Réflexion sur l'analyse d'un problème.

30 aout 2009

- A propos de ce document :

Je me présente, *Ahuri Serenity*, actuellement professeur de script à l' École SL sur France 3D. J'aime partager mes connaissances aux étudiants de l'école ainsi qu'aux autres, c'est pourquoi dans un esprit de partage et seulement de partage, j'ai décidé de rédiger ce document. Je suis loin d'être un bon rédacteur donc soyez tolérants ! ;)

Mon expérience dans SL, m'a montré que les gens qui s'intéressent aux scripts sans aucune connaissance antérieure dans le domaine, ont beaucoup de mal à rédiger des scripts d'eux même. Est ce par manque de pratique ? Certainement. Est ce par manque de méthodes ? Oui. Très clairement, on va le voir, un script ne se rédige pas n'importe comment, une certaine rigueur peut résoudre bien des problèmes. Il sera donc indispensable d'insister sur la méthodologie.

Que cela soit bien clair, j'écris cette note dans un esprit de partage et avec toute la modestie possible, je n'ai aucune prétention particulière et surtout pas celle de maitriser mon domaine. Je ne juge personne, je ne fais que transmettre une technique que j'utilise PERSONNELLEMENT.

Je tiens aussi à préciser que les extraits de codes présents dans ce document ne sont pas sensés être les meilleurs solutions ou les plus optimisées, mais des exemples pédagogiques, il y aura donc probablement des points discutables j'en suis conscient.

J'espère apporter au final, chez certains d'entre vous, des axes de réflexion, des points de départ, ou pourquoi pas une certaine autonomie non encore acquise. Vous l'aurez compris, cette note n'est pas destinée aux plus initiés d'entre vous, mais plutôt aux débutants. Il s'agit de montrer une façon de réfléchir à un problème et de faire naitre, de votre propre réflexion, les éléments de réponse à vos problèmes liés au développement de scripts en LSL.

Pour ceux qui sont vraiment grands débutants, qui n'ont jamais touché un script de leur vie, cette note va peut être vous choquer, vous traumatiser à vie, je demande donc la plus grande prudence .. :D évidemment je plaisante mais ca peut sérieusement être difficile pour les moins initiés. Cependant une lecture ne coute rien, et en plus une surprise vous attend pendant votre lecture …..... xD

Pour toute question, n'hésitez pas à me contacter in world.

Sommaire :

Introduction

Un problème, selon sa difficulté, peut se résoudre naturellement, sans aucune réflexion ou difficilement, après des heures de luttes acharnées et grâce aux bons conseils des plus expérimentés. Toujours est il qu'une bonne démarche de résolution peut être la clé pour débloquer des situations plus que difficiles.

Nous allons donc essayer de voir ensemble UNE méthode d'analyse et de résolution de problèmes liés aux scripts. Celle que j'utilise personnellement, dans ma petite vie de scripteur SLien. Nous allons utiliser un exemple simple et pédagogique pour montrer concrètement la façon de procéder.

A signaler tout de même, qu'avec une bonne utilisation de cette méthode, mais surtout des concepts qu'elle soulève, vous serez en mesure de rédiger un script d'un niveau bien au dessus du votre avec peu de connaissances en script.

Présentation du fil rouge

Je profite de ce chapitre, pour souligner l'importance de bien détailler le problème à résoudre ! Pour apprécier pleinement votre problème (au sens analyse du terme), il peut falloir un support : rédiger le problème sur une feuille de papier, schématiser le problème à l'aide flèches, bulles, cadres, etc. ... Personnellement, pour les scripts importants en quantité de travail, je m'efforce de schématiser.

