ÉCOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE ET DE MATHÉMATIQUES APPLIQUÉES DE GRENOBLE

BoiteMaker

Robin VINCENT-DELEUZE & Floran NARENJI-SHESHKALANI

15 novembre 2015

1 INTRODUCTION

Ce projet est réalisé dans le cadre du cours d'Ada suivi à l'Ensimag en première année. Cette application génère, selon les entrées saisies par l'utilisateur, un patron de conception pour une boîte. Cette boîte se veut personnalisable à travers une série de paramètre standards (dimensions, couleurs...).

La lecture du fichier **README.md**, située à la racine du repository est avisée. Dans le dossier **doc** se trouvent deux exemples ayant servi à expliquer la généricité et les pointeurs de fonction.

2 FONCTIONNALITÉS SUPPLÉMENTAIRES

Nous permettons à l'utilisateur d'afficher les informations de deboggage (`--show-debug`), d'enregistrer les informations de deboggage dans un fichier (`--log FILE`), de choisir la couleur de la bordure (`--border COLOR`), du remplissage (`--fill COLOR`) et de choisir le motif de remplissage (`--pattern FILE`).

⚠ La fonction `--pattern` requiert la présence d'**ImageMagick**.

3 CHOIX TECHNIQUES

3.1 GESTION DE VERSION

Nous utilisons le système de gestion de version **git** sur la plateforme **github** afin de maintenir l'historique de nos modifications et de synchroniser le code entre les différents lieux et

membres de l'équipe.

3.2 Système de build

Idéalement, nous souhaitions mettre notre code source dans un dossier, les fichiers de compilation dans un autre et les binaires dans un dernier. Réaliser cela avec gnatmake s'est avéré fastidieux : de nombreux paramètres se succèdent avec peu de maintenabilité, et le plus simple pour standardiser le processus est un fichier bash.

Nous choisissons d'employer le système de build **gprbuild**.**gprbuild** permet de décrire l'ensemble des comportements que l'on souhaite appliquer dans un unique fichier (ici *BoiteMaker.gpr*) qui est standardisé, claire et écrit dans un dérivé proche d'Ada.

Nous spécifions aussi à travers **gprbuild** l'ensemble des paramètres de compilations supplémentaires souhaités (notamment l'affichage de tout warning).

3.3 Responsabilité unique

Nous choisissons d'appliquer le principe de responsabilité unique. Ainsi, nous distinguons différents packages qui sont eux même séparés en deux dossiers :

src Fonctionnalités métiers liées directement à la boîte
box_info, box_parts, halfbox, halfbox_panel, svg

src/utilities Fonctionnalités annexes
commandline_args, generic_linked_list, imagemagick, logger, help, point, text_file, petit_poucet

Une autre règle que nous choisissons d'appliquer est de ne pas avoir d'objet partiellement initialisé : un object est soit non-initialisé, soit entièrement et complètement initialisé. Du point de vue du programmeur, les packages sont "atomiques" dans le sens où ils acceptent un jeu d'arguments et renvoient à partir de celui-ci un objet complet.

3.4 Décomposition en arbre

Le processus de génération de la boîte peut se représenter sous la forme d'un ensemble de relation père-fils.

Le premier enregistrement, **box_info_t**, est hydraté à l'aide des entrées de la ligne de commande. Ensuite, celui-ci est passé au package **box_parts**, qui extrait les informations nécessaires à la génération de chacune des demi-boîtes et les transmet au package **halfbox**, qui réalise la même opération pour chacune des faces de la demi-boîte avec le package **halfbox_panel**.

Conceptuellement, on observe d'abord la boîte, et on la divise progressivement en entités de plus en plus simples jusqu'à ce que l'on atteigne une entité d'une simplicité permettant son calcul.

3.5 Liste générique, point & petit poucet

On choisit de représenter le polygone sous la forme d'une liste simplement chaînée. Cette liste est générique, ce qui permet sa réutilisation ultérieure pour tout type. Ici, elle est utilisée avec l'enregistrement **point_t**, un ensemble de coordonées `(x, y)`.

Le petit poucet est un aide. Son objectif est de simplifier la programmation de la logique de dessin de chaque patron. On dirige le petit poucet sur chacun des axes à l'aide de fonctions permettant de réaliser des mouvements (gauche, droite, haut, bas) et celui-ci enregistre dans une liste chainée de point sa position à chaque instant. Une fois le petit poucet revenu en position initiale (définie à sa création), le dessin du patron est fini.

3.6 SVG

L'unique fonction du package svg accepte en entrée un **box_parts_t**. Elle permet de séléctionner une couleur de bordur, de remplissage et un motif de remplissage. Elle renvoie une chaîne de caractère contenant la représentation en SVG de la boîte, en procédant conceptuellement comme pour le processus de construction, c'est à dire de haut (boîte) en bas (face).

3.7 ImageMagick

Pour le remplissage de la boîte par un motif, le logiciel ImageMagick est utilisé. Celui-ci permet de récupérer l'image encodée en base64 et d'obtenir ses dimensions, trois données nécessaires pour embarquer une image dans un fichier SVG. ImageMagick est appelé directement en tant que processus. Cela est moins fiable qu'une bibliothèque liée à la compilation mais a pour avantage d'éviter l'intégration au projet d'un wrapper autour d'une API native d'ImageMagick, couteuse à compiler et difficile à intégrer. On sacrifie donc la résolution de la dépendance à la compilation pour une résolution à l'exécution, ce qui rend la bonne exécution du programme dépendante de la capacité de l'utilisateur à fournir une installation d'ImageMagick fonctionnelle.

3.8 Logging

Le logging, présent à travers tout le code, nous permet de tracer l'execution détaillée du programme et d'enregistrer sa trace dans un fichier ou sur la console. Les logs sont d'une grande utilité à la résolution des problèmes : ils permettent d'accéder à une vaste quantité d'information sans cependant avoir à utiliser un outil tel que gdb, peu évident pour des débutants.