Réunion MAAM SimulAtom

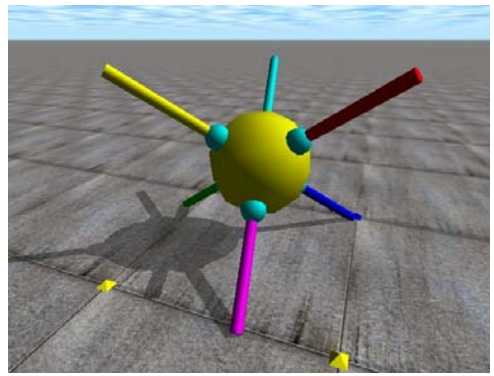
- Michel Dubois
- Yann Le Guyadec

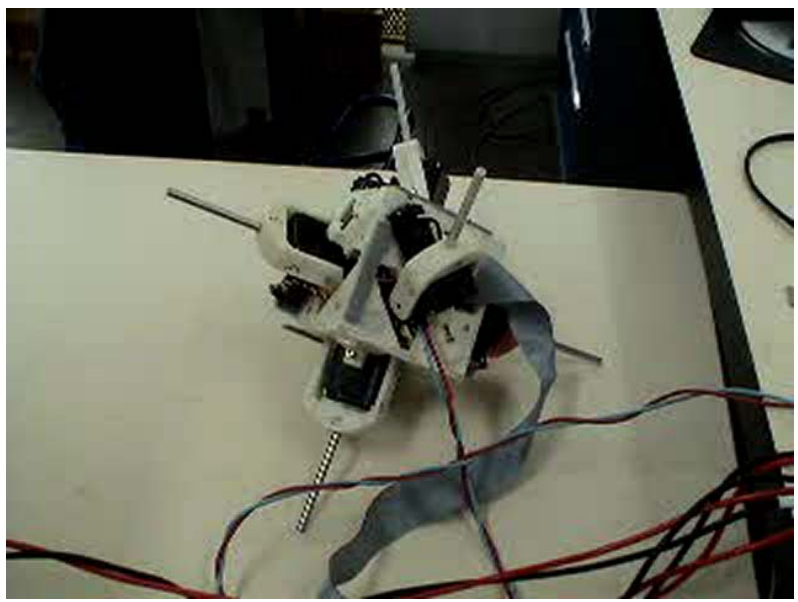
Plan

- Introduction
  - Le composant MAAM
  - Le projet MAAM
  - L'API MAAM
  - Les modes de communication
- La simulation
  - Pourquoi simuler
  - Fonctionnalités
  - Simulation dynamique versus simulation cinématique
- Les principales caractéristiques du simulateur
  - Open Dynamics Engine
  - Übersim
  - Modifications des composants
  - Des classes utilitaires
  - La programmation par consigne
- Exemples d'utilisation du simulateur
  - Le walker
  - L'araignée
  - Autres molécules …
- Futurs développements
  - Simulation de capteurs
  - Vers un simulateur distribué
- Perspectives et conclusions

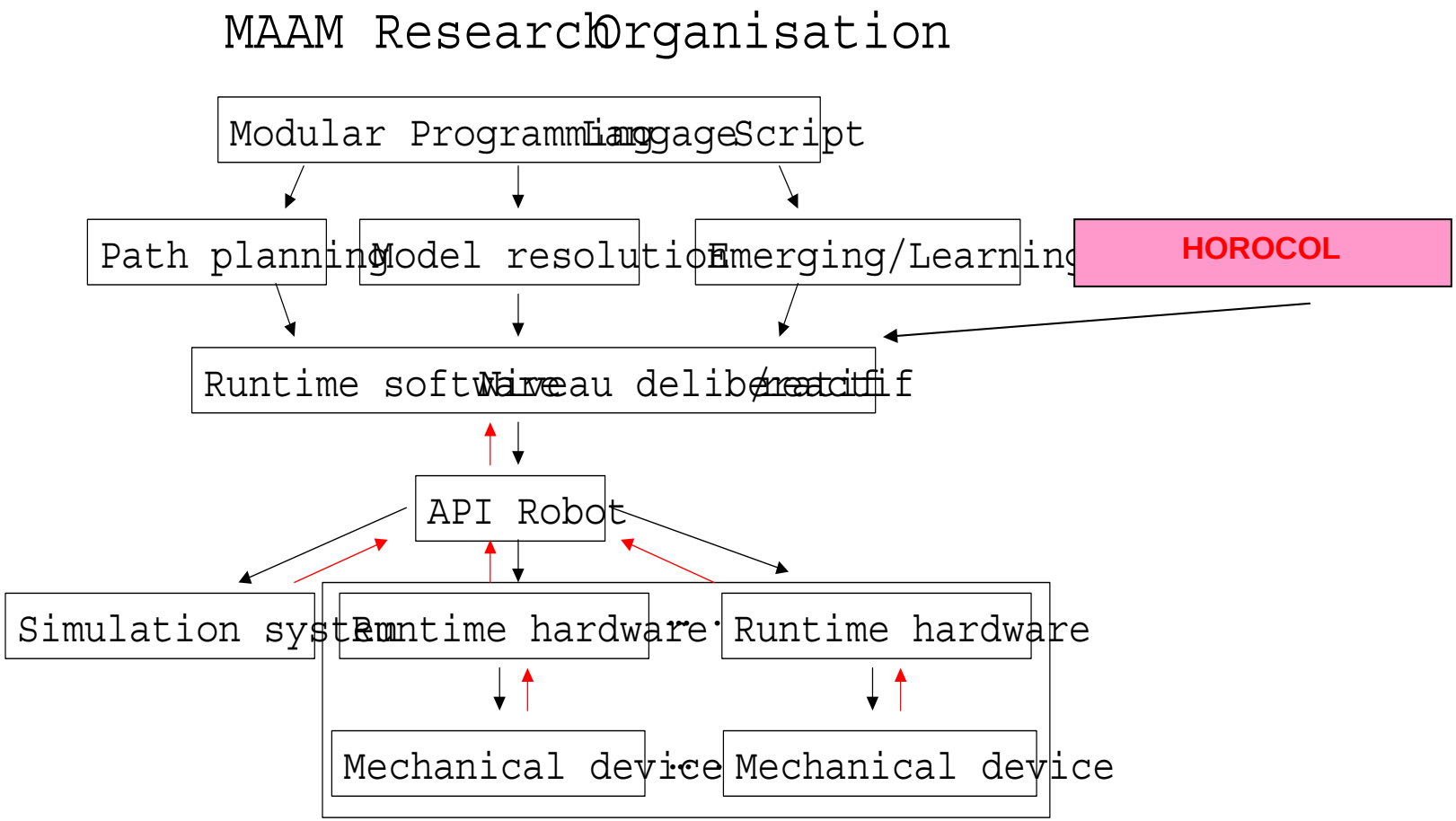
Description of our basic component (Atom)

- 6 legs :
  - 2 perpendicular dof {-pi/4 .. +pi/4}
  - interconnection mechanism to form *molecules*
- Communications :
  - radio (broadcast from a server)
  - point to point (between neighbours)
- Perception :
  - on each leg :
    - *1 IR sender/receiver*
    - *1 contact detection sensor*
    - *3 force sensors*
  - into the kernel :
    - *1 accelerometer*
    - *1 gyrometer (angular speed)*

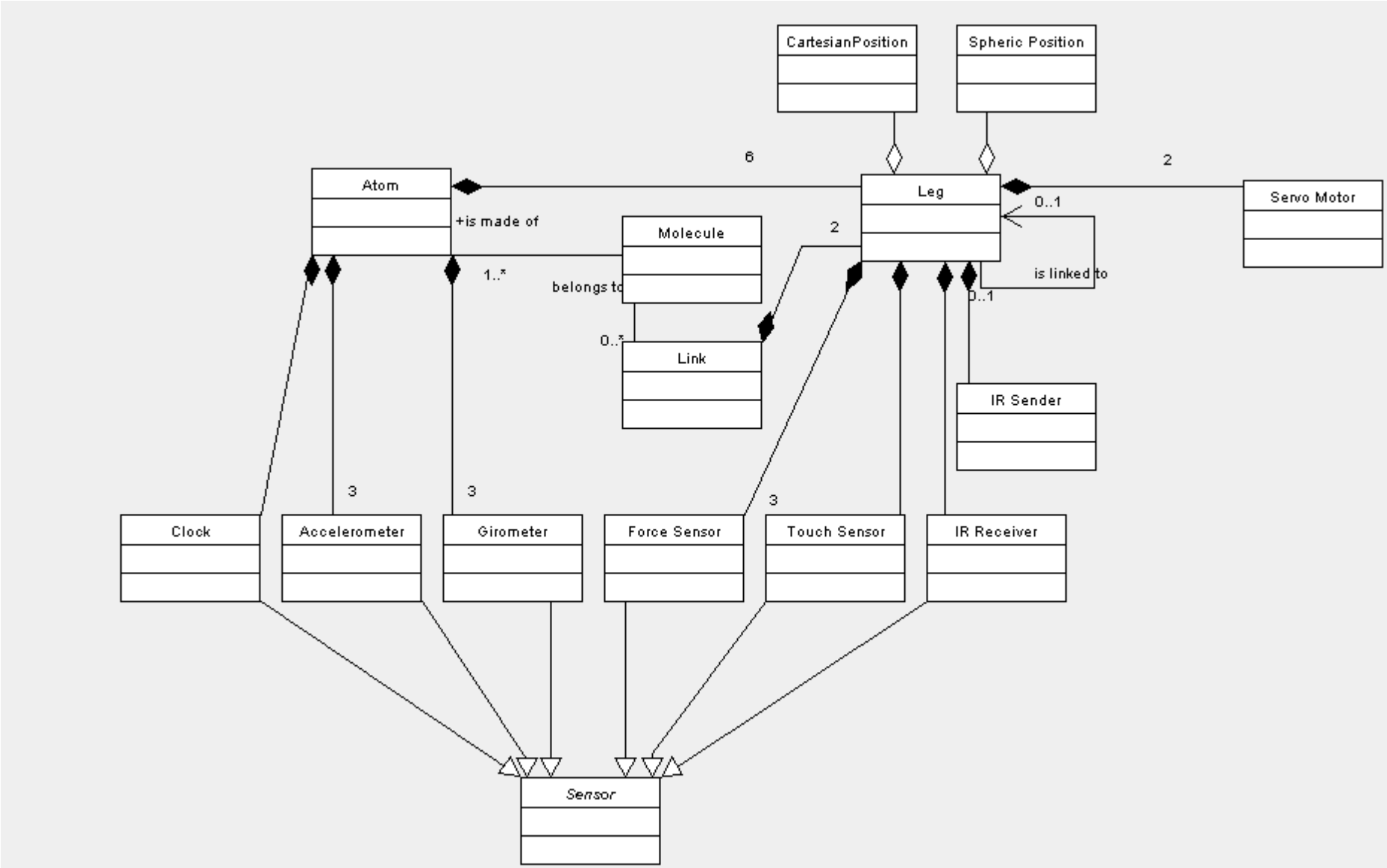




Teams in the MAAM Project



API

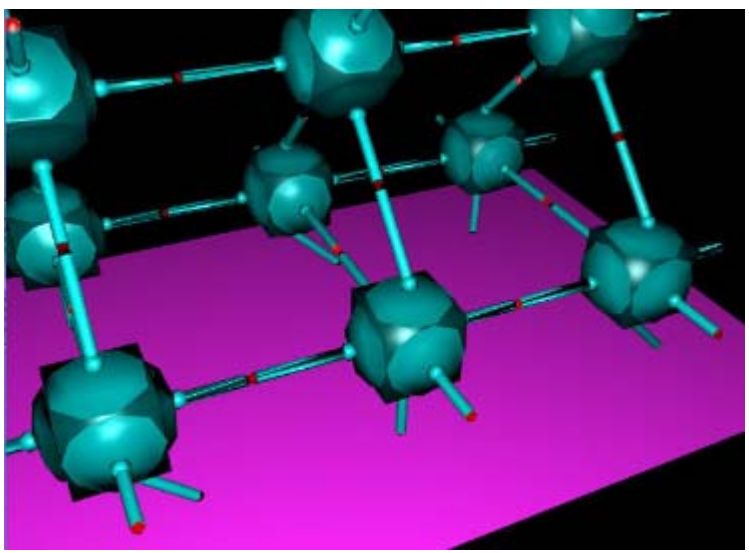

CartesianPosition
Spheric Position
6
2
Atom
Leg
Servo Motor
+is made of
0..1
Molecule
2
1..*
belongs to
is linked to
0..1
0..*
Link
IR Sender
3
3
3
Clock
Accelerometer
Girometer
Force Sensor
Touch Sensor
IR Receiver
Sensor

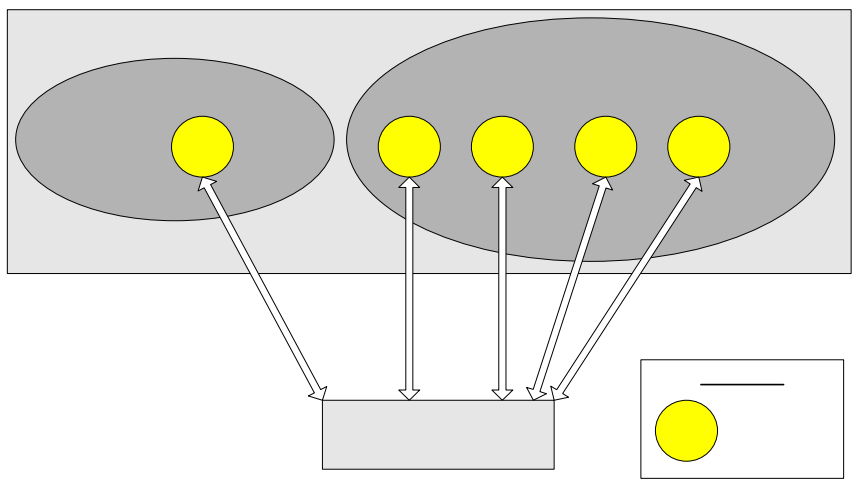
Communication directe

Chaque atome est accessible directement par l'ordonnanceur.