Si le problème ne vient pas à vous naturellement mais via une autre personne, un client par exemple, il peut se poser également des problèmes de communication. Pour cela, il existe des moyens simples tels que le cahier des charges (CdC). Un objectif important du CdC est d'offrir un support intermédiaire entre le demandeur et le développeur, il permet à chacun de comprendre les demandes et exigences de l'autre, dans les 2 sens, donc si vous désespérez avec vos demandeurs pour vous faire comprendre, n'hésitez pas à utiliser ce procédé. Le CdC se doit d'être précis et complet avant le début du développement du script.

Et notre fil rouge dans tout ca ? Très bien ! Sans plus attendre voici le CdC de notre mini projet :

Réalisation d'une flying box : (Ceux qui ont suivi mon cours sur le sujet vont se frotter les mains, les autres, j'attends de voir :D)

Petite présentation : Une flyingbox est un outils de travail (mais pas seulement), permettant à son utilisateur de disposer d'une plateforme aménagée à sa guise. On rez la flyingbox, on s'assoie dessus, elle monte jusqu'à une hauteur donnée, elle déploie la plateforme de travail. Lorsque le travail est terminé, on se rassoie sur la flyingbox, pour nettoyer la zone et redescendre au sol.

Critères :
- Utilisation Personnelle, que le owner.
- Déplacements verticaux, hauteur réglable facilement par le proprio du script. (*)
- Déplacements non physiques.
- La FlyingBox doit avoir la capacité de rezzer autant d'objets (non liés) que possible.
- Le déploiement de la plateforme se fait de manière automatique lorsqu'on arrive en haut et, se détruit, toujours automatiquement, lors de la descente.
- Le système doit être low lag.

(*) ne veut pas un système automatisé forcement, juste un paramètre explicite en haut du script

Maintenant qu'on a présenté notre fil rouge, et qu'on a vu l'importance de clarifier l'énoncé d'un problème, il va falloir l'observer.

Observer un problème

Nan, ne sortez pas votre microscope ou vos jumelles ! Pour observer un problème lié aux scripts rien de tel qu'un peu de jugeote !

En effet, il est important dans un premier temps d'avoir sous les yeux les éléments du problèmes, de les répertorier, de les considérer un à un et dans leur globalité. Chaque script doit être écrit après mure réflexion, on se lance pas dans un script sans anticiper. Au final, **un script n'est rien d'autre que la traduction dans un langage compréhensible par la machine d'un énoncé pensé et traité**. Ce que je veux dire c'est que lorsque vous écrivez vos scripts le problème est déjà résolu !! Si si !!

C'est bien là la difficulté pour moi pour vous faire comprendre ! Comment quelqu'un qui n'a pas l'habitude du langage de script peut être d'accord avec moi ? Je ne sais pas, en regardant par lui même. Faites moi confiance sur une chose, réfléchir avant d'agir est un proverbe qui prend tout son sens dans le monde du script.

Vous l'aurez compris on ne va pas rédiger le script dans cette partie loin de la ! ;)

Aller, appliquons une petite observation, qu'avons nous ? Une boite qui monte et qui descend, une plateforme qui se rez et qui disparait à volonté avec ses éléments non forcément liés. Voila pour les différents objets. Quand on observe encore un peu, on s'aperçoit qu'on n'a pas besoin de faire 36 scripts, 2 suffiront. Pourquoi ? Il nous faut un script pour la flyingbox forcement, elle va pas se déplacer toute seule hin ;) Il nous faut aussi un script dans la plateforme pour pouvoir disparaitre, encore une fois elle va pas disparaitre toute seule ;) Mais en relisant mon CdC je m'aperçoit que la plateforme doit être morcelée ! Ça veut donc dire que je dois faire un nouveau script pour chaque morceau ??? Heureusement non, ils ont tous le même comportement : disparaitre à volonté. Il faut donc faire la part des choses, identifier les comportements, regrouper les éléments de sorte de simplifier le problème. Ici, tous les morceaux de la plateforme doivent disparaître sur commande de la flyingbox de manière synchrone et strictement identique, l'utilisation d'un script identique va donc de soi. Ce script sera donc mis dans chaque morceau de la plateforme.

