Rapport du projet jeu

Par :
Steve Caya

Travail présenté à :
Louis Marchand

Dans le cadre du cours :
420-PRB-DM
PROGRAMMATION ORIENTÉE PAR OBJET II

Cégep de Drummondville
26 Mai 2016

Voici le jeu que j'ai programmé durant la session d'hiver 2016 dans le cadre du cours de programmation orientée objet II. Le jeu a été programmé en langage Eiffel et utilise la librairie Eiffel Game 2. Le jeu que je vous présente est le jeu MotoX est un jeu de plate-forme ayant comme but se conduire une moto munie d'un fusil et de se rendre jusqu'à la fin du niveau sans mourir. Dans ce document je vous présente un compte rendu du projet avec un guide d'installation, une description de l'évolution du projet, une autocritique du projet et un exemple de "design" objet. Le jeu MotoX est disponible à l'adresse suivante :

https://github.com/shipl0ad/MotoXeiffel

Guide d'installation du jeu :

Pour compiler le jeu MotoX sur votre ordinateur, il est essentiel d'avoir l'API EiffelStudio ou tout autre compilateur Eiffel. Pour ma part je vous conseille d'utiliser EiffelStudio, car il est très complet et relativement facile d'utilisation. Il est disponible à l'adresse suivante : https://www.eiffel.com/eiffelstudio/ . Ensuite il faut installer la librairie Eiffel Game 2 disponible à l'adresse suivante www.eiffelgame.2org et suivre l'installation suivante faite par Louis Marchand :

Installation de Eiffel Game 2 Sur Windows

Installer Eiffel Game 2 :

1. Télécharger Eiffel Game

2 : https://github.com/tioui/Eiffel_Game2/tree/windows_build 2. Copier le répertoire "game2" dans le sous-répertoire d'EiffelStudio contrib/library/

3. Copier les fichiers dll du fichier zip correspondant à votre version d'Eiffelstudio (DLL32.zip ou DLL64.zip) dans le répertoire de votre projet (le même répertoire que votre fichier .ecf).

Ceci était la méthode pour Windows, maintenant voici le guide d'installation de la librairie sur Linux.

Sur Linux

Installez les librairies C : (Chaque # signifie une ligne en bash)

sudo apt-get install git libopenal-dev libsndfile1-dev libgles2-mesa-dev
sudo apt-get install git libsdl2-dev libsdl2-gfx-dev libsdl2-image-dev libsdl2-ttf-dev
sudo apt-get install libepoxy-dev libgl1-mesa-dev libglu1-mesa-dev

Télécharger Eiffel Game 2 # git clone --recursive
https://github.com/tioui/Eiffel_Game2.git

Créer un lien entre le répertoire EiffelStudio et Eiffel_Game2 :

sudo ln -s `pwd`/Eiffel_Game2 /usr/lib/eiffelstudio-15.12/contrib/library/game2

Compiler la librairie :

cd Eiffel_Game2

./compile_c_library.sh

Une fois les installations terminées, il faut ouvrir le document MotoZX. Trouver dans le document le fichier suivant : motozx.ecf . Ensuite, il faut ouvrir ce fichier et le compiler.

Guide d'utilisation

Une fois le jeu lancé il est très simple d'y jouer. Dès l'ouverture du jeu, un menu s'offre à vous. Il est important de noter que la touche "Escape" peut être appuyée en tout temps pour quitter le jeu. Par contre, si vous quittez avant ou pendant que vous jouez votre pointage sera perdu. La touche "H" dans le menu vous affichera le meilleur pointage disponible sur le serveur. Ensuite la touche "Enter" débute le jeu. Pour contrôler la moto, il suffit d'utiliser les touches flèche gauche ou droite de votre clavier. Pour tirer avec le fusil il suffit d'appuyer sur la touche "espace", la touche ne peut pas être retenu enfoncer pour avoir l'effet d'un fusil automatique donc un coup à la fois. Le jeu se finit dès que vous touchez à l'objectif final sans vous être fait tuer et le score se détermine en fonction du plus rapide et le nombre d'ennemis tuer.

