Normes de codage ChatScript

Traduction Mathieu Rigard 02/12/2017 pour Utopia : m.rigard@utopia-french.tech

A faire : correction et ancrer les liens.

Contenu:

Les normes de codage vous permettent à vous, ainsi qu'aux autres, de comprendre votre code.
Ils peuvent également permettre à d'autres personnes qui ne sont pas des programmeurs de voir un résumé de votre code, généré par CS : c’est la fonctionnalité *:abstract*.
Pour utiliser *:abstract*, connectez-vous simplement en local avec un nouveau nom d'utilisateur (ou effacez le contenu du dossier USERS), puis tapez " :abstract ~mytopic" pour un topic. Le résumé ne sera pas lisible à l'écran mais est capturé dans le journal (log) de l'utilisateur. Renommez simplement ce fichier journal et déplacez-le ailleurs.

Indentation des règles

Ne pas indenter les règles de niveau supérieur.

Réponses en retrait (une tabulation) par niveau.

Laissez la ligne vide avant une règle de haut niveau.

Exemples :

t: Comment vas-tu ?
	a: (~goodness) content pour toi.
		b: (toi) pas si mal.
	a: (~badness) désolé pour toi.
	b: (toi) pas si mal. # à ne pas faire - mauvais niveau d'indentation

u: (Comment vas-tu) Je vais bien.
u: (et après) xxx # à ne pas faire - coincé immédiatement après la règle de haut niveau

	u: (pourquoi pas) parce que. # c'est une mauvaise indentation de la règle de haut niveau.

Explications : D'une part, vous voulez minimiser l'espace inutile sur une page. Alors
les règles de premier niveau ne doivent pas être indentées[1] (ni leurs exemples d'entrées). D'autre part, vous voulez voir facilement la structure du code, alors mettre des tabulation (indenter) vos répliques est de nature à clarifier la structure du dialogue. Ainsi vous pourrez repérer facilement l'endroit où un segment de règle se termine et où un autre commence. Donc, je sépare les règles de haut niveau et je garde les répliques ensemble.

1 typographie : une **indentation** est un retrait de la première ligne d'un paragraphe.

Étiquettes de règles

Faire des étiquettes significatives en utilisant des lettres majuscules.

Étiquetez toutes les règles qui ont leur propre texte de sortie.

Exemples :

t: DEMANDE_EMAIL () Quelle est votre adresse email ?
    a: EMAIL_DONNE (~email_url) Merci.
    a: () ^reuse (EMAIL_DONNE)

t: B55 () Quelle est votre adresse email ?# à ne pas faire - étiquette sans signification

Explications :

En utilisant des lettres majuscules pour les étiquettes de règles (à la fois à la règle et dans les endroits avec ^reuse où l'on réutilise la règle), il est facile de les voir dans une masse de texte comme on en obtient avec la commande :trace.

Si vous ne travaillez pas sur votre propre projet de bot perso, chaque règle qui génère une sortie devrait avoir un tel label qui apparaîtra dans un résumé. Cela permet aux autres de se référer à votre règle et peut-être de la trouver dans les fichiers journaux des clients.

Des modèles faciles à lire

Évitez les parenthèses superflues.

Pour des motifs multiples, placez chacun sur sa propre ligne.

Exemples:

```
?: QUE_PRODUIT_BRILLIG ([
        ( Que * "Brillig Understanding" )
        ( [dit parle savoir] * ~2 "à propos de" "Brillig Understanding" )
        ])
        Brillig Understanding fait des systèmes de langage naturel.


u: MAUVAIS_CHOIX ([
        (<< société Brillig >>) # parenthèses superflues
        (~monmot) # parenthèses superflues
        ([mot1 mot2]) # parenthèses superflues
        ])
```

Explications :

Les parenthèses superflues rendent les modèles plus difficiles à lire, donc plus difficiles à corriger visuellement. Ils ralentissent également le moteur CS, mais pas de manière sensible.

Lorsque vous utilisez plusieurs modèles dans une règle, les disposer chacun sur sa propre ligne vous permet (ou au correcteur du code) de les visualiser chacun séparément.

Donner des exemples d'entré

Donner des exemples d'entré pour les répondeurs et les répliques.

Donnez plusieurs échantillons pour des constructions de phrase follement différentes lorsque vous avoir plusieurs Modèle dans une règle.

Exemples :

```
t: De quelle année s'agit-il ?
        #! 1993
        a: (~ année) Super.


#! Comment vous appelez-vous
#! Qui es-tu?
?: ([
        ('tu [nom appelle])
        (qui est 'tu)
        ])
        Je m'appelle Rose.
```

Explications :

Les exemples d'entrées expliquent vos habitudes. Au lieu d'avoir à interpréter le Modèle, vous savez immédiatement ce que la règle a l'intention de faire.

Les exemples d'entrées vous permettent d'utiliser *:abstract* pour donner aux non-programmeurs une vue d'ensemble de votre code. Les exemples d'entrées permettent également à CS de tester votre code à l'aide de *:verify*.

