LSL

Guide des articulations

par Bestmomo Lagan

Version 1.1

02/11/2012

***Avertissement** : Ce guide n'est pas un document officiel de Second Life. Il n'engage que la responsabilité de son auteur.*

Table des matières

1.Introduction....3

2.Les axes....4

2.1Créer un axe....4
2.2Multiplier les axes....8
2.3Positionner les axes....8

3.Le setup....10

3.1Le script de setup....10
3.2Liaison des primitives avec les axes....17
3.3Positionnement dans le second état....18

4.Mise en action....21

4.1Le script de mise en action....21
4.2Les paramètres de fonctionnement....28
4.2.1Paramètres de durée....28
4.2.2Paramètres d'accès....29
4.2.3Paramètres de sons....29
4.2.4Paramètres de sensor....29
4.2.5Messages....29
4.2.6API....29

5.Setup multiples....30

5.1Un exemple de setup multiples....30
5.2La gomme....33

1. Introduction

Les rotations continuent à hanter les codeurs dans Second Life et la moindre primitive devant décrire une trajectoire circulaire peut rapidement devenir un cauchemar. J'ai tenté dans mon guide sur les rotation d'apaiser ces angoisses, de même que dans celui sur les machines qui a suivi. Mais il subsiste des zones délicates comme celui d'une banale porte pivotante ou d'un bras articulé.

D'autre part il y a ceux dont les poils se hérissent à la lecture d'une seule ligne de script, complètement hermétiques à ce langage ils s'en remettent à ceux qui « savent » mais qui hésitent aussi souvent à se lancer dans des opérations où les rotations peuvent les emporter dans des migraines tenaces.

C'est pour apporter une solution aux premiers et pour alléger la tache des seconds que j'ai entrepris de mettre au point un système « clé en mains » aussi facile à mettre en œuvre qu'il est possible dans ce domaine en 3 dimensions.

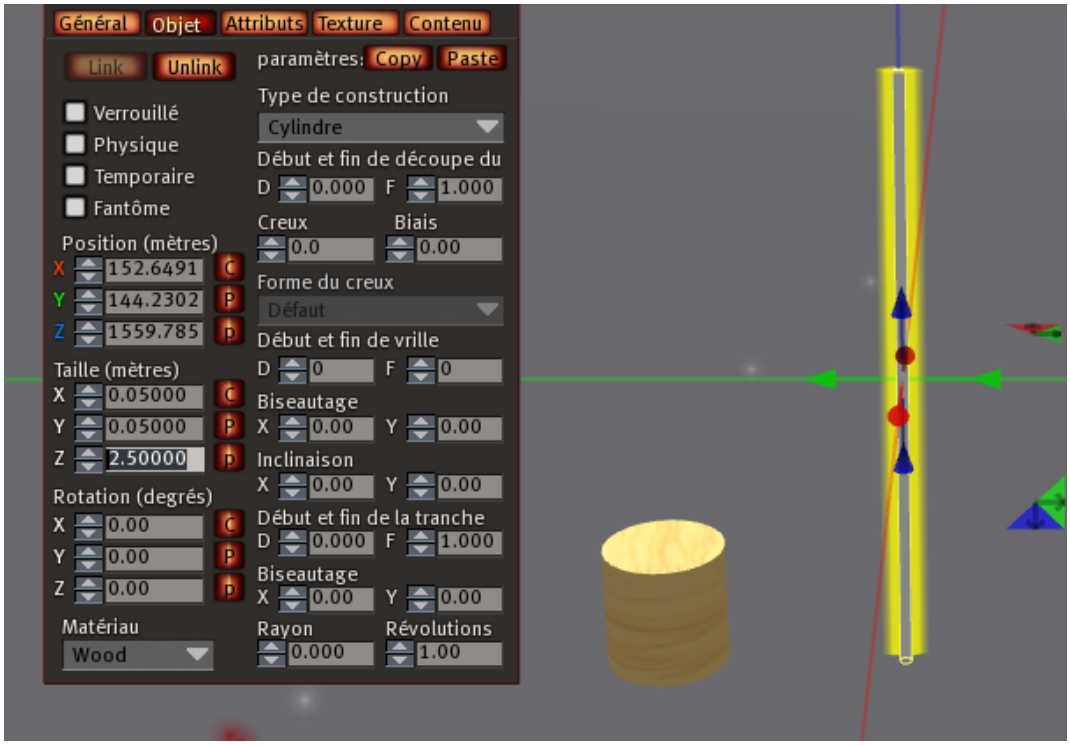
L'objet de ce petit manuel est de présenter ce système et de montrer comme l'utiliser.

2. Les axes

Le système est fondé sur l'utilisation d'axes pour représenter les rotations dans l'espace.

2.1 Créer un axe

Pour créer un axe il suffit d'utiliser un cylindre et de l'amincir suffisamment pour lui donner un aspect de « manche à balai » :



Ici vous voyez le cylindre de départ et sa version en « axe ». Il vaut mieux enlever la texture par défaut et lui donner une couleur blanche.

Une fois que c'est fait il faut équiper cet axe de son script :

```
////////////////////////////////////////////////////////////////////////////////
//                                                                            //
//                          Articulation Axe V 1.02                           //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//                  Copyright (c) 2012 by Bestmomo Lagan                      //
////////////////////////////////////////////////////////////////////////////////
//   This program is free software: you can redistribute it and/or modify     //
//    it under the terms of the GNU General Public License as published by    //
//    the Free Software Foundation, either version 3 of the License, or       //
```

```
//    (at your option) any later version.                                    //
//                                                                           //
//    Articulations is distributed WITHOUT ANY WARRANTY;                     //
//    See the GNU General Public License for more details.                   //
//                                                                           //
//    To get a copy of the GNU General Public License, see <http://www.gnu.org/licenses/>.    //
///////////////////////////////////////////////////////////////////////////////////////////////

// ---------------------------------------------
//                  Paramètres
// ---------------------------------------------

integer CANAL_AXE    = -2253600;
integer CANAL_SETUP  = -2253610;
list     COULEURS     = [<1,1,0>,<0,1,0>,<0,1,1>,<0,0,1>,<1,0,1>];

// ---------------------------------------------
//                  Variables de travail
// ---------------------------------------------
list   l_noms;
integer i_id;

// ---------------------------------------------
//                  Canal aléatoire
// ---------------------------------------------
integer genCanal() {return ~(integer)llFrand((float)DEBUG_CHANNEL);}

// ---------------------------------------------
//                  Paramètres primitive
// ---------------------------------------------
string get_nom() {return llList2String(llGetPrimitiveParams([PRIM_NAME]), 0);}
set_nom(string nom) {llSetLinkPrimitiveParamsFast(LINK_THIS, [PRIM_NAME, nom]);}
set_couleur(vector couleur) {llSetLinkPrimitiveParamsFast(LINK_THIS, [PRIM_COLOR, ALL_SIDES, couleur, 1]);}
set_texte(string texte) {llSetText(texte, <1,1,1>, 1);}

// ---------------------------------------------
//                  Test initial
// ---------------------------------------------
test_initial() {
   // Initialisations
   l_noms = [];
   set_texte("");
   // On écoute sur canal des axes
   llListen(CANAL_AXE, "", NULL_KEY, "");
   // Ecoute setup
   llListen(CANAL_SETUP + 1, "", NULL_KEY, "");
   // Le test est émis
   llWhisper(CANAL_AXE, "TEST");
   // Délai d'attente de la réponse
   llSetTimerEvent(2.0);
}

// ---------------------------------------------
//                  Etat initial de création des axes
// ---------------------------------------------
```

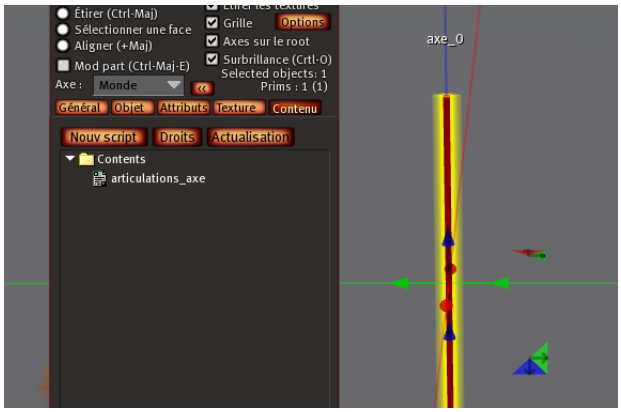
```
default
{
    state_entry() {
        test_initial();
    }

    on_rez(integer start_param){
        test_initial();
    }

    listen(integer channel, string name, key id, string message) {
        if(channel == CANAL_AXE) {
            list l = llParseString2List(message, [":"], []);
            // Retour de demande de nom
            if(llList2String(l, 0) == "TEST_BACK") l_noms += llList2String(l, 1);
            // Réception de demande de nom
            else if(llList2String(l, 0) == "TEST") llWhisper(CANAL_AXE, "TEST_BACK:" + get_nom());
        }
        else if(channel == CANAL_SETUP + 1 && message == "GO")
            state setup;
    }

    timer() {
        llSetTimerEvent(.0);
        // Réponses reçues
        if(llGetListLength(l_noms)) {
            // On passe en revue les axes présents pour connaitre l'index suivant
            list l;
            integer n = llGetListLength(l_noms);
            // Test de limite
            if(n > 5) {
                llOwnerSay("Limite dépassée ! Pas plus de 6 axes !");
                llDie();
            }
            else {
                while(n--) {
                    string nom = llList2String(l_noms, n);
                    l += (integer)llGetSubString(nom, -1, -1);
                }
                i_id = (integer)llListStatistics(LIST_STAT_MAX, l) + 1;
                set_nom("axe_" + (string)i_id);
                set_texte(get_nom());
                set_couleur(llList2Vector(COULEURS, i_id - 1));
                // On écoute sur canal de setup
                llListen(CANAL_SETUP + 1, "", NULL_KEY, "");
            }
        }
        // Pas de réponse reçue : premier axe
        else {
            set_nom("axe_0");
            set_texte("axe_0");
            i_id = 0;
            set_couleur(<1,0,0>);
            // On écoute sur canal de setup
            llListen(CANAL_SETUP, "", NULL_KEY, "");
```

```
        }
    }
}

// ---------------------------------------------
//                  Etat de setup
// ---------------------------------------------
state setup
{
    state_entry() {
        // Ecoute setup
        llListen(CANAL_SETUP + 1, "", NULL_KEY, "");
    }
    on_rez(integer start_param){
        llResetScript();
    }

    listen(integer channel, string name, key id, string message) {
        //if(message == "DIE") llDie();
        list l = llParseString2List(message, ["@"], []);
        // Est-ce que c'est pour moi ?
        if((integer)llList2String(l, 1) == i_id) {
            // Réponse à demande de position et rotation
            if(llList2String(l, 0) == "ASK")
                llSay(CANAL_SETUP, "SET@" + (string)llGetPos() + "@" + (string)llGetRot());
            // Ordre de mouvement
            else if(llList2String(l, 0) == "MOVE")
                llSetLinkPrimitiveParamsFast(LINK_THIS,
                [PRIM_POSITION, (vector)llList2String(l, 2),
                PRIM_ROTATION, llGetRot() * (rotation)llList2String(l, 3)]);
        }
    }
}
```

