Prototypage d'interfaces par langage de script

Projet Turtle

M1 Informatique

RUAN Henry - FRANCHINA Lucas

**Explication du code:**

Le code de la tortue se présente en trois programmes:

* editeurSimple.py : un éditeur simple qui ne gère pas les fonctionnalités complexes (Répéter)
* editeur.py : un éditeur ergonomique gérant des fonctionnalités complexes
* visualiseur.py : un visualiseur amélioré du bus Ivy.

Nous avons rencontré un problème avec la méthode IvySendMessage pour les deux éditeurs, car elle ne semblait pas envoyer de messages sur le bus Ivy. Toutefois, nous avons trouvé une solution alternative : exécuter le processus (subprocess) dans un thread (threading) et récupérer les flux entrants/sortants pour permettre au programme Python d'interagir avec le bus Ivy.

1. **editeurSimple.py**

Ce code optimisé crée un éditeur simple qui permet à l'utilisateur d'envoyer des commandes via le bus Ivy en utilisant une interface graphique basée sur Tkinter. Un seul thread est utilisé pour gérer l'envoi des commandes plutôt que d'en créer un à chaque pression sur la touche "Enter".

**Explication plus poussée :**

* Les bibliothèques nécessaires sont importées : tkinter, subprocess, threading et queue.
* Une instance de la fenêtre principale Tkinter (root) est créée avec une dimension spécifique de 210x20 pixels et un titre "Editeur Simple".
* Une file d'attente (commands\_queue) est créée pour stocker les commandes à envoyer.
* La fonction command\_sender() est définie pour traiter les commandes dans la file d'attente de manière continue. Elle récupère et envoie les commandes une par une et marque la tâche comme terminée.
* La fonction run\_command() est définie pour ajouter la commande à la file d'attente et réactiver la zone de saisie.
* La fonction on\_enter\_key(event) est définie pour réagir à l'événement appuyer sur la touche "Enter". Lorsque l'utilisateur appuie sur "Enter", cette fonction désactive temporairement la zone de saisie, affiche la commande envoyée dans la console et appelle la fonction run\_command().
* Une zone de saisie (Entry) est créée pour que l'utilisateur puisse entrer des commandes et liée à l'événement appuyer sur la touche "Enter".
* Le script ivyprobe.py est lancé avec la bibliothèque subprocess. Les flux stdin, stdout et stderr sont configurés pour communiquer avec ce processus.
* Un thread unique est créé pour gérer l'envoi des commandes et démarré avant d'exécuter la boucle principale de Tkinter.
* La boucle principale de Tkinter est exécutée pour afficher la fenêtre et gérer les événements de l'interface graphique.

En somme, ce code crée une interface utilisateur simple permettant de saisir des commandes, qui sont ensuite ajoutées à une file d'attente et traitées par un seul thread pour être envoyées sur le bus Ivy en utilisant le script ivyprobe.py lorsqu'on appuie sur la touche "Enter".

1. **editeur.py**

Dans ce fichier, une classe EditeurDeTexte a été créée, qui créé un éditeur de texte ergonomique permettant d’importer, d’exporter des fichiers XML,d’ajouter, de modifier ou de supprimer des commandes pour contrôler une turtle graphique en utilisant le bus Ivy.

De la même façon, editeur.py utilise une file d’attente traitée par un seul thread pouvant être envoyé sur le bus Ivy en utilisant le script ivyprobe.py lorsqu’on appuie sur le bouton Send.

Ce programme offre davantage de fonctionnalités :

Effectivement, l'ensemble des boutons est présent dans une interface ergonomique. Par exemple, l'utilisateur remplit le champ à côté du bouton "Avancer", puis clique sur le bouton, et à droite de l'écran s'affiche "AVANCER X", avec X étant la valeur entrée dans le champ Tk.Entry à côté de "Avancer".

Chaque Tk.Entry est associé au bouton situé à sa gauche.

Des boutons + et - sont disponibles avec plusieurs options :

Si l’utilisateur clique sur un label, puis clique sur + il peut ajouter une ligne au-dessus de ce label, s’il clique sur - le label sélectionné est supprimé.  
Pour ajouter une ligne en dessous du label, l’utilisateur peut soit clique droit, ajouter une ligne en dessous, soit sélectionner le label, et cliquer sur un des boutons pour ajouter une commande à la suite.

De la même façon, si un label est sélectionné, l’utilisateur en cliquant sur un bouton modifiera ce dernier.

De plus, la commande envoyée sur le bus Ivy est visible en temps réel grâce à un surlignage en jaune de la commande en cours d'exécution.

**Fonctionnalités de l'éditeur:**

* Utilisation ergonomique des commandes
* Importer un xml pour visualiserles commandes
* Exporter en xml
* Envoyer sur le bus Ivy avec la commande “Send”
* Suppression / Modification / Ajout de commande
* Gestion des répétitions, avec imbrication possible

.

1. **visualiseur.py**

Visualiseur.py comprend deux classes : une classe Turtle permettant de déplacer la tortue en fonction des messages Ivy, et une classe CustomDialog, qui est un menu textuel développé pour modifier l'historique des commandes reçues.

IvyBindMsg relie Ivy aux fonctions de la tortue à l'aide de Regex.

**Fonctionnalités du visualiseur:**

* Importer du xml si ce dernier n’a pas de REPETER
* Sauvegarder l’image actuel en JPEG
* Sauvegarder l’historique actuel en XML
* Nettoyer le dessin ainsi que l’historique grâce à Clear
* Activer ou Désactiver la gomme dans le menu Outils
* REPLAY l’historique
* Modifier / Supprimer / Ajouter des commandes à l’historique
* Modifier la vitesse de Replay de l’historique.
* Lancer l’éditeur

Grâce au bus Ivy, il est possible de lancer plusieurs visualiseurs simultanément afin de recevoir les données envoyées par l'éditeur de texte sur chaque visualiseur.