C'est une vision qui correspond aux prototypes centralisés du simulateur

- blender;
- Java 3D;
- Übersim+ODE.





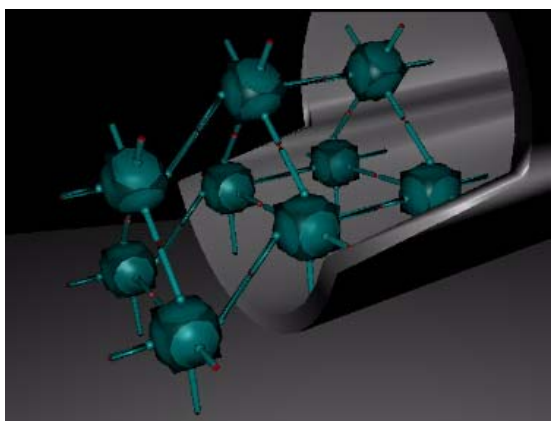
Communication Point à Point

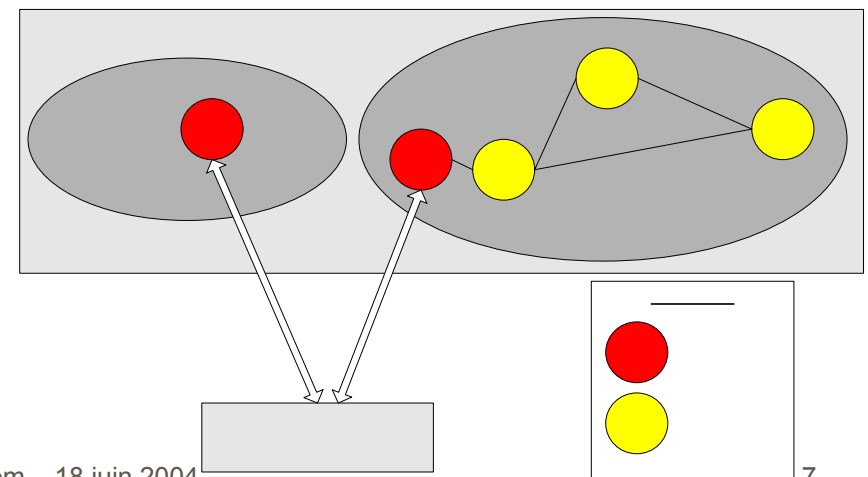
Que faire quand un atome de la molécule n'est plus joignable ?

Un seul atome est joignable et transfert les messages aux autres atomes de la molécule.

Projet DESS ASIR 2002 « AlgoAtome »

- Même interface que l'API centralisée.
- Implémentation en JAVA RMI.





Plan

- Introduction
  - Le composant MAAM
  - Le projet MAAM
  - L'API MAAM
  - Les modes de communication
- La simulation
  - Pourquoi simuler
  - Fonctionnalités
  - Simulation dynamique versus simulation cinématique
- Les principales caractéristiques du simulateur
  - Open Dynamics Engine
  - Übersim
  - Modifications des composants
  - Des classes utilitaires
  - La programmation par consigne
- Exemples d'utilisation du simulateur
  - Le walker
  - L'araignée
  - Autres molécules …
- Futurs développements
  - Simulation de capteurs
  - Vers un simulateur distribué
- Perspectives et conclusions

Simulation Tool

- Why simulation ?
  - Speed
  - Low cost
- How to close the ‘ reality gap ’ ?
  - Add noise
  - Sampling
  - Minimal simulation
- A physical simulation library :ODE

Fonctionalités

Fonctionnalités des atomes

- Mouvement
- *(Connexion)*/Déconnexion
- Communication avec le serveur ou les voisins
- Recepteur/Emetteur IR
- Capteur d'effort
- Capteur de contact
- Accelerometre/gyromètre

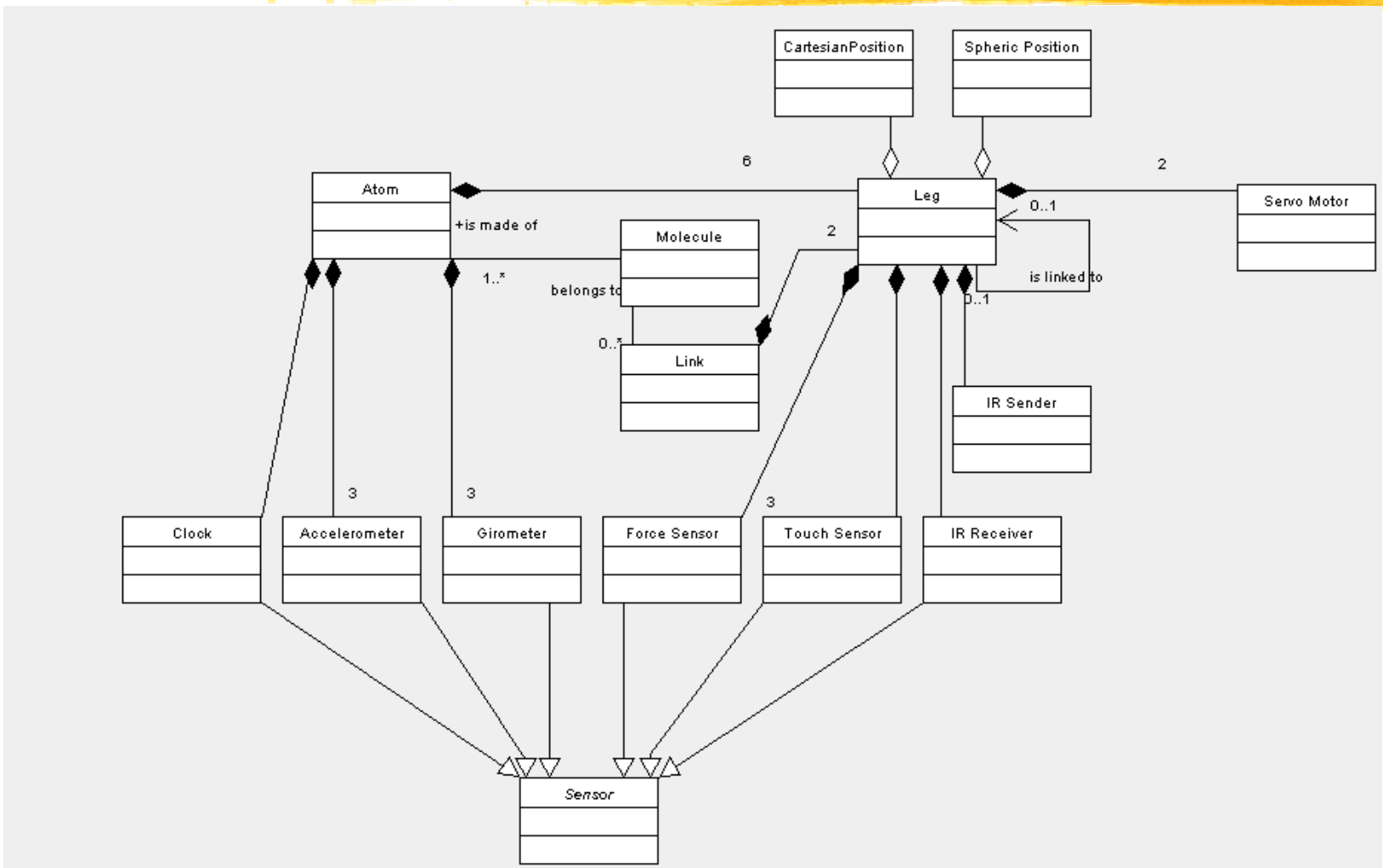
Environnement

- Modèle géométriques
- Modèle de collision
- Modèle de gravité
- Modèle de propagation d'effort

Contrôle

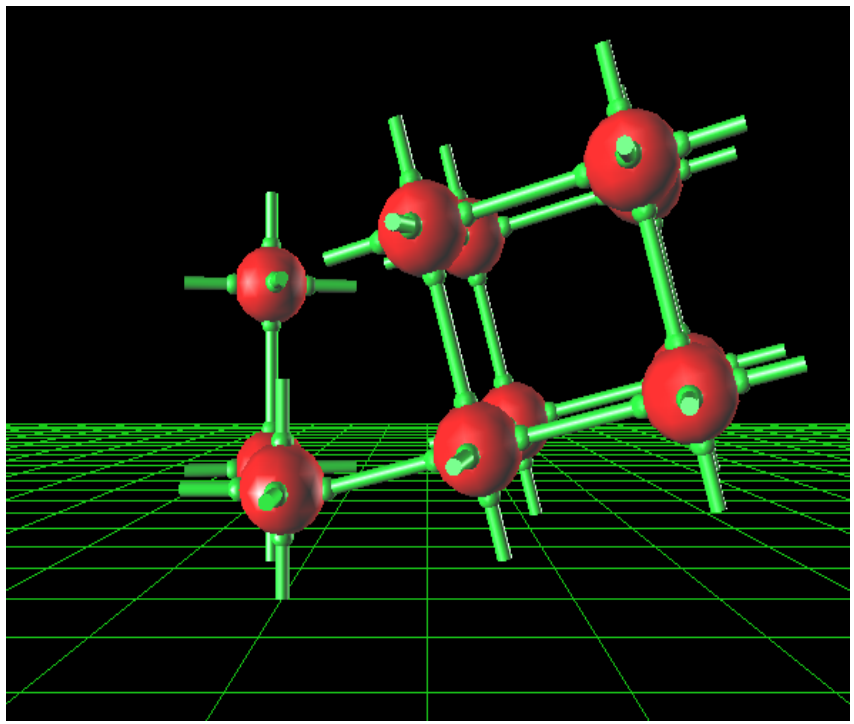
- Modèle de concurrence
- Modèle du temps

API


CartesianPosition
Spheric Position
6
2
Atom
Leg
Servo Motor
+is made of
0..1
Molecule
2
1..*
belongs to
is linked to
0..1
0..*
Link
IR Sender
3
3
3
Clock
Accelerometer
Girometer
Force Sensor
Touch Sensor
IR Receiver
Sensor

Simulation cinématique centralisée

- 2 prototypes Blender/python
- 1 prototype Java 3D
- Une application de construction de molécules



