Les triggers dans CocoMUD

Il est possible que les triggers soient la fonctionnalité la plus puissante de tout client MUD. CocoMUD propose un système simple et flexible de triggers, qui sera présenté ici avec de nombreux exemples.

Qu'est-ce qu'un trigger ?

Normalement, si vous entrez une commande dans le client, le serveur devrait vous "répondre" en envoyant une ou plusieurs lignes. Cette réponse est constituée en grande partie (sinon en totalité, à l'exception des couleurs) de texte, tout comme votre commande.

Les triggers peuvent intercepter une ligne particulière et réagir en fonction, en faisant quelque chose. C'est l'expression la plus simple d'un trigger : je surveille le texte envoyé par le serveur, je réagis si je rencontre une certaine ligne et je fais quelque chose en retour.

Les triggers étant potentiellement puissants, leur configuration a été répartie dans différentes couches de complexité : vous n'avez pas besoin de comprendre (ni même de connaître l'existence) des types de triggers les plus avancés pour utiliser cette fonctionnalité. Cette documentation va décrire dans l'ordre la mise en place, pas à pas, des triggers les plus simples jusqu'aux triggers les plus complexes.

Créer un trigger

Commençons par quelque chose de simple : un trigger qui joue un son quand quelqu'un (n'importe qui) parle sur un canal du jeu.

Cet exemple sera différent sur différents MUDs, bien entendu, n'hésitez donc pas à l'adapter en fonction de vos besoins. Pour notre exemple, admettons que nous pouvons utiliser la commande "hrp" pour envoyer un message sur le canal "hrp", comme ceci :

```
hrp bonjour à vous !
```

Et si tout va bien, le MUD devrait répondre avec :

```
[hrp] Vous dites : bonjour à vous !
```

Ou bien, si quelqu'un d'autre parle sur le canal "hrp", vous pourriez voir un message comme ceci :

```
[hrp] Aaron dit : coucou
```

En bref, si nous voulons créer un trigger qui intercepte ces lignes, il nous faut un trigger qui intercepte les lignes qui commencent par "[hrp]" ("hrp" entouré de crochets).

> Pourquoi intercepter ces messages ?

Si vous êtes nouveau dans l'utilisation des triggers, vous pourriez vous demander à quoi ils servent. La réponse dans ce cas précis pourrait être que, quand on reçoit un message sur le canal "hrp", le client joue un son. C'est sur cela que nous allons travailler.

Nous allons donc voir comment créer un trigger qui :

- Réagit quand le client reçoit une ligne commençant par "[hrp]" ;
- Joue un son à ce moment.

Comme la plupart des fonctionnalités sur CocoMUD, on peut créer des triggers en passant par l'interface ou par une instruction

SharpScript. La première étant la plus facile pour commencer, nous allons la voir d'abord.

Depuis l'interface

Pour ajouter, éditer ou suppriemr un trigger depuis l'interface de CocoMUD, ouvrez la barre de menu, **Jeu** -> **Triggers**.

Vous devriez vous trouver dans une boîte de dialogue listant les triggers actuellement configurés sur cet univers. Il est très possible que cette liste soit vide la première fois que vous ouvrez cette boîte de dialogue. Pour créer un trigger, cliquez sur le bouton **Ajouter**.

Une nouvelle boîte de dialogue s'ouvre alors. Le curseur devrait se trouver dans une zone d'édition, où l'on vous demande d'entrer le trigger (c'est-à-dire, son déclencheur, la partie qui va indiquer à CocoMUD quoi surveiller).

Pour notre exemple, nous avons déterminé que notre trigger devrait s'exécuter quand le client reçoit une ligne commençant par [hrp]. Vous pouvez écrire [hrp] dans cette zone de texte, mais ne la quittez pas encore : si vous créez un trigger déclenché par [hrp], le trigger ne sera exécuté que si le client envoie une ligne ne contenant **que** [hrp]. Ce n'est pas ce que l'on veut faire ici : nous voulons que le trigger réagisse à une ligne commençant par [hrp] . La solution est de mettre un signe astérisque (*) après {hrp}, pour dire au client que [hrp] peut être suivi de n'importe quoi.

