecteur

Renvoie

le vecteur appelant

Définition à la ligne **95** du fichier **Vecteur.cc**.

4.19.3.9 const Vecteur Vecteur : :operator- ( )

Prototype vecteur opposé

fait l'opposer du vecteur appelant

Renvoie

le vecteur appelant opposé

Définition à la ligne **122** du fichier **Vecteur.cc**.

4.19.3.10 Vecteur & Vecteur : :operator-= (Vecteur const & *v1*)

Prototype méthodes -=.

soustrait le vecteur appelant au vecteur v en paramètre et l'enregistre dans l'objet appelant

Paramètres

Vecteur *(p. 29)*	const& v : permet de soustraire le deuxième vecteur

Renvoie

le vecteur appelant

Définition à la ligne **109** du fichier **Vecteur.cc**.

4.19.3.11 bool Vecteur : :operator== (Vecteur const & *v*) const

Prototype comparateurs ==.

Paramètres

Vecteur *(p. 29)*	const& v : permet de comparer au deuxième vecteur

Renvoie

true s'il sont égale et false sinon

Définition à la ligne **68** du fichier **Vecteur.cc**.

4.19.3.12 Vecteur & Vecteur : :operator^= (Vecteur const & *v1*)

Prototype produit vectoriel.

permet de faire le produit vectoriel avec le vecteur en paramètre et l'enregistrer dans le vecteur appelant le calcul ce fait comme le produit vectoriel usuels

Paramètres

Vecteur *(p. 29)*	const& v1 : permet de calculer le produit vectoriel avec le deuxième vecteur

Renvoie

le vecteur appelant qui est le produit vectoriel des deux vecteurs

Définition à la ligne **148** du fichier **Vecteur.cc**.

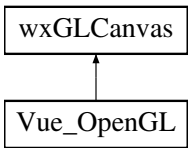
La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Vecteur.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Vecteur.cc**

4.20 Référence de la classe Vue_OpenGL

Prototype de la classe **Vue_OpenGL** (p. 33).

```
#include <Vue_OpenGL.h>
```

Graphe d'héritage de Vue_OpenGL :



Fonctions membres publiques

- **Vue_OpenGL** (wxWindow *parent, wxSize const &taille=wxDefaultSize, wxPoint const &position=wxDefaultPosition)
  constructeur ***Vue_OpenGL*** *(*p. *33)*
- **Vue_OpenGL** (wxWindow *parent, bool reglage, wxSize const &taille=wxDefaultSize, wxPoint const &position=wxDefaultPosition)
  constructeur ***Vue_OpenGL*** *(*p. *33)*
- virtual ~**Vue_OpenGL** ()
  Destructeur.
- void **InitOpenGL** ()
  prototype public de la méthode qui va appeler evolue de systeme

Fonctions membres protégées

- void **dessine** (wxPaintEvent &evenement)
  prototype de la méthode dessine
- void **OnSize** (wxSizeEvent &evenement)
- void **OnKeyDown** (wxKeyEvent &evenement)
  prototype de la méthode qui gère les événements du claviers
- void **OnEnterWindow** (wxMouseEvent &evenement)
  prototype de la méthode qui gères les éveénement de la souris
- void **OnTimer** (wxTimerEvent &event)
  prototype de la méthode du Timer qui va appeler la méthode évolue du système
- void **RotateTheta** (GLdouble deg)
  prototype de la méthode qui fait la rotation de l'angle theta pour la caméra
- void **RotatePhi** (GLdouble deg)
  prototype de la méthode qui fait la rotation de l'angle phi pour la caméra
- void **deplace** (double dr)
  prototype de la méthode qui fait le déplacement de la caméra

4.20.1 Description détaillée

Prototype de la classe **Vue_OpenGL** (p. 33).

Cette classe est ce qui remplira la fenêtre et qui gère les animations dans celle-ci comme notre simulation du gaz parfait. Elle est écrite avec du C++ et de l'OpenGL. representant la partie de l'OpenGL qui va instancier presque toutes les parties de notre simulation

Définition à la ligne **30** du fichier **Vue_OpenGL.h**.

4.20.2 Documentation des constructeurs et destructeur

4.20.2.1 Vue_OpenGL : :Vue_OpenGL (wxWindow * *parent,* wxSize const & *taille =* `wxDefaultSize`, wxPoint const & *position =* `wxDefaultPosition`)

constructeur **Vue_OpenGL** (p. 33)

Constructeur.

Constructeur de la Vue_OenGL qui initialise le système, le timer, la position de la caméra au départ

Définition à la ligne **43** du fichier **Vue_OpenGL.cc**.

4.20.2.2 Vue_OpenGL : :Vue_OpenGL (wxWindow * *parent,* bool *reglage,* wxSize const & *taille =* wxDefaultSize, wxPoint const & *position =* wxDefaultPosition)

constructeur **Vue_OpenGL** (p. 33)

Constructeur.

Constructeur de la Vue_OenGL qui initialise le système, le timer, la position de la caméra au départ

Définition à la ligne **61** du fichier **Vue_OpenGL.cc**.

4.20.2.3 virtual Vue_OpenGL : :~Vue_OpenGL ( ) [inline], [virtual]

Destructeur.

Destructeur de la classe **Vue_OpenGL** (p. 33)

Définition à la ligne **51** du fichier **Vue_OpenGL.h**.

4.20.3 Documentation des fonctions membres

4.20.3.1 void Vue_OpenGL : :deplace (double *dr*) [protected]

prototype de la méthode qui fait le déplacement de la caméra

Définition à la ligne **205** du fichier **Vue_OpenGL.cc**.

4.20.3.2 void Vue_OpenGL : :dessine (wxPaintEvent & *evenement*) [protected]

prototype de la méthode dessine

Définition à la ligne **74** du fichier **Vue_OpenGL.cc**.

4.20.3.3 void Vue_OpenGL : :InitOpenGL ( )

prototype public de la méthode qui va appeler evolue de systeme

Définition à la ligne **213** du fichier **Vue_OpenGL.cc**.

4.20.3.4 void Vue_OpenGL : :OnEnterWindow (wxMouseEvent & *evenement*) [inline], [protected]

prototype de la méthode qui gères les éveénement de la souris

Définition à la ligne **63** du fichier **Vue_OpenGL.h**.

4.20.3.5 void Vue_OpenGL : :OnKeyDown (wxKeyEvent & *evenement*) [protected]

prototype de la méthode qui gère les événements du claviers

Définition à la ligne **119** du fichier **Vue_OpenGL.cc**.

4.20.3.6 void Vue_OpenGL : :OnSize (wxSizeEvent & *evenement*) [protected]

Définition à la ligne **104** du fichier **Vue_OpenGL.cc**.

4.20.3.7 void Vue_OpenGL : :OnTimer (wxTimerEvent & *event*) [protected]

prototype de la méthode du Timer qui va appeler la méthode évolue du système

Définition à la ligne **239** du fichier **Vue_OpenGL.cc**.

4.20.3.8 void Vue_OpenGL : :RotatePhi (GLdouble *deg*) [protected]

prototype de la méthode qui fait la rotation de l'angle phi pour la caméra

Définition à la ligne **197** du fichier **Vue_OpenGL.cc**.

4.20.3.9 void Vue_OpenGL : :RotateTheta (GLdouble *deg*) [protected]

prototype de la méthode qui fait la rotation de l'angle theta pour la caméra

Définition à la ligne **189** du fichier **Vue_OpenGL.cc**.

La documentation de cette classe a été générée à partir des fichiers suivants :
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Vue_OpenGL.h**
- /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/**Vue_OpenGL.cc**

Chapitre 5

Documentation des fichiers

5.1 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Canon.cc

fichier permettant la définition de la classe **Canon** (p. 7)

```
#include "wx/wxprec.h"
#include "wx/wx.h"
#include "wx/glcanvas.h"
#include "Canon.h"
#include <iostream>
```

5.1.1 Description détaillée

fichier permettant la définition de la classe **Canon** (p. 7)

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **Canon.cc**.

5.2 Canon.cc

```
00001
00009 #include "wx/wxprec.h"
00010 #ifndef WX_PRECOMP
00011 #include "wx/wx.h"
00012 #endif
00013 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00014 #include "Canon.h"
00015 #include <iostream>
00016
00017 using namespace std;
00018
00019 /*================================================================================
00020  * Definition des constructeurs
```

```
00021  *================================================================================*/
00032 Canon::Canon()
00033 :rayon(3), longueur(10), cylindre(gluNewQuadric()), bouleFond(gluNewQuadric()), roue1(gluNewQuadric()),
00034     roue2(gluNewQuadric()), enjoliveur1(gluNewQuadric()), enjoliveur2(gluNewQuadric())
00035 {}
00036
00037 /*================================================================================
00038  * Definition des méthodes
00039  *================================================================================*/
00046 void Canon::dessine() const
00047 {
00048
00049     glColor4d(0.0, 0.0, 0.0, 1); // couleur vert, 100%
00051     glPushMatrix();
00052     glTranslated(-longueur/2,-longueur/2,-longueur/2);
00053     glRotated(55, -1,1,0);
00054     gluCylinder(cylindre, rayon, rayon, longueur, 10, 10);
00055     glPopMatrix();
00056
00058     glPushMatrix();
00059     glTranslated(-longueur/2,-longueur/2,-longueur/2);
00060     gluSphere(bouleFond, rayon, 10, 10);
00061     glPopMatrix();
00062
00064     glColor4d(0.7, 0.2, 0.2, 1);
00065     glPushMatrix();
00066     glTranslated(-longueur/1.5+rayon,-longueur/1.5,-longueur/1.5);
00067     glRotated(90, 1,1,0);
00068     gluDisk(enjoliveur1, 0, rayon/1.5, 10, 1);
00069     gluCylinder(roue1, rayon/1.5, rayon/1.5, longueur/10, 10, 10);
00070     glPopMatrix();
00071
00073     glColor4d(0.7, 0.2, 0.2, 1);
00074     glPushMatrix();
00075     glTranslated(-longueur/1.5,-longueur/1.5+rayon,-longueur/1.5);
00076     glRotated(90, 1,1,0);
00077     gluDisk(enjoliveur2, 0,rayon/1.5, 10, 1);
00078     gluCylinder(roue2, rayon/1.5, rayon/1.5, longueur/10, 10, 10);
00079     glPopMatrix();
00080 }
00081
00082
```

5.3 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Canon.h

est la classe qui contient l'objet enceinte qui est la boîte où seront nos particules

```
#include "wx/wxprec.h"
#include "wx/wx.h"
#include "wx/glcanvas.h"
#include "Dessinable.h"
```

Classes

— class **Canon**
*Prototype de la classe **Canon** (p. 7).*

5.3.1 Description détaillée

est la classe qui contient l'objet enceinte qui est la boîte où seront nos particules

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **Canon.h**.

