Raphael BOUILHOL
Bastien LE SAULNIER

25/01/2019

Rapport Projet C++
Thème Yellow

Introduction

Nous avons souhaité coder notre propre version du jeu mobile “Human Ressource Machine” du studio Tomorrow Corporation. Non seulement car nous apprécions le jeu, mais aussi parce que son fonctionnement convenait parfaitement à l’utilisation d’un langage orienté objet, et enfin car sa complexité nous permettrait de répondre aisément aux contraintes d’implémentation imposées.

Description de l’application

(Il serait bon d’avoir une photo de la dernière version de l’interface)

Le but de ce jeu est de traiter une “pile” de données (Entree) afin d’en calculer la bonne “pile” de sortie selon une consigne donnée. Le traitement se fait à l’aide de commande très bas niveaux décrites plus bas. Le joueur dispose aussi 8 cases mémoires pour effectuer son traitement, ainsi qu’une case “active” avec laquelle il fait le traitement.

Les données à traiter sont des nombres. L’entrée est donc représentée par une pile de nombre.

Les commandes sont :

- Entrée : Prend une donnée en entrée, la place dans la case active.
- Sortie : Place en sortie la case en case active.
- Addition n : Additionne la case active avec la case n. Le résultat est placé dans la case active. (n < 8)
- Soustraction n : Effectue la soustraction de la case active par la case n. Le résultat est placé dans la case active. (n < 8)
- Incrémentation : Incrémente de 1 la case active.
- Décrémentation : Décrémente de 1 la case active0
- Saut n : Fait un saut relatif dans le code.
- Saut si nul n : Le saut est effectué ssi la case active contient 0.
- Saut si négatif n : Le saut est effectué ssi la case active contient une donnée de valeur négative.
- Sauvegarder n : Stock la donnée dans la case active vers la case mémoire numéro n. (n < 8)
- Charger n : Copie dans la case active la donnée en case mémoire n. Une copie de la donnée reste dans la case active. (n < 8)

Lors de l’exécution du code, il est possible par l’utilisateur d’observer à tout moment le contenue de la mémoire, de la case active, de l’entrée et de la sortie.

Le code est entré par l’utilisateur dans une case texte à droite de l’écran. Une fois le code complet, il clique sur “Compiler”. Si le code ne donne pas d’erreur de compilation, le joueur peut alors lancer l’exécution de son programme selon 3 modes : pas à pas, continue,

instantanée. Ces trois modes d'exécutions sont choisis grâce aux différents boutons de lecture :

- Continue : Les lignes de codes sont exécutées automatiquement à une fréquence assez faible pour permettre à l'utilisateur d'observer le fil d'exécution de son programme. Cette exécution peut être stoppée à tout moment en cliquant sur le bouton correspondant.
- Pas à pas : Pour chaque clique sur ce bouton, une ligne de code est exécutée.
- Instant : Le code est exécuté instantanément.

L'exécution s'arrête lorsque le programme termine, ou lorsqu'il y a une erreur d'exécution. Le programme termine quand il n'y a plus d'instructions à exécuter ou lorsque l'instruction "ENTREE" est utilisé alors qu'il n'y a plus de donnée dans la pile d'entrée.

En cas d'erreur dans la compilation, la ligne responsable de l'erreur est affichée dans le cadre de texte en bas de l'écran.
De même, en cas d'erreur lors de l'exécution.
Enfin, la ligne en cours d'exécution ainsi que son numéro dans les code est affiché dans ce même cadre lors de l'exécution.

Exemple de niveau :

La consigne est : "Pour chaque 3 entrées, les additionner et placer le résultat en sortie".
Si l'entrée est : {1, 2, 3, 4, 5, 6}
Alors la sortie doit être : {6, 15}

Un code répondant à cette consigne est :

```
ENTREE
SAUVEGARDE 0
ENTREE
ADDITION 0
SAUVEGARDE 0
ENTREE
ADDITION 0
SORTIE
SAUT -8
```

Utilisation

Les bibliothèques utilisées sont SFML 2.5.1 et TGUI 0.8.
Nous avons fait le choix d'utiliser abondamment la librairie TGUI car elle proposait une utilisation similaire aux librairies que l'on a eu l'occasion d'utiliser en Java l'an dernier, basé sur des widgets simple, notamment les TextBox que nous avons utilisé en priorité pour faciliter le code et la manipulation de l'interface.

