SMILE :-)

Conception module Opale

SOMMAIRE

1. Introduction

Dans le cadre du projet OPALE, et plus particulièrement en ce qui concerne le projet SMILE, l'ensemble des informations concernant la conception se trouve dans le présent document. Cependant pour des raisons de clarté et de facilité d'accès, les documents de conceptions, tout commes les dossiers de spécifications sont répartis en 3 documents distinct... Vous avez entre les main le dossier de conception concernant **le module Opale**.

Ce document a été réalisé avec staroffice, et les diagrammes avec dia. Les sources sont disponibles sur le serveur d'IMERIR ou par notre site internet. Les numéros entre crochet font référence à un lien que l'on trouvera en dernière page.

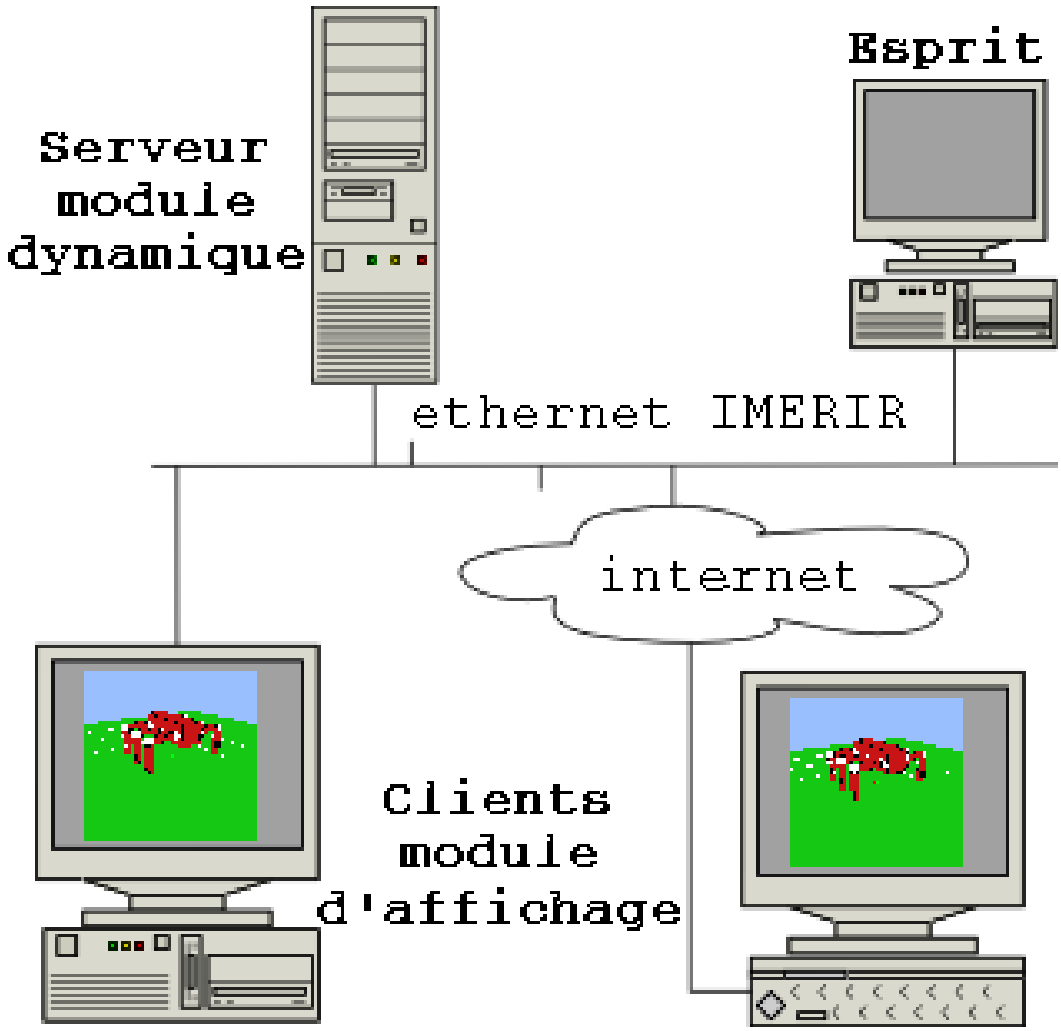
Vous trouverez dans ce document la description de l'architecture du système à différents niveaux.