5.4 Canon.h

```
00001
00009 #ifndef PRJ_CANON_H
00010 #define PRJ_CANON_H
00011 #include "wx/wxprec.h"
00012 #ifndef WX_PRECOMP
00013 #include "wx/wx.h"
00014 #endif
00015 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00016 #include "Dessinable.h"
00017
00029 class Canon : public Dessinable
00030 {
00031 /*============================================================================
00032  * Definition des attributs
00033  *============================================================================*/
00034     private:
00035     double rayon;
00036     double longueur;
00037     GLUquadric* cylindre;
00038     GLUquadric* bouleFond;
00039     GLUquadric* roue1;
00040     GLUquadric* roue2;
00041     GLUquadric* enjoliveur1;
00042     GLUquadric* enjoliveur2;
00043
00044
00045     public:
00046 /*============================================================================
00047  * Prototype du constructeur
00048  *============================================================================*/
00050     Canon();
00051
00053     ~Canon()
00054     {
00055         gluDeleteQuadric(cylindre);
00056         gluDeleteQuadric(bouleFond);
00057         gluDeleteQuadric(roue1);
00058         gluDeleteQuadric(roue2);
00059         gluDeleteQuadric(enjoliveur1);
00060         gluDeleteQuadric(enjoliveur2);
00061     }
00062
00063
00064 /*============================================================================
00065  * Prototype des méthodes
00066  *============================================================================*/
00067
00069     virtual void dessine() const;
00070 };
00071
00072 #endif // PRJ_CANON_H
```

5.5 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Dessinable.h

est la super-classe avec une méthode dessine qui permet d'avoir une spécialisation pour chaque type d'objet

Classes

— class **Dessinable**

5.5.1 Description détaillée

est la super-classe avec une méthode dessine qui permet d'avoir une spécialisation pour chaque type d'objet

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **Dessinable.h**.

5.6 Dessinable.h

```
00001
00009 #ifndef PRJ_DESSINABLE_H
00010 #define PRJ_DESSINABLE_H
00011
00012 class Dessinable
00013 {
00014     public :
00016     virtual void dessine() const = 0;
00017 };
00018
00019 #endif // PRJ_DESSINABLE_H
```

5.7 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Enceinte.cc

fichier permettant la définition de la classe enceinte

```
#include "wx/wxprec.h"
#include "wx/wx.h"
#include "wx/glcanvas.h"
#include <iostream>
#include "Enceinte.h"
```

5.7.1 Description détaillée

fichier permettant la définition de la classe enceinte

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **Enceinte.cc**.

5.8 Enceinte.cc

```
00001
00009 #include "wx/wxprec.h"
00010 #ifndef WX_PRECOMP
00011 #include "wx/wx.h"
00012 #endif
00013 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00014
00015 #include <iostream>
00016 #include "Enceinte.h"
00017
00018 using namespace std;
00019
00020 /*================================================================================
00021  * Definition des constructeurs
00022  *================================================================================*/
00027 Enceinte::Enceinte()
00028 :largeur(100), longueur(100), hauteur(100)
00029 {}
00030
00038 Enceinte::Enceinte(bool reglage)
00039 {
00040     do
00041     {
00042         largeur = wxAtoi(wxGetTextFromUser(wxT("Rentrez une largeur (entre 0 et 150)"), wxT("Paramétrage de
      l'enceinte")));
00043     } while (largeur <= 0 or largeur > 150);
00044
00045     do
00046     {
00047         longueur = wxAtoi(wxGetTextFromUser(wxT("Rentrez une longueur (entre 0 et 150)"), wxT("Paramétrage
      de l'enceinte")));
00048     } while (longueur <= 0 or longueur > 150);
00049
00050     do
00051     {
00052         hauteur = wxAtoi(wxGetTextFromUser(wxT("Rentrez une hauteur (entre 0 et 150)"), wxT("Paramétrage de
      l'enceinte")));
00053     } while (hauteur <= 0 or hauteur > 150);
00054 }
00055
00064 Enceinte::Enceinte(double largeur, double longueur, double hauteur)
00065 :largeur(largeur), longueur(longueur), hauteur(hauteur)
00066 {}
00067
00068 /*================================================================================
00069  * Definition des méthodes
00070  *================================================================================*/
00076 double Enceinte::getLargeur() const
00077 {
00078     return largeur;
00079 }
00085 double Enceinte::getLongueur() const
00086 {
00087     return longueur;
00088 }
00094 double Enceinte::getHauteur() const
00095 {
00096     return hauteur;
00097 }
00101 void Enceinte::display() const
00102 {
00103     cout << largeur << ", " << longueur << ", " << hauteur << endl;
00104 }
00105
00113 void Enceinte::dessine() const
00114 {
00115     glColor4d(0.0, 0.0, 0.0, 0.5);
00116     glPointSize(4);
00117
00118     //fond de l'enceinte
00119     glBegin(GL_LINE_STRIP);
00120     glVertex3d(0,0,0);
00121     glVertex3d(largeur,0,0);
00122     glVertex3d(largeur,longueur,0);
00123     glVertex3d(0,longueur,0);
00124     glVertex3d(0,0,0);
00125     glEnd();
00126
00127     //fond gauche de l'enceinte
00128     glBegin(GL_LINE_STRIP);
00129     glVertex3d(0,0,0);
00130     glVertex3d(0,0,hauteur);
00131     glEnd();
00132
```

```
00133     //fond droite de l'enceinte
00134     glBegin(GL_LINE_STRIP);
00135     glVertex3d(0,longueur,0);
00136     glVertex3d(0, longueur,hauteur);
00137     glEnd();
00138
00139     //devant gauche de l'enceinte
00140     glBegin(GL_LINE_STRIP);
00141     glVertex3d(largeur,0,0);
00142     glVertex3d(largeur,0,hauteur);
00143     glEnd();
00144
00145     //devant droite de l'enceinte
00146     glBegin(GL_LINE_STRIP);
00147     glVertex3d(largeur, longueur,0);
00148     glVertex3d(largeur,longueur,hauteur);
00149     glEnd();
00150
00151     //plafond  de l'enceinte
00152     glBegin(GL_LINE_STRIP);
00153     glVertex3d(0,0,hauteur);
00154     glVertex3d(largeur,0,hauteur);
00155     glVertex3d(largeur,longueur,hauteur);
00156     glVertex3d(0,longueur,hauteur);
00157     glVertex3d(0,0,hauteur);
00158     glEnd();
00159 }
00160
00161
```

5.9 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Enceinte.h

est la classe qui contient l'objet enceinte qui est la boîte où seront nos particules

```
#include "Dessinable.h"
```

Classes

— class **Enceinte**

5.9.1 Description détaillée

est la classe qui contient l'objet enceinte qui est la boîte où seront nos particules

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **Enceinte.h**.

5.10 Enceinte.h

```
00001
00009 #ifndef PRJ_ENCEINTE_H
00010 #define PRJ_ENCEINTE_H
00011
```

```
00012 #include "Dessinable.h"
00013
00019 class Enceinte : public Dessinable
00020 {
00021 /*============================================================================
00022  * Definition des attributs
00023  *===========================================================================*/
00024     private:
00025     double largeur;
00026     double longueur;
00027     double hauteur;
00028
00029     public:
00030 /*============================================================================
00031  * Prototypage des constructeurs
00032  *===========================================================================*/
00034     Enceinte();
00036     Enceinte(bool reglage);
00038     Enceinte(double largeur, double longueur, double hauteur);
00039
00040 /*============================================================================
00041  * Prototypage des méthodes
00042  *===========================================================================*/
00044     double getLargeur() const;
00046     double getLongueur() const;
00048     double getHauteur() const;
00050     void display() const;
00052     virtual void dessine() const;
00053 };
00054
00055 #endif // PRJ_ENCEINTE_H
```

5.11 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Fenetre.cc

est la définition de la classe fenêtre en OpenGL

`#include "Fenetre.h"`

5.11.1 Description détaillée

est la définition de la classe fenêtre en OpenGL

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **Fenetre.cc**.

5.12 Fenetre.cc

```
00001
00009 #include "Fenetre.h"
00010
00011 /* ================================================================
00012    Implementation de Fenetre
00013    ================================================================ */
00014
00015 BEGIN_EVENT_TABLE(Fenetre, wxFrame)
00016     EVT_MENU( wxID_EXIT, Fenetre::OnExit )
```

```
00017 END_EVENT_TABLE()
00018
00019 // ============================================================================
00020
00028 Fenetre::Fenetre( wxString const& titre
00029                 , wxSize   const& taille
00030                 , wxPoint  const& position
00031                 , long            style
00032                 )
00033 : wxFrame(0, wxID_ANY, titre, position, taille, style)
00034 , fogl(new Vue_OpenGL(this))
00035 {
00037   wxMenu* winMenu = new wxMenu;
00039   winMenu->Append(wxID_EXIT, wxT("&Close"));
00041   wxMenuBar* menuBar = new wxMenuBar;
00043   menuBar->Append(winMenu, wxT("&Window"));
00044
00045   SetMenuBar(menuBar);
00046
00048   Show(true);
00049
00051   fogl->InitOpenGL();
00052 }
00053
00065 Fenetre::Fenetre( wxString const& titre
00066                 , bool reglage
00067                 , wxSize   const& taille
00068                 , wxPoint  const& position
00069                 , long            style
00070                 )
00071 : wxFrame(0, wxID_ANY, titre, position, taille, style)
00072 , fogl(new Vue_OpenGL(this, reglage))
00073 {
00075   wxMenu* winMenu = new wxMenu;
00077   winMenu->Append(wxID_EXIT, wxT("&Close"));
00079   wxMenuBar* menuBar = new wxMenuBar;
00081   menuBar->Append(winMenu, wxT("&Window"));
00082
00083   SetMenuBar(menuBar);
00084
00086   Show(true);
00087
00089   fogl->InitOpenGL();
00090 }
```

5.13 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Fenetre.h

est le prototype de la classe fenetre qui permettra de créer une fentre contenant notre application

```
#include "wx/wx.h"
#include "wx/glcanvas.h"
#include "Vue_OpenGL.h"
```

Classes

— class **Fenetre**

5.13.1 Description détaillée

est le prototype de la classe fenetre qui permettra de créer une fentre contenant notre application

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **Fenetre.h**.

5.14 Fenetre.h

```
00001
00009 #ifndef PRJ_FENETRE_H
00010 #define PRJ_FENETRE_H
00011
00012 #ifndef WX_PRECOMP
00013 #include "wx/wx.h"
00014 #endif
00015 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00016 #include "Vue_OpenGL.h"
00017
00018
00023 class Fenetre: public wxFrame
00024 {
00025     public:
00026
00028     Fenetre( wxString const& titre
00029          , wxSize   const& taille   = wxDefaultSize
00030          , wxPoint  const& position = wxDefaultPosition
00031          , long            style    = wxDEFAULT_FRAME_STYLE
00032          );
00033
00035     Fenetre( wxString const& titre
00036          , bool reglage
00037          , wxSize   const& taille   = wxDefaultSize
00038          , wxPoint  const& position = wxDefaultPosition
00039          , long            style    = wxDEFAULT_FRAME_STYLE
00040          );
00042     virtual ~Fenetre() {}
00043
00044     protected:
00046     void OnExit(wxCommandEvent& event) { Close(true); }
00047
00048     //attibut
00049     Vue_OpenGL* fogl;
00050
00051 DECLARE_EVENT_TABLE()
00052 };
00053 #endif
```

5.15 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/GenerateurAleatoire.cc

Définition de la classe qui permet de générer des nombres aléatoires.

