Editeur de niveaux
Prog & Play

Benjamin BONTEMPS

25 mars 2016

Résumé

Ce document a pour objectifs de récapituler l'essentiel du travail réalisé dans le cadre du projet de conception d'un éditeur de niveaux pour le jeu sérieux Prog&Play et de permettre sa maintenabilité en cas de nécessité. Il suppose acquis les concepts de base liés à Prog&Play, à Spring (moteur sur lequel est basé le jeu) et au développement en Lua.

Table des matières

1 Architecture

Cette section présente les choix effectués quant à l'architecture adoptée pour la réalisation de l'éditeur de niveux.

Ce dernier est composé de :
- Un widget central permettant d'afficher les éléments de l'interface utilisateur et de capter les interactions avec cette dernière ou avec le moteur pour effectuer des actions en conséquence. (editor_user_interface.lua)
- Un gadget permettant l'utilisation de code synchronisé [1]. (editor_gadget.lua)
- Un fichier contenant toutes les chaînes de caractères à afficher. (EditorStrings.lua)
- Un fichier définissant une classe [2] machine à états. (StateMachine.lua)
- Un fichier contenant des fonctions utilitaires. (Misc.lua)
- Des fichiers de description de tables Lua. (Actions.lua, Conditions.lua, TextColors.lua, Filters.lua)

2 Description détaillée des modules

2.1 Widget

Le widget est l'élément central de l'éditeur des niveaux autour duquel s'articulent les autres modules. Le choix d'un widget en tant qu'élément central se justifie par le fait que la majorité des interactions se font de façon asynchrone. L'unicité du widget vient du fait qu'il n'est possible de n'avoir des variables communes à plusieurs widgets qu'en passant par la table WG, ce qui aurait été gênant étant donné le très grand nombre de variables devant être utilisées à de nombreux endroits distincts.

2.1.1 Framework Chili UI

Afin de simplifier le développement de l'interface graphique, j'ai choisi d'utiliser un framework d'interface graphique développé spécifiquement pour Spring dénommé Chili UI.

Ce framework a l'avantage de ne pas avoir à utiliser l'API OpenGL de Spring et les différents listeners pour afficher et permettre l'interaction avec des fenêtres, boutons etc. Tout y est déjà implémenté, ce qui a permis de réduire considérablement le temps requis pour obtenir les premiers résultats. Le problème majeur est que le framework est extrêmement mal documenté, et la meilleure façon de comprendre son fonctionnement pour s'en servir correctement voire même le modifier pour l'adapter à des besoins particuliers est de se référer au code source présent sur github [3].

Les modifications que j'y ai personnellement apporté concernent les boutons, qui possèdent un état supplémentaire *chosen* permettant d'avoir un feedback visuel lorsqu'on les sélectionne (les modifications de cet état sont gérées "à la main", mais le feedback visuel est géré automatiquement par le framework).

1. Par exemple, la création d'une unité sur le terrain.
2. Au sens de Lua.
3. https ://github.com/jk3064/chiliui/

Le widget comporte de nombreuses fonctions reprenant les fonctions *New* du framework en n'utilisant qu'un certain nombre de paramètres, ce qui permet une définition plus compacte de chaque élément de l'interface. Le point négatif est qu'il faut soit connaître par cœur l'ordre dans lequel les paramètres doivent être renseignés dans le prototype de la fonction soit se référer en permanence à la définition des fonctions. Ces fonctions sont définies dans la catégorie *Chili UI functions*.

Il faut faire extrêmement attention à une chose en particulier en ce qui concerne le framework : il ne faut surtout pas modifier les éléments d'interface qui ne sont pas affichés à l'écran (c'est-à-dire les éléments d'interface qui n'ont ni Screen0, ni un descendant de Screen0 en tant que parent). Ceci ne produit pas d'erreur lors de l'exécution, mais les éléments d'interface concernés ne s'actualiseront plus correctement, ce qui est très gênant d'un point de vue utilisateur (les opérations seront effectuées, mais l'utilisateur n'aura aucun feedback visuel).

2.1.2 Initialisation de l'interface

L'initialisation de la quasi-intégralité de l'interface se fait par les fonctions d'initialisation appelées dans la méthode *widget :Initialize()*, et décrites dans la catégorie *Initialisation functions*. La grande majorité des fenêtres y est initialisée, en les affichant toutes d'un coup (pour les raisons évoquées précedemment), puis elles sont masquées pour n'afficher que la fenêtre correspondant à l'état actuel de la machine à états globale (voir 2.3).

Les fonctions des catégories *Top bar functions* et *Forces window buttons functions* s'occupent de changer l'état courant de la machine à états globale et de choisir quelles fenêtres doivent être affichées. Il s'agit pour la plupart de fonctions appelées lors de la pression sur des boutons.

2.2 Gadget

2.3 Machines à états

2.4 Fonctions utilitaires

2.5 Fichiers de description