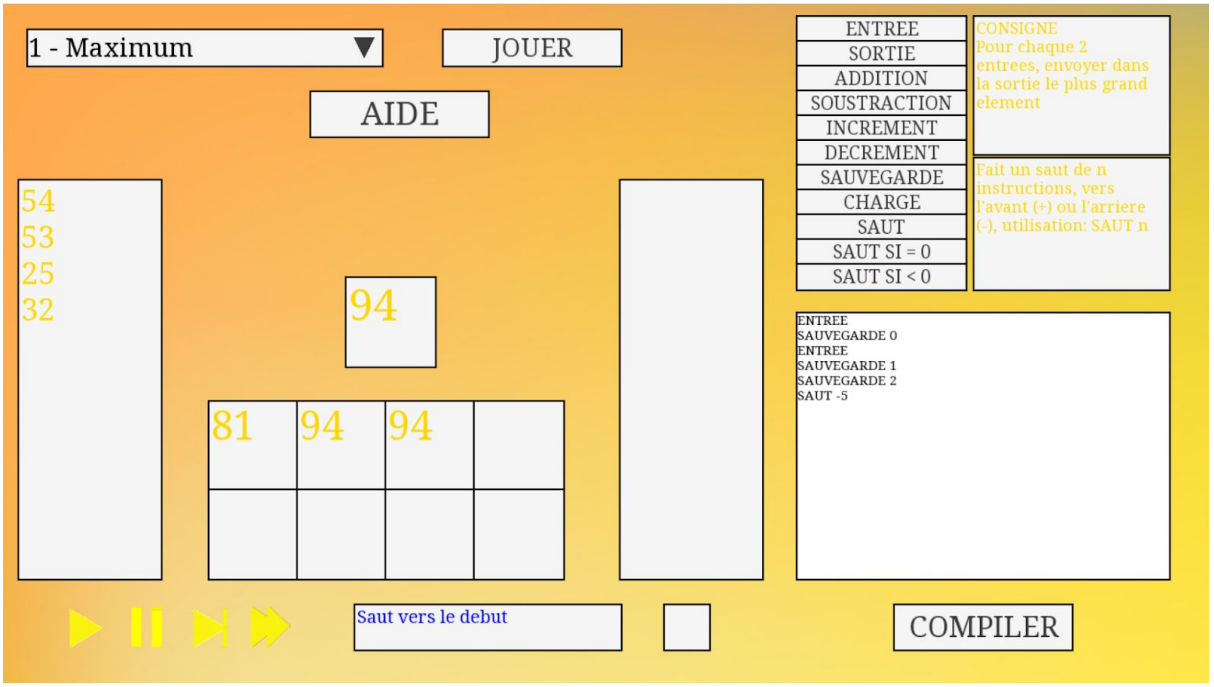
La librairies TGUi peut-être installé sur la distribution Ubuntu avec les commandes suivantes:

sudo add-apt-repository ppa:texus/tgui-0.8

sudo apt-get update

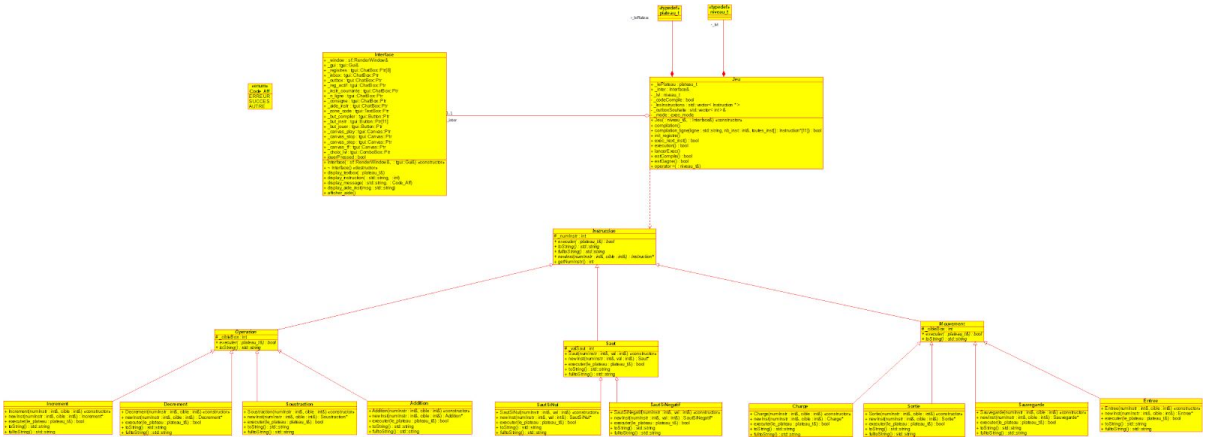
sudo apt-get install libtgui-dev

Pour lancer le programme, il suffit d'utiliser le makefile (commande make) qui s'occupera de compiler le code puis de lancer le jeu.



Une consigne est présentée en haut à droite de l'écran. Il faut remplir la case en bas à droite avec du code pour répondre à cette consigne. Il est possible de cliquer sur chacune des instructions rappelées en haut à droite afin d'avoir un rappel de l'utilité et de l'utilisation d'une instruction.
Le code écrit, il suffit de la compiler en cliquant sur le bouton associé. Si la syntaxe est bonne, il vous sera indiqué que la compilation a été réussi, sinon la ligne de code posant problème sera indiqué en rouge.
Une fois le code compilé, il sera nécessaire de cliquer sur une icône en bas à gauche correspondant à un mode de fonctionnement. Le code sera alors exécuté à la vitesse correspondante.
L'utilisateur peut à tout moment arrêter l'exécution pour changer son code, il suffira juste de recompiler et de relancer l'exécution.

Description du code



L'utilisation du langage orienté objet concerne principalement l'implémentation des instructions. En effet, toutes les instructions contiennent un numéro d'instruction ainsi qu'une commande d'exécution. Cependant, seules certaines instructions ont besoin d'un paramètre (comme Saut n).
Le choix d'implémentation a été fait avec l'optique de la lisibilité. Les différentes instructions ont été séparées en 3 familles :

- Mouvement [de données] : un déplacement d'une donnée depuis une case vers une autre.
- Saut : Un saut dans l'exécution du code qui peut être conditionnel.
- Opération : Un calcul qui dépend de la case active et parfois aussi d'une autre donnée.

A chaque instruction correspond un objet. Lors de la compilation du code de l'utilisateur, un tableau (vecteur) d'instructions est construit correspondant au code. Les instructions de ce tableau sont ensuite exécutés les unes après les autres (sauf en cas de saut) selon les commandes de l'utilisateur.

Tout en haut de la hiérarchie se trouve le Jeu. Il contient à la fois le tableau d'instructions, mais aussi le plateau. Le plateau est une structure contenant toutes les données de la partie: l'entrée, la sortie, la case active et les cases mémoire.
C'est l'instance de l'objet Jeu qui s'occupe de la gestion de la compilation, mais aussi de l'exécution.

Concernant les contraintes, le choix du jeu nous a beaucoup aidé puisque les 8 classes, 3 niveaux de hiérarchies et les conteneurs différents ont tout de suite paru évident, en particulier l'utilisation de deque. Les surcharges d'opérateurs ont cependant posées problèmes. La surcharge du = permettant à la classe Jeu de changer de niveau est une idée pertinente mais venu tard, et nous n'avons pas trouvé d'idées pour la seconde surcharge. En effet, la surcharge de l'opérateur << de la classe Instruction prévu initialement n'a pas pu être implémentée.

Enfin, la compilation et l'exécution sur code sont les parties dont nous sommes le plus fier. La compilation a nécessité beaucoup de tests pour déceler tous les cas particuliers pouvant la faire échouer et la construction du container d'Instruction a été laborieux et nous a permis d'avoir une compréhension plus net des subtilitées des relations de classes induits par la POO. L'exécution quant-à elle a nécessité beaucoup de réflexions sur comment la mettre en oeuvre, avec l'intégration des différentes vitesses d'exécution. Grâce à cela la mise en oeuvre a été plutôt rapide et nous sommes très content du rendu obtenu.
Au final, l'application est plus réactive et aboutie que nous ne l'espérons initialement, nous sommes très fier du rendu final et d'à quel point nous avons été mesure de nous rapprocher des concepts du jeu pris en exemple.