`#include "GenerateurAleatoire.h"`

5.15.1 Description détaillée

Définition de la classe qui permet de générer des nombres aléatoires.

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GenerateurAleatoire.cc**.

5.16 GenerateurAleatoire.cc

```
00001
00009 #include "GenerateurAleatoire.h"
00010
00011 using namespace std;
00012
00016 GenerateurAleatoire::GenerateurAleatoire()
00017  : generateur(std::random_device()())
00018 {}
00019
00023 GenerateurAleatoire::GenerateurAleatoire(unsigned int graine)
00024  : generateur(graine)
00025 {}
00026
00032 int GenerateurAleatoire::uniforme_entier(int min, int max) {
00033     return distribution_uniforme_entier(generateur, std::uniform_int_distribution<int>::param_type{min, max
      });
00034 }
00035
00041 double GenerateurAleatoire::uniforme_reel(double min, double max) {
00042     return distribution_uniforme_reel(generateur, std::uniform_real_distribution<double>::param_type{min,
      max});
00043 }
00044
00050 double GenerateurAleatoire::gaussienne(double moyenne, double ecart_type) {
00051     return distribution_gaussienne(generateur, std::normal_distribution<double>::param_type{moyenne,
      ecart_type});
00052 }
```

5.17 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/GenerateurAleatoire.h

Prototype de la classe qui permet de générer des nombres aléatoires.

`#include <random>`

Classes

— class **GenerateurAleatoire**
Représente des objets capables de générer des nombres aléatoires.

5.17.1 Description détaillée

Prototype de la classe qui permet de générer des nombres aléatoires.

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GenerateurAleatoire.h**.

5.18 GenerateurAleatoire.h

```
00001
00009 #ifndef PRJ_GENERATEURALEATOIRE_H
00010 #define PRJ_GENERATEURALEATOIRE_H
00011
00012 #include <random>
00013
00020 class GenerateurAleatoire {
00021
00022     public:
00023
00025     GenerateurAleatoire();
00027     GenerateurAleatoire(unsigned int graine);
00028
00030     int uniforme_entier(int min, int max);
00032     double uniforme_reel(double min, double max);
00034     double gaussienne(double moyenne, double ecart_type);
00035
00036     private:
00037
00039     std::default_random_engine generateur;
00041     std::uniform_int_distribution<int> distribution_uniforme_entier;
00043     std::uniform_real_distribution<double> distribution_uniforme_reel;
00045     std::normal_distribution<double> distribution_gaussienne;
00046
00047 };
00048
00049 #endif // PRJ_GENERATEURALEATOIRE_H
```

5.19 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/GLArgon.cc

est la définition de la classe de la particule argon en OpenGL

`#include "GLArgon.h"`

5.19.1 Description détaillée

est la définition de la classe de la particule argon en OpenGL

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GLArgon.cc**.

5.20 GLArgon.cc

```
00001
00009 #include "GLArgon.h"
00010
00011 /*================================================================================
00012  * Definition des constructeurs et destructeur
00013  *================================================================================*/
00020 GLArgon::GLArgon(Vecteur position, Vecteur vitesse)
00021  : Particule(position, vitesse, 39.948, 0.71), sphere(gluNewQuadric())
00022 {}
00023
```

```
00031 GLArgon::GLArgon(Enceinte const& enceinte, GenerateurAleatoire& tirage, double temperature)
00032  : Particule(enceinte, tirage, temperature, 39.948, 0.71), sphere(gluNewQuadric())
00033 {}
00034
00035 GLArgon::~GLArgon()
00036 {
00037     gluDeleteQuadric(sphere);
00038 }
00039
00040 /*================================================================================
00041  * Definition des méthodes
00042  *================================================================================*/
00047 void GLArgon::evolue(double dt)
00048 {
00049     Particule::evolue(dt);
00050 }
00051
00056 void GLArgon::dessine() const
00057 {
00058     glColor4d(0.0, 1.0, 0.0, 1);
00059
00060     //ajouter le déplacement
00061     glPushMatrix();
00062
00063     glTranslated(this->getPosition().getX(), this->getPosition().getY(), this->
      getPosition().getZ());
00064
00065     gluSphere(sphere,rayon,10,10);
00066     glPopMatrix();
00067 }
```

5.21 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/GLArgon.h

est le prototype de la classe de la particule argon en OpenGL

```
#include "wx/wxprec.h"
#include "wx/wx.h"
#include "wx/glcanvas.h"
#include "Particule.h"
```

Classes

— class **GLArgon**
*Prototype de la classe **GLArgon** (p. 13).*

5.21.1 Description détaillée

est le prototype de la classe de la particule argon en OpenGL

Prototype de la classe de la particule Neon en version texte.

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **GLArgon.h**.

5.22 GLArgon.h

```
00001
00009 #include "wx/wxprec.h"
00010 #ifndef WX_PRECOMP
00011 #include "wx/wx.h"
00012 #endif
00013 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00014
00015 #include "Particule.h"
00016
00017
00025 class GLArgon : public Particule
00026 {
00027     public:
00028 /*============================================================================
00029  * Prototyopes des constructeurs et destructeur
00030  *============================================================================*/
00032     GLArgon(Vecteur position, Vecteur vitesse);
00034     GLArgon(Enceinte const& enceinte, GenerateurAleatoire& tirage, double temperature);
00036     virtual ~GLArgon();
00037
00038 /*============================================================================
00039  * Prototypes des methodes
00040  *============================================================================*/
00042     virtual void evolue(double dt);
00044     virtual void dessine() const;
00045
00046 /*============================================================================
00047  * Definition des attributs
00048  *============================================================================*/
00049     private:
00050     GLUquadric* sphere;
00051
00052 };
00053
```

5.23 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/GLFluor.cc

est la définition de la classe de la particule fluor en OpenGL qui a en plus une mémorisation et affichage de la trajectoire

`#include "GLFluor.h"`

5.23.1 Description détaillée

est la définition de la classe de la particule fluor en OpenGL qui a en plus une mémorisation et affichage de la trajectoire

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GLFluor.cc**.

5.24 GLFluor.cc

```
00001
00009 #include "GLFluor.h"
00010
00011 /*================================================================================
00012  * Definition des constructeurs et destructeur
00013  *================================================================================*/
00022 GLFluor::GLFluor(Vecteur position, Vecteur vitesse)
00023 :Particule(position, vitesse, 18.998, 0.42),  sphere(gluNewQuadric())
00024 {}
00034 GLFluor::GLFluor(Enceinte const& enceinte, GenerateurAleatoire& tirage, double temperature)
00035 :Particule(enceinte, tirage, temperature, 18.998, 0.42), sphere(gluNewQuadric())
00036 {}
00037
00038 GLFluor::~GLFluor()
00039 {
00040     gluDeleteQuadric(sphere);
00041 }
00042
00043 /*================================================================================
00044  * Definition des méthodes
00045  *================================================================================*/
00054 void GLFluor::evolue(double dt)
00055 {
00056     Particule::evolue(dt);
00057     enregistrerCoordonnee();
00058 }
00063 void GLFluor::enregistrerCoordonnee()
00064 {
00066     if(dequePositions.size()>150)
00067     {
00068         dequePositions.pop_front();
00069     }
00071     dequePositions.push_back(new Vecteur(this->getPosition().getX(), this->
      getPosition().getY(), this->getPosition().getZ()));
00072 }
00073
00080 void GLFluor::dessine() const
00081 {
00082
00083     glColor4d(0.6, 0.34, 0.9, 1);
00084     glBegin(GL_LINE_STRIP);
00086     if(!dequePositions.empty())
00087     {
00088         for(size_t i(0);i<dequePositions.size()-1;i++)
00089         {
00090             glVertex3f((*dequePositions[i]).getX(), (*dequePositions[i]).getY(), (*dequePositions[i]).getZ(
      ));
00091         }
00092     }
00093     glEnd();
00095     glColor4d(0.6, 0.34, 0.9, 1);
00096
00097     //ajouter le déplacement
00098     glPushMatrix();
00099     glTranslated(this->getPosition().getX(), this->getPosition().getY(), this->
      getPosition().getZ());
00100     gluSphere(sphere,rayon,10,10);
00101     glPopMatrix();
00102
00103 }
```

5.25 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/GLFluor.h

est le prototype de la classe de la particule fluor en OpenGL qui a en plus une mémorisation et affichage de la trajectoire

```
#include "wx/wxprec.h"
#include "wx/wx.h"
#include "wx/glcanvas.h"
#include <vector>
#include <deque>
#include "Vecteur.h"
#include "Particule.h"
```

Classes

— class **GLFluor**
Prototype de la classe GLluor.

5.25.1 Description détaillée

est le prototype de la classe de la particule fluor en OpenGL qui a en plus une mémorisation et affichage de la trajectoire

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GLFluor.h**.

5.26 GLFluor.h

```
00001
00010 #include "wx/wxprec.h"
00011 #ifndef WX_PRECOMP
00012 #include "wx/wx.h"
00013 #endif
00014 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00015
00016 #include <vector>
00017 #include <deque>
00018 #include "Vecteur.h"
00019 #include "Particule.h"
00020
00032 class GLFluor : public Particule
00033 {
00034     public:
00035 /*============================================================================
00036  * Prototyopes des constructeurs et destructeur
00037  *============================================================================*/
00039     GLFluor(Vecteur position, Vecteur vitesse);
00041     GLFluor(Enceinte const& enceinte, GenerateurAleatoire& tirage, double temperature);
00043     virtual ~GLFluor();
00044
00045 /*============================================================================
00046  * Prototypes des methodes
00047  *============================================================================*/
00049     virtual void evolue(double dt);
00051     virtual void dessine() const;
00053     void enregistrerCoordonnee();
00054
00055 /*============================================================================
00056  * Definition des attributs
00057  *============================================================================*/
00058     private:
```

```
00059     GLUquadric* sphere;
00060     std::deque<Vecteur*> dequePositions;
00061 };
00062
```

5.27 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/GLHelium.cc

est la définition de la classe de la particule Helium en OpenGL

`#include "GLHelium.h"`

5.27.1 Description détaillée

est la définition de la classe de la particule Helium en OpenGL

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GLHelium.cc**.