Peut-être reconnaissez-vous cette syntaxe : c'est la même que pour créer des alias avec variables, et ce n'est pas une coïncidence, nous verrons plus tard pourquoi.

Pour l'instant, vous pouvez écrire dans ce champ de texte :

```
[hrp]*
```

Si vous appuyez sur Tab, vous devriez vous trouver dans une liste d'actions que l'on peut associer à ce trigger. Dans notre exemple, nous voudrions jouer un son quand ce trigger se déclenche : parcourez la liste jusqu'à trouver l'action :

```
Joue un son
```

Et cliquez sur le bouton suivant : **Ajouter l'action**.

On vous demande maintenant de configurer cette action (dans notre cas, choisir le fichier sonore à jouer quand ce trigger se déclenche). Vous avez un bouton "Parcourir" qui vous permet de trouver le fichier sonore sur votre disque dur. Le bouton "Test" permet de vérifier que CocoMUD parvient bien à lire et jouer ce fichier (CocoMUD lit les fichiers .wav et .ogg). Si tout se passe bien, cliquez sur **OK**. L'action sera ajoutée au trigger. vous vous trouverez dans la liste des actions liées à ce trigger :

```
#play mon_fichier.wav
```

Bien sûr, "mon_fichier.wav" sera remplacé par le chemin et le nom du fichier que vous avez sélectionné. La représentation de l'action est en version courte (version SharpScript), cela explique pourquoi vous voyez #play mon_fichier.wav. Ne vous en inquiétez pas trop, c'est surtout un point de repère pour vérifier que l'action a bien été ajoutée, ainsi qu'un moyen de l'éditer, si vous voulez par exemple jouer un autre fichier sonore à la place.

Ces deux listes et les quelques boutons qui l'entourent forment l'éditeur SharpScript, permettant de configurer des actions assez complexes sans toucher au SharpScript de près ou de loin. Cette fenêtre peut s'avérer un peu intimidante au premier abord, voici donc un résumé de ce qu'elle contient. Notez que vous trouverez la même hiérarchie pour configurer les alias ou les macros et d'autres fonctionnalités de CocoMUD dépendantes du SharpScript :

- La première chose à renseigner pour les triggers est le déclencheur, comme nous l'avons fait plus haut. C'est une zone de texte toute simple ;
- Au-dessous se trouve une liste d'actions connectés à ce trigger. C'est une liste, vous pouvez naviguer avec les flèches pour la parcourir ou sélectionner une action précise ;
- À droite se trouve le bouton pour éditer la ligne d'action sélectionnée (dans notre cas, nous pourrions vouloir changer le fichier sonore joué par le trigger) ;
- Toujours à droite se trouve le bouton pour supprimer la ligne d'action. Notez que ces trois champs (liste d'actions, bouton éditer et bouton supprimer) n'apparaissent pas si aucune action n'est associée à ce trigger ;
- Au-dessous se trouve une liste d'actions que l'on pourrait vouloir lier à ce trigger. C'est dans cette liste que nous avons

sélectionné "joue un son" dans l'exemple ci-dessus. Elle apparaît dans tous les cas, car un trigger peut être connecté à aucune, une, deux ou de nombreuses actions, il n'y a pas vraiment de limite.
- À droite se trouve le bouton pour ajouter l'action sélectionnée.
- Le reste de la fenêtre contient d'autres cases à cocher et options qui seront détaillées dans la suite de cette documentation.

Nous avons créé notre trigger [hrp]* connecté à une action pour jouer un son : vous pouvez donc cliquer sur **OK**. Vous devriez vous retrouver dans la liste des triggers actuels, sur le trigger que vous venez d'ajouter. Appuyez de nouveau sur **OK** pour fermer la boîte de dialogue en sauvegardant. Si vous quittez la boîte de dialogue autrement, le trigger ne sera pas ajouté.