On doit donc réaliser 2 scripts. De notre observation bête et méchante nait des conclusions, certes évidentes pour la plupart d'entre vous, mais moins évidentes quand il s'agit d'un très gros projet faisant par exemple intervenir des acteurs externes à SL comme des serveurs Web eux même reliés à des acteurs externes.

Bref, une bonne observation est sensée délimiter le problème. Il faut fixer les limites générales permettant de débuter une analyse.

Analyser un problème

Nous y voilà, l'analyse du problème ! Non j'ai dis on range les loupes, on range les détecteurs infra rouges et autres scanners ! Ce qu'il va nous falloir ici c'est notre léger socle de connaissances en LSL. Je vous l'avais dis, il faut quand même un minimum de connaissances, mais pas énormément ;)

C'est pour cette étape qu' il est important d'avoir une petite culture LSLienne :) Connaitre un

peut ce qui est possible ou non, ce qui existe ou pas, etc. Le niveau de connaissances que vous avez, déterminera le niveau d'analyse du problème et donc influera sur la complexité de vos scripts ! **N'HESITEZ PAS A MANGER DU WIKI, VOTRE BIBLE** ! Euh ... n'attaquez pas votre écran svp sinon comment je fais moi pour vous montrer la suite huuh ?

Bon, l'analyse se fait pas en claquant des doigts, il faut prendre le mal par les cornes et le taureau en patience ! et lui faire face !

Et si on commençait par le commencement ? Oui ? Bon début :)

Il existe différentes démarches pour analyser un problème, je ne vais pas vous l'apprendre : chronologique, thématique, évènementielle, ... Nous allons opter ici pour la démarche chronologique mais on aurait pu faire autrement bien entendu (on applique donc une démarche séquentielle).

Avant de commencer tout de même, rappelons rapidement le concept d' état en LSL :

Prenons une porte, elle peut être ouverte ou fermée, ceci représente ses *états*. Un script peut présenter différents états également, mais, jamais être dans 2 états simultanés ! Et oui la porte est soit ouverte soit fermée ! N'importe quel objet a forcement un état en cours, il ne peut pas être sans état, on met donc initialement un état par défaut aux scripts, l'*état default*. C'est donc l'état considéré en tout premier temps dans un script.

Notre porte, lorsqu'elle est dans un état donné, on peut intervenir dessus, la pousser, la cogner, la peindre, souffler dessus, etc. Du point de vue de la porte, il arrive des *évènements*. Ces évènements sont donc à la base des différentes instructions dans un script LSL. Il y a forcement un facteur. La porte quand on la pousse, elle tourne autour de son axe, le fait de tourner correspondrait aux actions de nos instructions. On repère ces évènements grâce aux *events* en LSL. On va pouvoir donc savoir si quelqu'un touche notre objet, le cogne, lui parle, etc .... afin d'exécuter nos instructions. (N'y aurait il pas un soupçon d' OO dans l'idée la dedans ? gagné :D)

Les *instructions* vont définir les actions à faire, ces actions sont conditionnées en fonction des circonstances, elles sont paramétrées.

Si pour vous, tout cela semble étranger, voir totalement inconnu, c'est que vous n'avez pas une culture assez satisfaite en matière de LSL. Je vous conseillerai donc, dans ce cas, d'aller vous documenter sur la question avant de poursuivre, il est nécessaire d'avoir un minimum de connaissances en LSL (les bases) pour comprendre ce qui suit correctement.

Top Chrono ! - Comme dirait mon ami Barry -

- *On rez la flyingbox, que doit il se passer ? initialisation éventuelle ? définir un état particulier ?*

=> Le script dois savoir au rez qu'il est en bas, donc on doit s'arranger pour que l'état initial soit celui "au sol"

=> utilisation d'évènements liés au rez, aux changements d'états.