5.28 GLHelium.cc

```
00001
00009 #include "GLHelium.h"
00010
00011 /*================================================================================
00012  * Definition des constructeurs et destructeur
00013  *================================================================================*/
00020 GLHelium::GLHelium(Vecteur position, Vecteur vitesse)
00021 :Particule(position, vitesse, 4.002602, 0.31), sphere(gluNewQuadric())
00022 {}
00030 GLHelium::GLHelium(Enceinte const& enceinte, GenerateurAleatoire& tirage, double temperature)
00031 :Particule(enceinte, tirage, temperature, 4.002602, 0.31), sphere(gluNewQuadric())
00032 {}
00033
00034 GLHelium::~GLHelium()
00035 {
00036     gluDeleteQuadric(sphere);
00037 }
00038
00039 /*================================================================================
00040  * Definition des méthodes
00041  *================================================================================*/
00046 void GLHelium::evolue(double dt)
00047 {
00048     Particule::evolue(dt);
00049 }
00050
00055 void GLHelium::dessine() const
00056 {
00057     glColor4d(1.0, 0.5, 0.0, 1);
00058
00059     //ajouter le déplacement
00060     glPushMatrix();
00061
00062     glTranslated(this->getPosition().getX(), this->getPosition().getY(), this->
      getPosition().getZ());
00063
```

```
00064     gluSphere(sphere,rayon,10,10);
00065     glPopMatrix();
00066 }
```

5.29 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/GLHelium.h

est le prototype de la classe de la particule Helium en OpenGL

```
#include "wx/wxprec.h"
#include "wx/wx.h"
#include "wx/glcanvas.h"
#include "Particule.h"
```

Classes

— class **GLHelium**
*Prototype de la classe **GLHelium*** (p. *17*).

5.29.1 Description détaillée

est le prototype de la classe de la particule Helium en OpenGL

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GLHelium.h**.

5.30 GLHelium.h

```
00001
00009 #include "wx/wxprec.h"
00010 #ifndef WX_PRECOMP
00011 #include "wx/wx.h"
00012 #endif
00013 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00014
00015 #include "Particule.h"
00016
00023 class GLHelium : public Particule
00024 {
00025     public:
00026 /*============================================================================
00027  * Prototyopes des constructeurs et destructeur
00028  *============================================================================*/
00030     GLHelium(Vecteur position, Vecteur vitesse);
00032     GLHelium(Enceinte const& enceinte, GenerateurAleatoire& tirage, double temperature);
00034     virtual ~GLHelium();
00035
00036 /*============================================================================
00037  * Prototypes des methodes
00038  *============================================================================*/
00040     virtual void evolue(double dt);
00042     virtual void dessine() const;
```

```
00043
00044 /*================================================================================
00045  * Definition des attributs
00046  *================================================================================*/
00047     private:
00048     GLUquadric* sphere;
00049 };
00050
```

5.31 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/GLNeon.cc

est la définition de la classe de la particule Néon en OpenGL

`#include "GLNeon.h"`

5.31.1 Description détaillée

est la définition de la classe de la particule Néon en OpenGL

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GLNeon.cc**.

5.32 GLNeon.cc

```
00001
00009 #include "GLNeon.h"
00010
00011 /*================================================================================
00012  * Definition des constructeurs et destructeur
00013  *================================================================================*/
00020 GLNeon::GLNeon(Vecteur position, Vecteur vitesse)
00021 :Particule(position, vitesse, 20.1797, 0.38), sphere(gluNewQuadric())
00022 {}
00030 GLNeon::GLNeon(Enceinte const& enceinte, GenerateurAleatoire& tirage, double temperature)
00031 :Particule(enceinte, tirage, temperature, 20.1797, 0.38), sphere(gluNewQuadric())
00032 {}
00033
00034 GLNeon::~GLNeon()
00035 {
00036     gluDeleteQuadric(sphere);
00037 }
00038
00039 /*================================================================================
00040  * Definition des méthodes
00041  *================================================================================*/
00046 void GLNeon::evolue(double dt)
00047 {
00048     Particule::evolue(dt);
00049 }
00050
00055 void GLNeon::dessine() const
00056 {   glColor4d(1.0, 0.0, 0.0, 1);
00057
00058     //ajouter le déplacement
00059     glPushMatrix();
00060
```

```
00061     glTranslated(this->getPosition().getX(), this->getPosition().getY(), this->
      getPosition().getZ());
00062
00063     gluSphere(sphere,rayon,10,10);
00064     glPopMatrix();
00065 }
```

5.33 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/GLNeon.h

est le prototype de la classe de la particule néon en OpenGL

```
#include "wx/wxprec.h"
#include "wx/wx.h"
#include "wx/glcanvas.h"
#include "Particule.h"
```

Classes

— class **GLNeon**
*Prototype de la classe **GLNeon*** (p. *19*).

5.33.1 Description détaillée

est le prototype de la classe de la particule néon en OpenGL

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GLNeon.h**.

5.34 GLNeon.h

```
00001
00009 #include "wx/wxprec.h"
00010 #ifndef WX_PRECOMP
00011 #include "wx/wx.h"
00012 #endif
00013 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00014
00015 #include "Particule.h"
00016
00024 class GLNeon : public Particule
00025 {
00026     public:
00027 /*============================================================================
00028  * Prototyopes des constructeurs et destructeur
00029  *============================================================================*/
00031     GLNeon(Vecteur position, Vecteur vitesse);
00033     GLNeon(Enceinte const& enceinte, GenerateurAleatoire& tirage, double temperature);
00035     virtual ~GLNeon();
00036
00037 /*============================================================================
00038  * Prototypes des methodes
```

```
00039  *=============================================================================*/
00041     virtual void evolue(double dt);
00043     virtual void dessine() const;
00044
00045 /*=============================================================================
00046  * Definition des attributs
00047  *=============================================================================*/
00048     private:
00049     GLUquadric* sphere;
00050
00051 };
00052
```

5.35 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/GUI.cc

est la définition de l'application princpal qui lance tout le programme

```
#include "GUI.h"
#include "Fenetre.h"
```

5.35.1 Description détaillée

est la définition de l'application princpal qui lance tout le programme

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GUI.cc**.

5.36 GUI.cc

```
00001
00009 #include "GUI.h"
00010 #include "Fenetre.h"
00011
00012 /* ============================================================
00013    Implementation de l'application principale
00014    ============================================================ */
00015
00024 bool GUI::OnInit()
00025 {
00027     bool reglage(false);
00030     if (wxMessageBox(wxT("Voulez-vous paramétrer le système ?"), wxT("Lancement de la simulation"),
00031         wxYES_NO | wxICON_QUESTION)==wxYES)
00032     { reglage = true; }
00033
00034     Fenetre* f (nullptr);
00035
00036     if (reglage) { f = new Fenetre(wxT("Simulation temps reel"),
00037                                    reglage, wxSize(800, 600)); }
00038     else { f = new Fenetre(wxT("Simulation temps reel"),
00039                            wxSize(800, 600)); }
00040
00041     SetTopWindow(f);
00042     return (f != 0);
00043 }
00044
```

```
00045 /* ================================================================
00046    Equivalent de main()
00047    ================================================================ */
00049 IMPLEMENT_APP(GUI)
```

5.37 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/GUI.h

est le prototype de l'application princpal qui lance tout le programme

```
#include "wx/wx.h"
#include "wx/glcanvas.h"
```

Classes

— class **GUI**
Application principale.

5.37.1 Description détaillée

est le prototype de l'application princpal qui lance tout le programme

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mai 2014

Définition dans le fichier **GUI.h**.

5.38 GUI.h

```
00001
00009 #ifndef PRJ_GUI_H
00010 #define PRJ_GUI_H
00011
00012 #ifndef WX_PRECOMP
00013 #include "wx/wx.h"
00014 #endif
00015 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00016
00020 class GUI : public wxApp
00021 {
00022     public:
00024     bool OnInit();
00025
00026 };
00027 #endif
```

5.39 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Particule.cc

Définition de la classe mère **Particule** (p. 22).

```
#include "Particule.h"
#include <cmath>
```

Fonctions

— ostream & **operator**<< (ostream &sortie, **Particule** const &p)

5.39.1 Description détaillée

Définition de la classe mère **Particule** (p. 22).

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **Particule.cc**.

5.39.2 Documentation des fonctions

5.39.2.1 ostream& operator<< (ostream & *sortie,* Particule const & *p*)

Paramètres

p	**Particule** (p. 22) à afficher
sortie	ostream sur lequel on doit afficher p

Renvoie

ostream modifié par l'affichage de p

Définition à la ligne **160** du fichier **Particule.cc**.

5.40 Particule.cc

```
00001
00009 #include "Particule.h"
00010 #include <cmath>
00011
00012 using namespace std;
00013
00014 /*================================================================================
00015  * Definition des constructeurs
00016  *================================================================================*/
00018 Particule::Particule()
00019     : position(Vecteur(0,0,0)), vitesse(Vecteur(0,0,0)), masse(0), rayon(0) {}
00020
00027 Particule::Particule(Vecteur position, Vecteur vitesse, double masse, double rayon)
00028     : position(position), vitesse(vitesse), masse(masse), rayon(rayon) {}
00029
00033 Particule::Particule(Enceinte const& enceinte, GenerateurAleatoire& tirage, double temperature, double
      masse, double rayon)
00034     : position(Vecteur(tirage.uniforme_reel(0, enceinte.getLargeur()),
00035                        tirage.uniforme_reel(0, enceinte.getLongueur()),
00036                        tirage.uniforme_reel(0, enceinte.getHauteur()))),
00037       masse(masse), rayon(rayon)
```

```
00038 {
00039     double ecart_type (sqrt(8314.472 * temperature / masse));
00040     vitesse = Vecteur(tirage.gaussienne(0, ecart_type),
00041                       tirage.gaussienne(0, ecart_type),
00042                       tirage.gaussienne(0, ecart_type));
00043 }
00044
00045 /*============================================================================
00046  * Definition des méthodes
00047  *============================================================================*/
00051 Vecteur Particule::getPosition() const
00052 {
00053     return position;
00054 }
00057 ostream& Particule::afficher(ostream& sortie) const
00058 {
00059     return sortie << "pos : " << position << " ; v : " << vitesse
00060                   << " ; m : " << masse;
00061 }
00062
00064 void Particule::evolue(double dt)
00065 {
00066     position += vitesse * dt;
00067 }
00068
00073 void Particule::gere_sorties(Enceinte const& enceinte)
00074 {
00075     // Par rapport a X
00076     while (position.getX() - rayon < 0 or position.getX() + rayon > enceinte.
      getLargeur()) {
00077         if (position.getX() - rayon < 0)
00078         {
00079             position = Vecteur (-(position.getX() - rayon) + rayon, position.
      getY(), position.getZ());
00080             vitesse = Vecteur (-vitesse.getX(), vitesse.getY(), vitesse.getZ());
00081             // Rebond sur la face 1
00082         } else
00083         {
00084             position = Vecteur (enceinte.getLargeur() - (position.getX() + rayon - enceinte.
      getLargeur()) - rayon, position.getY(), position.getZ());
00085             vitesse = Vecteur (-vitesse.getX(), vitesse.getY(), vitesse.getZ());
00086             // Rebond sur la face 2
00087         }
00088     }
00089
00090     // Par rapport a Y
00091     while (position.getY() - rayon < 0 or position.getY() + rayon > enceinte.
      getLongueur()) {
00092         if (position.getY() - rayon < 0)
00093         {
00094             position = Vecteur (position.getX(), -(position.getY() - rayon) +
      rayon, position.getZ());
00095             vitesse = Vecteur (vitesse.getX(), -vitesse.getY(), vitesse.getZ());
00096             // Rebond sur la face 3
00097         } else
00098         {
00099             position = Vecteur (position.getX(), enceinte.getLongueur() - (
      position.getY() + rayon - enceinte.getLongueur()) - rayon, position.getZ());
00100             vitesse = Vecteur (vitesse.getX(), -vitesse.getY(), vitesse.getZ());
00101             // Rebond sur la face 4
00102         }
00103     }
00104
00105     // Par rapport a Z
00106     while (position.getZ() - rayon < 0 or position.getZ() + rayon > enceinte.
      getHauteur()) {
00107         if (position.getZ() - rayon < 0)
00108         {
00109             position = Vecteur (position.getX(), position.getY(), -(position.
      getZ() - rayon) + rayon);
00110             vitesse = Vecteur (vitesse.getX(), vitesse.getY(), -vitesse.getZ());
00111             // Rebond sur la face 5
00112         } else
00113         {
00114             position = Vecteur (position.getX(), position.getY(), enceinte.
      getHauteur() - (position.getZ() + rayon - enceinte.getHauteur()) - rayon);
00115             vitesse = Vecteur (vitesse.getX(), vitesse.getY(), -vitesse.getZ());
00116             // Rebond sur la face 6
00117         }
00118     }
00119 }
00120
00124 Vecteur Particule::pavageCubique(double epsilon) const
00125 { return Vecteur (floor(position.getX()*(1/epsilon)),
00126                   floor(position.getY()*(1/epsilon)),
00127                   floor(position.getZ()*(1/epsilon))); }
00128
```

```
00132 void Particule::collision(Particule& p, GenerateurAleatoire& tirage)
00133 {
00134     Vecteur vg (masse/(masse + p.masse) * vitesse + p.masse/(masse + p.masse) * p.
      vitesse);
00135
00136     double L (sqrt((vitesse-vg) * (vitesse-vg))); // racine carrée du produit scalaire d'un vecteur avec
       lui-même = norme de ce vecteur
00137
00138     double z(tirage.uniforme_reel(-L, L));
00139     double phi(tirage.uniforme_reel(0, 2 * M_PI));
00140
00141     double r (sqrt(L * L - z * z));
00142     Vecteur v0 (r * cos(phi), r * sin(phi), z);
00143
00144     vitesse = vg + v0;
00145     p.vitesse = vg - (masse/p.masse) * v0;
00146 }
00147
00150 double Particule::get_rayon()
00151 { return rayon; }
00152
00153 /*================================================================================
00154  * Definition des fonctions
00155  *================================================================================*/
00156
00160 ostream& operator<<(ostream& sortie, Particule const& p)
00161 {
00162     return p.afficher(sortie);
00163 }
00164
```