Pour vérifier que notre nouveau trigger marche correctement, essayons d'envoyer un message sur le canal "hrp" :

```
hrp Est-ce que ça marche ?
```

Si tout va bien (et partant du principe que le serveur répond comme on l'attend), on devrait recevoir la ligne :

```
[hrp] Vous dites : est-ce que ça marche ?
```

Et vous devriez entendre le son que vous avez sélectionné à l'étape précédente. Voilà ! Ce n'était pas si difficile, si ?

Depuis le SharpScript

Comme toujours, l'interface permet de manipuler des options potentiellement complexes, mais elles sont toutes converties en SharpScript à la fin, même si vous n'avez pas vraiment besoin de vous en occuper. Créer le trigger de notre exemple en SharpScript est assez facile : vous pouvez entrer cette ligne directement dans votre client, ou bien dans le fichier "config.set".

```
#trigger [hrp]* {#play mon_fichier.wav}
```

C'est une instruction SharpScript. De gauche à droite, nous avons :

- #trigger est le nom de l'action SharpScript. Ici, #trigger crée simplement un nouveau trigger ;
- [hrp]* est notre déclencheur, ce que le trigger doit surveiller quand on reçoit des messages du serveur. Tout comme audessus, on a précisé [hrp]* , qui veut dire "toute ligne commençant par [hrp]" ;
- Ensuite, on doit préciser la ou les actions à exécuter quand ce trigger se déclenche. Ici, #play mon_fichier.wav. #play est une autre action SharpScript, qui permet simplement de jouer le fichier passé en argument. Puisque notre argument contient un espace entre le nom de l'action #play et l'argument de la fonction mon_fichier.wav, on doit entourer l'argument d'accolades.

Ajouter un trigger en SharpScript est peut-être bien plus rapide, mais le système ne vous pardonnera pas facilement si vous faites des erreurs de syntaxe. L'interface nécessite d'avantage d'étapes, mais elle est généralement plus sûre.

Editer un trigger

Dans la boîte de dialogue des triggers (menu **Jeu** -> **Triggers**), vous pouvez éditer un trigger. Vous aurez besoin d'éditer un trigger dans deux cas : si vous voulez changer son déclencheur (la partie qui indique au trigger quel morceau de texte surveiller) on ses actions (la partie qui décrit quoi faire quand ce trigger se déclenche).

Supprimer un trigger

Supprimer un trigger peut se faire depuis l'interface également. Cliquez simplement sur le bouton **Supprimer** après avoir sélectionné un trigger existant. Confirmez que vous voulez supprimer ce trigger. N'oubliez pas de fermer la boîte de dialogue en cliquant sur **OK**, sans quoi votre trigger ne sera pas effacé.

Utiliser des variables dans les triggers

Il est tant de regarder plus attentivement le signe astérisque (*) que nous avons utilisé plus haut. Ce n'est pas un hasard si vous reconnaissez cette syntaxe depuis la documentation des alias, puisqu'il s'agit de la même chose : l'astérisque veut dire "tout et n'importe quoi".

Voici une liste des syntaxes possibles, pour vous donner une idée :

Syntaxe	Signifie
Bienvenue*	N'importe quelle ligne commençant par Bienvenue .
*classe	N'importe quelle ligne finissant par classe .
passage	N'importe quel ligne contenant (au début, à la fin ou au milieu) passage. Notez que ce trigger sera aussi déclenché si la ligne contient passager, par extension.
* passage *	N'importe quelle ligne contenant le mot passage entouré d'espaces. Le mot passager ne déclenchera plus ce trigger cette fois.
Vous gagnez * crédits en *.	Des lignes comme Vous gagnez 80 crédits en combat. ou Vous gagnez 10 crédits en management. déclencheront ce trigger. La ligne Vous gagnez un certain nombre de crédits en quelque chose. déclenchera le trigger également.

En bref, un signe astérisque veut dire n'importe quoi : une lettre, un chiffre, un mot, un nombre, un espace, un signe de ponctuation, un message assez long... voire rien du tout.