Ce script a plusieurs fonctions :

1. Vérifier s'il existe d'autres axes présents,
2. Définir un index et afficher un texte flottant d'identification,
3. Répondre à la demande du setup sur sa position et son orientation
4. Se détruire sur l'ordre du setup.

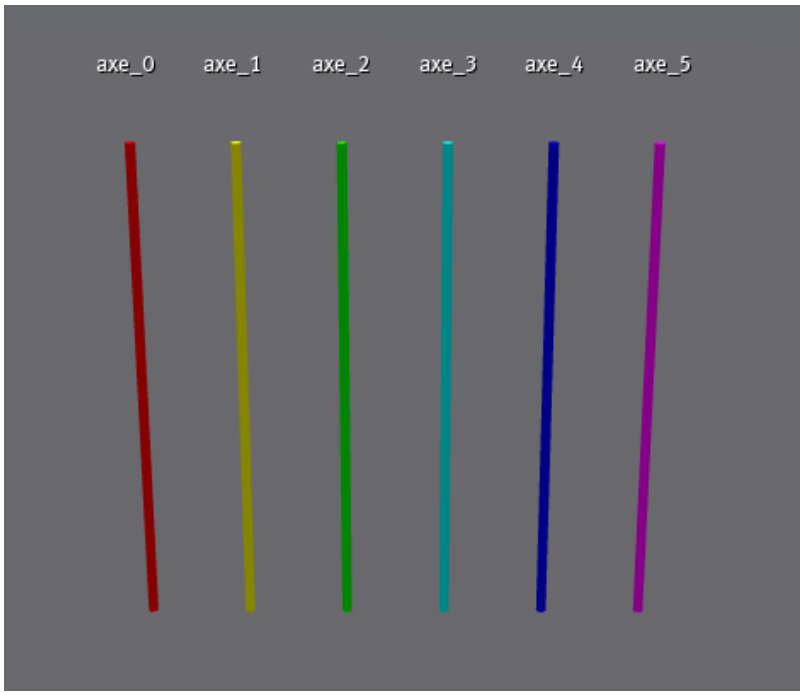
Lorsqu'on l'équipe de son script voilà ce qu'il se passe :



On voit que l'axe est devenu rouge et son texte flottant affiche « axe_0 » (remarquez que l'index commence à zéro).

2.2 Multiplier les axes

Voilà ce qu'il se passe quand on fait des copies de cet axe :



L'index progresse jusqu'à 5 et des couleurs différentes permettent de différencier les axes. Si vous essayez de créer un axe supplémentaire vous obtenez ce message :

Limite dépassée ! Pas plus de 6 axes !

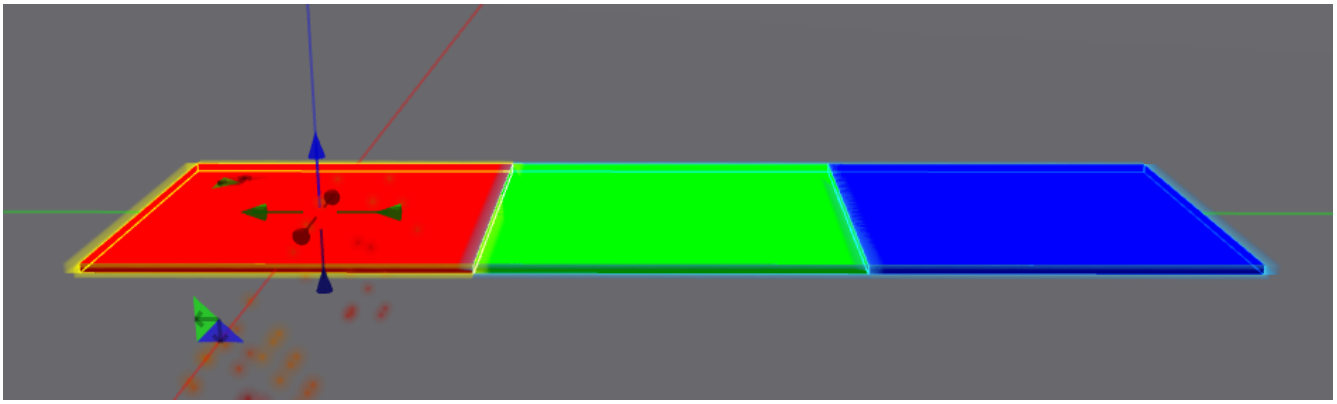
Les axes peuvent être créés par copie ou par création à partir de l'inventaire. Je vous conseille d'ailleurs de sauver votre premier axe dans votre inventaire parce que le setup détruit les axes et vous seriez obligé d'en récréer un pour un autre setup.

> ☝ *Respectez un délai d'au moins 2 secondes entre deux générations d'axe pour éviter d'en avoir deux identiques.*

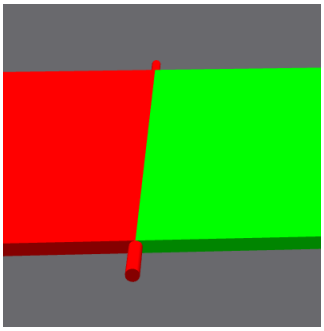
2.3 Positionner les axes

Il y a quelques règles à respecter concernant le positionnement des axes. La principale est qu'ils doivent être toujours parallèles !

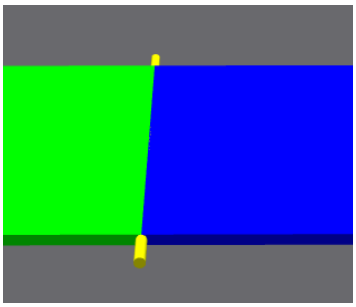
Prenons un exemple de réalisation. Voici 3 primitives liées :



On veux que ces primitives soient articulées entre elles. La primitive racine figurée en rouge ne bougera évidemment pas. On a donc besoin de deux axes à disposer entre d'une part la primitive racine rouge et la primitive enfant verte :



et entre cette dernière et la primitive enfant bleue :



Remarquez l'ordre de disposition des axes, on commence par l'axe rouge (axe_0) pour l'articulation proximale et l'axe jaune (axe_1) pour l'articulation distale. Si on avait d'autres articulations on prendrait dans l'ordre :

- axe vert (axe_2)
- axe cyan (axe_3)
- axe bleu (axe_4)
- axe magenta (axe_5)

3. Le setup

Une fois que l'objet est créé et les axes positionnés on peut passer à l'étape du setup.