5.41 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Particule.h

Prototype de la classe mère **Particule** (p. 22).

```
#include "Vecteur.h"
#include "Dessinable.h"
#include "Enceinte.h"
#include "GenerateurAleatoire.h"
```

Classes

— class **Particule**
Classe mère dont hérite toutes les classes de type : TXTNom et GLNom Représentation d'une particule (ici version graphique)

Fonctions

— std : :ostream & **operator**<< (std : :ostream &sortie, **Particule** const &p)
Surcharge de l'opérateur d'affichage.

5.41.1 Description détaillée

Prototype de la classe mère **Particule** (p. 22).

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **Particule.h**.

5.41.2 Documentation des fonctions

5.41.2.1 std : :ostream& operator<< (std : :ostream & *sortie,* Particule const & *p*)

Surcharge de l'opérateur d'affichage.

5.42 Particule.h

```
00001
00009 #ifndef PRJ_PARTICULE_H
00010 #define PRJ_PARTICULE_H
00011
00012 #include "Vecteur.h"
00013 #include "Dessinable.h"
00014 #include "Enceinte.h"
00015 #include "GenerateurAleatoire.h"
00016
00027 class Particule : public Dessinable
00028 {
00029 /*============================================================================
00030  * Definition des attributs
00031  *============================================================================*/
00032     protected :
00034     Vecteur position;
00036     Vecteur vitesse;
00038     double const masse;
00040     double const rayon;
00041
00042 /*============================================================================
00043  * Prototypage des constructeurs
00044  *============================================================================*/
00045     public :
00047     Particule();
00049     Particule(Vecteur position, Vecteur vitesse, double masse, double rayon);
00051     Particule(Enceinte const& enceinte, GenerateurAleatoire& tirage, double temperature, double
      masse, double rayon);
00052
00053 /*============================================================================
00054  * Prototypage des methodes
00055  *============================================================================*/
00057     Vecteur getPosition() const;
00059     std::ostream& afficher(std::ostream& sortie) const;
00061     virtual void evolue(double dt);
00063     void gere_sorties(Enceinte const& enceinte);
00065     Vecteur pavageCubique(double espsilon) const;
00067     void collision(Particule& p, GenerateurAleatoire& tirage);
00069     double get_rayon();
00070
00071 };
00072
00074 std::ostream& operator<<(std::ostream& sortie, Particule const& p);
00075
00076 #endif // PRJ_PARTICULE_H
```

5.43 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Systeme.cc

Définition de la classe **Systeme** (p. 24) (objet formé d'une enceinte et de particules)

```
#include "Systeme.h"
#include "GLNeon.h"
#include "GLHelium.h"
#include "GLArgon.h"
#include "GLFluor.h"
#include "wx/wx.h"
#include "wx/glcanvas.h"
```

5.43.1 Description détaillée

Définition de la classe **Systeme** (p. 24) (objet formé d'une enceinte et de particules)

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **Systeme.cc**.

5.44 Systeme.cc

```
00001
00009 #include "Systeme.h"
00010 #include "GLNeon.h"
00011 #include "GLHelium.h"
00012 #include "GLArgon.h"
00013 #include "GLFluor.h"
00014
00015 #ifndef WX_PRECOMP
00016 #include "wx/wx.h"
00017 #endif
00018 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00019
00020 using namespace std;
00021
00022 /*================================================================================
00023  * Definition des constructeurs et du destructeur
00024  *================================================================================*/
00025
00027 Systeme::Systeme()
00028  : temperature(298) // Température normale : 25°C
00029 { initialise(); }
00030
00037 Systeme::Systeme(bool reglage)
00038  : enceinte(reglage), temperature(298)
00039 {
00040     do
00041     {
00042         temperature = wxAtoi(wxGetTextFromUser(wxT("Rentrez une température (entre 0 et 400 Kelvin)"), wxT(
      "Température")));
00043     } while (temperature <= 0 or temperature > 400);
00044
00045     double nbrArgon(0);
00046     do
00047     {
00048         nbrArgon = wxAtoi(wxGetTextFromUser(wxT("Rentrez le nombre de particules d'Argon (entre 0 et 425,
       Attention : ne pas créer plus de 425 particules en tout)"), wxT("Paramétrage du nombre de particules")));
00049     } while (nbrArgon < 0 or nbrArgon > 425);
00050
00051     double nbrFluor(0);
00052     do
00053     {
00054         nbrFluor = wxAtoi(wxGetTextFromUser(wxT("Rentrez le nombre de particules de Fluor (entre 0 et 3,
       Attention : ne pas créer plus de 425 particules en tout)"), wxT("Paramétrage du nombre de particules")));
```

```
00055     } while (nbrFluor < 0 or nbrFluor > 3);
00056
00057     double nbrHelium(0);
00058     do
00059     {
00060         nbrHelium = wxAtoi(wxGetTextFromUser(wxT("Rentrez le nombre de particules d'Helium (entre 0 et 425,
      Attention : ne pas créer plus de 425 particules en tout)"), wxT("Paramétrage du nombre de particules")));
00061     } while (nbrHelium < 0 or nbrHelium > 425);
00062
00063     double nbrNeon(0);
00064     do
00065     {
00066         nbrNeon = wxAtoi(wxGetTextFromUser(wxT("Rentrez le nombre de particules de Néon (entre 0 et 425,
      Attention : ne pas créer plus de 425 particules en tout)"), wxT("Paramétrage du nombre de particules")));
00067     } while (nbrNeon < 0 or nbrNeon > 425);
00068
00069     initialise(nbrArgon,nbrFluor,nbrHelium,nbrNeon);
00070 }
00071
00079 Systeme::Systeme(double largeur, double longueur, double hauteur, double temperature)
00080  : enceinte(largeur, longueur, hauteur), temperature(temperature)
00081 { initialise(); }
00082
00084 Systeme::~Systeme()
00085 { supprimerParticules(); }
00086
00087 /*============================================================================
00088  * Definition des methodes
00089  *============================================================================*/
00090
00092 void Systeme::ajouterParticule(Particule* particule)
00093 {
00094     if(particule != nullptr)
00095     { collectionParticules.push_back(unique_ptr<Particule>(particule)); }
00096 }
00097
00099 void Systeme::dessine() const
00100 {
00101     glPushMatrix();
00102     glTranslated(-enceinte.getLargeur()/2,-enceinte.getLongueur()/2,-enceinte.
      getHauteur()/2);
00103     enceinte.dessine();
00104     canon.dessine();
00105
00106     for(auto const& particule : collectionParticules)
00107     { particule->dessine(); }
00108     glPopMatrix();
00109 }
00110
00114 bool Systeme::supprimerParticules()
00115 {
00116     for(auto& particule : collectionParticules)
00117     { particule.reset(); }
00118     return true;
00119 }
00120
00126 void Systeme::evolue(double dt)
00127 {
00128     for (auto const& particule : collectionParticules)
00129     {
00130         particule->evolue(dt);
00131
00132         particule->gere_sorties(enceinte);
00133
00134         vector<size_t> collisions_possibles (determine_collisions_possibles(particule));
00135         if (collisions_possibles.size() != 0)
00136         { particule->collision(*collectionParticules[choisi_collision(collisions_possibles)], tirage); }
00137     }
00138 }
00139
00143 vector<size_t> Systeme::determine_collisions_possibles(unique_ptr<Particule> const& p1)
00144 {
00145     vector<size_t> tableau;
00146     for(size_t i(0);i<collectionParticules.size();++i)
00147     {
00148         if((p1 != collectionParticules[i]) and (p1->pavageCubique(epsilon) == collectionParticules[i]->
      pavageCubique(epsilon)))
00149         { tableau.push_back(i); }
00150     }
00151     return tableau;
00152 }
00153
00156 size_t Systeme::choisi_collision(vector<size_t> const& collisions_possibles)
00157 { return collisions_possibles[tirage.uniforme_entier(0, collisions_possibles.size() - 1)]; }
00158   //le tirage uniforme d'ENTIER se faisant sur un intervalle fermé [a, b], il faut donc tirer sur [0,
      collisions_possibles.size() - 1] pour que ce soit juste
00159
```

```
00164 void Systeme::ajouterArgon(unsigned int nbr_argon, bool canon)
00165 {
00166     GLArgon argon(Vecteur(0,0,0), Vecteur(0,0,0));
00167     double rayon_argon(argon.get_rayon());
00168     if (nbr_argon != 0) { change_epsilon(rayon_argon); }
00169     if (canon) {
00170         for(int i(1) ; i <= nbr_argon ; ++i)
00171         {
00172             double pas(i*rayon_argon);
00173             ajouterParticule(new GLArgon(Vecteur(pas,pas,pas), Vecteur(250,250,250)));
00174         }
00175     } else {
00176         for(int i(1) ; i <= nbr_argon ; ++i)
00177         { ajouterParticule(new GLArgon(enceinte, tirage, temperature)); }
00178     }
00179 }
00180
00185 void Systeme::ajouterFluor(unsigned int nbr_fluor, bool canon)
00186 {
00187     GLFluor fluor(Vecteur(0,0,0), Vecteur(0,0,0));
00188     double rayon_fluor(fluor.get_rayon());
00189     if (nbr_fluor != 0) { change_epsilon(rayon_fluor); }
00190     if (canon) {
00191         for(int i(1) ; i <= nbr_fluor ; ++i)
00192         {
00193             double pas(i*rayon_fluor);
00194             ajouterParticule(new GLFluor(Vecteur(pas,pas,pas), Vecteur(250,250,250)));
00195         }
00196     } else {
00197         for(int i(1) ; i <= nbr_fluor ; ++i)
00198         { ajouterParticule(new GLFluor(enceinte, tirage, temperature)); }
00199     }
00200 }
00201
00206 void Systeme::ajouterHelium(unsigned int nbr_helium, bool canon)
00207 {
00208     GLHelium helium(Vecteur(0,0,0), Vecteur(0,0,0));
00209     double rayon_helium(helium.get_rayon());
00210     if (nbr_helium != 0) { change_epsilon(rayon_helium); }
00211     if (canon) {
00212         for(int i(1) ; i <= nbr_helium ; ++i)
00213         {
00214             double pas(i*rayon_helium);
00215             ajouterParticule(new GLHelium(Vecteur(pas,pas,pas), Vecteur(250,250,250)));
00216         }
00217     } else {
00218         for(int i(1) ; i <= nbr_helium ; ++i)
00219         { ajouterParticule(new GLHelium(enceinte, tirage, temperature)); }
00220     }
00221 }
00222
00227 void Systeme::ajouterNeon(unsigned int nbr_neon, bool canon)
00228 {
00229     GLNeon neon(Vecteur(0,0,0), Vecteur(0,0,0));
00230     double rayon_neon(neon.get_rayon());
00231     if (nbr_neon != 0) { change_epsilon(rayon_neon); }
00232     if (canon) {
00233         for(int i(1) ; i <= nbr_neon ; ++i)
00234         {
00235             double pas(i*rayon_neon);
00236             ajouterParticule(new GLNeon(Vecteur(pas,pas,pas), Vecteur(250,250,250)));
00237         }
00238     } else {
00239         for(int i(1) ; i <= nbr_neon ; ++i)
00240         { ajouterParticule(new GLNeon(enceinte, tirage, temperature)); }
00241     }
00242 }
00243
00248 void Systeme::initialise(unsigned int nbr_argon, unsigned int nbr_fluor,
00249                          unsigned int nbr_helium, unsigned int nbr_neon)
00250 {
00251     epsilon = enceinte.getLargeur();
00252     ajouterArgon(nbr_argon);
00253     ajouterFluor(nbr_fluor);
00254     ajouterHelium(nbr_helium);
00255     ajouterNeon(nbr_neon);
00256 }
00257
00259 void Systeme::initialise()
00260 {
00261     initialise(tirage.uniforme_entier(0, 75),
00262                tirage.uniforme_entier(0, 3),
00263                tirage.uniforme_entier(0, 100),
00264                tirage.uniforme_entier(0, 75));
00265 }
00266
00269 void Systeme::change_epsilon(double rayon)
```

```
00270 { if (epsilon > rayon) { epsilon = rayon; }}
```