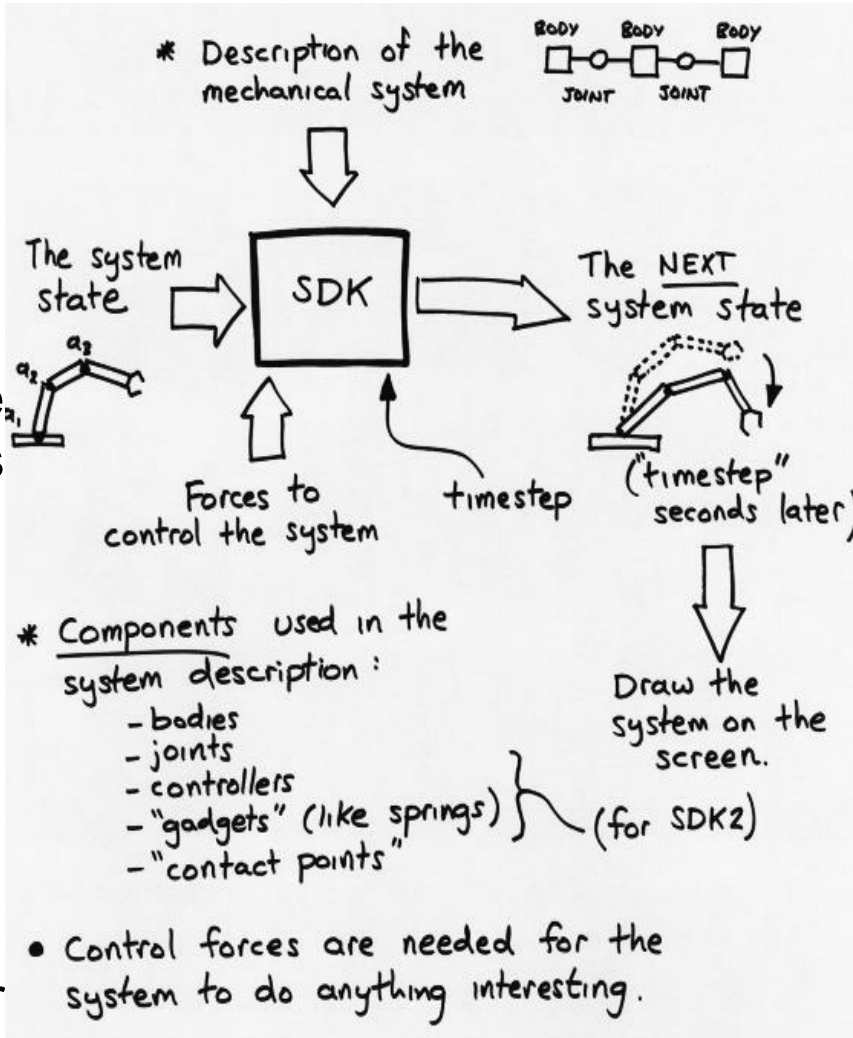
- Les collisions ne sont pas gérées.
- Les lois physiques ne sont pas simulées.

Plan

- Introduction
  - Le composant MAAM
  - Le projet MAAM
  - L'API MAAM
  - Les modes de communication
- La simulation
  - Pourquoi simuler
  - Fonctionnalités
  - Simulation dynamique versus simulation cinématique
- **Les principales caractéristiques du simulateur**
  - **Open Dynamics Engine**
  - **Übersim**
  - **Modifications des composants**
  - **Des classes utilitaires**
  - **La programmation par consigne**
- Exemples d'utilisation du simulateur
  - Le walker
  - L'araignée
  - Autres molécules …
- Futurs développements
  - Simulation de capteurs
  - Vers un simulateur distribué
- Perspectives et conclusions

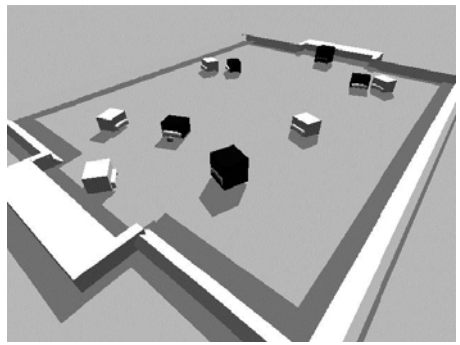
Dynamics library ODE

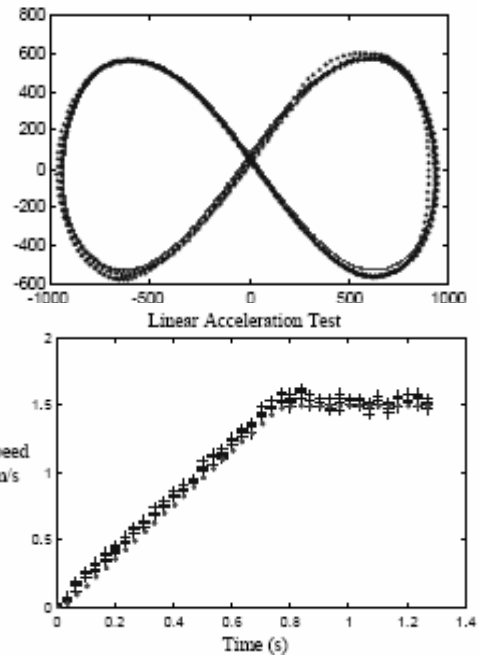
- ODE : Open Dynamics Engine
  `http://opende.sourceforge.net`
- Open Source project.
- Cross platform development.
- A rigid body simulation engine initiated by Russel SMITH.
- It has reached a maturity level: accurate, methods are fast and stable and it is reasonably documented.
- Its methods are based on following works:
  - *Mirtich's method for collisions,*
  - *Stewart/Trinkle and Anitescu/Potra for integration*
  - *Baraff for the solver.*
- It provides Open GL routines to render the 3D simulated environment.



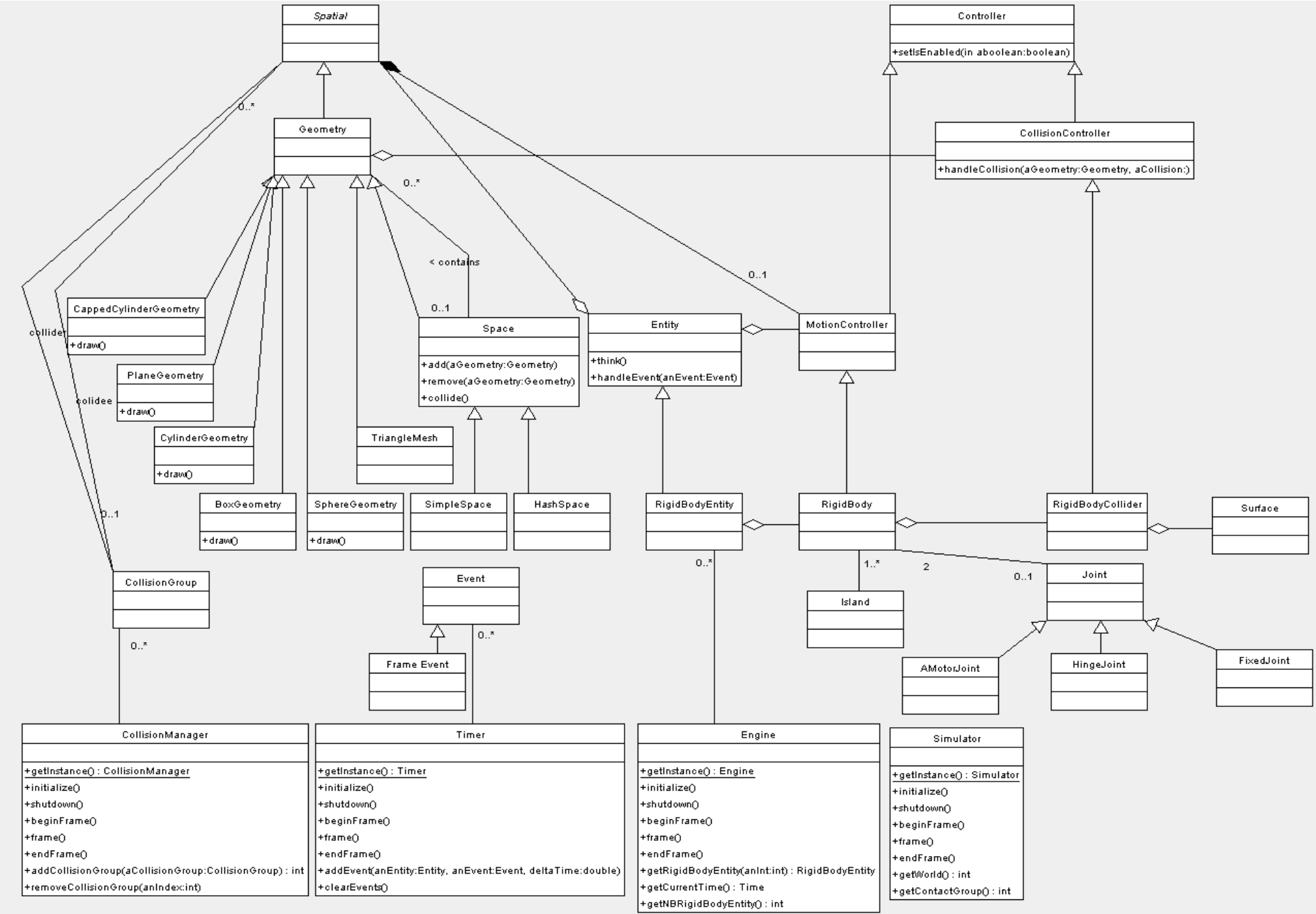
Übersim Simulation tool Version 0.56

- http://www-2.cs.cmu.edu/~coral/ubersim.html
- Open source project, Linux platform.
- Goal : create a simulation environment that enables control systems to be developed rapidly and transferred to the real system easily.
- A comparison between a simulated robot and a real robot validate the Übersim simulator.
- Internal :
  - Discrete event driven simulation
  - An Object Oriented framework
  - Based on Open Dynamics Engine
  - Store the World State via the Scene Graph
  - The particular parameters are read from text configuration files
  - The perception layer is not relevant.
  - Only for wheeled robots not for legged one.
  - Not for distributed simulation