3.1 Le script de setup

Le script de setup a pour mission :

1. de repérer les axes présents et d eleur demander leur position et leur orientation
2. de permettre de « lier » les axes avec les primitives qui en dépendent
3. de permettre un réglage des rotations sur tous les axes à partir d'un menu
4. de sauvegarder toutes les informations dans les primitives enfants concernées par le mouvements
5. de donner l'ordre aux axes de se détruire et de se détruire lui-même

Voici ce script :

```
////////////////////////////////////////////////////////////////////////////////
//                                                                            //
//                          Articulation Setup V 1.02                         //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//                  Copyright (c) 2012 by Bestmomo Lagan                      //
////////////////////////////////////////////////////////////////////////////////
//   This program is free software: you can redistribute it and/or modify     //
//    it under the terms of the GNU General Public License as published by    //
//    the Free Software Foundation, either version 3 of the License, or       //
//    (at your option) any later version.                                     //
//                                                                            //
//    Articulations is distributed WITHOUT ANY WARRANTY;                      //
//    See the GNU General Public License for more details.                    //
//                                                                            //
//    To get a copy of the GNU General Public License, see <http://www.gnu.org/licenses/>.    //
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//                  Parameters that you can change                            //
////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////
//          Dont change code below unless you know what you do                //
////////////////////////////////////////////////////////////////////////////////

// ------------------------------------------------
//                Variables de travail
// ------------------------------------------------
integer    i_canal_menu;
integer    i_step;
float      f_angle;
integer    i_menu;
```

```
// Valeurs d'état intermédiaire de la position de l'axe
list       l_v_centre_state_int;

// Axe
vector     v_axe;

// Links primitives avec déjà un setup
list       l_link_setup;

// Id de ce setp
string     s_setup = "0";

// -----------------------------------------------
//                  Variables de paramétrage
// -----------------------------------------------
// Numéro de liaison de la prim
list       l_link;

// Position des axes pour les deux états
list       l_v_centre_state_1;
list       l_v_centre_state_2;

// Rotation des prims pour les deux états
list       l_r_rot_state_1;
list       l_r_rot_state_2;

// Position des prims pour les deux états
list       l_v_pos_state_1;
list       l_v_pos_state_2;

// Rotation des vecteurs tournants
list                                                                                    l_vec_rot                =
[ZERO_ROTATION,ZERO_ROTATION,ZERO_ROTATION,ZERO_ROTATION,ZERO_ROTATION,ZERO_ROTA
TION];

// -----------------------------------------------
//                  Paramètres à ne pas toucher
// -----------------------------------------------
integer    CANAL_SETUP    = -2253610;

// -----------------------------------------------
//                  Canal aléatoire
// -----------------------------------------------
integer genCanal() {
   return ~(integer)llFrand((float)DEBUG_CHANNEL);
}
// -----------------------------------------------
//                  Gestion des menus
// -----------------------------------------------
// Envoi du menu à l'owner
set_menu(string texte, list menu) {
   llDialog(llGetOwner(), texte, menu, i_canal_menu);
}
// Menu de sélection des liaisons
```

```
menu_liaisons() {
    list l = ["Next"];
    if(i_step) l = ["Back"] + l;
     set_menu("\nClick the primitive linked to axe_" + (string)i_step + "\n(click again to unlink) \nand click on \"Next\" when done", l);
}
// Menu de réglage rotation
menu_setup() {
     set_menu("\nAdjust prim link_axe_" + (string)i_step + " \nand click on another one when it's done\nor \"End\" if all prims are adjusted"
        , ["-1","End","+1","-5","< >","+5","-20","< >","+20","-90","< >","+90"]);
}

// ----------------------------------------------
//                  Passage de position globale <-> locale
// ----------------------------------------------
vector global_to_local(vector position) {
    return (position - llGetRootPosition()) / llGetRootRotation();
}
vector local_to_global(vector position) {
    return llGetRootPosition() + position * llGetRootRotation();
}

// ----------------------------------------------
//                  Passage de rotation globale <-> locale
// ----------------------------------------------
rotation rot_to_local(rotation rot) {
    return rot / llGetRootRotation();
}
rotation rot_to_global(rotation rot) {
    return rot * llGetRootRotation();
}

// ----------------------------------------------
//                  Paramètres primitive
// ----------------------------------------------
string get_texte(integer id) {
    return llList2String(llGetLinkPrimitiveParams(id, [PRIM_TEXT]), 0);
}
set_texte(integer id, vector col) {
    llSetLinkPrimitiveParamsFast(id, [PRIM_TEXT, "Link to axe_" + (string)i_step, col, 1]);
}
unset_texte(integer id) {
    llSetLinkPrimitiveParamsFast(id, [PRIM_TEXT, "", <0,0,0>, 0]);
}
vector get_pos(integer id) {
    return llList2Vector(llGetLinkPrimitiveParams(id, [PRIM_POS_LOCAL]), 0);
}
rotation get_rot(integer id) {
    return llList2Rot(llGetLinkPrimitiveParams(id, [PRIM_ROT_LOCAL]), 0);
}

// ----------------------------------------------
//                  Position instantanée
// ----------------------------------------------
```

```
set_pos_inst() {
    integer n = llGetListLength(l_v_centre_state_1);
    integer i;
    // Initialisations
    l_v_pos_state_2 = [];
    l_r_rot_state_2 = [];
    l_v_centre_state_2 = [];
    // Boucle
    for(; i < n; ++i) {
        integer id = llList2Integer(l_link, i);
        l_v_pos_state_2 += get_pos(id);
        l_r_rot_state_2 += get_rot(id);
        l_v_centre_state_2 += llList2Vector(l_v_centre_state_int, i);
    }
    f_angle = 0;
}

// ------------------------------------------------
//        Vérification des setup déjà présents
// ------------------------------------------------
get_infos() {
    // Recherche des primitives avec déjà un setup
    integer n = llGetNumberOfPrims();
    list l_id;
    integer i;
    for(i = 2; i <= n; ++i) {
        string s = get_texte(i);
        if(s != "") {
            if(llGetSubString(s, 0, 0) == "!") {
                l_link_setup += i;
                l_id += (integer)llGetSubString(s, 1, 1);
            }
        }
    }
    // Détermination de l'id de ce setup
    if(llGetListLength(l_id)) s_setup = (string)((integer)llListStatistics(LIST_STAT_MAX, l_id) + 1);
}

// ------------------------------------------------
//        Sauvegardes informations
// ------------------------------------------------
save_infos() {
    integer n = llGetListLength(l_link);
    while(n--) {
        string s = "!" + s_setup + (string)n
            + "@" + compact_info(llList2Vector(l_v_centre_state_1, n))
            + "@" + compact_info(llList2Vector(l_v_centre_state_2, n))
            + "@" + compact_info(llRot2Euler(llList2Rot(l_r_rot_state_1, n)) * RAD_TO_DEG)
            + "@" + compact_info(llRot2Euler(llList2Rot(l_r_rot_state_2, n)) * RAD_TO_DEG)
            + "@" + compact_info(llList2Vector(l_v_pos_state_1, n))
            + "@" + compact_info(llList2Vector(l_v_pos_state_2, n))
            + "@" + compact_info(llRot2Euler(llList2Rot(l_vec_rot, n)) * RAD_TO_DEG);
        llSetLinkPrimitiveParamsFast(llList2Integer(l_link, n), [PRIM_TEXT, s, ZERO_VECTOR, 0]);
    }
}
```