Évolution du projet

C’est la deuxième fois que je faisais ce cours puisque l’an dernier je n’avais tout simplement rien remis suite à des problèmes personnels, le jeu ne partait de rien. Je n’avais pas vraiment programmé d’autre jeu auparavant. Les premières semaines de la session pour ma part étaient très occupées. J’avais un projet dans le cadre du cours de DM2 qui se terminait à la sixième semaine. Donc j’ai donc fait mon diagramme de classe en début de session, mais suite au cours de DM2 je savais que le premier diagramme n’est jamais le même lorsque le jeu est fini parce que lors de l’analyse tu vois le programme au plus simple et quelques complexité font que tu dois modifier le diagramme de classe.

Par la suite, il fallait donc me familiariser avec le langage Eiffel et la librairie Eiffel Game 2. Les exemples fournis par Louis Marchand sont très simples à la compréhension du langage et de sa librairie. Donc une fois la familiarisation faite, il faut débuter la programmation. J’ai donc pris les exemples à Louis et j’ai adapté un peu à mes besoins. Par la suite, j’ai donc utilisé GIMP pour faire mes dessins. La moto a été produite entièrement par moi. À la base j’étais supposé géré des rotations de la moto, mais je me suis rendu compte que c’était un peu complexe et inutile à la fonctionnalité du jeu. La programmation allait bien avec quelques petits “bugs” mais sans plus ça avançait très bien.

Ensuite j'ai eu une embûche et cela va me permettre de grandir de cette expérience. Pendant environ deux semaines j'ai continué la programmation sur mon ordinateur portable sans envoyer des mises à jour sur le GIT. Malheureusement il est arrivé un accident avec mon ordinateur portable et le disque dur et le reste de l'ordinateur a été brisé. Donc j'ai perdu deux semaines de travail, par contre cela m'a appris à toujours utiliser GIT à chaque changement que l'on fait pour ne rien perdre. Donc j'ai dû reprendre la programmation du jeu et couper quelques fonctionnalités que je voulais faire.

Auto critique

Personnellement le projet jeu du cours de programmation orienté objet II ma fait découvrir que la programmation de jeu vidéo n'était pas faite pour moi. La création du jeu a souvent été remise en question lors de sa programmation. Ce que je veux dire par là, c'est que pendant la programmation j'avais l'impression que je devrais faire telle fonction d'une autre manière ou bien de changer telle chose à place de telle chose. Je me remets toujours en question dans les choix du jeu. Ce qui ralentit énormément la progression du jeu. L'analyse du programme a été faite rapidement environ deux semaines et c'est sans doute pourquoi je remettais mes décisions de programmation. Ensuite, l'an passé je ne comprenais pas vraiment la programmation orientée objet et cette année je crois me débrouiller déjà pas mal plus que la dernière fois que j'ai fait ce cours. Je ne continuerai pas la programmation de ce jeu dans un avenir rapproché par

contre je vais continuer dans la programmation orientée objet, car cela semble très performant.

Exemple de “design” objet

Dans cette section je vais vous démontrer la méthode redéfinie. Pour cela je vais prendre la classe “SOUND” qui est une classe abstraite. Cette classe sert seulement pour gérer les sons. Les classes héritant de cette classe sont “NIVEAU” et “BULLET” qui joue chacun leur son différent. Donc c’est pourquoi dans la classe “SOUND” j’ai mis une fonction qui s’appelle “jouer_son” qui est vide. Dans la classe “NIVEAU” le son qui est ajouté est une chanson qui est mise en une boucle infinie et qui est seulement lancée automatiquement avec le jeu. Pour ce qui est de la classe “BULLET”, la fonction “jouer_son” à un autre sens elle sert donc seulement à faire le son du fusil qui tire quand l'utilisateur appuie sur la touche “espace”. Donc je modifie dans deux classes enfants la fonction qui est dans la classe parent. C’est donc un exemple de “design objet” qui est appelé la méthode redéfinie.