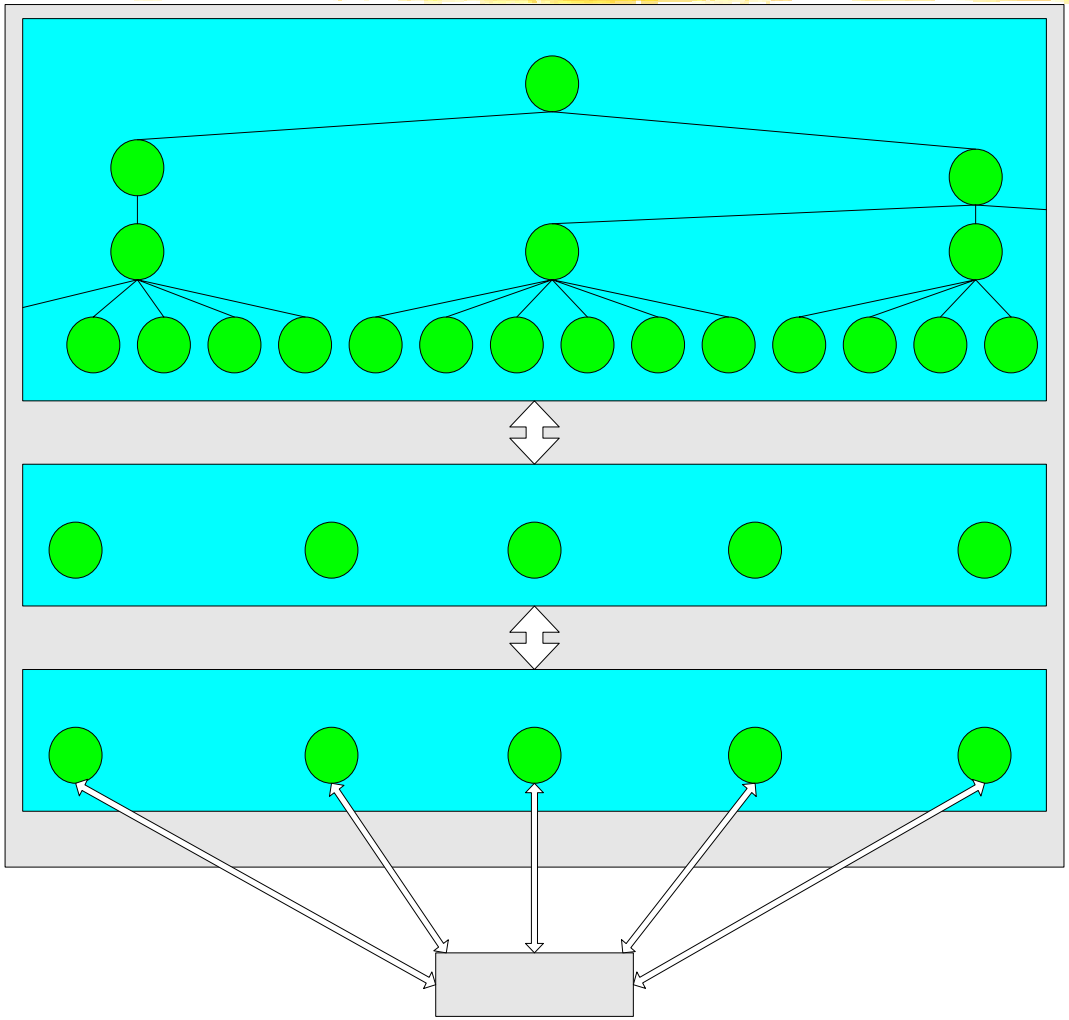




Framework

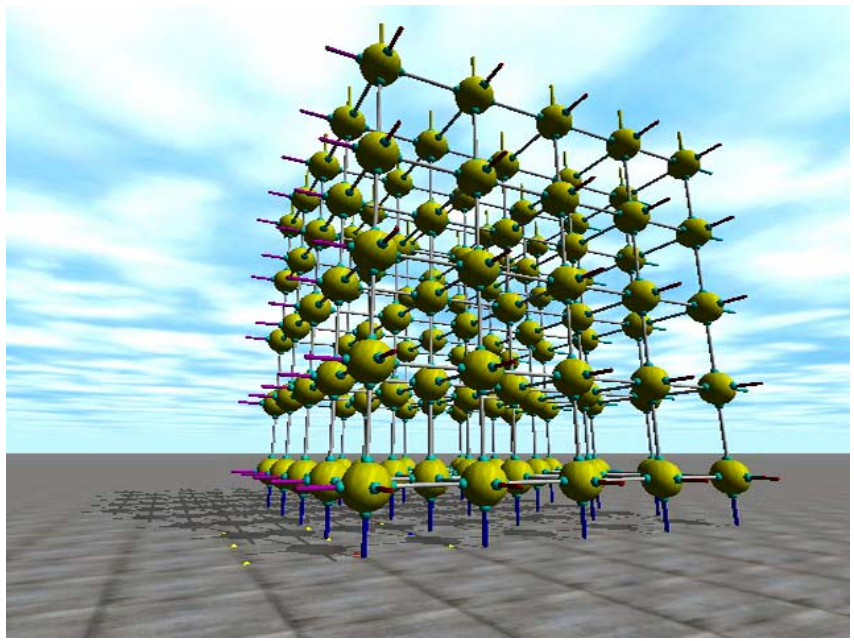

Spatial
Controller
+setIsEnabled(in aboolean:boolean)
0..*
Geometry
CollisionController
+handleCollision(aGeometry:Geometry, aCollision:)
0..*
< contains
0..1
0..1
CappedCylinderGeometry
+draw()
collidee
Space
+add(aGeometry:Geometry)
+remove(aGeometry:Geometry)
+collide()
Entity
+think()
+handleEvent(anEvent:Event)
MotionController
PlaneGeometry
+draw()
colidee
CylinderGeometry
+draw()
TriangleMesh
BoxGeometry
+draw()
SphereGeometry
+draw()
SimpleSpace
HashSpace
RigidBodyEntity
RigidBody
RigidBodyCollider
Surface
0..1
CollisionGroup
0..*
Event
0..*
Frame Event
0..*
1..*
2
0..1
Island
Joint
AMotorJoint
HingeJoint
FixedJoint
CollisionManager
+getInstance() : CollisionManager
+initialize()
+shutdown()
+beginFrame()
+frame()
+endFrame()
+addCollisionGroup(aCollisionGroup:CollisionGroup) : int
+removeCollisionGroup(anIndex:int)
Timer
+getInstance() : Timer
+initialize()
+shutdown()
+beginFrame()
+frame()
+endFrame()
+addEvent(anEntity:Entity, anEvent:Event, deltaTime:double)
+clearEvents()
Engine
+getInstance() : Engine
+initialize()
+shutdown()
+beginFrame()
+frame()
+endFrame()
+getRigidBodyEntity(anInt:int) : RigidBodyEntity
+getCurrentTime() : Time
+getNBRigidBodyEntity() : int
Simulator
+getInstance() : Simulator
+initialize()
+shutdown()
+beginFrame()
+frame()
+endFrame()
+getWorld() : int
+getContactGroup() : int

Architecture de SimulAtom (C++)



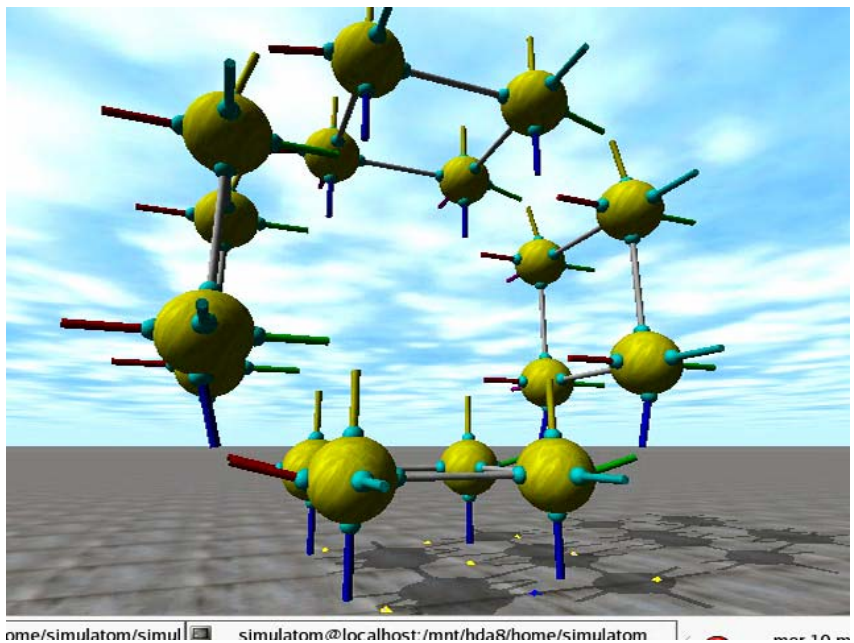
Modification des composants

- Un framework étendu aux robots à pattes
- Un framework corrigé pour les configurations cycliques
- O.D.E. corrigé pour une meilleure extensibilité
- Le rendu des sphères a été amélioré



Notion d'état initial

- En statique
  - Positionnement des atomes dans l'espace
  - Accrocher les atomes à l'environnement
- Avec la dynamique :
  - Déplacement des pattes
  - Connexions actives souhaitables
- Une fois la molécule formée, on peut décrocher les atomes



Classes utilitaires

Des classes propres à la simulation sont nécessaires :

- Pour constituer l'état initial d'une molécule

  `(util->setCartesianPosition(,,))`

- Pour remplacer les capteurs (`util->isOnGround()`)
- Pour récupérer des informations précieuses

  `(util->getCartesianPosition())`

- Chaque classe de l'API possède une classe interne.
- La classe `SimWorld` permet de créer le monde et de configurer la simulation

Documentation du simulateur

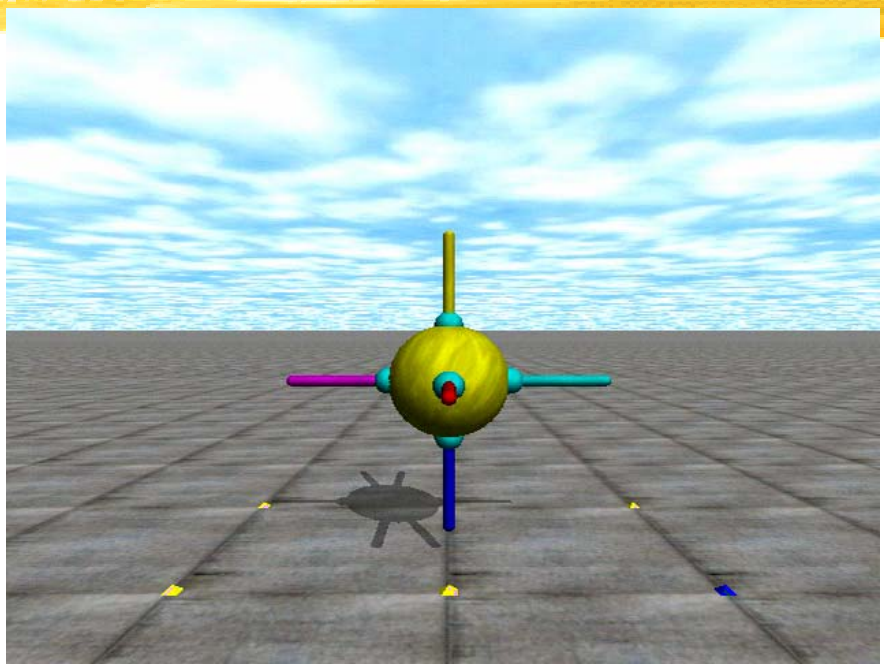
- Documentation fournie avec Doxygen
  - Documentation des classes de l'API
  - Documentation des méthodes de l'API
  - Documentation de la structure du framework ÜBERSIM
- Fichier README
  - Les paquetages à installer
  - Différentes installations : mode mono ou multi-utilisateur
  - Documentation des applications fournies à titre d'exemple
  - Ecrire un programme de contrôle
- Les présentations du projet MAAM
- Ma page web :

`www-valoria.univ-ubs.fr/Michel.Dubois`

Programmation par consigne - Principe

- L'appel à l'API est non bloquant. L'appel à un accesseur se contente de la lecture d'une valeur retournée. L'appel à un modificateur permet de fixer une consigne. Puis la main est redonnée au programme appelant. La consigne sera exécutée sur plusieurs pas d'intégration.
- Il y a une queue pour mémoriser les différentes consignes. Le mode KEEP permet de mettre la consigne dans la queue. Le mode REPLACE vide la queue avant l'ajout de la consigne.
- Lors d'un pas d'intégration, si l'objectif de la consigne à exécuter n'est pas atteint, elle est traitée puis elle est ajoutée dans la queue pour être de nouveau traitée prioritairement au prochain pas d'intégration. Sinon, c'est la consigne suivante qui sera traitée.
- La consigne est étiquetée temporellement pour déterminer quand elle doit être exécutée. Un délai peut être précisé pour retarder son exécution.

Programmation par consigne – Exemple 1



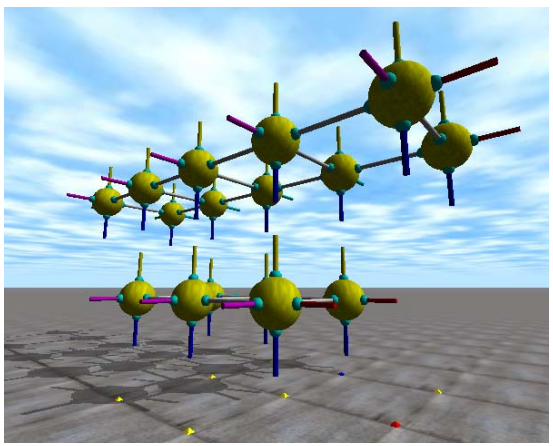
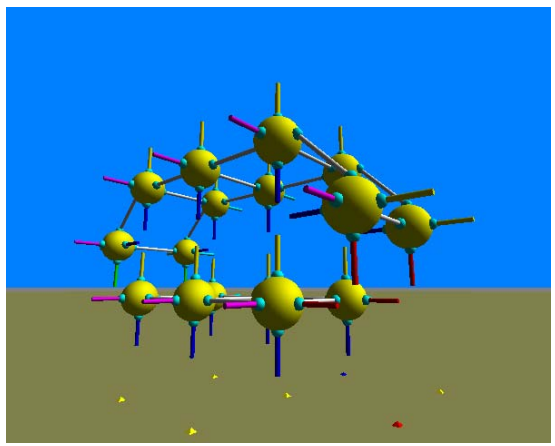
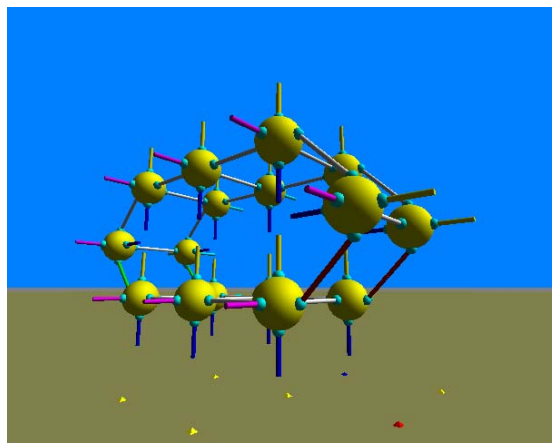
```
atom->getLeg(0)->getServoMotor(0)->setTargetPosition(0,REPLACE);
atom->getLeg(0)->getServoMotor(0)->setTargetPosition(250,KEEP);
atom->getLeg(0)->getServoMotor(0)->setTargetPosition(125,KEEP);
atom->getLeg(0)->getServoMotor(1)->setTargetPosition(0,REPLACE,10)
atom->getLeg(0)->getServoMotor(1)->setTargetPosition(250,KEEP,10);
atom->getLeg(0)->getServoMotor(0)->setTargetPosition(125,KEEP,10);
```

Programmation par consigne - Synchronisation

- Il faut permettre au programme de contrôle de savoir quand un ensemble de consignes a été totalement exécutées.
- L'ajout d'une étiquette de synchronisation à la consigne permet de définir les éléments de l'ensemble de synchronisation.
- Pour connaître si un ensemble de consigne a été totalement exécuté, il suffit de faire appel à la méthode `isFinished()` de la classe `Molécule`.