5.45 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/Simulation d'un gaz parfait/Systeme.h

Prototype de la classe de la classe **Systeme** (p. 24) (objet formé d'une enceinte et de particules)

```
#include <memory>
#include <vector>
#include "Particule.h"
#include "Enceinte.h"
#include "Canon.h"
```

Classes

— class **Systeme**
*Permet de créer, de dessiner et de faire évoluer des objets de type **Systeme** (p. 24) formés d'une enceinte et de particules.*

5.45.1 Description détaillée

Prototype de la classe de la classe **Systeme** (p. 24) (objet formé d'une enceinte et de particules)

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **Systeme.h**.

5.46 Systeme.h

```
00001
00009 #ifndef PRJ_SYSTEME_H
00010 #define PRJ_SYSTEME_H
00011
00012 #include <memory>
00013 #include <vector>
00014
00015 #include "Particule.h"
00016 #include "Enceinte.h"
00017 #include "Canon.h"
00018
00023 class Systeme : public Dessinable
00024 {
00025 /*============================================================================
00026  * Definition des attributs
00027  *============================================================================*/
00028     private :
00030
00031     std::vector<std::unique_ptr<Particule>> collectionParticules;
00033     Enceinte enceinte;
00035
00037     double temperature;
00039     double epsilon;
```

```
00041     GenerateurAleatoire tirage;
00043
00044     Canon canon;
00045
00046 /*================================================================================
00047  * Prototypes des constructeurs et du destructeur
00048  *================================================================================*/
00049     public :
00051     Systeme();
00053     Systeme(bool reglage);
00055     Systeme(double largeur, double longueur, double hauteur, double temperature);
00056     ~Systeme();
00058
00059     private :
00060     Systeme(Systeme const& autre) = delete; //suppression du constructeur de copie
00061
00062
00063 /*================================================================================
00064  * Prototypes des methodes
00065  *================================================================================*/
00066     public :
00068     void ajouterArgon(unsigned int nbr_argon, bool canon = false);
00070     void ajouterFluor(unsigned int nbr_fluor, bool canon = false);
00072     void ajouterHelium(unsigned int nbr_helium, bool canon = false);
00074     void ajouterNeon(unsigned int nbr_neon, bool canon = false);
00076     void dessine() const override;
00078     void evolue(double dt);
00079
00080     private :
00082     void ajouterParticule(Particule* particule);
00084     bool supprimerParticules();
00085
00086     Systeme& operator=(Systeme aCopier) = delete; //suppression de l'operateur = pour un systemme
00087
00089     std::vector<size_t> determine_collisions_possibles(std::unique_ptr<Particule> const& p1);
00092     size_t choisi_collision(std::vector<size_t> const& collisions_possibles);
00094     void initialise(unsigned int nbr_argon, unsigned int nbr_fluor, unsigned int nbr_helium, unsigned int
      nbr_neon);
00096     void initialise();
00098     void change_epsilon(double rayon);
00099 };
00100
00101 #endif // PRJ_SYSTEME_H
```

5.47 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/TXTArgon.cc

Définition de la classe de la particule Argon en version texte.

```
#include "TXTArgon.h"
```

5.47.1 Description détaillée

Définition de la classe de la particule Argon en version texte.

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **TXTArgon.cc**.

5.48 TXTArgon.cc

```
00001
00009 #include "TXTArgon.h"
00010
00017 TXTArgon::TXTArgon(Vecteur position, Vecteur vitesse)
00018  : Particule(position, vitesse, 39.948) {}
00019
00022 void TXTArgon::dessine() const
00023 {
00024     std::cout << "particule TXTAr : " << *this << endl;
00025 }
```

5.49 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/TXTArgon.h

Prototype de la classe de la particule Argon en version texte.

```
#include "Particule.h"
```

Classes

— class **TXTArgon**

5.49.1 Description détaillée

Prototype de la classe de la particule Argon en version texte.

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **TXTArgon.h**.

5.50 TXTArgon.h

```
00001
00009 #include "Particule.h"
00010
00018 //???? est-ce possible ?
00019
00020 class TXTArgon : public Particule
00021 {
00022     public :
00023
00025     TXTArgon(Vecteur position, Vecteur vitesse);
00026
00028     void dessine() const override;
00029 };
```

5.51 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/TXTHelium.cc

Définition de la classe de la particule Helium en version texte.

`#include "TXTHelium.h"`

5.51.1 Description détaillée

Définition de la classe de la particule Helium en version texte.

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **TXTHelium.cc**.

5.52 TXTHelium.cc

```
00001
00009 #include "TXTHelium.h"
00010
00017 TXTHelium::TXTHelium(Vecteur position, Vecteur vitesse)
00018  : Particule(position, vitesse, 4.002602) {}
00019
00022 void TXTHelium::dessine() const
00023 {
00024     std::cout << "particule TXHe : " << *this << endl;
00025 }
```

5.53 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/TXTHelium.h

Prototype de la classe de la particule Helium en version texte.

`#include "Particule.h"`

Classes

— class **TXTHelium**

5.53.1 Description détaillée

Prototype de la classe de la particule Helium en version texte.

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **TXTHelium.h**.

5.54 TXTHelium.h

```
00001
00009 #include "Particule.h"
00010
00018 class TXTHelium : public Particule
00019 {
00020     public :
00021
00023     TXTHelium(Vecteur position, Vecteur vitesse);
00024
00026     void dessine() const override;
00027 };
```

5.55 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/TXTNeon.cc

Définition de la classe de la particule Neon en version texte.

`#include "TXTNeon.h"`

5.55.1 Description détaillée

Définition de la classe de la particule Neon en version texte.

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

avril 2014

Définition dans le fichier **TXTNeon.cc**.

5.56 TXTNeon.cc

```
00001
00009 #include "TXTNeon.h"
00010
00017 TXTNeon::TXTNeon(Vecteur position, Vecteur vitesse)
00018  : Particule(position, vitesse, 20.1797) {}
00019
00022 void TXTNeon::dessine() const
00023 {
00024     std::cout << "particule TXTNe : " << *this << endl;
00025 }
```

5.57 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/TXTNeon.h

```
#include "Particule.h"
```

Classes

— class **TXTNeon**

5.58 TXTNeon.h

```
00001
00009 #include "Particule.h"
00010
00018 class TXTNeon : public Particule
00019 {
00020     public :
00021
00023     TXTNeon(Vecteur position, Vecteur vitesse);
00024
00026     void dessine() const override;
00027 };
```

5.59 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Vecteur.cc

est la définition de la classe qui nous pourmet de gérer la position et la vitesse de nos particules mais aussi de tous l'espace qui les entours.

```
#include "Vecteur.h"
```

Fonctions

— ostream & **operator**<< (ostream &sortie, **Vecteur** const &v1)
— const **Vecteur operator+** (**Vecteur** v1, **Vecteur** const &v2)
Prototype fonction + entre deux vecteurs.
— const **Vecteur operator-** (**Vecteur** v1, **Vecteur** const &v2)
Prototype fonction - soustraction entre deux vecteurs.
— const **Vecteur operator**∗ (double scalaire, **Vecteur** v1)
fonction qui surcharge l'opérateur ∗
— const **Vecteur operator**∗ (**Vecteur** v1, double scalaire)
Prototype fonction multiplication par un scalire arrière.
— const **Vecteur operator**^ (**Vecteur** v1, **Vecteur** const &v2)
Prototype fonction ^ *prduit vectoriel.*

5.59.1 Description détaillée

est la définition de la classe qui nous pourmet de gérer la position et la vitesse de nos particules mais aussi de tous l'espace qui les entours.

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mars 2014

Définition dans le fichier **Vecteur.cc**.