- *Quels évènements pourront changer cet état d'attente ?*

=> On a dit qu'il fallait que le owner s'assoie sur la flying box

=> évènements liés au sit, comment le repère t- on ?

- Q*ue se passe t-il alors ?*

=> on monte/descend . Comment monter ? Quel type de déplacement ? Quelle solution est plus adaptée ?

- *Changements d'états éventuels ?*
  => bien lister mes différents états, il semble ici qu'il y en ai 2 (au sol, en l'air).
  => Notre script doit faire un cycle de fonctionnement, il nous font donc des états qui s'appelent en formant une boucle. ;)

- *Comment mes objets de plateforme savent qu'ils doivent disparaitre ?*
  => Quels moyens de communication ? Le plus adapté ?
  => Shout / Listen (shout car les objets peuvent être assez loin du centre de la plateforme)

- Etc.

On cherche les réponses à nos questions dans les différentes sources à disposition, la première : la mémoire. Ensuite il y a le **wiki**, vous savez, votre bible ! Etc.
On doit revenir autant de fois que nécessaire sur les étapes d'analyse tant que des questions subsistent.

Élaboration des scripts

Enfin ! on va pouvoir se lâcher les amis !! A ce stade là normalement, votre problème est pratiquement résolu, en tous cas vous avez votre script en tête. Il ne reste plus qu'a traduire l'énoncé de votre réflexion en langage compréhensible par la machine. On en est à l'étape 0+3 = 4 ! Et oui le zéro compte :) Et toujours pas un code d'écrit !

Mais bon toute bonne chose a une faim, ce qui n'empêchera pas de nourrir votre enthousiasme gourmand afin d'écrire comme il se doit le code tant pensé et repensé aussi digeste qu'il se peut être !

La démarche est alors la suivante :

1. Définir un éventuel **paramétrage** naturel, les attributs, les données système évidentes, …
2. Extraire les différentes **fonctions à concevoir**. (ne pas les remplir forcement, et c'est là le secret de tout vous allez voir !!)
3. Une fois les fonctions mises en relief, on aménage le **corps du script** (*) en utilisant la logique mise en valeur dans notre partie précédente, vous verrez ca en pratique, ca parlera mieux.
4. On finit d'écrire les fonctions (**remplissage**). C'est sensé être la partie qui demande le plus de connaissances techniques si on suit ma démarche.
5. On **test**.

(*) Ce que je nomme corps du script, est l'ensemble complémentaire de l'ensemble { variables globales + fonctions personnelles } => il s'agit donc du bloc des états, des évènements, des instructions logiques et appels de fonctions en rapport avec l'analyse du problème.

Le passage d'une étape à la suivante se fait prudemment. Il n'est pas question de faire des aller-retours entre chaque étape.

Je rappelle que les procédés utilisés sont à transposer, évidemment, certaines étapes diffèrent selon les cas car cela dépend de trop du problème à traiter.

Aller, on va essayer de détailler notre transcription, analyse → script rédigé.

- Script des éléments de la plateforme : PLATE

✗ *Paramétrage* :

Les morceaux de plateforme sont commandés à distance, le moyen choisit précédemment pour communiquer est Shout / Listen, on doit donc définir un channel de communication. On définit aussi un message particulier envoyé par la flyingbox qui certifiera qu'on veut bien faire disparaître la plateforme.

```
integer iCHANNEL = -458;          // Channel dans lequel on ordonne de disparaitre.
string  sCOMMANDE = "die";        // Commande de disparition.
```

✗ *Fonctions à concevoir* :

Notre script n'a qu'une seule action à exécuter : faire disparaitre l'objet qui le contient.

```
Disparaitre( )
{

}
```

Cela suffira donc :)

✗ *Corps du script* :

Le script doit écouter sur le bon channel et réagir quand le bon message est perçu. On utilise donc : llListen / listen. On remarquera l'utilisation de notre nouvelle fonction sans même avoir pensé forcément comment la remplir.

```
default
{
   state_entry()
   {
      llListen(iCHANNEL, "", NULL_KEY, sCOMMANDE);
   }
   listen(integer channel, string name, key id, string m)
   {
      Disparaitre( );
   }
}
```

Le corps du script me semble convenir à la logique de mon analyse, je continue donc.