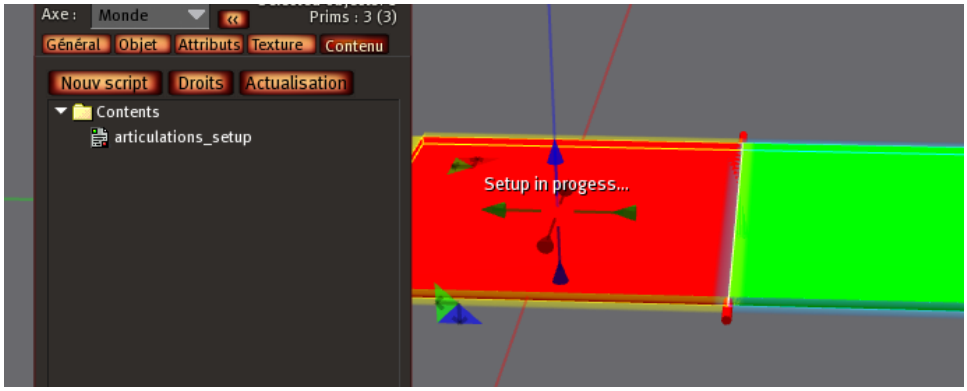
```
string compact_info(vector v) {
   return get_part(v.x) + "," + get_part(v.y) + "," + get_part(v.z);
}
string get_part(float f) {
   if(f == .0) return "0";
   else return llGetSubString((string)f, 0, 5);
}

// ---------------------------------------------
//                  Récupération des infos des axes
// ---------------------------------------------
default
{
   state_entry() {
      llOwnerSay(llGetScriptName() + " ready.");
      llSetText("Setup in progess...", <1,1,1>, 1);
      // Récupération infos setup antérieurs éventuels
      get_infos();
      // Ecoute pour les axes
      llListen(CANAL_SETUP, "", NULL_KEY, "");
      // Message de début de setup
      llSay(CANAL_SETUP + 1, "GO");
      // Petit délai
      llSleep(2.0);
      // Message au premier axe
      llSay(CANAL_SETUP + 1, "ASK@0");
      llSetTimerEvent(2.0);
   }

   listen(integer channel, string name, key id, string message) {
      // Réception messages des axes
      list l = llParseString2List(message, ["@"], []);
      // Récupération des données de l'axe
      if(llList2String(l, 0) == "SET") {
         vector v_centre_global = (vector)llList2String(l, 1);
         rotation r_global = (rotation)llList2String(l, 2);
         vector v_centre_local = global_to_local(v_centre_global);
         rotation r_local = rot_to_local(r_global);
         l_v_centre_state_1 += v_centre_local;
         l_v_centre_state_int += v_centre_local;
         if(!i_step) v_axe = llVecNorm(v_centre_local + <.0,.0,1.0> * r_local  - v_centre_local);
         // Message axe suivant
         llSay(CANAL_SETUP + 1, "ASK@" + (string)(++i_step));
         llSetTimerEvent(2.0);
      }
   }

   timer() {
      llSetTimerEvent(.0);
      if(i_step == 0) llOwnerSay("I didn't find an axis !");
      else
         state liaisons;
   }
}
```

```
// ------------------------------------------------
//                  Définition des liaisons
// ------------------------------------------------
state liaisons
{
   state_entry() {
      // Message pour axe_0
      i_step = 0;
      llOwnerSay("Click on the primitive linked to axe_0");
   }

   touch_start(integer total_number) {
      integer id = llDetectedLinkNumber(0);
      // On ne doit pas lier le root ni deux fois la même primitive !
      if(id != 1 && llListFindList(l_link, [id]) == -1) {
         set_texte(id, <1,1,1>);
         // Mémorisation des informations
         l_link += id;
         l_v_pos_state_1 += get_pos(id);
         l_r_rot_state_1 += get_rot(id);
         // Nouvelle liaison
         if(++i_step < llGetListLength(l_v_centre_state_1))
            llOwnerSay("Click on the primitive linked to axe_" + (string)i_step);
         // Fin des liaisons
         else {
            i_step = 0;
            state etats;
         }
      }
   }
}

// ------------------------------------------------
//                  Réglage du deuxième état
// ------------------------------------------------
state etats
{
   state_entry() {
      // Canal pour le menu
      i_canal_menu = genCanal();
      // Ecoute pour le menu
      llListen(i_canal_menu, "", llGetOwner(), "");
      // Message
      llOwnerSay("Set all elements for state 1 and click on the primitive you want to adjust for state 2");
   }

   touch_start(integer total_number) {
      integer i_link = llDetectedLinkNumber(0);
      integer i = llListFindList(l_link, [i_link]);
      integer i_bis = llListFindList(l_link_setup, [i_link]);
      if(~i && i_bis == -1) {
         i_step = 0;
         integer n = llGetListLength(l_v_centre_state_1);
         for(; i_step < n; ++i_step) set_texte(llList2Integer(l_link, i_step), <1,1,1>);
         i_step = i;
```

```
        set_texte(i_link, <1,0,0>);
        set_pos_inst();
        if(!i_menu) {
            i_menu = TRUE;
            menu_setup();
        }
    }
}

listen(integer channel, string name, key id, string message) {
    // Réception messages des menus
    if(channel == i_canal_menu) {
        // Touche neutre
        if(message == "< >" || message == " ") menu_setup();
        // Fin du réglage
        else if(message == "End") {
            // Mémorisation état 2
            set_pos_inst();
            // Effacement des textes flottants
            unset_texte(1);
            integer n = llGetListLength(l_v_centre_state_1);
            while(n--) unset_texte(llList2Integer(l_link, n));
            // Destruction des axes
            llSay(CANAL_SETUP + 1, "DIE");
            // Enregistrement des informations
            save_infos();
            // Fin
            llOwnerSay("Drop \"Articulation action\" script in object inventory");
            llRemoveInventory(llGetScriptName());
        }
        // Réglage en cours
        else {
            float f_valeur = (float)message;
            f_angle += f_valeur;
            integer n = llGetListLength(l_v_centre_state_1);
            integer i = i_step;
            // Mise en mouvement global
            for(; i < n; ++i) {
                integer id_link = llList2Integer(l_link, i);
                rotation r = llAxisAngle2Rot(v_axe, f_angle * DEG_TO_RAD);
                vector v_tournant = llList2Vector(l_v_pos_state_2, i) - llList2Vector(l_v_centre_state_2, i_step);
                if(i == i_step) l_vec_rot = llListReplaceList(l_vec_rot, [r], i, i);
                llSetLinkPrimitiveParamsFast(id_link, [
                    PRIM_POSITION, llList2Vector(l_v_centre_state_2, i_step) + v_tournant * r,
                    PRIM_ROT_LOCAL, llList2Rot(l_r_rot_state_2, i) * r]);
                // Encore un axe
                if(i < n - 1) {
                    v_tournant = llList2Vector(l_v_centre_state_2, i + 1) - llList2Vector(l_v_centre_state_2, i_step);
                    vector axe_pos = llList2Vector(l_v_centre_state_2, i_step) + v_tournant * r;
                    // Mémorisation de la position
                    l_v_centre_state_int = llListReplaceList(l_v_centre_state_int, [axe_pos], i + 1, i + 1);
                    // Ordre de mouvement à l'axe
                    llSay(CANAL_SETUP + 1, "MOVE@" + (string)(i + 1)
                        + "@" + (string)(local_to_global(axe_pos))
                        + "@" + (string)ZERO_ROTATION
```

```
                    );
                }
            }
            menu_setup();
        }
    }
}
```

Il suffit de mettre ce script dans l'inventaire de la primitive racine pour démarrer le setup :

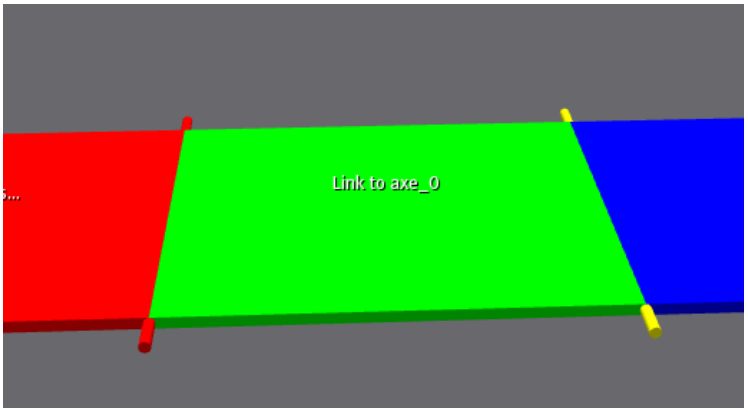


3.2 Liaison des primitives avec les axes

Un message apparaît dans le Chat :

Click on the primitive linked to axe_0

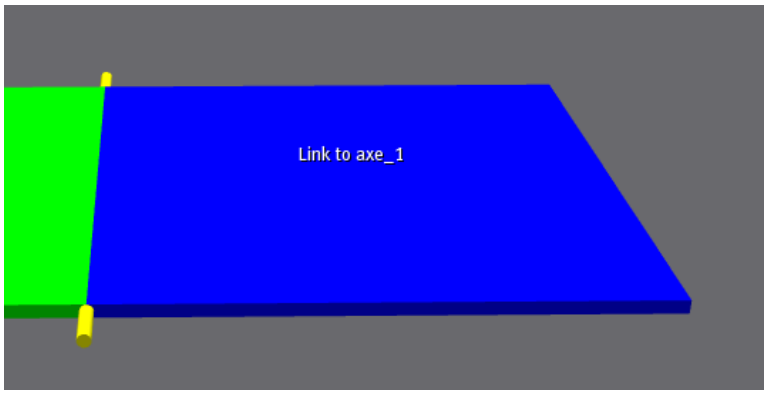
On vous invite à cliquer sur la primitive qui dépend du premier axe (axe_0). On clique donc sur la primitive verte :



Le texte flottant de cette primitive nous informe que l'action a bien été prise en compte et nous rappelle avec quel axe elle est liée. On reçoit un nouveau message :

Click on the primitive linked to axe_1

Un nouveau clic, cette fois sur la primitive bleue :



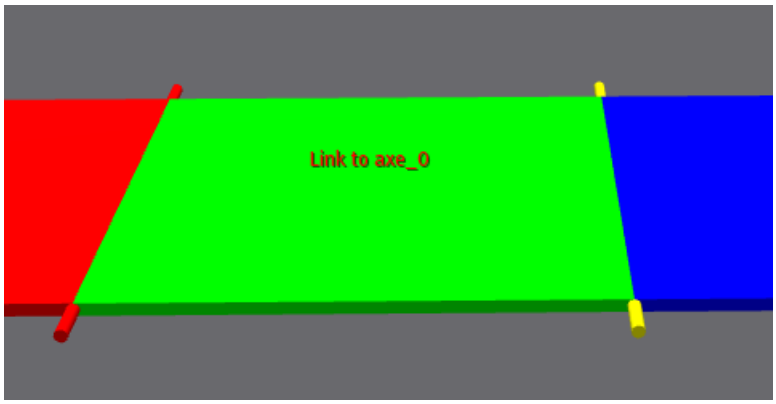
3.3 Positionnement dans le second état

Vous recevez à nouveau un message :

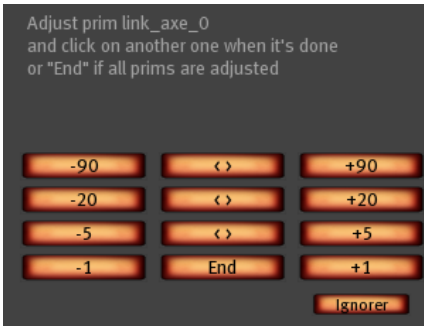
Set all elements for state 1 and click on the primitive you want to adjust for state 2

En fait tous les éléments doivent normalement être déjà correctement positionnés dans le premier état avec les axes bien en place. C'est juste un rappel. Si ce n'est pas le cas reprenez la procédure depuis le début.