Un mot de mise en garde : la syntaxe de vos déclencheurs est très importante, et vous devriez vérifier avec soin les lignes que vous souhaitez intercepter.

```
*table*
```

Ce trigger se déclenchera quand le mot table se trouve dans une ligne, au début, au milieu ou à la fin. Ce trigger se déclencherait donc avec les lignes suivantes :

```
~ un tableau noir (commande board) se trouve ici
Dans l'angle nord-est de la cour se trouve un long bâtiment de bois peint en rouge, probablement u
ne étable.
```

Souvenez-vous qu'un trigger est autant facile à restrindre que facile à déclencher. Il vous faut trouver le bon équilibre entre les deux.

Retournons aux variables. Le signe astérisque (*) fait deux choses :

- Il aide à décrire quand le trigger doit se déclencher (il veut dire "n'importe quoi") ;
- Il écrit dans une ou plusieurs variables.

Utilisons le même exemple, avec notre trigger [hrp[*. Qu'arrive-t-il lorsque vous recevez une ligne comme celle-ci :

```
[hrp] Edgar dit : je n'y comprend rien, quelqu'un pourrait-il m'aider ?
```

D'abord, le trigger [hrp]* est déclenché, puis la partie après [hrp] (celle décrite par le signe astérisque) est capturée dans une variable. Les variables permettent de conserver des informations, et c'est justement ce qu'elles font ici. Les variables sont numérotées en partant de $1, $2, $3 et ainsi de suite. Donc dans notre exemple, si on reçoit la ligne suivante, le trigger va créer une variable $1 contenant la partie après [hrp] :

```
 Edgar dit : je n'y comprend rien, quelqu'un pourrait-il m'aider ?
```

Que peut-on faire avec cette variable ? Beaucoup de choses. Chaque paramètre de nos actions liées au trigger peuvent utiliser les variables du trigger. Nous verrons des exemples concrets un peu plus bas, mais pour l'heure, nous pouvons commencer par

l'afficher. Vous pouvez entrer l'instruction suivante dans votre client :

```
#say $1
```

Qui devrait vous afficher :

```
 Edgar dit : je n'y comprend rien, quelqu'un pourrait-il m'aider ?
```

> Pourquoi la ligne commence par un espace ?

Si vous vous posez cette question, essayez d'écrire le déclencheur du trigger et la ligne reçue l'un à côté de l'autre, cela devrait vous aider à comprendre :

- Déclencheur : [hrp]* ;
- Ligne reçue : [hrp] Edgar dit : je n'y comprend rien, quelqu'un pourrait-il m'aider ? .

Vous avez trouvé ? La ligne reçue par le client commence par "hrp" entre crochets, un espace puis le nom de la personne qui parle... alors que notre déclencheur capture tout ce qu'il y a après le crochet fermant de "[hrp]", ce qui inclue notre signe espace dans ce cas.

La solution : modifier quelque peu notre déclencheur.

```
[hrp] *
```

Cette fois, on met un espace entre le crochet fermant et le signe astérisque. Si l'on reçoit la ligne :

```
[hrp] Edgar dit : je n'y comprend rien, quelqu'un pourrait-il m'aider ?
```

Et qu'on affiche $1, on devrait voir :

```
Edgar dit : je n'y comprend rien, quelqu'un pourrait-il m'aider ?
```

Les espaces dans les déclencheurs pourraient bien être l'une des erreurs principales quand on crée ses propres triggers la première fois. Le meilleur conseil que je puisse vous donner est de regarder avec attention les lignes que vous voulez surveiller à l'aide de triggers, et de n'utiliser le signe astérisque (*) que quand vous n'avez pas de moyen de savoir ce qui s'y trouve.