✗ *Remplissage* :

Nous devons toujours avoir le CdC dans la tête, surtout pendant l'étape précédente et celle ci. C'est pourquoi on va faire attention à ce que ce soit le owner qui commande la disparition et non le copain à coté qui utilise la même flyingbox ;) ensuite on utilise la commande llDie pour faire disparaître l'objet.

```
Disparaitre( key id )
{
   if( llGetOwnerKey(id) == llGetOwner() )
   {
      llDie();
   }
}
```

Remarque : Nous avons eu besoin de rajouter un paramètre à la fonction, il faudra donc bien le rajouter à l'appel de la fonction dans le corps du script. Attention je ne répèterai pas cette remarque par la suite, alors qu'on répètera ce principe dans les prochaines fonctions.

✗ *Tests* :

Un test simple permet de valider ce script : mettre iCHANNEL égal à 1, prononcer ou faire prononcer par un objet nous appartenant le message « die » dans le channel 1. Notre objet disparaît sans broncher c'est gagné.

- Script de la flyingbox : FLYINGBOX

✗ *Paramétrage* :

On va avoir besoin du channel et du message pour la disparition (les mêmes que précédemment bien entendu), du nom de la plateforme pour savoir quel objet rezzé (pour ceux qui connaissent on peut aussi prendre le premier objet de l'inventaire, ca évite l'utilisation de cette variable et ca permet une programmation plus dynamique), de la hauteur (absolue) de la plateforme souhaitée et enfin d'un vecteur position relative de la plateforme par rapport à la flying box au rez.

```
string  sPlateforme   = "Box";       // Nom de la plateforme (pour le rez)
float   fHauteur      = 35.0;        // hauteur de montée en mètres.

integer iCHANNEL  = -458;            // Channel dans lequel on ordonne de disparaitre.
string  sCOMMANDE = "die";           // Commande de disparition.

vector vOffSet        = < 0.0, 0.0, -2.0>;  // Position de la plateforme rezzée par rapport à la flying
box (relative)
```

✗ *Fonctions à concevoir* :

Quelles actions sont à effectuer pour la box ? On a dit : monter, descendre et réagir lorsqu'une personne s'assoie. Soit ! On écrit ces 3 fonctions :

```
monter( )
{

}

descendre( )
{
```

```
}

// onSit va permettre d'agir comme il se doit quand on est sur que quelqu'un s'assoit

onSit( )
{

}
```

- ✗ *Corps du script* :

  Je vais détailler un peu plus que dans le script précédent.

  Dans notre analyse, on a montré que 2 états suffiront : « au sol » et « en l'air ».

```
default
{

}

state top
{

}
```

Ensuite, on se pose la question des évènements à considérer pour notre objet. Sachant que notre script boucle sur 2 états, on aura besoin de state_entry (événement appelé à l'entrée dans l'état correspondant) dans chaque état pour définir une situation initiale de l'état. (Waaah c'est brouillon ! En effet, il ne vous reste plus qu'a vous venger sur le **WIKI**.

….....
…. ….. …..... ….....
…..
….
…
…
..