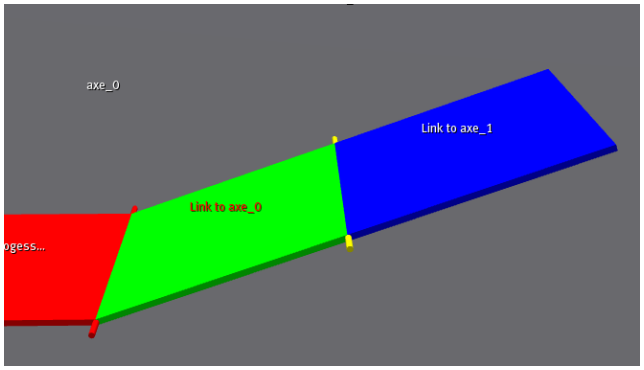
Quand on clique sur une primitive son texte flottant passe au rouge pour nous informer que notre action a été efficace :



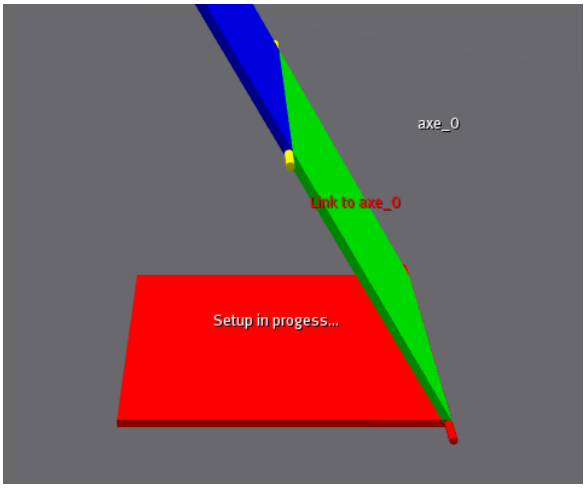
Il apparaît aussi un menu :



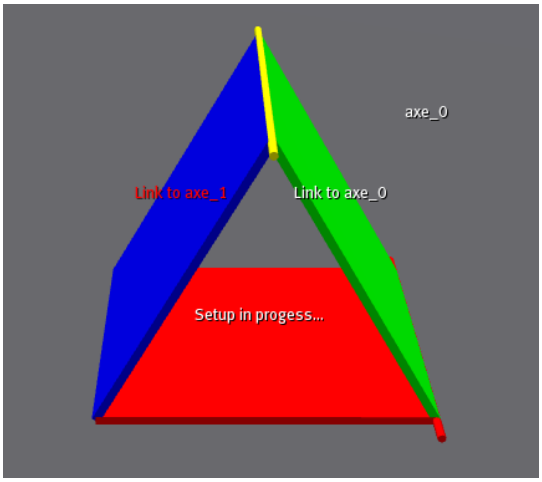
Ce menu nous sert pour régler la rotation de cette primitive sur son axe. Voici par exemple une rotation de 20 degrés :



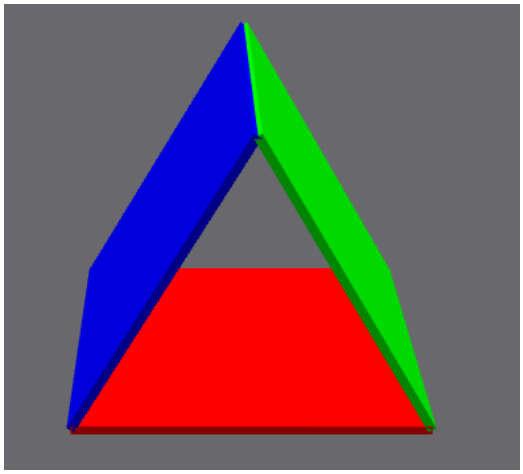
De façon logique tout ce qui suit cette primitive dans la hiérarchie des axes subit le mouvement. Continuons le réglage jusqu'à obtenir un angle de 120 degrés :



Nous cliquons ensuite sur la primitive bleue, son texte flottant vire au rouge, le menu reste disponible, appliquons aussi une rotation de 120 degrés :



Nous avons fini le réglage, il ne nous reste plus qu'à cliquer sur le bouton « End » du menu :



Les axes disparaissent, ainsi que le script de setup dans l'inventaire.

Les paramètres du setup ont été sauvegardés dans les textes flottants rendus invisibles des primitives concernées par le mouvement.

4. Mise en action

Nous en arrivons à la dernière étape, celle de mise en action du mouvement.

4.1 Le script de mise en action

Le script de mise en action à les fonctions suivantes :

1. lecture des informations de setup dans les textes flottants cachés des primitives enfants,
2. mémorisation de ces informations,
3. lecture des paramètres de fonctionnement,
4. mise en action du mouvement sur sollicitation

Voici ce script :

```
////////////////////////////////////////////////////////////////////////////////////
//                                                                                //
//                              Articulation Action V 1.04                        //
//                                                                                //
////////////////////////////////////////////////////////////////////////////////////
//                  Copyright (c) 2012 by Bestmomo Lagan                          //
////////////////////////////////////////////////////////////////////////////////////
//   This program is free software: you can redistribute it and/or modify         //
//    it under the terms of the GNU General Public License as published by        //
//    the Free Software Foundation, either version 3 of the License, or           //
//    (at your option) any later version.                                         //
//                                                                                //
//    Articulations is distributed WITHOUT ANY WARRANTY;                          //
//    See the GNU General Public License for more details.                        //
//                                                                                //
//    To get a copy of the GNU General Public License, see <http://www.gnu.org/licenses/>.    //
////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////
//                  Parameters that you can change                                //
////////////////////////////////////////////////////////////////////////////////////

// Time
float      TIME          = 2;       // Transition time in seconds
integer    AUTO          = FALSE;   // TRUE for loop mode
float      RETURNTIME    = 0;       // Time for auto return to first position (0 for no auto-return)
integer    API           = FALSE;   // True for only API

// Access
string     ACCESS        = "All";   // Access must be "All", "Group", "Owner" or "List"
list       ACCESSLIST    = [];      // Access names list if ACCESS is on "List"

// Sounds
string     SECONDSOUND   = "none";  // Sound for second position ("none" if none)
string     FIRSTSOUND    = "none";  // Sound for first position ("none" if none)
```

```
// Sensor
float     SENSOR_RANGE   = 0;       // Sensor range (0 for none), no touch action if sensor
float     SENSOR_DELAY   = 4;       // Sensor delay (no matter if no sensor)

// Messages
string    ACCESSMESSAGE  = "Sorry but your are not allowed to command this object";
string    CLICKMESSAGE   = "Come closer to command this object";

// API
integer   API_CLICK      = 111;     // Like a "click"
integer   API_AUTO_START = 112;     // Start in "auto" mode
integer   API_AUTO_STOP  = 113;     // Stop in "auto" mode
integer   API_GO_POS_1   = 114;     // Go to first position
integer   API_GO_POS_2   = 115;     // Go to second  position
integer   API_GET_POS    = 116;     // To get position
integer   API_POS_ANSWER = 117;     // POsition answer

////////////////////////////////////////////////////////////////////////////////
//              Dont change code below unless you know what you do            //
////////////////////////////////////////////////////////////////////////////////

// ------------------------------------------------
//                  Variables de paramétrage
// ------------------------------------------------
// Numéro de liaison de la prim
list      l_link;

// Position des axes pour les deux états
list      l_v_centre_state_1;
list      l_v_centre_state_2;

// Rotation des prims pour les deux états
list      l_r_rot_state_1;
list      l_r_rot_state_2;

// Position des prims pour les deux états
list      l_v_pos_state_1;
list      l_v_pos_state_2;

// Rotation des vecteurs tournants
list                                                                    l_vec_rot               =
[ZERO_ROTATION,ZERO_ROTATION,ZERO_ROTATION,ZERO_ROTATION,ZERO_ROTATION,ZERO_ROTA
TION];

// ------------------------------------------------
//                  Paramètres à ne pas toucher
// ------------------------------------------------
float     TEMPS_BASE       = .04;

// ------------------------------------------------
//                  Paramètres primitive
// ------------------------------------------------
string get_texte(integer id) {
   return llList2String(llGetLinkPrimitiveParams(id, [PRIM_TEXT]), 0);
```

```
}
vector get_pos(integer id) {
    return llList2Vector(llGetLinkPrimitiveParams(id, [PRIM_POS_LOCAL]), 0);
}

// ---------------------------------------------
//                  Variables de travail
// ---------------------------------------------
integer     i_auto;

// ---------------------------------------------
//        Test d'accès
// ---------------------------------------------
integer test_access(key id) {
    if(ACCESS == "List" && ~llListFindList(ACCESSLIST, [llKey2Name(id)])) return TRUE;
    else if(ACCESS == "Owner" && llGetOwner() == id) return TRUE;
    else if(ACCESS == "Group" && llSameGroup(id)) return TRUE;
    else if(ACCESS == "All") return TRUE;
    else return FALSE;
}

// ---------------------------------------------
//                  Interpolation
// ---------------------------------------------
rotation r_cos(rotation r0, rotation r1, float t){
    float f = (1 - llCos(t * PI)) / 2;
    float f_ang = llAngleBetween(r0, r1);
    if(f_ang > PI) f_ang -= TWO_PI;
    return r0 * llAxisAngle2Rot(llRot2Axis(r1 / r0) * r0, f_ang * f);
}

// ---------------------------------------------
//        Transition
// ---------------------------------------------
transition(integer i_back) {
    float f_step = 1.0 / (TIME / TEMPS_BASE);
    float step;
    while((step += f_step) <= 1.0) {
        integer n = llGetListLength(l_link);
        integer i;
        rotation r_tot;
        rotation r;
        vector v_centre = llList2Vector(l_v_centre_state_1 , 0);
        list l_params;
        for(; i < n; ++i) {
            if(i_back) r = r_cos(llList2Rot(l_vec_rot, i), ZERO_ROTATION, step);
            else r = r_cos(ZERO_ROTATION, llList2Rot(l_vec_rot, i), step);
            r_tot *= r;
            vector v_tournant = llList2Vector(l_v_pos_state_1, i) - llList2Vector(l_v_centre_state_1 , i);
            vector pos = v_centre + v_tournant * r_tot;
            if(i) l_params += [PRIM_LINK_TARGET, llList2Integer(l_link, i)];
            l_params += [PRIM_POSITION, pos, PRIM_ROT_LOCAL, llList2Rot(l_r_rot_state_1, i) * r_tot];
            // Encore un axe
            if(i < n - 1) {
                v_tournant = llList2Vector(l_v_centre_state_1 , i + 1) - llList2Vector(l_v_centre_state_1 , i);
```