Exemple – barrières de synchronisation

- Pour créér une chenille, on peut passer par plusieurs étapes :

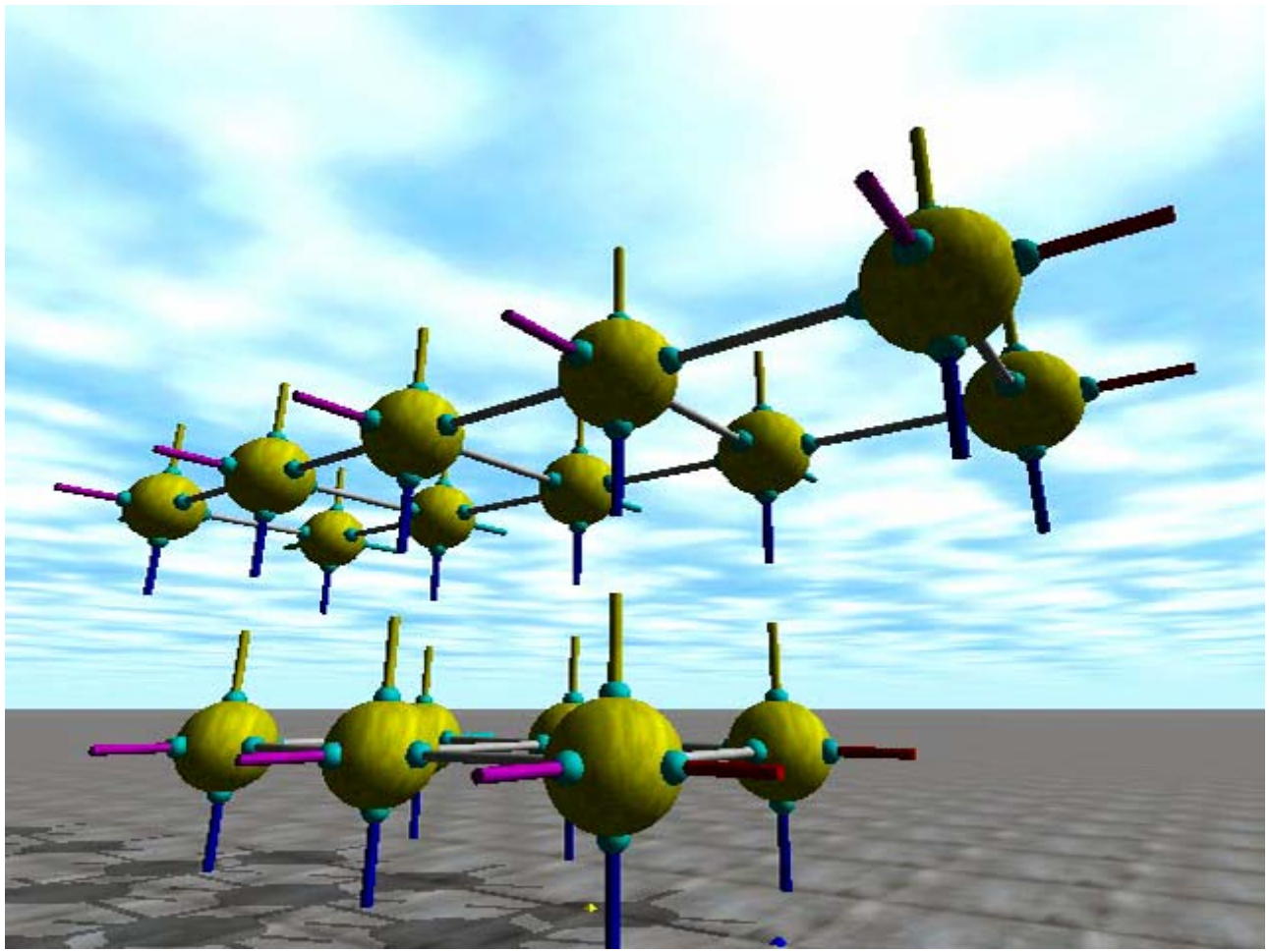




- On peut définir deux ensembles de synchronisation

Programmation par consigne – Exemple 2

```
unsigned int synchro1=atoms[0]->getMolecule()->getNextSyncIndex();
atoms[3]->getLeg(0)->getServoMotor(0)->setTargetPosition(0,REPLACE,0,synchro1);
atoms[4]->getLeg(1)->getServoMotor(0)->setTargetPosition(250,REPLACE,0,synchro1);
atoms[7]->getLeg(1)->getServoMotor(0)->setTargetPosition(250,REPLACE,0,synchro1);
atoms[6]->getLeg(0)->getServoMotor(0)->setTargetPosition(0,REPLACE,0,synchro1);
…
unsigned int synchro2=atoms[0]->getMolecule()->getNextSyncIndex();
atoms[0]->getLeg(1)->getServoMotor(0)->setTargetPosition(0,REPLACE,0,synchro2);
atoms[3]->getLeg(1)->getServoMotor(0)->setTargetPosition(250,REPLACE,0,synchro2);
atoms[7]->getLeg(0)->getServoMotor(0)->setTargetPosition(0,REPLACE,0,synchro2);
atoms[2]->getLeg(0)->getServoMotor(0)->setTargetPosition(250,REPLACE,0,synchro2);
…
for(unsigned int i=0;i<atoms[0]->getMolecule()->getSyncIndexCount();i++)
    {
      printf("Ensemble de synchronisation[%i] indexe par %i :",
             i,atoms[0]->getMolecule()->getSyncIndex(i));
      if (atoms[0]->getMolecule()->isFinished(atoms[0]->getMolecule()->getSyncIndex(i))==true)
        printf ("termine\n");
      else
        printf ("non termine\n");
    }
```

Programmation par consigne – Video 2

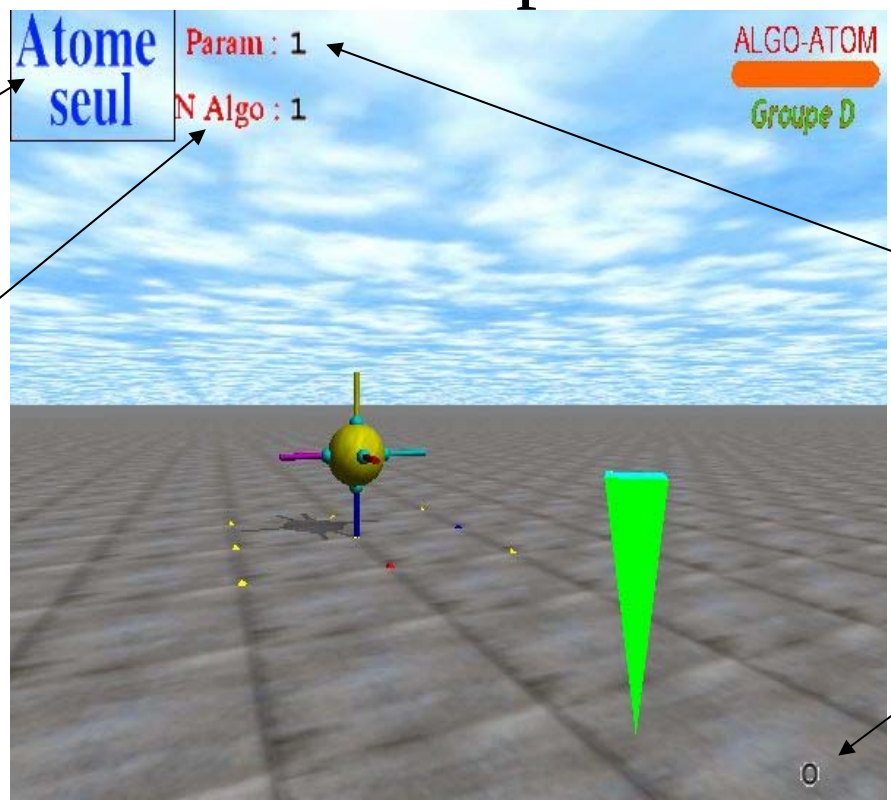


Plan

- Introduction
  - Le composant MAAM
  - Le projet MAAM
  - L'API MAAM
  - Les modes de communication
- La simulation
  - Pourquoi simuler
  - Fonctionnalités
  - Simulation dynamique versus simulation cinématique
- Les principales caractéristiques du simulateur
  - Open Dynamics Engine
  - Übersim
  - Modifications des composants
  - Des classes utilitaires
  - La programmation par consigne
- **Exemples d'utilisation du simulateur**
  - **Le walker**
  - **L'araignée**
  - **Autres molécules …**
- Futurs développements
  - Simulation de capteurs
  - Vers un simulateur distribué
- Perspectives et conclusions

Projet Etudiant DUT Info

- L'interface mise en place :



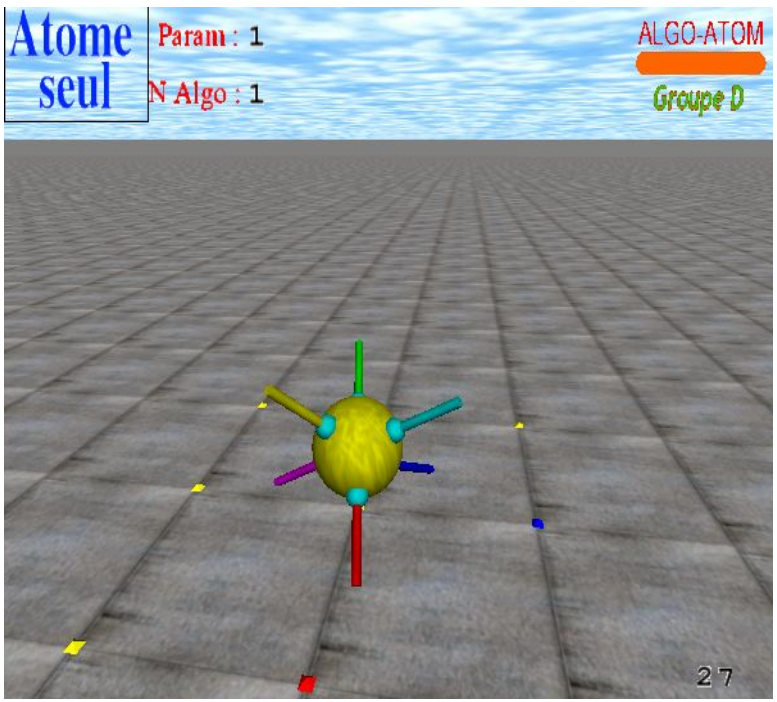
Nom de la Molécule

Numéro de l'algorithme

Paramètre de construction

Temps simulé

Projet Etudiant DUT Info



- L’atome seul :