La première partie décrit son contexte, sa composition, et donne un diagramme de classe general.

La deuxième partie présente un diagramme de classe détaillé.

2. Conception préliminaire

1. Le système dans son contexte



2. Les composants du système

Le système que nous développons cette année se décompose en trois parties :

- un module Opale
- un module Dynamique
- un module Affichage

Le schéma suivant illustre la répartition de chaque module et les relations qui les font communiquer.

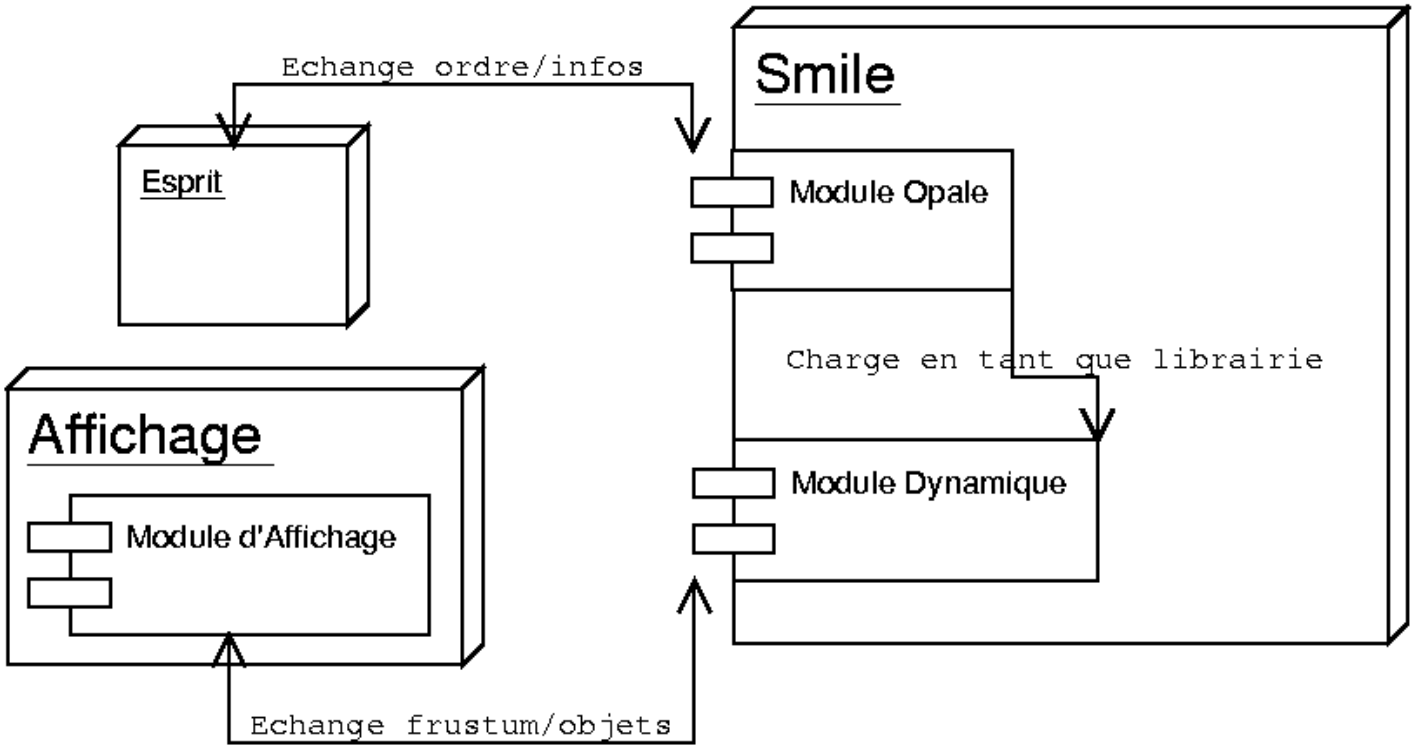
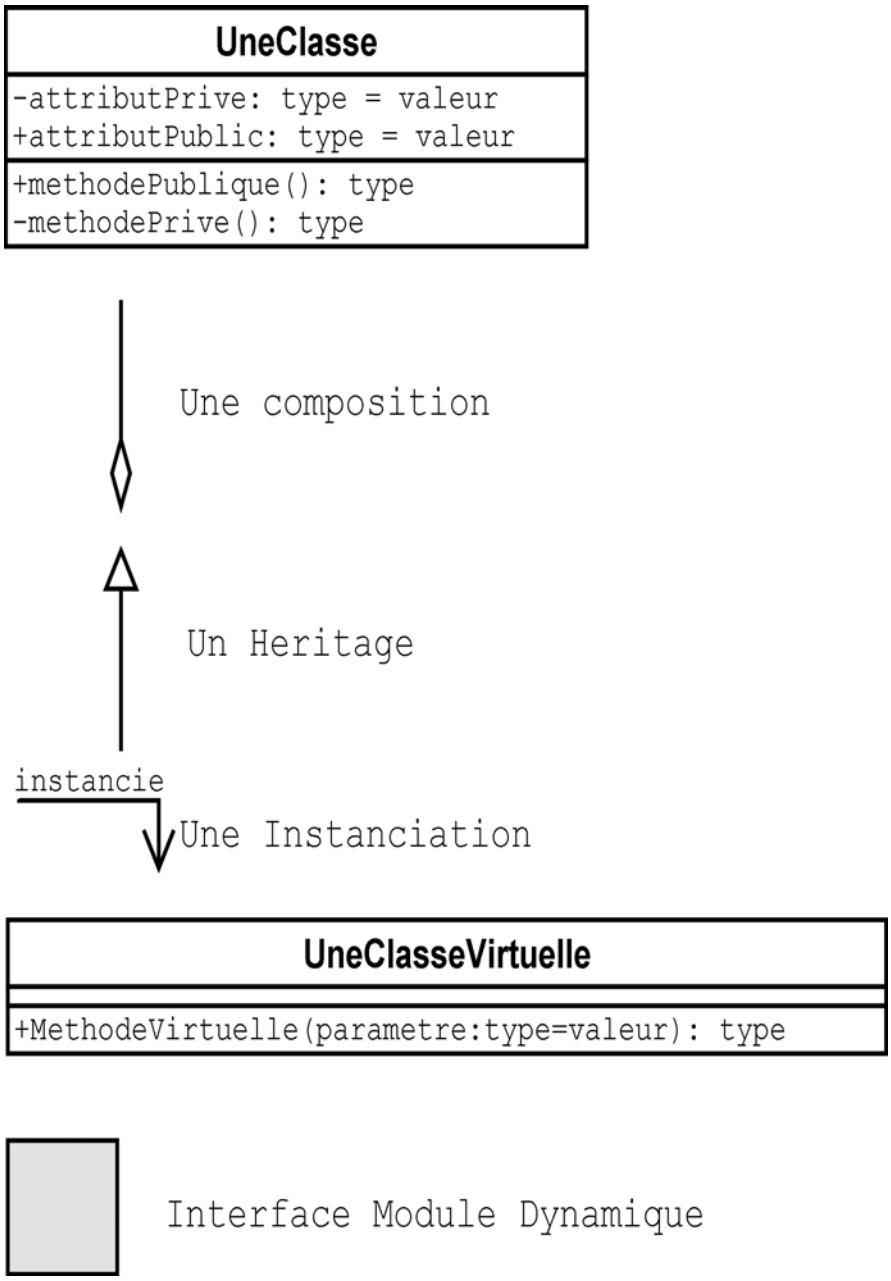
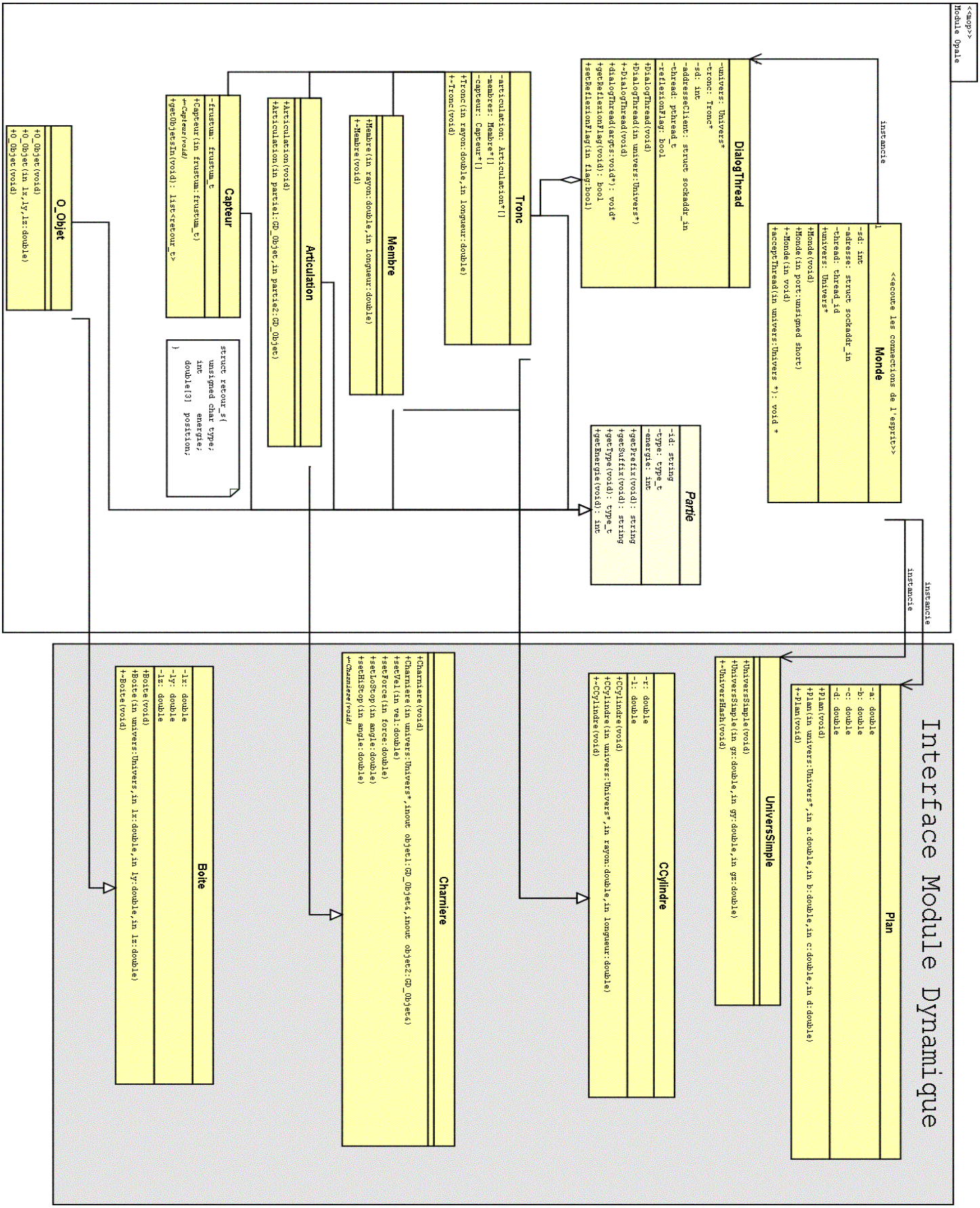