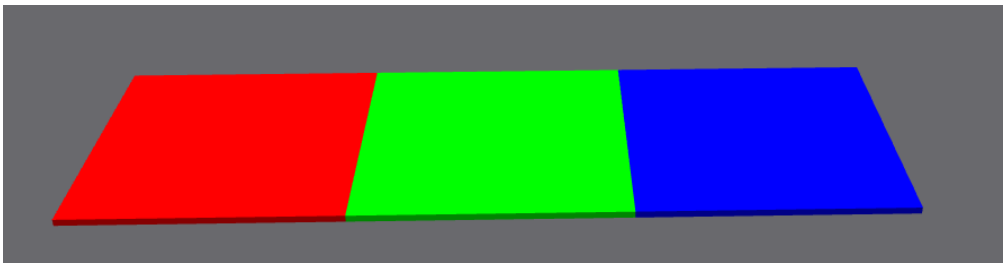
```
            v_centre += v_tournant * r_tot;
          }
        }
        llSetLinkPrimitiveParamsFast(llList2Integer(l_link, 0), l_params);
        llSleep(TEMPS_BASE);
    }
}

// ---------------------------------------------
//       Manoeuvre de l'objet
// ---------------------------------------------
// Test de première position
integer test_first() {
    return get_pos(llList2Integer(l_link, 0)) == llList2Vector(l_v_pos_state_1, 0);
}

// Test de seconde position
integer test_second() {
    return get_pos(llList2Integer(l_link, 0)) == llList2Vector(l_v_pos_state_2, 0);
}

// Manoeuvre sur clic
action_click() {
    // Vers second
    if(test_first()) {
        go_second();
    }
    // vers premier
    else if(test_second()) {
        if(RETURNTIME == .0) go_first();
    }
    // Cas où il y a eu une modification de build
    else {
        llSetText("", ZERO_VECTOR, 0);
        llResetScript();
    }
}

// Passage en seconde position
go_second() {
    transition(FALSE);
    // Fixation de l'état
    integer n = llGetListLength(l_link);
    while(n--) {
        llSetLinkPrimitiveParamsFast(llList2Integer(l_link, n), [
            PRIM_POSITION, llList2Vector(l_v_pos_state_2, n),
            PRIM_ROT_LOCAL, llList2Rot(l_r_rot_state_2, n)]);
    }
    if(SECONDSOUND != "none") llTriggerSound(SECONDSOUND, 1.0);
    else if(RETURNTIME != .0 && SENSOR_RANGE == .0 && !AUTO) llSetTimerEvent(RETURNTIME);
}

// Passage en première position
go_first() {
    transition(TRUE);
```

```
    // Fixation de l'état
    integer n = llGetListLength(l_link);
    while(n--) {
        llSetLinkPrimitiveParamsFast(llList2Integer(l_link, n), [
            PRIM_POSITION, llList2Vector(l_v_pos_state_1, n),
            PRIM_ROT_LOCAL, llList2Rot(l_r_rot_state_1, n)]);
    }
    if(FIRSTSOUND != "none") llTriggerSound(FIRSTSOUND, 1.0);
}

// ------------------------------------------------
//        Détermination de l'index du script
// ------------------------------------------------
integer trouve_index() {
    list l = llParseString2List(llGetScriptName(), [" "], []);
    if(llGetListLength(l) == 1) return 0;
    else return (integer)llList2String(l, 1);
}

// ------------------------------------------------
//        Restitution informations
// ------------------------------------------------
integer get_infos() {
    // Recherche des primitives avec texte
    list l_textes;
    list l_id;
    integer n = llGetNumberOfPrims();
    integer i;
    string s_id = (string)trouve_index();
    for(i = 2; i <= n; ++i) {
        string s = get_texte(i);
        if(s != "") {
            if(llGetSubString(s, 0, 1) == "!" + s_id) {
                l_textes += s;
                l_id += i;
            }
        }
    }
    // Traitement des informations
    n = llGetListLength(l_textes);
    if(n) {
        i = n;
        while(i--) {
            l_link += 0;
            l_v_centre_state_1 += 0;
            l_v_centre_state_2 += 0;
            l_r_rot_state_1 += 0;
            l_r_rot_state_2 += 0;
            l_v_pos_state_1 += 0;
            l_v_pos_state_2 += 0;
        }
        while(n--) {
            string s = llList2String(l_textes, n);
            list l = llParseString2List(s, ["@"], []);
            integer id = (integer)llGetSubString(s, 2, 2);
```

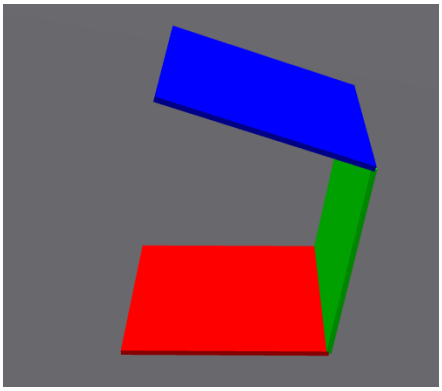
```
            l_link = llListReplaceList(l_link, [llList2Integer(l_id, n)], id, id);
            l_v_centre_state_1 = llListReplaceList(l_v_centre_state_1, [get_vector(llList2String(l, 1))], id, id);
            l_v_centre_state_2 = llListReplaceList(l_v_centre_state_2, [get_vector(llList2String(l, 2))], id, id);
            l_r_rot_state_1 = llListReplaceList(l_r_rot_state_1, [get_rotation(llList2String(l, 3))], id, id);
            l_r_rot_state_2 = llListReplaceList(l_r_rot_state_2, [get_rotation(llList2String(l, 4))], id, id);
            l_v_pos_state_1 = llListReplaceList(l_v_pos_state_1, [get_vector(llList2String(l, 5))], id, id);
            l_v_pos_state_2 = llListReplaceList(l_v_pos_state_2, [get_vector(llList2String(l, 6))], id, id);
            l_vec_rot = llListReplaceList(l_vec_rot, [get_rotation(llList2String(l, 7))], id, id);
        }
        return TRUE;
    }
    return FALSE;
}

rotation get_rotation(string s) {
    return llEuler2Rot((vector)("<" + s + ">") * DEG_TO_RAD);
}
vector get_vector(string s) {
    return (vector)("<" + s + ">");
}

// ------------------------------------------------
//                  Etat de fonctionnement
// ------------------------------------------------
default
{
    state_entry() {
        // Test présence d'infos
        if(get_infos()) {
            // Retour en première position après setup et après reset éventuel
            integer n = llGetListLength(l_link);
            while(n--) {
                llSetLinkPrimitiveParamsFast(llList2Integer(l_link, n), [
                    PRIM_POSITION, llList2Vector(l_v_pos_state_1, n),
                    PRIM_ROT_LOCAL, llList2Rot(l_r_rot_state_1, n)]);
            }
            // Sensor éventuel
                if(SENSOR_RANGE != .0 && !API) llSensorRepeat("", NULL_KEY, AGENT, SENSOR_RANGE, PI,
SENSOR_DELAY);
            // Marche automatique
            if(AUTO) {
                i_auto = TRUE;
                llSetTimerEvent(.01);
            }
        }
    }

    touch_start(integer total_number) {
        // On vérifie que le clic se fait sur une prim qui doit bouger
        if(~llListFindList(l_link, [llDetectedLinkNumber(0)]) && !API) {
            key k = llDetectedKey(0);
            if(test_access(k)) {
                if(SENSOR_RANGE == .0) action_click();
                else if(test_first()) llInstantMessage(k, CLICKMESSAGE);
            }
```

```
            else llInstantMessage(k, ACCESSMESSAGE);
        }
    }

    sensor(integer total_number) {
        // Gestion si objet en première position
        if(test_first()) {
            // Test de personnes autorisées
            while(total_number--) if(test_access(llDetectedKey(total_number))) {
                go_second();
                return;
            }
            // Si personne autorisée on met en seconde si première
            else if(test_second()) go_first();
        }
    }

    no_sensor() {
        // Si personne on met en première
        if(test_second()) go_first();
    }

    link_message(integer sender_number, integer number, string message, key id) {
        if(number == API_CLICK && SENSOR_RANGE == .0) action_click();
        else if(number == API_GO_POS_1) go_first();
        else if(number == API_GO_POS_2) go_second();
        else if(number == API_GET_POS) {
            llMessageLinked(LINK_THIS, API_POS_ANSWER, (string)test_second(), NULL_KEY);
        }
        else if(AUTO) {
            if(number == API_AUTO_START) {
                i_auto = TRUE;
                llSetTimerEvent(.01);
            }
            else if(number == API_AUTO_STOP) i_auto = FALSE;
        }
    }

    timer() {
        llSetTimerEvent(.0);
        if(AUTO) {
            if(i_auto) {
                if(test_first()) go_second();
                else go_first();
                llSetTimerEvent(.01);
            }
        }
        else go_first();
    }
}
```

Il suffit de le mettre dans l'inventaire de la primitive racine. L'objet passe dans son état initial :



Il suffit ensuite de cliquer dessus pour déclencher le mouvement :



Celui-ce s'effectue avec les paramètres par défaut.