Les canaux CocoMUD dans les triggers

CocoMUD offre un système assez puissant de canaux. Cette fonctionnalité est à différencier des canaux en jeu, qui sont dépendants du jeu où vous vous connectez. Les canaux CocoMUD sont des listes d'`'evènements dans lesquels vous mettez ce que vous voulez. Ce n'est pas obligatoire de mettre des canaux en jeu dans des canaux CocoMUD, c'est juste plus comfortable. Nous allons voir cela (pourquoi l'utiliser et comment faire).

Vous trouverez plus d'informations sur ce concept et leur utilisation avec triggers en lisant la documentation des canaux.

Les triggers muets

Dans certains cas, quand on reçoit une certaine ligne, on ne veut pas l'afficher sur le client.

Cela arrive vraiment ?

Parfois. Par exemple, certains MUD envoient des messages d'ambiance régulièrement (toutes les 3-5 secondes) quand vous vous trouvez dans une certaine situation. Ce peut être agréable pour mettre dans l'ambiance, mais pas toujours avec un lecteur d'écran.

```
Un noeud dans le bois éclate en une pluie d'étincelles.
```

Joli et, bien il faut le reconnaître, près d'un feu de camp il est normal qu'il pétille, lance des étincelles et craque de temps en temps en temps. Mais avec un lecteur d'écran, ce n'est pas le plus utile. Nous allons donc retirer cette ligne de l'affichage.

Pour ce faire, créez un nouveau trigger. En passant par l'interface :

- Ouvrez le menu **Jeu** -> **Trigger** ;
- Ajoutez un trigger en cliquant sur **Ajouter** ;
- Collez la ligne dans le déclencheur : Un noeud dans le bois éclate en une pluie d'étincelles. ;
- Pas besoin de sélectionner une action, sauf si vous voulez jouer un son approprié, cela dit.
- Tabulez jusqu'à trouver la case "Trigger muet". Cochez-la.
- Validez plusieurs fois sur **OK** pour sortir de la boîte de dialogue en sauvegardant.

Si le client reçoit cette ligne, il ne l'affichera plus.

Créer ce trigger en SharpScript est assez simple :

```
#trigger {Un noeud dans le bois éclate en une pluie d'étincelles.} {} +mute
```

Notez que le second paramètre (définissant la liste d'actions associées à ce trigger) est vide dans notre exemple. Le troisième paramètre quant à lui est un flag, commençant par + ou - et suivi du nom du flag (ici mute pour créer un trigger muet).

Vous pouvez très bien avoir un trigger muet qui exécute cependant des actions, cela n'est pas du tout incompatible.

Les triggers marqués

Les triggers marqués peuvent être utiles pour l'accessibilité. Quand un trigger marqué se déclenche, il place le curseur directement sur la ligne ayant déclenché ce trigger. Vous pourriez avoir un trigger sur la ligne décrivant les sorties d'une salle, par exemple. Si ce trigger est marqué, quand vous explorerez plusieurs salles, le curseur sera toujours déplacé sur la listes des sorties, plutôt que ramené en bas de la fenêtre, où vous devrez appuyez plusieurs fois sur la flèche haut pour lire les sorties disponibles.

C'est le même principe pour créer ce trigger :

- Dans la barre de menu, sélectionnez **Jeue** -> **Triggers**.
- Cliquez sur le bouton **Ajouter** pour créer un nouveau trigger.
- Écrivez dans le déclencheur quelque chsoe comme : Sorties :* .
- Tabulez plusieurs fois pour cocher la case "trigger marqué".

La prochaine fois que vous recevrez une ligne commençant par Sorties : , le curseur sera déplacé automatiquement dessus.