C'est bon ? Vous revoilà ? Je suis sur qu'en passant vous avez chercher comment on récupère l'évènement, « une personne s'assoie ». xD Aller pour les plus feignants : Il s'agit d'un événement plus général que çà, changed, qui détecte un changement sur l'objet ! Cela peut être une changement de texture, de proprio, de région, de link, etc. C'est ce dernier qui nous intéresse, quand un personne s'assoie ca change le link, elle devient une nouvelle prim du link set ;) (WIKI !!!). Enfin, il faudra utiliser la fonction llAvatarOnSitTarget pour savoir si une personne est bel et bien assise ;) (Mon ami le Wiki me souffle à l'oreille que cela implique de définir un llSitTarget, soit ! Les state_entry sont là pour ca :D)

```
default
{
   state_entry()
   {
      llSitTarget(<0,0,0.1>,ZERO_ROTATION);
   }

   changed(integer change)
   {
      if (change & CHANGED_LINK)
      {
         // On prend l'identité de la personne assise
         // test : il y a effectivement quelqu'un qui s'assoi.
         if( (agent = llAvatarOnSitTarget()) != NULL_KEY )
         {
            onSit( );
            state top;
         }
      }
   }
}

state top
{
   state_entry()
   {
      llSitTarget(<0,0,0.1>,ZERO_ROTATION);
   }

   // La presence du on_rez répond à un des éléments du CdC je vous laisse chercher :)
  // on_rez est appellé lorsque le script est rezzé in world.
   on_rez(integer start_param)
   {
      llResetScript();
   }

   changed(integer change)
   {
      if (change & CHANGED_LINK)
      {
         // On prend l'identité de la personne assise
         // test : il y a effectivement quelqu'un qui s'assoi.
         if( (agent = llAvatarOnSitTarget()) != NULL_KEY )
         {
            onSit( );
            state default;
         }
      }
   }
}
```

Revenons rapidement, sur un point ou deux :

llAvatarOnSitTarget() renvoi la clé de la personne assise si il y en une et NULL_KEY sinon. On place cette clé dans une variable agent que j'ai mis globale, on verra pourquoi plus tard (agent = llAvatarOnSitTarget()). Ensuite on teste si elle est différente de NULL_KEY ((agent = llAvatarOnSitTarget()) != NULL_KEY), si oui, on est sur qu'une personne est assise, on exécute alors onSit(). Dans chaque état cela fonctionne de la même façon, sauf que la redirection de l'état se fait forcément différemment (state top / state default). C'est par ces dernières instructions qu'on peut changer d'état.

Remarque : Pas besoin d'une connaissance approfondie en script LSL pour faire cela, si vous en êtes pas convaincus, essayez de prendre du recul sur la façon de procéder : on a effectué une analyse qu'on a juste retransmis en langage LSL. Pas de réflexion poussée ici, un peu de documentation à chercher sur le wiki peut être, mais c'est pas la fin du monde et ça apprend pas mal de choses.

- ✗ *Remplissage* :

Bon, je vous avais prévenu, la partie remplissage peut s'avérer technique, mais d'autant moins difficile que vous aurez pensé votre problème et qu'il existera de sous fonctions divisant votre problème en sous problèmes. N'abusez cependant pas des fonctions, c'est un coup à s'emmêler les pinceaux, à oublier l'existence de certaines fonctions ou pire à compliquer le problème !

Il faut coder nos 3 fonctions, qu'attendons nous ? Go !

```
monter()
{
   // On conserve la position au sol
   vInit = llGetPos();

   // Deplacement ascendant
   while ( llVecDist(llGetPos(), <vInit.x,vInit.y,fHauteur>) > 0.001 )
   {
      llSetPos(<vInit.x,vInit.y,fHauteur>);
   }

   // Rez de la plateforme de travail
   llRezObject(sPlateforme, llGetPos()+vOffSet, ZERO_VECTOR, ZERO_ROTATION, 0);

   // On fait se lever le proprio de la flyingbox
   llUnSit(agent);
}

descendre()
{
   // On détruit la plateforme
   llShout(iCHANNEL, sCOMMANDE);

   // Deplacement descendant
   while ( llVecDist(llGetPos(), vInit) > 0.001 )
   {
      llSetPos(vInit);
   }
```

```
    // On fait se lever le proprio de la flyingbox
    llUnSit(agent);
}

onSit(  integer estAuSol)
{
    // Si c'est le owner on agit sinon on fait le ver la personne
    // CF CdC
    if(agent == llGetOwner())
    {
        // Si le cube est au sol on monte sinon on descend.
        if( estAuSol )
            monter();
        else
            descendre();

    }
    else // Forcement différent de NULL_KEY
    {
        llUnSit(agent);
    }
}
```