4.2 Les paramètres de fonctionnement

Ces paramètres sont tous situés en tête du script :

```
// Time
float      TIME          = 2;       // Transition time in seconds
integer    AUTO          = FALSE;   // TRUE for loop mode
float      RETURNTIME    = 0;       // Time for auto return to first position (0 for no auto-return)
integer    API           = FALSE;   // True for only API

// Access
string     ACCESS        = "All";   // Access must be "All", "Group", "Owner" or "List"
list       ACCESSLIST    = [];      // Access names list if ACCESS is on "List"

// Sounds
string     SECONDSOUND   = "none";  // Sound for second position ("none" if none)
string     FIRSTSOUND    = "none";  // Sound for first position ("none" if none)

// Sensor
float      SENSOR_RANGE  = 0;       // Sensor range (0 for none), no touch action if sensor
float      SENSOR_DELAY  = 4;       // Sensor delay (no matter if no sensor)

// Messages
string     ACCESSMESSAGE = "Sorry but your are not allowed to command this object";
string     CLICKMESSAGE  = "Come closer to command this object";
```

```
// API
integer    API_CLICK      = 111;     // Like a "click"
integer    API_AUTO_START  = 112;     // Start in "auto" mode
integer    API_AUTO_STOP   = 113;     // Stop in "auto" mode
integer    API_GO_POS_1    = 114;     // Go to first position
integer    API_GO_POS_2    = 115;     // Go to second  position
integer    API_GET_POS     = 116;     // To get position
integer    API_POS_ANSWER  = 117;     // POsition answer
```

4.2.1 Paramètres de durée

Les trois premiers paramètres concernent la durée :

- TIME : c'est la durée totale du mouvement exprimée en secondes
- AUTO : c'est un mode spécial où le mouvement est ininterrompu, on passe d'un état à l'autre indéfiniment, à moins d'une commande particulière d'API
- RETURNTIME : si une valeur autre que 0 est précisée l'objet retourne automatiquement à sa position initiale au bout de ce délai

4.2.2 Paramètres d'accès

Si vous voulez limiter l'accès à la commande de l'objet vous avez deux paramètres

- ACCESS : avec comme valeur « All » (par défaut), « Group », « Owner » ou « List », dans ce dernier cas il faut renseigner le paramètre suivant,
- ACCESSLIST: liste des noms des personnes autorisées exprimés sous cette forme :

[« Bernard Tarvel », « Sigmund Certouille », « Claire Mand »]

4.2.3 Paramètres de sons

Si vous voulez déclencher un son pour chaque état vous avez ces deux paramètres :

- FIRSTSOUND : nom du son pour le premier état
- SECONDSOUND : nom du son pour le second état

Les fichiers des sons doivent se trouver dans l'inventaire de la primitive racine.

4.2.4 Paramètres de sensor

Si vous voulez un déclenchement par sensor (déclenchement à l'approche), renseignez ces deux paramètres :

- SENSOR_RANGE : distance de détection en mètres
- SENSOR_DELAY : fréquence de la détection en secondes

Attention au délai, des valeurs trop rapides impactent la charge du simulateur !

4.2.5 Messages

Vous pouvez personnaliser deux messages :

- ACCESSMESSAGE : information aux personnes non autorisées
- CLICKMESSAGE : information pour les utilisateurs de se rapprocher pour commander le mouvement

4.2.6 API

Ces paramètres concernent les scripteurs :

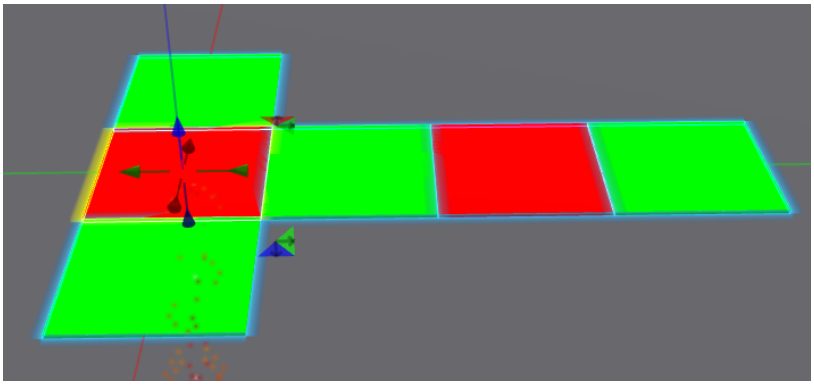
- API_CLICK : numéro du canal pour simuler un « clic »
- API_AUTO_START : numéro du canal pour démarrer un mouvement automatique (paramètre AUTO à TRUE)
- API_AUTO_STOP : numéro du canal pour arrêter un mouvement automatique (paramètre AUTO à TRUE)
- API_GO_POS_1 : numéro du canal pour commande vers position 1
- API_GO_POS_2 : numéro du canal pour commande vers position 2
- API_GET_POS : numéro du canal pour pour récupérer la position
- API_POS_ANSWER : numéro du canal pour la réponse de la position

5. Setup multiples

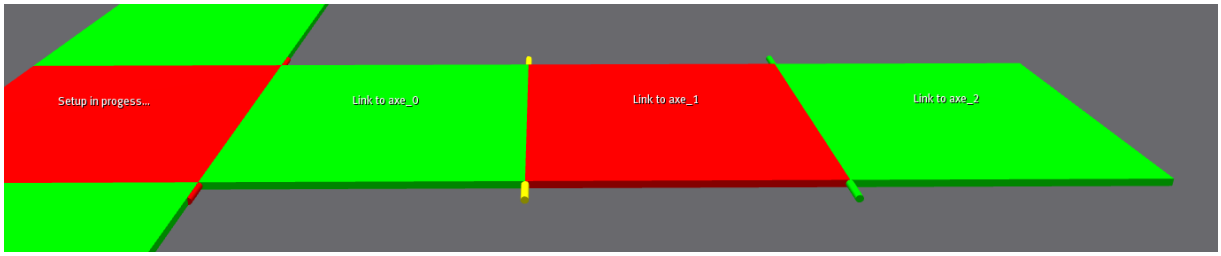
Nous avons vu que les axes doivent être parallèles. Comment faire lorsque nous voulons des mouvements avec des orientations différentes ? Le système permet de faire plusieurs setup (jusqu'à 10 pour un même objet).

5.1 Un exemple de setup multiples

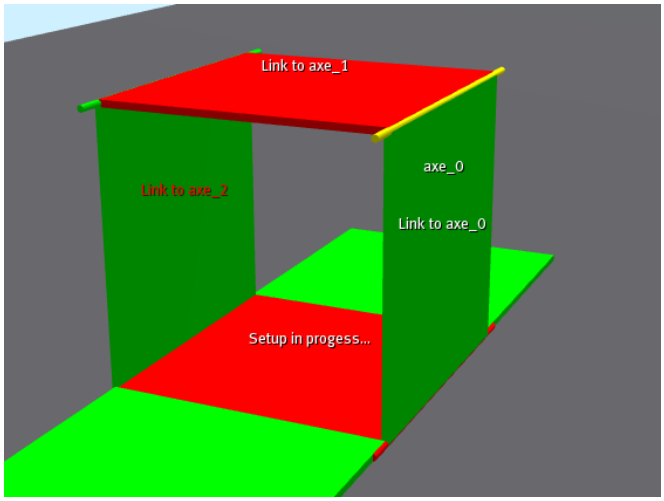
Prenons un exemple avec un cube décomposé dont voici le patron :



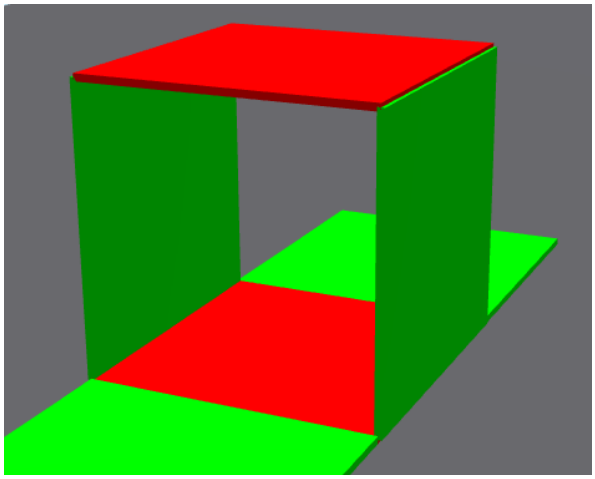
Nous commençons avec un setup pour les trois faces situées à droite :



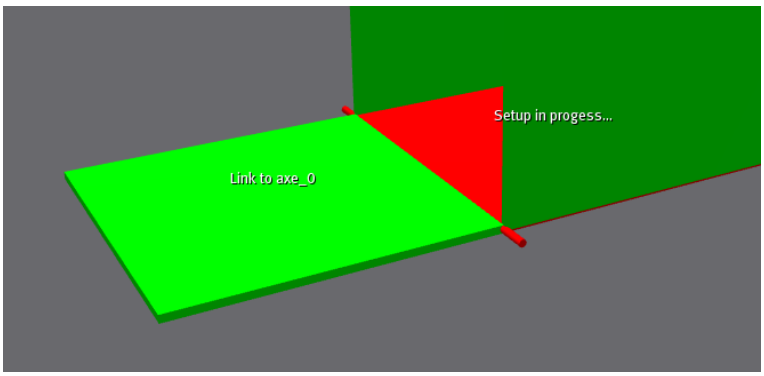
On a besoin de 3 axes et on fait le réglage de la seconde position :