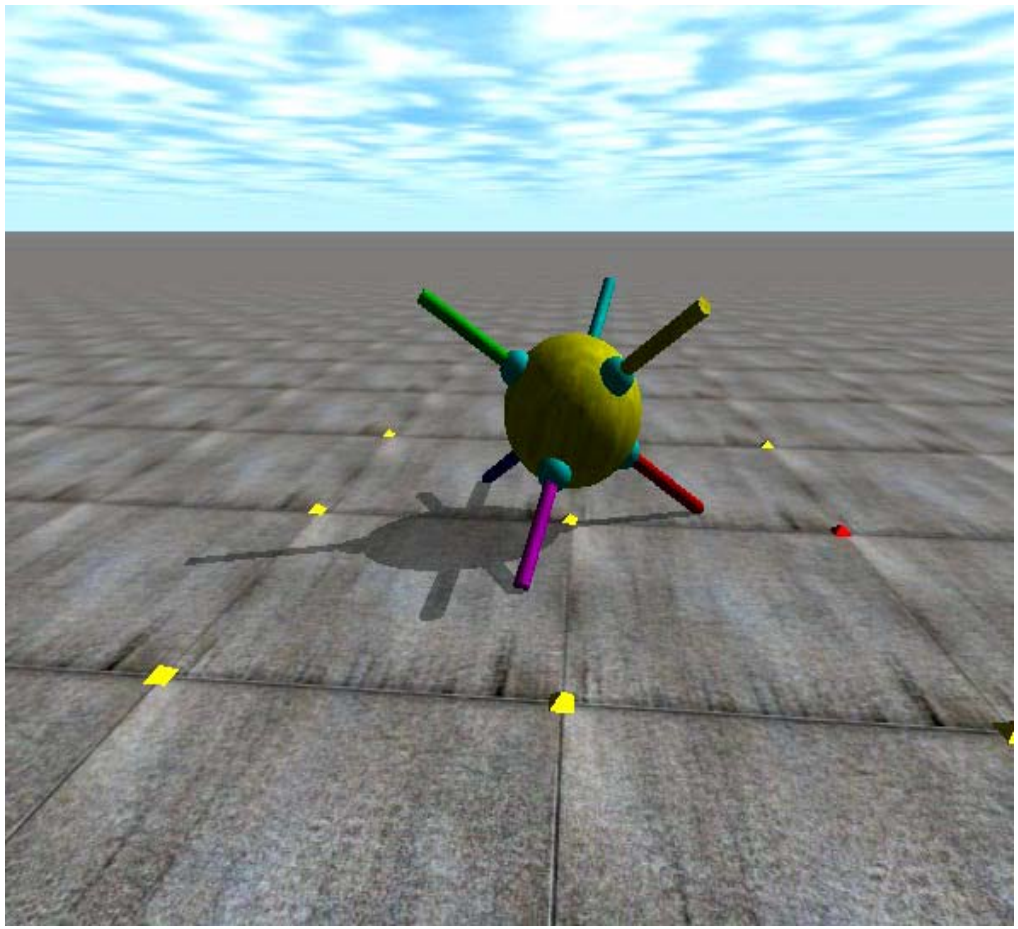
  Création : *

  Complexité : *

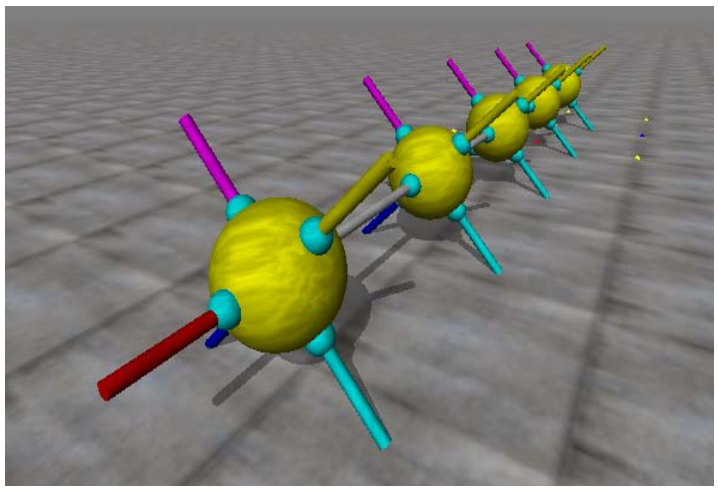
  Mouvements :

  - Déplacement par glissement.
  - Déplacement par glissement avec heuristique.
  - Déplacement par basculement.

Déplacement par roulement



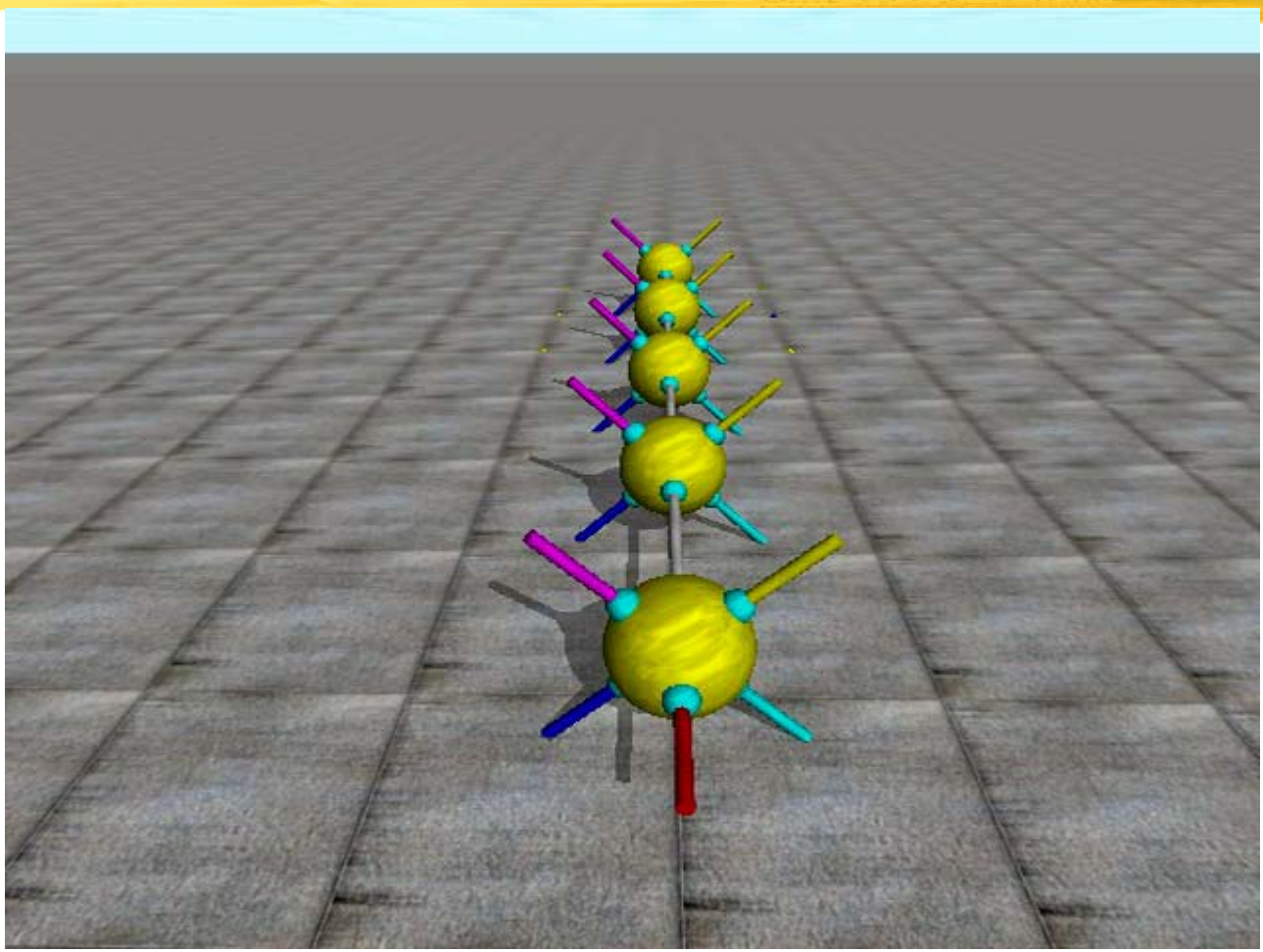
Projet Etudiant DUT Info



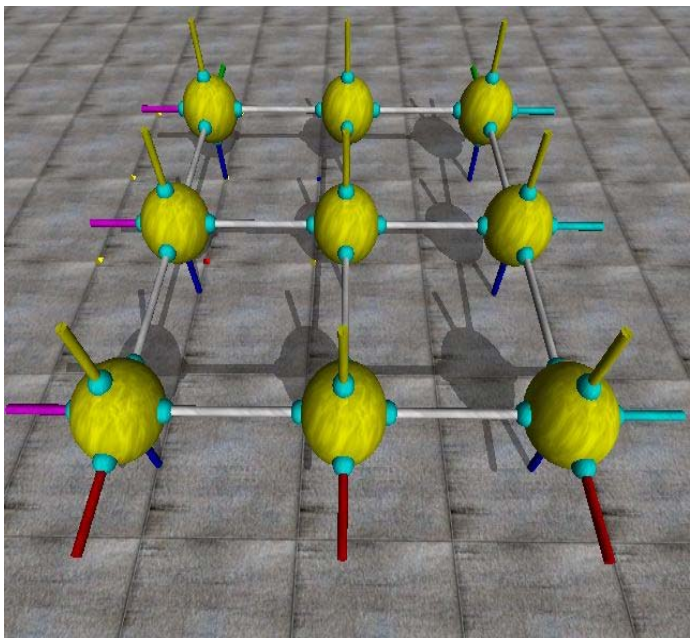
- Rangée
  - Création : *
  - Complexité : *

  Mouvements :
  - Marche pas à pas
  - Marche fluide
  - Retournement

Retournement de la rangée



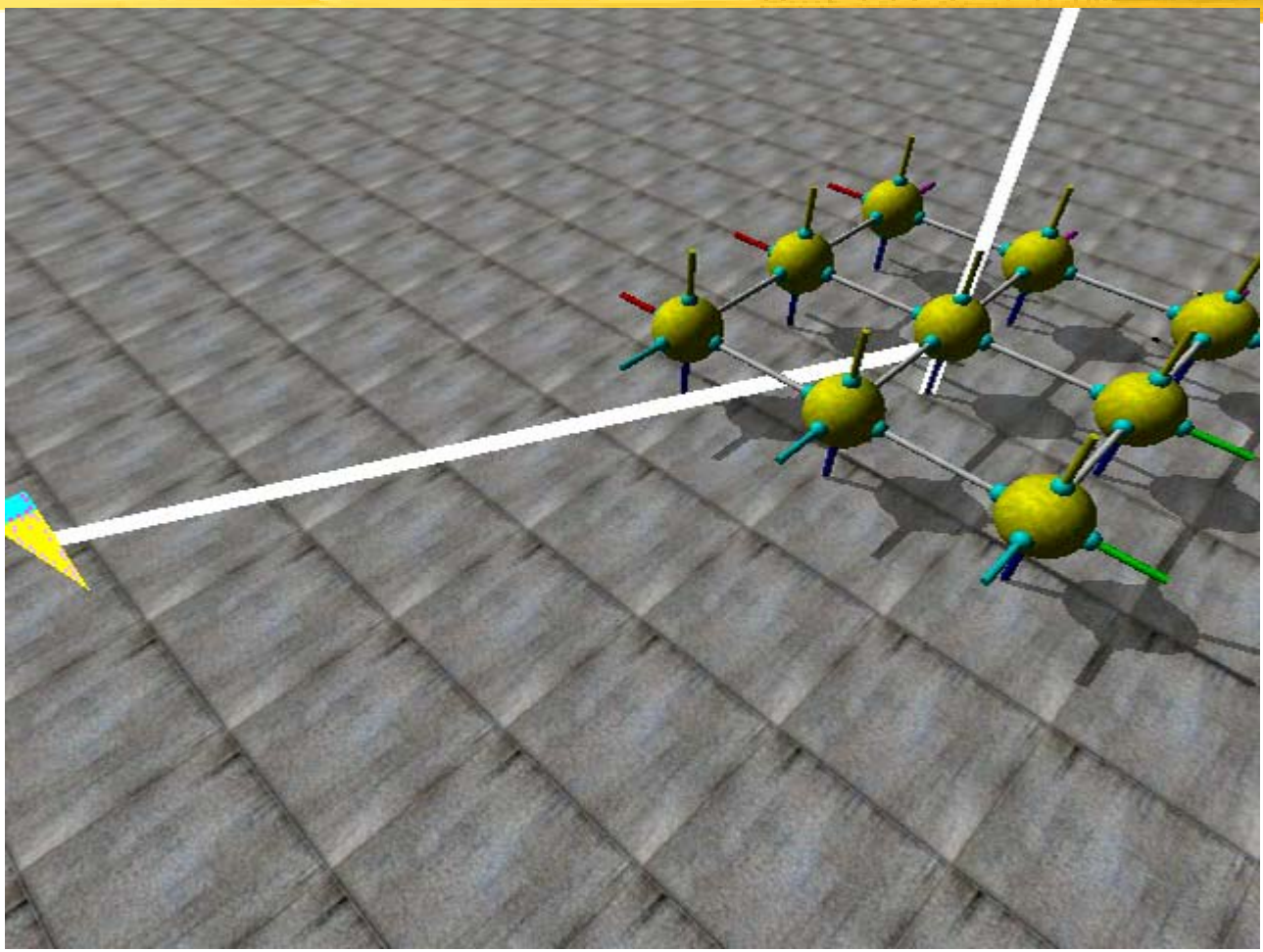
Projet Etudiant DUT Info



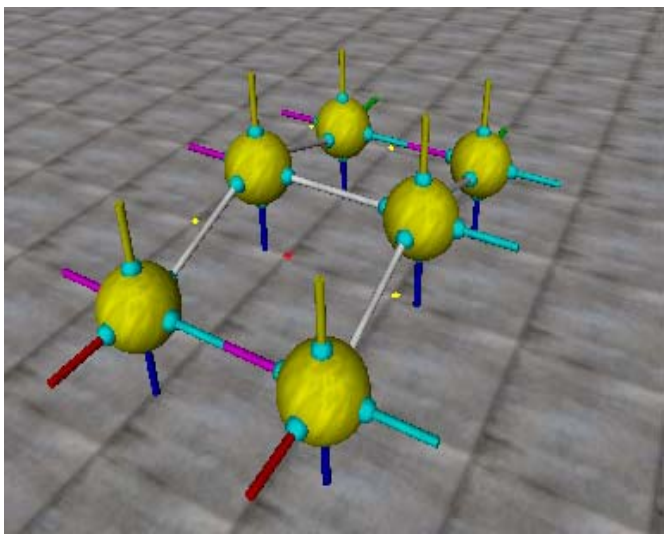
- Tapis
  - Création : *
  - Complexité : *