Remarque 1 : On ne va pas aborder la méthode utilisée pour le déplacement d'autant plus qu'on peut largement l'optimiser ! Cependant on voit bien l'intérêt maintenant d'avoir garder la position initiale de la flyingbox, car dans la descente, on en a besoin :)

Remarque 2 : On s'aperçoit que faire des fonctions simplifie largement le problème en le découpant en de plus petits problèmes. Dans nos fonctions de déplacement, il suffit alors de respecter un démarche itérative ;)

Remarque 3 : Oui la fonction onSit est très moche !!!!!! Rassurez vous quand même, elle fonctionne. Elle reçoit la clé de la personne assise, et doit gérer commune une grande, la montée ou la descente et le fait d'empêcher quiconque autre que le owner de s'assoir, on lui a donc donné un coup de main en lui mettant un paramètre pour savoir ou elle en est. CA C'EST VRAIMENT BROUILLON !!! Je vous l'accorde ! Cependant j'avais besoin d'organiser mon code comme ca pour vous montrer qu'on peut coder le corps du script et sans connaître le contenu des fonctions !! Ici ca donne un truc presque a refaire mais on va s'en contenter hin :p Après libre à vous de revenir sur vos structures pour la remplacer en beaucoup mieux :)

✗ *Tests* :

On va faire les tests avec notre jeu de scripts PLATE + FLYINGBOX :

```
// PLATE :

integer iCHANNEL = -458; // Channel dans lequel on ordonne de disparaitre.
string sCOMMANDE = "die"; // Commande de disparition.

Disparaitre( key id )
{
   if( llGetOwnerKey(id) == llGetOwner() )
   {
      llDie();
   }
}

default
{
   state_entry()
   {
      llListen(iCHANNEL, "", NULL_KEY, sCOMMANDE);
   }
   listen(integer channel, string name, key id, string m)
   {
      Disparaitre( id);
   }
}
```

```
// FLYINGBOX ::

string sPlateforme = "Box"; // Nom de la plateforme (pour le rez)
float  fHauteur = 35.0;        // hauteur de montée en mètres.

integer iCHANNEL = -458; // Channel dans lequel on ordonne de disparaitre.
string sCOMMANDE = "die"; // Commande de disparition.

vector vOffSet = < 0.0, 0.0, -2.0>; // Position de la plateforme rezzée par rapport à la flying box
(relative)

vector vInit;    // Position initiale de la box au sol
key agent;        // Clé de la personne qui se déplace sur la box.

monter()
{
   vInit = llGetPos();
   while ( llVecDist(llGetPos(), <vInit.x,vInit.y,fHauteur>) > 0.001 )
   {
      llSetPos(<vInit.x,vInit.y,fHauteur>);
   }
```

```
    llRezObject(sPlateforme, llGetPos()+vOffSet, ZERO_VECTOR, ZERO_ROTATION, 0);

    llUnSit(agent);
}

descendre()
{
    llShout(iCHANNEL, sCOMMANDE);

    while ( llVecDist(llGetPos(), vInit) > 0.001 )
    {
        llSetPos(vInit);
    }

    llUnSit(agent);
}

onSit(  integer estAuSol)
{
    // Si c'est le owner on agit sinon on fait le ver la personne
    // CF CdC
    if(agent == llGetOwner())
    {
        // Si le cube est au sol on monte sinon on descend.
        if( estAuSol )
            monter();
        else
            descendre();

    }
    else // Forcement différent de NULL_KEY
    {
        llUnSit(agent);
    }
}

default
{
    state_entry()
    {
        // Le fait qu'on utilise une assise oblige à définir un sit target , cf le wiki !!!!
        llSitTarget(<0,0,0.1>,ZERO_ROTATION);
    }

    changed(integer change)
    {
        if (change & CHANGED_LINK)
        {
            // On prend l'identité de la personne assise
            // test : il y a effectivement quelqu'un qui s'assoi.
            if( (agent = llAvatarOnSitTarget()) != NULL_KEY )
            {
```

```
            onSit( TRUE );
            state top;
        }
    }
  }
}

state top
{
  state_entry()
  {
    llSitTarget(<0,0,0.1>,ZERO_ROTATION);
  }

  // La présence du on_rez répond à un des éléments du CdC je vous laisse chercher :)
  on_rez(integer start_param)
  {
    llResetScript();
  }

  changed(integer change)
  {
    if (change & CHANGED_LINK)
    {
      // On prend l'identité de la personne assise
      // test : il y a effectivement quelqu'un qui s'assoi.
      if( (agent = llAvatarOnSitTarget()) != NULL_KEY )
      {
        onSit( FALSE );
        state default;
      }
    }
  }
}
```