Créer ce trigger avec une instruction SharpScript donne :

```
#trigger {Sorties : *} {} +mark
```

Les triggers avec substitution

Dans certains cas, quand un trigger s'exécute, on veut modifier la ligne qui a déclenché le trigger. L'un des exemples les plus fréquents est pour réduire une ligne un peu longue. Certains MUDs ont de longues lignes de texte et mettent l'information importante

à la fin, ce qui n'est pas super pratique avec un lecteur d'écran. Par exemple :

```
Quelqu'un parle publiquement sur le canal 'hrp' avec une petite voix inquiète : c'est sans danger ?
```

Bien que cette ligne de texte en apprenne beaucoup, l'information vraiment importante (le message) se trouve tout à la fin. Ce pourrait être bien de raccourcir un petit peu cette ligne de texte, et peut-être changer l'ordre des informations.

```
[hrp] Quelqu'un : c'est sans danger ? (une petite voix inquiète)
```

Pour faire cela, il faut créer un trigger, et créer une ligne de substitution. Quand le trigger est appelé, la ligne de substitution s'affichera à la place de la ligne d'origine qui a déclenché le trigger.

- Dans la barre de menu **Jeu** -> **Triggers** ;
- Cliquez sur **Ajouter** pour créer un nouveau trigger ;
- Écrivez le déclencheur comme d'habitude :

```
* parle publiquement sur le canal '*' avec * : *
```

- Notez que $1 contient le nom de l'auteur du message, $2 contient le nom du canal, $3 contient la voix utilisée et $4 contient le message lui-même ;
- Tabulez jusqu'à trouver la zone de texte appelée "Message de remplacement de la ligne ayant déclenché le trigger".
- Dedans, écrivez :

```
[$2] $1 : $4 ($3)
```

C'est compréhensible ? Si ce n'est pas le cas, prenez le temps de relire à quoi correspond chaque variable.

Créer le même trigger en SharpScript serait :

```
#trigger {* parle publiquement sur le canal '*' avec * : *} {} {[$2] $1 : $4 ($3)}
```

Note importante : nous avons trois arguments ici, dans notre action #trigger. Le premier contient toujours le déclencheur du trigger, le second la liste d'actions (vide ici). Le troisième contient la chaîne de substitution. Si il n'y a pas de troisième paramètre, ou que le paramètre est vide, la ligne est affichée telle qu'elle (c'est le cas dans tous nos exemples précédents).

Les triggers déclenchés par expressions régulières

Cette section et celles qui suivent sont un peu plus avancées.

Le symbole astérisque (*) est très pratique. Mais il n'est pas très précis. CocoMUD permet d'écrire des expressions régulières . Le but ici n'est pas de décrire la syntaxe de ces expressions, c'est un sujet à part entière, mais vous trouverez de nombreuses ressources (à commencer par le lien ci-dessus).

Pour utiliser des expressions régulières dans des déclencheurs de trigger, il suffit de commencer le trigger avec le signe ^. CocoMUD comprend que ce déclencheur doit être une expression régulière. Par exemple :

```
^Vous recevez \d+ XP.$
```

Ce trigger sera déclenché si vous recevez la ligne "Vous recevez ... XP.", avec ... étant un ou plusieurs chiffres. Ce trigger ne sera

pas déclenché par la ligne : "Vous recevez un peu d'XP."

Une chose importante à garder à l'esprit quand on utilise des triggers avec expressions régulières, cependant, est que si on veut capturer des informations, il faut utiliser des groupes de capture (des parenthèses).

```
^Vous recevez (\d+) XP.$
```

Ici, le nombre d'XP sera mis dans $1. Vous pouvez aussi utiliser des groupes nommés et y faire référence avec $nom_du_groupe.

Des triggers avancés en Python

Le moteur SharpScript est léger et puissant, mais ce reste un langage de script qui ne permet pas tout. Et qui ne permettra pas tout : il est censé resté léger et optimisé. Ce n'est pas un langage de programmation. Mais Python en est un, et CocoMUD est développé en Python. Le moteur SharpScript a une syntaxe particulière pour envoyer du code à Python directement, ce qui permet d'outre-passer les limites du moteur SharpScript de façon assez définitive. Depuis le code Python, on peut toujours utiliser les fonctions SharpScript, ce qui rend l'élaboration de triggers plus complexes assez simple, dès lors que l'on connaît Python.

Nous allons prendre un exemple, comme toujours, mais souvenez-vous qu'il n'y a pas de réelle limite à ce que vous pouvez faire, tant que Python le permet.