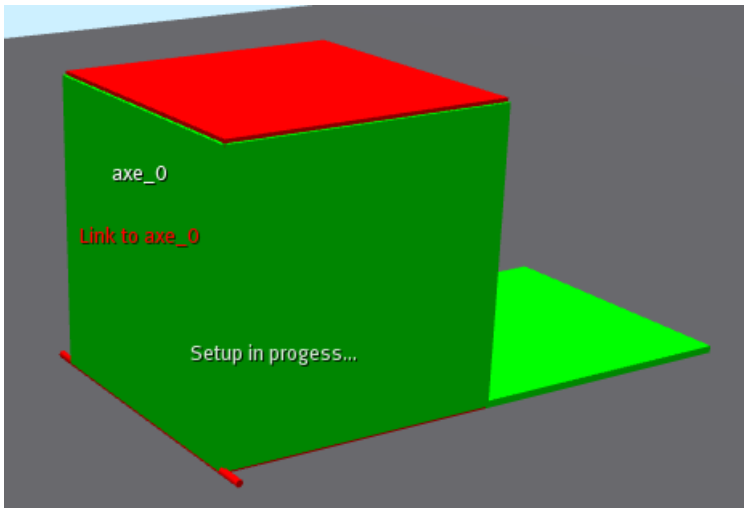


On termine le setup mais on ne met pas encore de script d'action (ça fonctionnerait mais pour les autres setups ça serait un peu gênant d'avoir des primitives qui bougent quand on clique) :

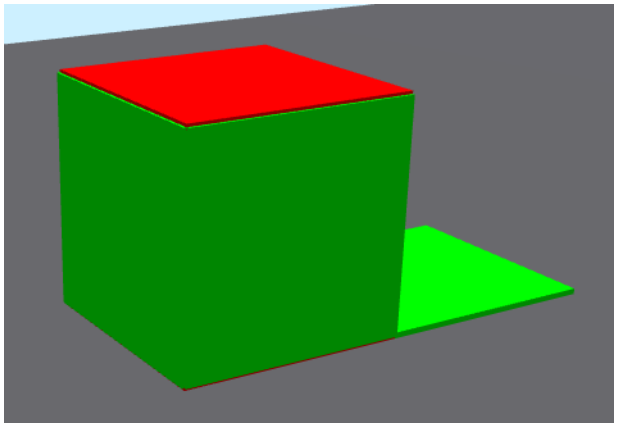


On peut alors passer au second setup :

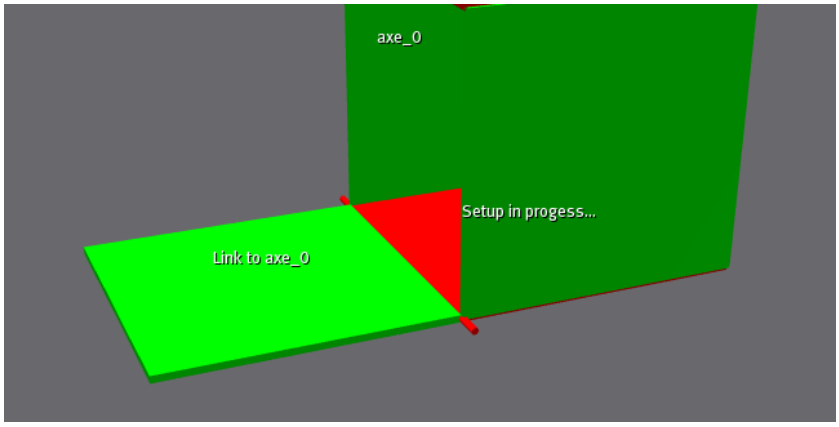




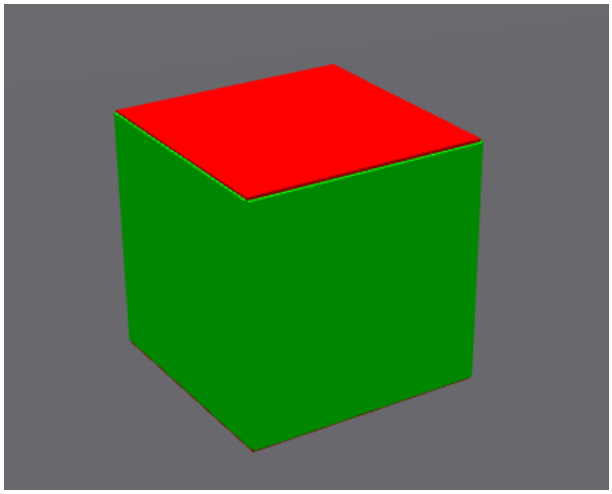
On termine le setup :



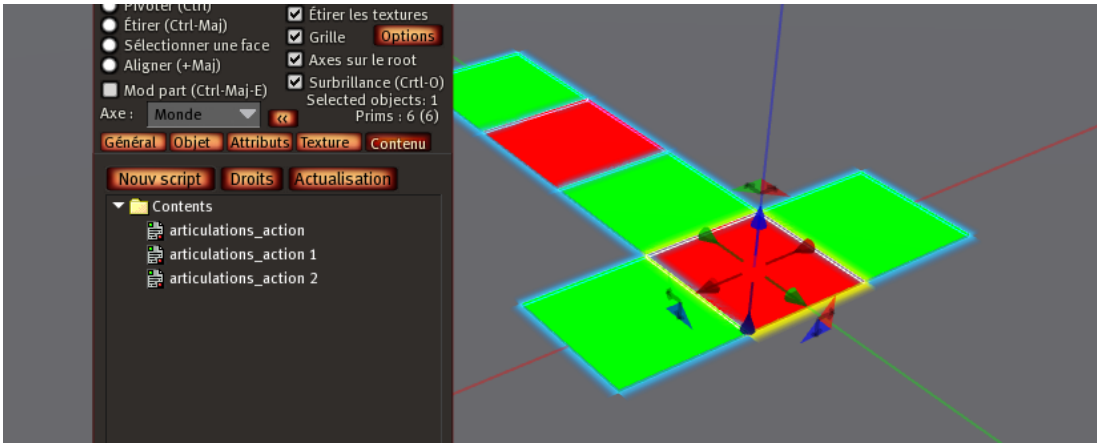
On ne met toujours pas de script d'action, on passe au dernier setup :



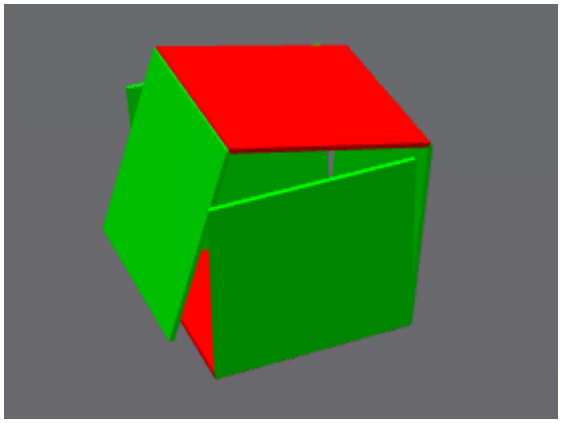
Et on termine ce setup :



Maintenant on met 3 scripts d'actions dans l'inventaire de la primitive racine :



Le cube se remet à plat, et maintenant il fonctionne :



5.2 La gomme

Les informations étant mémorisées dans le texte flottant des primitives vous ne pouvez pas réaliser un nouveau setup sur les mêmes primitives. C'est une bonne chose quand vous faites un setup multiples. Mais ça peut devenir gênant si vous voulez faire des reprises. Dans ce cas il faut effacer le texte flottant, ce qui est impossible à faire « à la main ».

Pour effacer toutes les informations de setup d'un objet utilisez le script gomme :

```
////////////////////////////////////////////////////////////////////////////////
//                                                                            //
//                        Articulation Gomme V 1.01                           //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//                  Copyright (c) 2012 by Bestmomo Lagan                      //
////////////////////////////////////////////////////////////////////////////////
//   This program is free software: you can redistribute it and/or modify     //
//    it under the terms of the GNU General Public License as published by    //
//    the Free Software Foundation, either version 3 of the License, or       //
//    (at your option) any later version.                                     //
//                                                                            //
```

```
//    Articulations is distributed WITHOUT ANY WARRANTY;                              //
//    See the GNU General Public License for more details.                            //
//                                                                                    //
//    To get a copy of the GNU General Public License, see <http://www.gnu.org/licenses/>.    //
////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////
//              Dont change code below unless you know what you do                    //
////////////////////////////////////////////////////////////////////////////////////////

// ------------------------------------------------
//                  Paramètres primitive
// ------------------------------------------------
string get_texte(integer id) {
   return llList2String(llGetLinkPrimitiveParams(id, [PRIM_TEXT]), 0);
}
unset_texte(integer id) {
   llSetLinkPrimitiveParamsFast(id, [PRIM_TEXT, "", <0,0,0>, 0]);
}

// ------------------------------------------------
//                  Etat unique
// ------------------------------------------------
default
{
   state_entry() {
      llOwnerSay(llGetScriptName() + " processing cleaning...");
      // Recherche des primitives avec un setup
      integer n = llGetNumberOfPrims();
      integer i;
      for(i = 2; i <= n; ++i) {
         string s = get_texte(i);
         if(s != "") if(llGetSubString(s, 0, 0) == "!") unset_texte(i);
      }
      llOwnerSay("Object is now clean !");
      llRemoveInventory(llGetScriptName());
   }
}
```

Il suffit de le mettre dans l'inventaire de la primitive racine pour tout effacer et retrouver un objet vierge de toute information de setup.