Illustration 1 Schéma des composants du système

3. Diagramme de classe du module opale dans le systèmes




Interface Module Dynamique
Module Opale
Monde
DialogThread
Tronc
Membre
Articulation
Capteur
O_Objet
Partie
Plan
UniversSimple
CCylindre
Charniere
Boite
instancie

3. Conception détaillée

1. Environement de développement

Description de l'environnement de développement par ressources :

Systèmes d'exploitation:

Windows NT/2000/XP.

GNU/Linux.

Logiciels de développement:

Microsoft Visual C++

gcc

Bibliothèques :

ODE / OpenGL / Glut ...

Matériels :

Les contraintes matérielles se situent sur les points ci-dessous qui seront très sollicités durant la simultation :

Reseau Ethernet / Internet (celui d'IMERIR)

Cartes Graphiques OpenGL (pour le module graphique)

Ram (pour le serveur qui fera tourner le module dynamique)

Fréquence processeur élevé (pour le serveur qui fera tourner le module dynamique)

Humain :

l'équipe projet (5 personnes).

Choix du protocole réseau :

Le module de communication du projet Opale étant un projet complet, nous n'implémentons qu'une version simplifiée de ce module. Pour cela nous utiliserons les sockets car elles sont utilisables sous Linux comme sous Windows et de mise en place simple.

2. Description par classe

Classe Capteur :

Hérite de la classe **Partie.**

Les attributs .

Frustum: (frustum) forme du cone de vision

Les constructeurs .

`void Capteur(frustum)`
`:Partie(char type,int energie)`
Créé le tronc de la créature
`Instancie les attributs de l'objet Capteur.`
Tronc.capteurs[nb_capteurs]= this;
Tronc.nb_capteurs++

Les accesseurs .

Les autres methodes :

```
retour_t getObjetIn(void)
    creé une liste de type retour_s contenant les caracteristiques des objets présent dans le volume visioné
                                        struct retour_s : type de l'objet, niveau d'énergie, coordonnées
Pour chaque objet dans univers faire :
        si objet.isIn(m_frustum) faire
                  position=objet.getAABB()
                partie=(cast)objet.userData()
                energie=partie.getEnergie()
                type=partie.getType()
                actuel=concatenation(type,position,energie,partie)
fsi
retour_t=concatenation(actuel,total)
fpour

retour retour_t
```

Informations sur la valeur à retourner pour définir l'objet :

Type	constante
Parties	
Tronc	T
Articulation	A
Membre	M
Capteur	C
Decors	
Sol	S
Obstacle	O
Nourriture	N

Classe Monde :

Les attributs .

sd: (entier) Idntifiant du socket

adresse: socket de communication

thread: Identifiant

univers: pointeur sur UniversSimple du module dynamique

Les constructeurs .

```
void Monde(unsigned short port)
                                        Créé un environement pour accueilir des créatures
instancie un univers : univers = new UniversSimple(0,0,-9.81)
ouvre un port sur lequel écouter des demandes de connection
lance un thread pour écouter des connections entrantes en y passant en
parametres le pointeur sur l'univers
```

Les accesseurs .

Les autres methodes :

```
void* acceptThread(Univers *univers)
                                        ecoute et établie une connection avec un esprit
boucle infnie
  écoute les demandes de connection
  si la conection est établie faire
      Rechercher une position libre
              Instancier un dialogThread (une créature) en lui passant en
      parametre le pointeur vers l'univers.
  Fsi
    Pour chaque créature faire
              Tantque refexionFlag = = 0   FtantQue
//optimisation bloquage du jeu
    FPour     //tous les flags sont alors a 1
    Pour chaque créature faire  reflexionFlag = 0 FPour
fboucle
```

Classe dialogThread :