  Mouvements :
  - - Marche en avant

Tapis avec heuristique



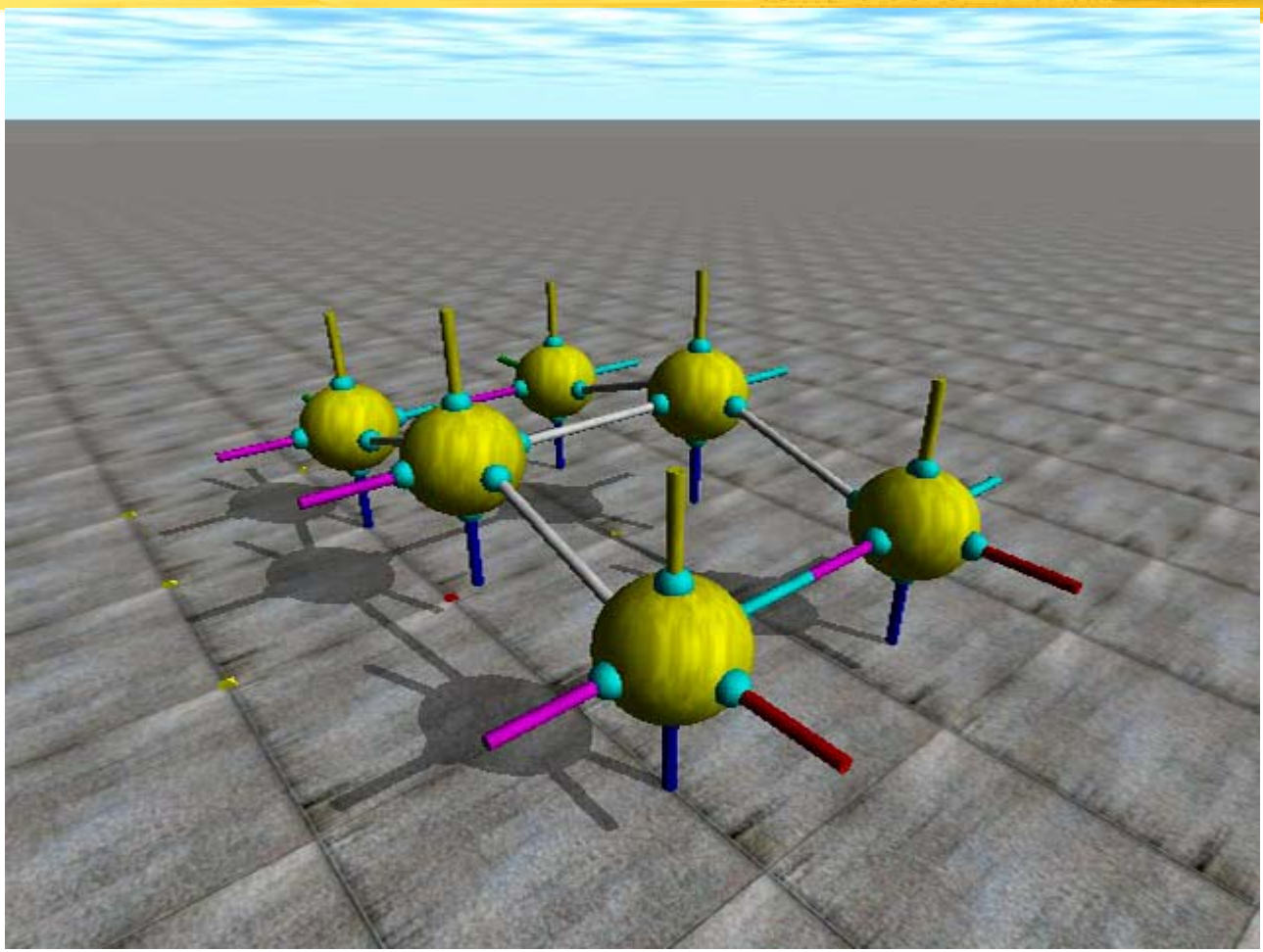
Projet Etudiant DUT Info



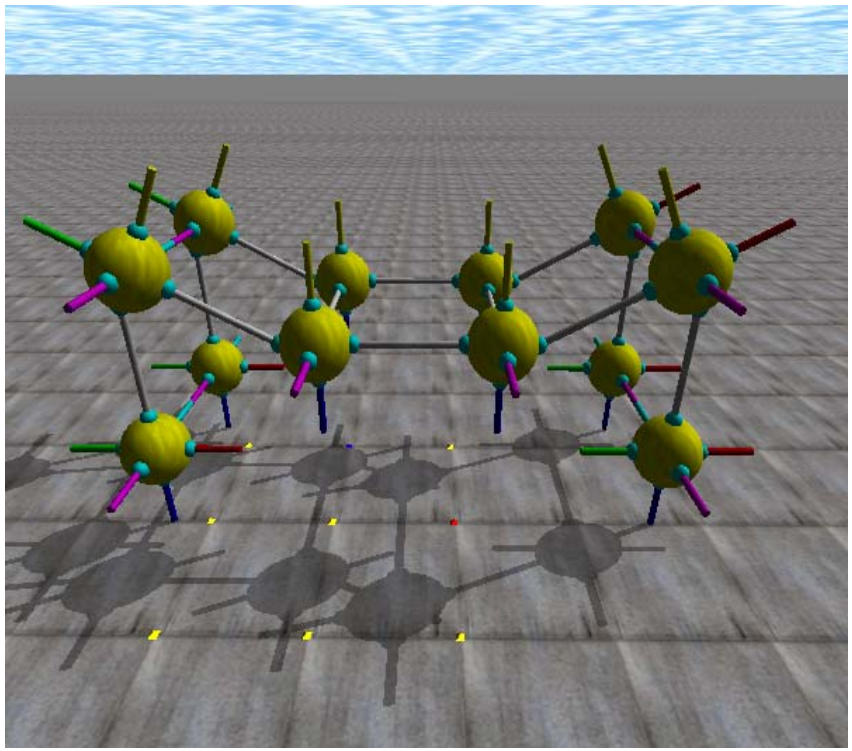
- Walker
  - Création : *
  - Complexité : **

  Mouvements :
  - Marche en avant
  - virage droite et gauche

Walker qui rame



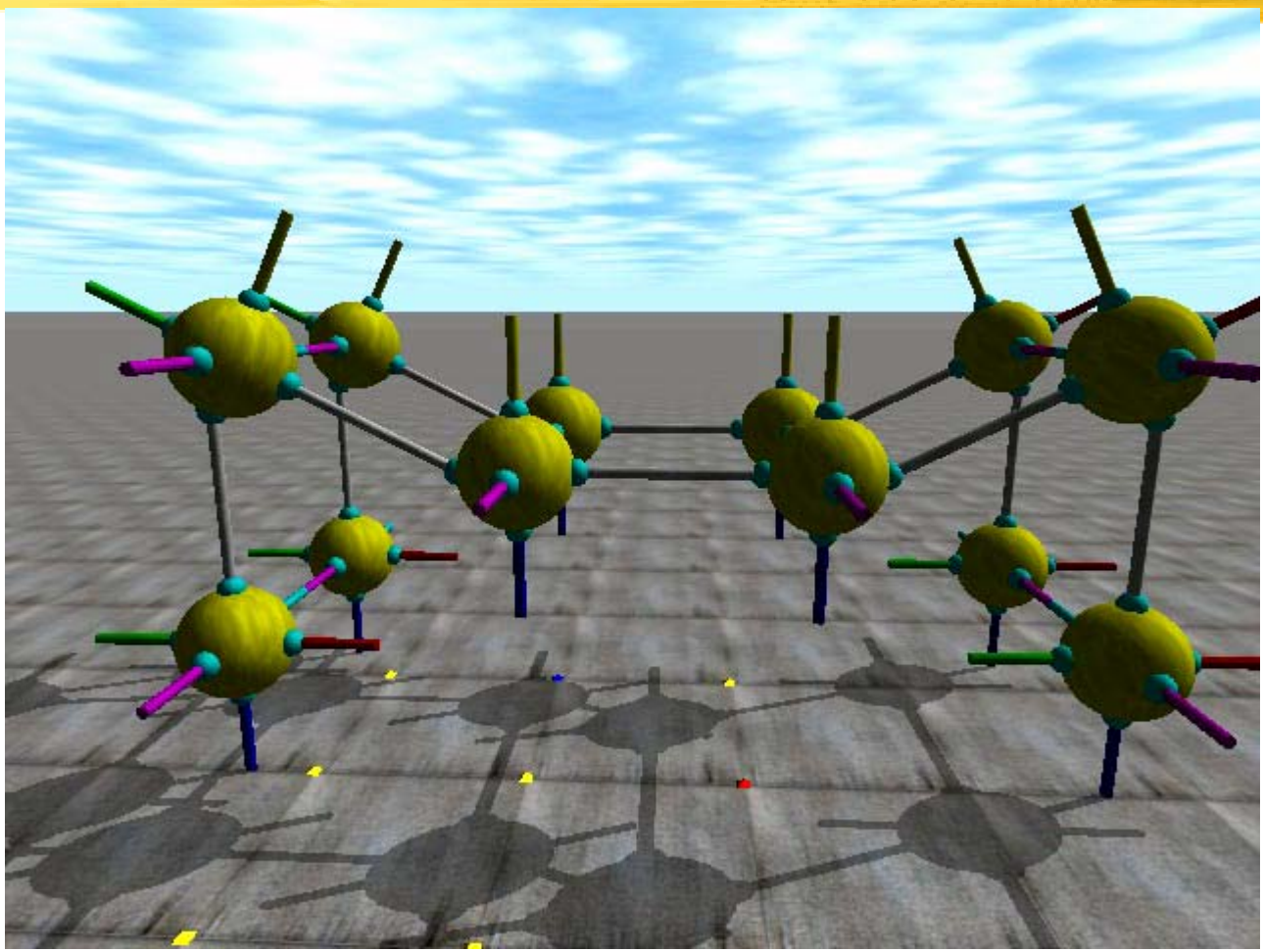
Projet Etudiant DUT Info



- Araignée
  - Création : ***
  - Complexité : ***

  Mouvements :
  - - Marche en avant
  - - Récupération d'objets

Saisie d’objet par l’araignée



Plan

- Introduction
  - Le composant MAAM
  - Le projet MAAM
  - L'API MAAM
  - Les modes de communication
- La simulation
  - Pourquoi simuler
  - Fonctionnalités
  - Simulation dynamique versus simulation cinématique
- Les principales caractéristiques du simulateur
  - Open Dynamics Engine
  - Übersim
  - Modifications des composants
  - Des classes utilitaires
  - La programmation par consigne
- Exemples d'utilisation du simulateur
  - Le walker
  - L'araignée
  - Autres molécules …
- **Futurs développements**
  - **Simulation de capteurs**
  - **Vers un simulateur distribué**
- Perspectives et conclusions