Admettons que le jeu envoie une ligne quand on reçoit de l'XP mais précise aussi le nombre d'XP nécessaire pour passer au niveau suivant.

Par exemple :

```
Vous recevez 37 XP et avez besoin de 500 pour passer au niveau suivant.
```

On va vouloir extraire ces deux nombres et afficher un pourcentage à la place : xp / total * 100. Ce n'est pas possible en utilisant simplement du SharpScript.

Pour l'instant, ce n'est pas possible d'utiliser l'interface pour manipuler du code Python. Vous devrez faire les essais en éditant directement le fichier "config.set".

```
#trigger {Vous recevez * XP et avez besoin de * pour passer au niveau suivant.} {+
    # $1 contient le nombre d'XP gagné
    # $2 contient le total pour passer au niveau suivant
    xp = args["1"]
    total = args["2"]

    # On convertit ces deux nombres
    try:
        xp = int(xp)
        total = int(total)
    except ValueError:
        # La conversion n'a pas marché, mais on ne fait rien
        pass
    else:
        pourcent = xp * 100.0 / total
        say("Vous recevez {}/{} XP ({}%).".format(xp, total, int(pourcent)))
}
```

Voilà du trigger plutôt avancé ! Quelques explications :

- Le déclencheur du trigger ne devrait pas être une grosse surprise pour vous à ce stade ;
- Le second paramètre commence par {+ (accolade gauche suivi du signe +). C'est la syntaxe pour dire à CocoMUD que ce qui se trouve entre les accolades est du code Python.
- Notez que tout le code est indenté. Il ne s'agit plus d'un comfort de lecture ici, mais d'une nécessité. Python a besoin de

l'indentation pour fonctionner. J'ai utilisé 4 espaces, mais vous pouvez en utiliser un (ou un signe de tabulation si vous voulez) ;
- Quelques commentaires. Cela peut toujours être utile.
- On extrait les deux nombres xp et total. Pour accéder aux variables du trigger, on utilise args qui est un dictionnaire les contenant. La première variable étant $1, on y accède en Python grâce à args["1"].
- On a besoin de convertir ces variables. Pourquoi ? Ce sont toujours des chaînes de caractère à ce stade, il faut les convertir en nombre. CocoMUD ne vérifie pas que ce sont des nombres, on doit donc le faire manuellement. C'est pourquoi on convertit dans un bloc try/except/else. Notez qu'on ne fait rien si la conversion échoue pour X raison.
- On crée ensuite le pourcentage. Puisqu'il s'agit de Python2, il faut lui indiquer de faire attention aux virgules.
- On affiche notre message ensuite, en utilisant la fonction say|(). C'est exactement comme appeler la fonction #say en SharpScript, c'est la même fonction qui est appelée. De même façon, on peut utiliser les fonctions send(), play(), feed() et autre. La syntaxe diffère car on est en Python, mais ce sont les mêmes fonctions et mêmes arguments.

Bien... je ne suis pas vraiment satisfait par le trigger que je viens de créer... on pourrait le raccourcir un petit peu, et le rendre plus lisible. Il ne faut pas toujours chercher la concision, mais ça aide. Si on était sûr que nos variables contiennent des nombres, on aurait moins de code à faire :

```
#trigger {^Vous recevez (\d+) XP et avez besoin de (\d+) pour passer au niveau suivant.$} {+
    # $1 contient le nombre d'XP gagné
    # $2 contient le total pour passer au niveau suivant
    xp = int(args["1"])
    total = int(args["2"])
    pourcent = xp * 100.0 / total
    say("Vous recevez {}/{} XP ({}%).".format(xp, total, int(pourcent)))
}
```

Utiliser les expressions régulières ici ajoute un peu de complexité au déclencheur, mais ça rend le code bien plus lisible je trouve.

Si vous voulez plus d'informations sur l'utilisation de code Python dans les instructions SharpScript, vous trouverez une section détaillée dans la documentation du SharpScript.