Les attributs.

univers: pointeur sur l'objet UniversSimple dumodule dynamique

tronc: pointeur sur le tronc de la créature

sd: (entier) Identifiant du socket

addresseClient :

thread :

reflexionFlag: (bool) permet de gerer le tour de jeu (creation creature et action)

Les constructeurs .

```
void dialogThread(Univers *univers)
                                    permet de communiquer entre l'esprit et le moteur dynamique
boucle infinie   //attente de la reception d'une decision
Si décision   reflexionFlag = 1      Fsi
TantQue   reflexionFlag différent de 0     FtantQue
Agir(decision)
FBoucle
```

Les accesseurs .

Les autres methodes :

Classe Tronc :

Hérite de la classe **Partie.**

Les attributs .

articulation: tableau de pointeur contenant toutes les articulations de la créature.
membres: tableau de pointeur contenant tous les membres de la créature.
capteurs: tableau de pointeur contenant tous les capteurs de la créature.
rayon : (double) rayon du tronc physique de la créature.
longueur: (double) longueur du tronc physique de la créature.
masse: (double) masse de la créature.
nb_membres: (double) nombre de membres de la creature;
nb_articulations: (double) nombre d'articulations de la creature
nb_capteurs: (double) nombre de capteurs de la creature
energie: (entier) correspond a l'energie de la créature

Les constructeurs .

`void Tronc(double rayon,double longueur,double masse,int energie)` `:Partie(int ID)` Créé le tronc de la créature
`Instancie les attributs de l'objet tronc.`
`Appelle aux constructeurs de menbres,capteurs,articulations.`

Les accesseurs .

Les autres methodes :

Classe Partie :

Les attributs .

ID: (string) defini l'Identité de la créature (type+numéro d'identification)

type: type de partie

energie: energie possédée par la partie

Les constructeurs .

Appartiennent à ses classes héritières

Les accesseurs .

Les autres methodes :

`type_t getType(void)`
Retourne le type de l'élément
`A partir de l'ID, on retrouve le type : membre,tronc,articulation,capteur ou O_Objet`

`string getPrefix(void)`
Retourne l'ID commun avec la partie IA de la Creature
`A partir de l'ID, on retrouve le prefixe de l'id`

`string getSuffix(void)`
Retourne l'ID commun avec la partie dynamique de Smile
`A partir de l'ID, on retrouve le suffixe de l'id`

`void GetEnergie(void)` Niveau d'energie d'un objet Retour energie

Classe Membre :

Hérite de la classe **Partie.**

Les attributs .

rayon : (double) rayon du tronc physique de la créature.

longueur: (double) longueur du tronc physique de la créature.

Les constructeurs .

`void` *`Membre`*`(double rayon,double longueur)`
Créé un membre de la créature
`Instancie les attributs de l'objet Membre.`
`Tronc``.membres[nb_membres]= this;`
`Tronc``.nb_membres`

Les accesseurs .

Les autres methodes :

Classe Articulation :

Hérite de la classe **Partie.**

Les attributs .

angle : (double) angle de rotation possible

Les constructeurs .

`void` *`Articulation`*`(angle)`
 	
Créé une articulation de la créature
`Instancie les attributs de l'objet Articulation.`
`Tronc``.articulations[nb_articulations]= this;`
`Tronc``.nb_articulations++`

Les accesseurs .

Les autres methodes :

Classe O_Objet :

Hérite de la classe **Partie**

Les attributs .

Forme: forme de l'objet

px : position en x

py : position en y

pz position en z

lx : (double) taille sur l'axe x

ly : (double) taille sur l'axe y

lz : (double) taille sur l'axe z

Les constructeurs .

`void O_Objet(lx,ly,lz,px,py,pz)`
 	
Créé un obstacle du décor
 	
`Vérifie que px,py,pz appartiennent au sol`
`Instancie les attributs de l'objet Obstacle.`

Les accesseurs .

Les autres methodes :