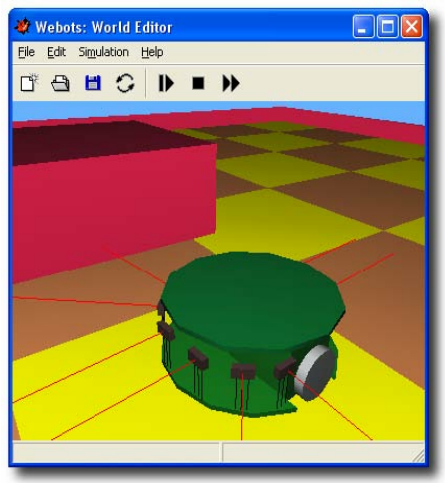
Webots

Commercial

Robots à roues et à pattes

Des modèles des robots existants

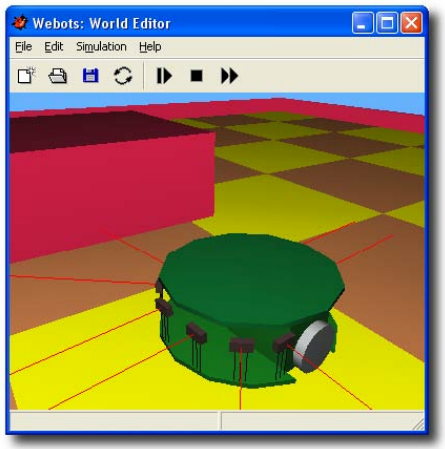
Une panoplie de capteurs



KHEPERA 2D - WEBOTS

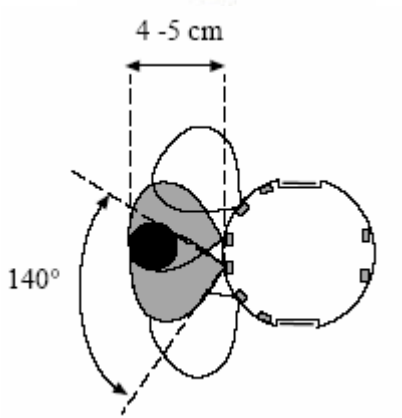
Feature	*Khepera 2D*	*Webots*
Developers	**O. Michel, M. Cergues**	**O. Michel, J. Porta**
History	**ancestor**	**successor**
Copyright	**free software**	**Cyberbotics S.A.**
Simulated robots	**Khepera (basic mod-ule only)**	**Khepera with some extension modules, …, customized mobile robot (future)**
Dimensionality	**2D**	**3D**
Interaction	**kinematic**	**Dynamic (ODE)**
Supervisor	**no**	**yes**
Transfer to real robot	**yes, serial protocol**	**yes, serial protocol and download**
Acceleration ratio	**20**	**38**

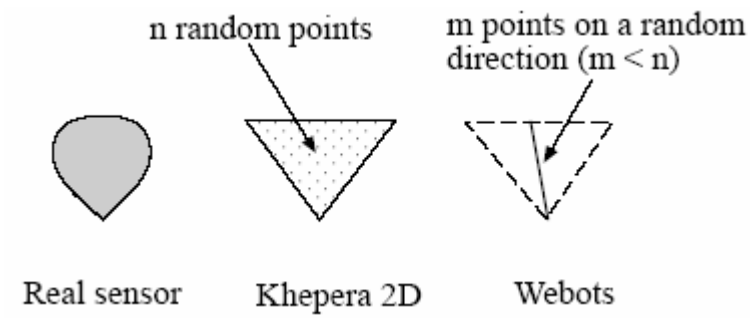
A Sensor based Simulator



Real sensors :
Proximity signal is a function of:
• Object IR reflexivity (color, surface,...).
• Type of sensor (emitter/receiver pair).
• Sensitivity of the sensor (e.g. 50% differences from data sheet for SFH-900).

Simulated sensors :
Weighted sum of n or m random points.
• Reflexivity is a function of object's color.
• All sensors are equal but ±10% of white noise



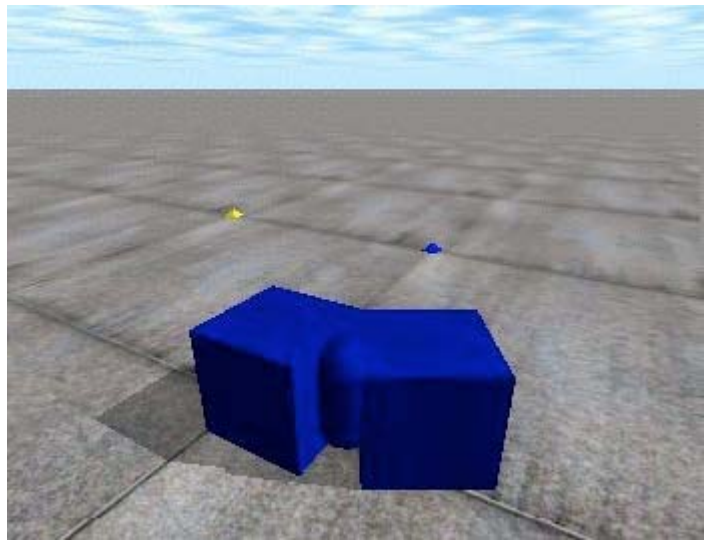


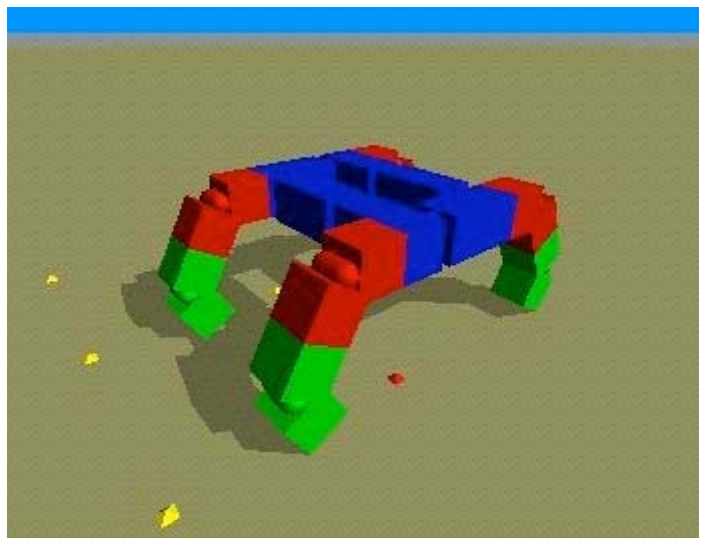
ADAM

http://lslwww.epfl.ch/birg/PROJECTS/daniel_marbach/projet.shtml
Goals
• Develop a simulator for modular robots
• Develop a script to build the robots
• Implement a genetic algorithm (GA)

Modular robots
– No self-reconfiguration
– No cycles
– Homogenous

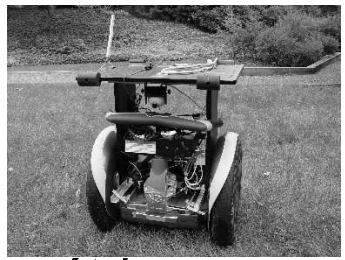
Modules are like LEGO blocks: Each type has certain qualities and positions where it is possible to put another building stone
The homogeneous component : only the Hinge Module is tested



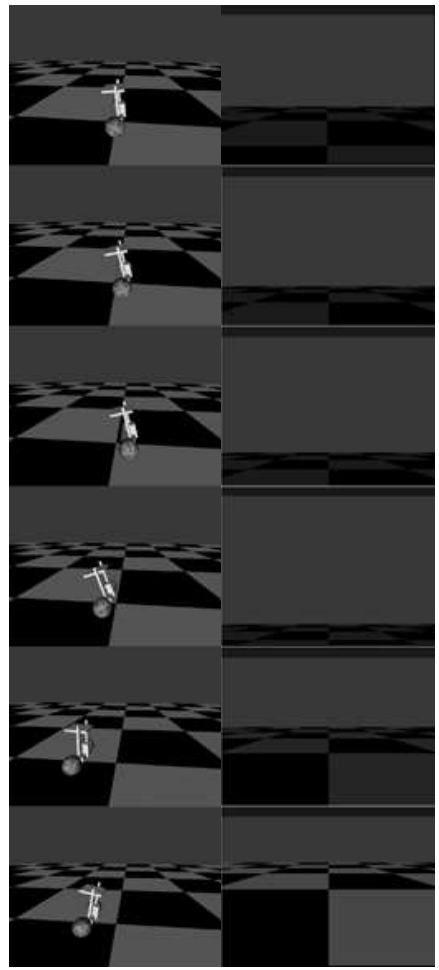


Übersim Simulation tool Version 2.0

- Code indisponible. Uniquement description dans un article. Pas de comparaison avec des robots rééls.
- Robot gouverné par la vision
  - Segway RPM voire AIBO
  - Boucle de contrôle à haute fréquence

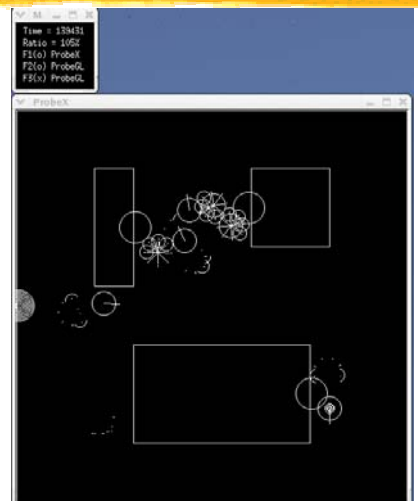


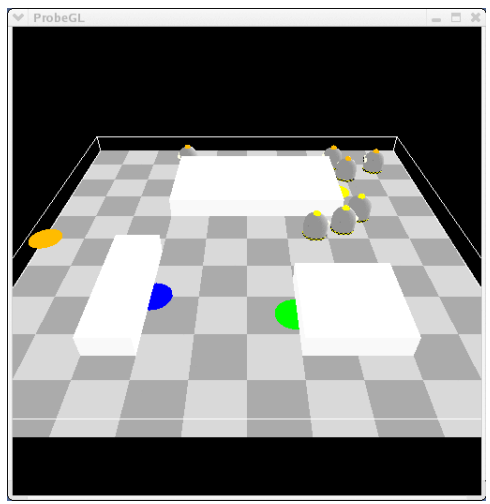
- Petit nombre de capteurs : Caméra et Odométrie
- Le serveur contient les géométries, les capteurs, les actionneurs.
- Modélisation des délais dans le serveur:
  - Délais concernant les capteurs et les actionneurs
  - Plusieurs modèles de délais disponibles
  - Gestion d'une file d'attente par client
- Le client se charge de l'auto-instanciation (transmission de sa description en XML) et du traitement des messages des capteurs et de l'envoi des messages vers les actionneurs en fonction du programme de contrôle.



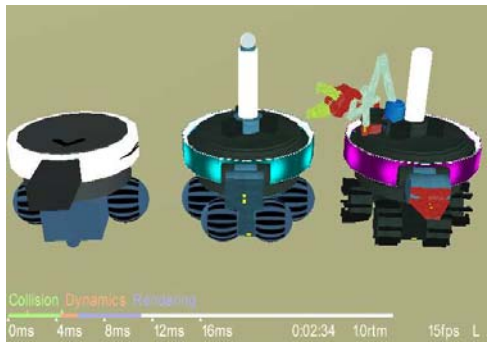
SPEAR

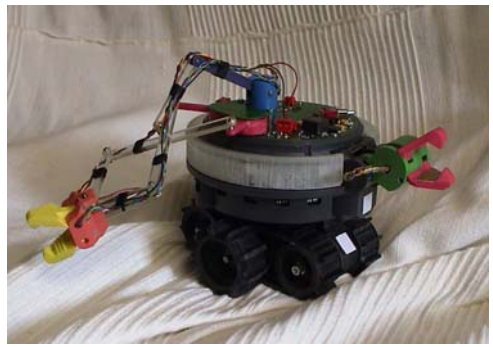
- Permet de contrôler plusieurs agents (animats)
- Utilise O.D.E.
- Le robot de référence : Khépéra
- Capteurs simulés : contact, lumière.
- Peu documenté





SWARMBOT 3D





Pour simuler une formation de SWARMBOT.

Simulation multi-modèles : modèle basique, modèle moyen et modèle détaillé.

Les sources sont disponibles.

Utilise un moteur physique commercial : Vortex.

Capteurs disponibles : proximité IR, capteur d'effort.

Perspectives

- Programmation des atomes et des molécules
- Définition du langage ;
- Environnement de développement pour la conception, l'exécution et la validation de programmes :
  - Conception d'un Interprète/compilateur ;
  - Optimisation ;
  - Étude sémantique ;
  - Débuggeur ;
  - ...
- Production de code pour les atomes physiques.