Sortie de Règle facile à lire

Indenter chaque phrase et chaque énoncé de code CS sur des lignes distinctes.

Exemples :

```
#! Test
u: (test)
	$status += 1
	$_x = ^compute(1 + 2)
	La réponse est $_x.
	L'avez-vous manqué ?


#! Où êtes-vous né
u: (où * vous * né) # mauvais - difficile à lire
Je suis né à San Francisco près de la vieille église sur la colline. Je suis né fils unique mais mes parents en ont toujours voulu d’autres.


#! Où êtes-vous né
	u: (où * vous * né) # correct
	Je suis né à San Francisco près de la vieille église sur la colline.
	Je suis né un enfant unique mais mes parents en ont toujours voulu d’autres.
```

Explications :

La sortie de l'utilisateur à plusieurs phrases est plus difficile à lire sur une même ligne, cela peut donner de très longues lignes. Le code est certainement plus difficile à comprendre quand plusieurs actions sont sur la même ligne. D'un autre côté, si la la sortie est minuscule, vous pouvez le mettre sur une seule ligne comme ceci:

```
u: (test) OK. $status += 1
```

Ensembles de règles connexes

Mettre les règles associées dans un même Topic.
Mettez des règles plus étroitement liées ensemble lorsque vous le pouvez.

Exemples :

topic: ~aliens ()
…

topic: ~ family ()

#! x *** Parents

#! Qui est ta mère
?: (qui * mère) Maman.

#! Que fais ta mère?
?: (Que * mère * fait) Elle travaille.

#! Qui est ton père? # Note que nous avions toutes les choses de la mère ensemble avant le père
?: (qui * père) Papa.

#! Que fait ton père?
?: (Que * père * fait) Il travaille.

#! x *** Frères et sœurs
…

Explications :

Avec le groupage approprié, vous devez immédiatement savoir où ajoutez une nouvelle règle ou découvrez que plusieurs règles effectuent le même travail.

On peut même annoter des groupes de règles pour utiliser :abstract en marquant #!x commentaires qui seront alors visibles pour étiqueter le groupe et le rendre facile à suivre.

Le seul problème peut être lorsque certaines règles en amont viennent bloquer l’expression d’une leur succédant et vous oblige à déplacer quelque chose. Envisagez d'utiliser les exclusion *!mots-clés*.

Localisation de concepts/fonctions

Si un concept ou une fonction est utilisé uniquement dans un fichier, placez-le dans ce fichier.

Si un concept ou une fonction est utilisé à plusieurs endroits, utilisez un fichier global functions.top ou concepts.top.

Pour les tables utilisées uniquement dans un fichier, cela dépend de la taille de la table.

Explications :

Lorsque vous pouvez localiser une donnée dans le fichier qui l'utilise, cela le rend plus facile d’en trouver la définition quand vous voulez l'inspecter.

Casse des mots-clés et fautes d'orthographe

Utilisez un dictionnaire de casse standard (ou votre dictionnaire de casse propre dans le cas de mots non répertoriés) pour les mots-clés dans concepts ou dans les Modèles.

Évitez de mettre des fautes d'orthographe dans les concepts ou les mots-clés. Utilisez *replace:* dans votre fichier de scripts pour ajouter efficacement des entrées aux fichiers de substitutions prédéfinis dans ChatScript. Mais utilisez *:replace* seulement quand le mot mal orthographié a une seule valeur évidente possible. *replace:* change le dictionnaire pour TOUT LES BOTS.

Pour les fautes d'orthographe qui sont des mots acceptables, ajoutez-les à concepts ou aux mots-clés.

Exemples :

concept: ~protocoles (Blu-ray WiFi)
replace: blue_ray Blu-ray # Toujours. Mettre cela comme orthographe correcte
replace: MS Microsoft # Mauvais – vrai pour les PC, pas pour le médico-social

#! I used my brakes
u: ([brake break]) Don’t hit the brakes too hard. # misspell is a real word, list it

#! J'aime mars
u: (_~planets) '_0 # produit Mars même si l'utilisateur a tapé mars

#! Aimez-vous les moutons?
u: ([mouton moutton bouton]) J'aime les moutons # c’est généralement une mauvaise idée de faire cela

Justification: Si un mot existe avec un seule casse dans le dictionnaire, CS peut ajuster automatiquement l'entrée pour la rendre correcte.

Même si un mot existe dans plusieurs casses, si un ensemble de concepts l'a dans d’une seule façon (par exemple ~planets ont le nom Mars et non le mois de mars), si vous le mémorisez, CS peut mémoriser la casse correcte.
Parfois, vous vous attendez à des fautes de frappe. Mettre la forme erronée dans vos modèles (comme moutton), signifie le faire entrer dans le dictionnaire comme mot acceptable. Cela signifie que la correction orthographique peut changer les "fautes d'orthographe" de fautes d'orthographe, provoquant la dérive des mots pour une entrée comme "hep". Préfère utiliser *replace:* si une faute d'orthographe ne peut aller qu’à un endroit évident.