5.59.2 Documentation des fonctions

5.59.2.1 const Vecteur operator∗ (double *scalaire,* Vecteur *v1*)

fonction qui surcharge l'opérateur ∗

Prototype fonction multiplication par un scalire avant.

Paramètres

double	scalaire : pour la multiplaction avant le vecteur
Vecteur *(p. 29)*	v1 : sert pour la multiplaction par le scalaire

Renvoie

retourne le vecteur modifier

Définition à la ligne **228** du fichier **Vecteur.cc**.

5.59.2.2 const Vecteur operator∗ (Vecteur *v1,* double *scalaire*)

Prototype fonction multiplication par un scalire arrière.

Paramètres

Vecteur *(p. 29)*	v1 : sert pour la multiplaction par le scalaire
double	scalaire : pour la multiplaction avant le vecteur

Renvoie

retourne le vecteur modifier

Définition à la ligne **239** du fichier **Vecteur.cc**.

5.59.2.3 const Vecteur operator+ (Vecteur *v1,* Vecteur const & *v2*)

Prototype fonction + entre deux vecteurs.

Paramètres

Vecteur *(p. 29)*	v1 : sert pour l'addition, on prends la copie pour ne pas changer le vecteur 1
Vecteur *(p. 29)*	const& v2 : sert pour l'addition

Renvoie

retourne le vecteur modifier

Définition à la ligne **204** du fichier **Vecteur.cc**.

5.59.2.4 const Vecteur operator- (Vecteur *v1,* Vecteur const & *v2*)

Prototype fonction - soustraction entre deux vecteurs.

Paramètres

Vecteur *(p. 29)*	v1 : sert pour la soustraction, on prends la copie pour ne pas changer le vecteur 1
Vecteur *(p. 29)*	const& v2 : sert pour la soustraction

Renvoie

retourne le vecteur modifier

Définition à la ligne **215** du fichier **Vecteur.cc**.

5.59.2.5 ostream& operator<< (ostream & *sortie,* Vecteur const & *v1*)

fonction qui permet de prendre un ostream& et va permettre de "multiplier" l'affichage des vecteurs facillement en surchargeant l'opérateur <<

Paramètres

ostream&	sortie : permet de choisir le flot de sortie
Vecteur *(p. 29)*	const& v1 : le vecteur que l'on veut donner a ostream

Renvoie

un ostream& pour pouvoir faire des "appels multiples"

Définition à la ligne **190** du fichier **Vecteur.cc**.

5.59.2.6 const Vecteur operator^ (Vecteur *v1,* Vecteur const & *v2*)

Prototype fonction ^ prduit vectoriel.

pour le produit vectoriel de deux vecteurs

Paramètres

Vecteur *(p. 29)*	v1 : sert pour la multiplaction par le scalaire
Vecteur *(p. 29)*	const& v2 : on ne modifie pas le vecteur mais on le veut comme référence

Renvoie

retourne le vecteur modifier

Définition à la ligne **252** du fichier **Vecteur.cc**.

5.60 Vecteur.cc

```
00001
00009 #include "Vecteur.h"
00010
00011 using namespace std;
00012
00013 /*============================================================================
00014  * Definition des constructeurs
00015  *============================================================================*/
00020 Vecteur::Vecteur()
00021     : x(0), y(0), z(0) {}
00022
00030 Vecteur::Vecteur(double x, double y, double z)
00031     : x(x), y(y), z(z) {}
00032
00033 /*============================================================================
00034  * Definition des méthodes
00035  *============================================================================*/
00041 double Vecteur::getX() const
00042 {
00043     return x;
```

```
00044 }
00050 double Vecteur::getY() const
00051 {
00052     return y;
00053 }
00059 double Vecteur::getZ() const
00060 {
00061     return z;
00062 }
00063
00068 bool Vecteur::operator==(Vecteur const& v) const
00069 {
00070     if (x == v.x and y == v.y and z == v.z)
00071     {
00072         return true;
00073     }else
00074     {
00075         return false;
00076     }
00077 }
00078
00084 bool Vecteur::operator!=(Vecteur const& v) const
00085 {
00086     return not(*this == v);
00087 }
00088
00095 Vecteur& Vecteur::operator+=(Vecteur const& v1)
00096 {
00097     x += v1.x;
00098     y += v1.y;
00099     z += v1.z;
00100     return *this;
00101 }
00102
00109 Vecteur& Vecteur::operator-=(Vecteur const& v1)
00110 {
00111     x -= v1.x;
00112     y -= v1.y;
00113     z -= v1.z;
00114     return *this;
00115 }
00116
00122 const Vecteur Vecteur::operator-()
00123 {
00124     return ((*this)*= (-1.));
00125 }
00126
00133 Vecteur& Vecteur::operator*=(double const& scalaire)
00134 {
00135     x *= scalaire;
00136     y *= scalaire;
00137     z *= scalaire;
00138     return *this;
00139 }
00140
00148 Vecteur& Vecteur::operator^=(Vecteur const& v1)
00149 {
00150     double x_(x), y_(y);
00151     x = (y * v1.z - z * v1.y);
00152     y = (z * v1.x - x_ * v1.z);
00153     z = (x_ * v1.y - y_ * v1.x);
00154     return *this;
00155 }
00156
00163 double Vecteur::operator*(const Vecteur& v1) const
00164 {
00165     return (v1.x*x + v1.y*y + v1.z*z);
00166 }
00167
00172 ostream& Vecteur::afficher(ostream& sortie) const
00173 {
00174     return sortie << x << " " << y << " " << z;
00175 }
00176
00177 /*============================================================================
00178  * Definition des fonctions
00179  *============================================================================*/
00180
00181
00190 ostream& operator<<(ostream& sortie, Vecteur const& v1)
00191 {
00192     return v1.afficher(sortie);
00193 }
00194
00195 /* Les méthodes suivantes sont écrites sur 2 lignes pour éviter la copie
00196  * inutile faîtes par certain compilateurs
00197  */
```

```
00198
00204 const Vecteur operator+(Vecteur v1, Vecteur const& v2)
00205 {
00206     v1 += v2;
00207     return v1;
00208 }
00209
00215 const Vecteur operator-(Vecteur v1, Vecteur const& v2)
00216 {
00217     v1 -= v2;
00218     return v1;
00219 }
00220
00228 const Vecteur operator*(double scalaire, Vecteur v1)
00229 {
00230     v1 *= scalaire;
00231     return v1;
00232 }
00233
00239 const Vecteur operator*(Vecteur v1, double scalaire)
00240 {
00241     v1 *= scalaire;
00242     return v1;
00243 }
00244
00252 const Vecteur operator^(Vecteur v1, Vecteur const& v2)
00253 {
00254     v1 ^= v2;
00255     return v1;
00256 }
```

5.61 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Vecteur.h

est le prototype de la classe qui nous pourmet de gérer la position et la vitesse de nos particules mais aussi de tous l'espace qui les entours.

```
#include <iostream>
```

Classes

- class **Vecteur**
  *Prototype de la classe **Vecteur*** (p. *29*).

Fonctions

- std : :ostream & **operator**<< (std : :ostream &sortie, **Vecteur** const &v1)
  Prototype fonction <<.
- const **Vecteur operator+** (**Vecteur** v1, **Vecteur** const &v2)
  Prototype fonction + entre deux vecteurs.
- const **Vecteur operator-** (**Vecteur** v1, **Vecteur** const &v2)
  Prototype fonction - soustraction entre deux vecteurs.
- const **Vecteur operator**∗ (double scalaire, **Vecteur** v1)
  Prototype fonction multiplication par un scalire avant.
- const **Vecteur operator**∗ (**Vecteur** v1, double scalaire)
  Prototype fonction multiplication par un scalire arrière.
- const **Vecteur operator**^ (**Vecteur** v1, **Vecteur** const &v2)
  Prototype fonction ^ prduit vectoriel.

5.61.1 Description détaillée

est le prototype de la classe qui nous pourmet de gérer la position et la vitesse de nos particules mais aussi de tous l'espace qui les entours.

Auteur

Emma Geoffray et Jonathan Burkhard

Version

1.0

Date

mars 2014

Définition dans le fichier **Vecteur.h**.

5.61.2 Documentation des fonctions

5.61.2.1 const Vecteur operator∗ (double *scalaire,* Vecteur *v1*)

Prototype fonction multiplication par un scalire avant.

Prototype fonction multiplication par un scalire avant.

Paramètres

double	scalaire : pour la multiplaction avant le vecteur
Vecteur *(p. 29)*	v1 : sert pour la multiplaction par le scalaire

Renvoie

retourne le vecteur modifier

Définition à la ligne **228** du fichier **Vecteur.cc**.

5.61.2.2 const Vecteur operator∗ (Vecteur *v1,* double *scalaire*)

Prototype fonction multiplication par un scalire arrière.

Paramètres

Vecteur *(p. 29)*	v1 : sert pour la multiplaction par le scalaire
double	scalaire : pour la multiplaction avant le vecteur

Renvoie

retourne le vecteur modifier

Définition à la ligne **239** du fichier **Vecteur.cc**.

5.61.2.3 const Vecteur operator+ (Vecteur *v1,* Vecteur const & *v2*)

Prototype fonction + entre deux vecteurs.

Paramètres

Vecteur *(p. 29)*	v1 : sert pour l'addition, on prends la copie pour ne pas changer le vecteur 1
Vecteur *(p. 29)*	const& v2 : sert pour l'addition

Renvoie

retourne le vecteur modifier

Définition à la ligne **204** du fichier **Vecteur.cc**.

5.61.2.4 const Vecteur operator- (Vecteur *v1*, Vecteur const & *v2*)

Prototype fonction - soustraction entre deux vecteurs.

Paramètres

Vecteur *(p. 29)*	v1 : sert pour la soustraction, on prends la copie pour ne pas changer le vecteur 1
Vecteur *(p. 29)*	const& v2 : sert pour la soustraction

Renvoie

retourne le vecteur modifier

Définition à la ligne **215** du fichier **Vecteur.cc**.

5.61.2.5 std : :ostream& operator<< (std : :ostream & *sortie,* Vecteur const & *v1*)

Prototype fonction <<.

5.61.2.6 const Vecteur operator^ (Vecteur *v1,* Vecteur const & *v2*)

Prototype fonction ^ prduit vectoriel.

pour le produit vectoriel de deux vecteurs

Paramètres

Vecteur *(p. 29)*	v1 : sert pour la multiplaction par le scalaire
Vecteur *(p. 29)*	const& v2 : on ne modifie pas le vecteur mais on le veut comme référence

Renvoie

retourne le vecteur modifier

Définition à la ligne **252** du fichier **Vecteur.cc**.

