Mathias DESOYER, Sully MILLET, Axel LE FAUCHEUR, Mathéo GUENEGAN

Projet de synthèse L3 UBO IFA

Classes :

réseau :

- InterpreteurServeur.java : calcule les resultats du script reçu par le server et convertis les images en base 64
- InterpreteurServeurTextuel.java : calcule les resultats du script reçu par le server et convertis les images un script simplifié

- Message.java : definit le format d'un message en json
- RobiClient.java : envoi le script au server et reçois les résultats en base64 puis affiche les images
- RobiClientTextuel.java : envoi le script au server et reçois les résultat en script simplifié puis affiche les images
- RobiServer.java : reçois le script du client puis envoi renvoi le résultat de l'interpréteur au client
- RobiServerTextuel.java : reçois le script du client puis envoi renvoi le résultat de l'interpréteur au client

IHM :

- CloneBloc.java : Permet la duplication de blocs
- CommandeBloc.java : Definit des commandes avec son nom et ses arguments
- FonctionnaliteBloc.java : Permet la suppression via un clic droit sur un bouton
- IHMController.java : Instance qui gère toute les interfaces
- RobiCoderALaMain.java : Interface permettant de coder en ligne
- RobiScratch.java : Interface permettant de coder en blocs
- ScriptsAssembleur.java : Permet d'assembler les blocs de scripts
- ScriptsInterpreteur.java : Permet de compiler les blocs en script lisible (format string)

Fonctionnement :

- 1) démarrer RobiServeur ; 2) démarrer IHMControlleur ; 3) créer des scripts
- Côté Scratch -> les blocs sont dupliqués et assemblés de part leur hauteur, lors de l'exécution, le scriptInterpreteur traduit les blocs chaînées en commandes exécutables, puis l'IHMController envoie les commandes exécutables au serveur. Le serveur fait fonctionner les commandes reçues en une fois, puis renvoie les images en base64 (en format String), l'IHMController reçoit les images et traduit la chaîne courante en Image (class) et l'affiche sur l'afficheur (en haut à droite de l'IHM scratch).

- Côté Coder à la main -> l'utilisateur écrit du script en format texte, lors de la demande pour envoyer au serveur, le script est récupéré et est envoyé au serveur qui fait comme pour le côté scratch, run les différentes commandes, et renvoie les image en base64. L'interface s'occupe de retraduire l'image courante base64 en Image (class) et l'affiche en haut à droite

- Côté serveur implémenté pour l'IHM : Le script reçu est joué dans un interpreteur puis renvoie les différentes images générées et traduites en base64.

- Côté serveur textuel : Le script reçu (non implémenté dans l'IHM donc script définit dans un Main) est joué pas à pas dans un Interpreteur qui pour chaque référence définit un Component et effectue des commandes qui modifient les positions de ces Components ex : (robi translate 20 20) si robi avait de base une position (x : 20 et y : 30) alors sa nouvelle position est (x : 40 et y : 50). Une fois les commandes jouées , le serveur renvoie tous ses components dans un String de type (RECTANGLE couleur x y w h) où couleur, x, y, w, h sont des variables. Puis le client côté textuel récupère les différents éléments d'un état et les traduit en GElement puis les ajoute dans un GSpace.

Bilan critique :

- L'exercice 6 n'a pas été fait (Le fonctionnement des SNodes a été compris trop tard)
- Dans l'exercice 5, quand une référence est supprimée, les enfants de cette référence existent toujours, il aurait fallu pour chaque référence créé, définir un environnement local et l'ajouter à cette référence.
- Le côté scratch implémente des commandes trop basiques, il aurait été intéressant d'ajouter des loops, des ajouts img etc.. De plus, les scripts générés sont générés d'une manière qui fait qu'un élément effectue tout son script avant le démarrage d'un autre élément. Il aurait été possible de traduire le script en ajoutant la commande 1 de l'objet puis la commande 1 de l'objet 2 etc.. (Manque de temps).
- L'Interface côté scratch aurait pu être simplifié niveau code, avec notamment l'utilisation de layout à la place de calculer durement les positions des JPanels.
- Le mode pas à pas ne correspond pas à un vrai mode "pas à pas", dans le sens où tout le script est joué en une fois côté serveur. Il y a eu un problème de compréhension sur la consigne.
- L'utilisation de Git a été hasardeuse par moment, faisant perdre un temps assez conséquent.
- Si un client écrit un script avec des erreurs, le serveur ne gère pas les erreurs. Pour le mode pas à pas, le serveur aurait pu renvoyer des erreurs si une commande envoyée génère un problème (problème de syntaxe, de référence etc) puis, il aurait pu y avoir un affichage de ces erreurs côté client.
- En général pour le développement, la mise en œuvre a été chaotique, dans le sens où il y a eu un manque de communication entre les développeurs, manque d'une direction claire pour le projet (de notre faute). Des fonctionnalités ont été ajoutées alors qu'elles n'étaient pas prévues, ce qui a généré plein de bugs, de conflits et

donc de perte de temps . Plus concrètement, il aurait fallu définir un chef de projet, créer dans un premier temps un diagramme UML de toute l’application, compris par tous et définir les limites du projet.