- Mettre le script PLATE dans différents objets, qu'on prend tels quels dans l'inventaire en une seule fois. Nommer le tout "Box".
- Insérer Box dans un cube au sol, on insère dans le même cube le script FLYING BOX
- Conserver une copie de ce cube.
- S'assoir sur le cube.
- conclusion ?
- Si on se rassoit dessus ?
- conclusion finale ?

Soit ca fonction, dans ce cas youpiiii, sinon il faut faire la démarche inverse. On regarde un pas en arrière, étape d'élaboration, voir si une faute ne s'est pas glissée, si on pense que non on recule encore. Si pas de faute de remplissage, on recule encore, si pas de faute de logique, on recule encore, etc. Normalement, on avance prudemment donc les erreurs doivent être fraiches, d'où la démarche précédente.

Remarque de conclusion :

Il est important de voir que les étapes d'élaboration des scripts sont en appui permanent sur l'édifice que représente l'observation du problème et son analyse (cf parties précédentes), il ne faut donc pas bâcler ces dernières parties auquel cas il sera parfois difficile voir suicidaire de commencer à écrire le script. Donc en gros respectez les étapes et tout ira bien :))))

Réflexion Post Développement

Cette étape n'est pas obligatoire mais reste importante surtout au niveau débutant car au début c'est pas automatique (comme les antibiotiques ?) et après ca le devient. Il s'agit de réfléchir à comment améliorer son script, quelles solutions pour quels points ? Peser le pour et le contre. Se renseigner sur d'éventuelles ouvertures. Vous devez prendre du recul sur vos scripts, avoir un soucis de d'**optimisation**. Les proprios de vos créations en seront ravis (no lag, performances, simplicité, etc), les serveurs de LL et les autres résidents également :) N'hésitez pas à montrer vos créations aux plus expérimentés pour profiter de leurs critiques (positives ou négatives).

Conclusion générale

Nous avons vu ensemble, que développer un script n'est pas forcement accessible qu'aux plus doués d'entre nous. Il suffit d'avoir un peu de logique et de méthodologie pour faire des choses sympa. Cette méthodologie dont j'ai montré un exemple ici, et que je défend fermement, témoigne de l'ouverture d'esprit et du bon sens. Prise de connaissance de l'énoncé, observation du système, analyse du problème, conception méthodique et tests précis, sont les étapes maitresses de ma façon de travailler que j'ai voulu partager avec vous au travers de cette note. J'espère que certains d'entre vous auront un déclic grâce à ce texte, et pourront venir me montrer à leur tour les piliers de leurs raisonnements, dans la conception de leurs scripts, réponses à des problèmes précis.

Merci à vous d'avoir eu le courage de me lire jusqu'ici,

A bientôt sur le bac a sable :D

Ahuri Serenity