5.62 Vecteur.h

```
00001
00010 #ifndef PRJ_VECTEUR_H
00011 #define PRJ_VECTEUR_H
00012
00013 #include <iostream>
00014
00015
00023 class Vecteur
00024 {
00025     private :
00026 /*============================================================================
00027  * Definition des attributs
00028  *===========================================================================*/
00029     double x;
00030     double y;
00031     double z;
00032
00033     public:
00034 /*============================================================================
00035  * Prototyopes des constructeurs
00036  *===========================================================================*/
00038     Vecteur();
00040     Vecteur(double x, double y, double z);
00041
00042 /*============================================================================
00043  * Prototypes des methodes
00044  *===========================================================================*/
00046     double getX() const;
00048     double getY() const;
00050     double getZ() const;
00052     bool operator==(Vecteur const&) const;
00054     bool operator!=(Vecteur const& v) const;
00056     Vecteur& operator+=(Vecteur const& v1);
00058     Vecteur& operator-=(Vecteur const& v1);
00060     Vecteur& operator*=(double const& scalaire);
00062     double operator*(const Vecteur& v1) const;
```

```
00064     Vecteur& operator^=(Vecteur const& v1);// ATTENTION XOR a une plus grande priorité à cause du ou
       exclusif donc toujours mettre des parentaises entre !!!!!
00066     const Vecteur operator-();
00068     std::ostream& afficher(std::ostream& sortie) const;
00069
00070 };
00071
00072 /*============================================================================
00073  * Prototypes des fonctions
00074  *============================================================================*/
00076 std::ostream& operator<<(std::ostream& sortie, Vecteur const& v1);
00078 const Vecteur operator+(Vecteur v1, Vecteur const& v2);
00080 const Vecteur operator-(Vecteur v1, Vecteur const& v2);
00082 const Vecteur operator*(double scalaire, Vecteur v1);
00084 const Vecteur operator*(Vecteur v1, double scalaire);
00086 const Vecteur operator^(Vecteur v1, Vecteur const& v2);// ATTENTION XOR a une plus grande priorité à cause
       du ou exclusif donc toujours mettre des parentaises entre !!!!!
00087
00088 #endif // PRJ_VECTEUR_H
```

5.63 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Vue_OpenGL.cc

```
#include "Vue_OpenGL.h"
```

5.64 Vue_OpenGL.cc

```
00001
00009 #include "Vue_OpenGL.h"
00010
00011 /*============================================================================
00012  * Implementation de Vue_OpenGL
00013  *============================================================================*/
00014
00025 int Vue_OpenGL::TIMER_ID(1);
00026
00027 BEGIN_EVENT_TABLE(Vue_OpenGL, wxGLCanvas)
00028     EVT_PAINT(          Vue_OpenGL::dessine       )
00029     EVT_SIZE(           Vue_OpenGL::OnSize        )
00030     EVT_KEY_DOWN(       Vue_OpenGL::OnKeyDown     )
00031     EVT_ENTER_WINDOW(   Vue_OpenGL::OnEnterWindow )
00032     EVT_TIMER(TIMER_ID, Vue_OpenGL::OnTimer       )
00033 END_EVENT_TABLE()
00034
00043 Vue_OpenGL::Vue_OpenGL( wxWindow*      parent
00044                       , wxSize  const& taille
00045                       , wxPoint const& position
00046                       )
00047 : wxGLCanvas(parent, wxID_ANY, position, taille,
00048              wxSUNKEN_BORDER|wxFULL_REPAINT_ON_RESIZE , wxT(""))
00049 , theta(0.0), phi(0.0), r(150.0)
00050 , timer(new wxTimer(this, TIMER_ID))
00051 {}
00052
00061 Vue_OpenGL::Vue_OpenGL( wxWindow*      parent
00062                       , bool reglage
00063                       , wxSize  const& taille
00064                       , wxPoint const& position
00065                       )
00066 : wxGLCanvas(parent, wxID_ANY, position, taille,
00067              wxSUNKEN_BORDER|wxFULL_REPAINT_ON_RESIZE , wxT(""))
00068 , theta(0.0), phi(0.0), r(150.0)
00069 , timer(new wxTimer(this, TIMER_ID))
00070 , systeme(reglage)
00071 {}
00072
00073
00074 void Vue_OpenGL::dessine(wxPaintEvent&)
00075 {
00076   if (!GetContext()) return;
00077
00078   SetCurrent();
00079
00080   // commence par effacer l'ancienne image
00081   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00082
```

```
00083   /* part du système de coordonnées de base
00084    * (dessin à l'origine : matrice identité) */
00085   glMatrixMode(GL_MODELVIEW);
00086   glLoadIdentity();
00087
00088   // place la caméra
00089   gluLookAt(r,   0.0, 0.0,
00090             0.0, 0.0, 0.0,
00091             0.0, 0.0, 1.0);
00092   glRotated(theta, 0.0, 1.0, 0.0);
00093   glRotated(phi,   0.0, 0.0, 1.0);
00094
00095     //appelle de la méthode dessine de systeme
00096     systeme.dessine();
00097
00098   // Finalement, on envoie le dessin à l'écran
00099   glFlush();
00100   SwapBuffers();
00101 }
00102
00103 // ======================================================================
00104 void Vue_OpenGL::OnSize(wxSizeEvent& event)
00105 {
00106   // Nécessaire pour mettre à jour le contexte sur certaines plateformes
00107   wxGLCanvas::OnSize(event);
00108
00109   if (GetContext()) {
00110     // set GL viewport (not called by wxGLCanvas::OnSize on all platforms...)
00111     int w, h;
00112     GetClientSize(&w, &h);
00113     SetCurrent();
00114     glViewport(0, 0, (GLint) w, (GLint) h);
00115   }
00116 }
00117
00118 // ======================================================================
00119 void Vue_OpenGL::OnKeyDown( wxKeyEvent& event )
00120 {
00121   switch( event.GetKeyCode() ) {
00122   case WXK_LEFT:
00123     RotatePhi( 2.0);
00124     Refresh(false);
00125     break;
00126
00127   case WXK_RIGHT:
00128     RotatePhi( -2.0);
00129     Refresh(false);
00130     break;
00131
00132   case WXK_UP:
00133     RotateTheta( 2.0);
00134     Refresh(false);
00135     break;
00136
00137   case WXK_DOWN:
00138     RotateTheta(-2.0);
00139     Refresh(false);
00140     break;
00141
00142   case WXK_HOME:
00143     r     =  5.0;   // On revient à un point fixé
00144     theta = 35.0;
00145     phi   = 20.0;
00146     Refresh(false);
00147     break;
00148
00149   case WXK_PAGEUP:
00150     deplace(-1.0);    // On se rapproche
00151     Refresh(false);
00152     break;
00153
00154   case WXK_PAGEDOWN:
00155     deplace(1.0);     // On s'éloigne
00156     Refresh(false);
00157     break;
00158
00159   // Pause sur la touche "espace"
00160   case ' ':
00161     if (timer->IsRunning()) {
00162       timer->Stop();
00163     } else {
00164       timer->Start();
00165     }
00166     break;
00167
00168   case 'A':
00169     systeme.ajouterArgon(20, true);
```

```
00170     break;
00171
00172   case 'F':
00173     systeme.ajouterFluor(1, true);
00174     break;
00175
00176   case 'H':
00177     systeme.ajouterHelium(20, true);
00178     break;
00179
00180   case 'N':
00181     systeme.ajouterNeon(20, true);
00182     break;
00183   }
00184
00185   event.Skip();
00186 }
00187
00188 // ========================================================================
00189 void Vue_OpenGL::RotateTheta(GLdouble deg)
00190 {
00191   theta += deg;
00192   while (theta < -180.0) { theta += 360.0; }
00193   while (theta >  180.0) { theta -= 360.0; }
00194 }
00195
00196 // ========================================================================
00197 void Vue_OpenGL::RotatePhi(GLdouble deg)
00198 {
00199   phi += deg;
00200   while (phi <   0.0) { phi += 360.0; }
00201   while (phi > 360.0) { phi -= 360.0; }
00202 }
00203
00204 // ========================================================================
00205 void Vue_OpenGL::deplace(GLdouble dr)
00206 {
00207   r += dr;
00208   if (r < 1.0) r = 1.0;
00209   else if (r > 1000.0) r = 1000.0;
00210 }
00211
00212 // ========================================================================
00213 void Vue_OpenGL::InitOpenGL()
00214 {
00215   // Initialisation OpenGL
00216
00217   SetCurrent();
00218
00219   // active la gestion de la profondeur
00220   glEnable(GL_DEPTH_TEST);
00221
00222   // active la transparence
00223   glEnable (GL_BLEND);
00224   glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
00225
00226   // fixe la perspective
00227   glMatrixMode(GL_PROJECTION);
00228   glLoadIdentity();
00229   gluPerspective(65.0, 4./3., 1.0, 1000.0);
00230
00231   // fixe la couleur du fond à noir
00232   glClearColor(1.0, 1.0, 1.0, 1.0);
00233
00234   // lance le Timer
00235   timer->Start(20);
00236 }
00237
00238 // ========================================================================
00239 void Vue_OpenGL::OnTimer(wxTimerEvent& event)
00240 {
00241   systeme.evolue(event.GetInterval() * 0.0001);
00242
00243   // demande l'affichage
00244   Refresh(false);
00245 }
```

5.65 Référence du fichier /Users/burkhard/Dropbox/projet d'info/maximilian-g065/↩ Simulation d'un gaz parfait/Vue_OpenGL.h

```
#include "wx/wxprec.h"
```

```
#include "wx/wx.h"
#include "wx/glcanvas.h"
#include "Systeme.h"
#include <GL/gl.h>
#include <GL/glu.h>
```

Classes

— class **Vue_OpenGL**
*Prototype de la classe **Vue_OpenGL** (p. 33).*

5.66 Vue_OpenGL.h

```
00001
00009 #ifndef PRJ_VUE_OPENGL_H
00010 #define PRJ_VUE_OPENGL_H
00011
00012 #include "wx/wxprec.h"
00013 #ifndef WX_PRECOMP
00014 #include "wx/wx.h"
00015 #endif
00016 #include "wx/glcanvas.h" // Pour combiner wxWidgets et OpenGL
00017 #include "Systeme.h"
00018 #include <GL/gl.h>
00019 #include <GL/glu.h>
00020
00021
00030 class Vue_OpenGL : public wxGLCanvas
00031 {
00032     public:
00034     Vue_OpenGL( wxWindow*      parent
00035             , wxSize  const& taille   = wxDefaultSize
00036             , wxPoint const& position = wxDefaultPosition
00037
00038             );
00040     Vue_OpenGL( wxWindow*      parent
00041             , bool reglage
00042             , wxSize  const& taille   = wxDefaultSize
00043             , wxPoint const& position = wxDefaultPosition
00044
00045             );
00051     virtual ~Vue_OpenGL() {};
00052
00054     void InitOpenGL();
00055
00056     protected:
00058     void dessine(wxPaintEvent& evenement);
00059     void OnSize(wxSizeEvent& evenement);
00061     void OnKeyDown(wxKeyEvent& evenement);
00063     void OnEnterWindow(wxMouseEvent& evenement) { SetFocus(); }
00065     void OnTimer(wxTimerEvent& event);
00067     void RotateTheta(GLdouble deg);
00069     void RotatePhi(GLdouble deg);
00071     void deplace(double dr);
00072
00073     private:
00074     Systeme systeme;
00075     double theta;
00076     double phi;
00077     double r;
00078     // le "Timer"
00079     wxTimer* timer;
00080     static int TIMER_ID;
00081
00082 DECLARE_EVENT_TABLE()
00083 };
00084 #endif
```