Projet jeu POO 2

Travail fait par

Jérémie Daem

Groupe : 01

Travail remis à

Louis Marchand

Programmation orientée par objet 2

420-PRB-DM

Rapport final du projet

Adresse dépôt Git: https://github.com/MercPlente/tp_poo.git
Nom d'utilisateur Git: Deweam

Guide d'installation sur Windows :

1.Installer EiffelStudio :

1.1Télécharger EiffelStudio à l'adresse:
http://sourceforge.net/projects/eiffelstudio/files/EiffelStudio%2015.12/Build_98308/
1.2 Installer EiffelStudio

2.Installer Eiffel Game 2 :

2.1 Télécharger Eiffel Game 2 :
https://github.com/tioui/Eiffel_Game2/tree/windows_build
2.2 Copier le répertoire "game2" dans le sous-répertoire d'EiffelStudio contrib/library/
2.3 Copier les fichiers dll du fichier zip correspondant à votre version d'Eiffelstudio (DLL32.zip ou DLL64.zip) dans le répertoire dans le répertoire ou se trouve le .ecf

3. Utiliser l'application:

3.1 Démarrer le serveur (serveur_tp_poo.ecf dans le répertoire serveur_tp_poo)
3.2 Démarrer le jeu (tp.ecf dans le répertoire tp_poo)
3.3 Profiter !

Guide d'installation sur Linux Ubuntu :

1. Installez EiffelStudio

```
# sudo add-apt-repository ppa:eiffelstudio-team/ppa
# sudo apt-get update
# sudo apt-get install eiffelstudio
```

2. Installez les librairies C :

```
# 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
```

3. Télécharger Eiffel Game 2

```
# git clone --recursive https://github.com/tioui/Eiffel_Game2.git
```

4. Créer un lien entre le répertoire EiffelStudio et Eiffel_Game2 :

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

5. Compiler la librairie :

```
# cd Eiffel_Game2
# ./compile_c_library.sh
```

6. Pour commencer à jouer:

Veuillez démarrer le serveur (serveur_tp_poo.ecf dans le répertoire serveur_tp_poo) et par la suite démarrer le jeu (tp.ecf dans le répertoire tp_poo).

Évolution du projet

Semaine 1: Présentation du projet

Semaine 2: Création de quelques sons tests, d'images et d'événements. On apprend Eiffel.

Semaine 3 & 4: Création des premières classes (Application, game_engine, image, sound). On s'inspire des exemples à Louis (surtout de Maryo en game surfaced). Création d'images avec GIMP et musiques venant du jeu "Diablo 1".

Semaine 5: Nouvelles classes de menu, nouvelles images.

Semaine 6: Ajout de classes vides et de commentaires en se fiant au diagramme UML. Ajout de préconditions, postconditions et documentation.

Semaine 7: Aucune modification de ma part, Marc a cependant retravaillé les menus pour utiliser un "design" objet.

Semaine 8: Marc continue son travaille sur les menus et crée une classe abstraite de menus pour revenir dans les menus précédents. Ajout de documentation et de la fonction quitter.

Semaine 9: Création d'un thread simple pour comprendre son fonctionnement ainsi que des modifications d'images.

Semaine 10: On refait le game_engine au complet. Une grosse partie du code s'en va dans des classes, dont entity. Ajout de tests pour les mouvements des entités On fait bouger le joueur sur l'écran.

Semaine 11 & 12: On tente de mettre une caméra sur le joueur, mais en vain. On continue d'explorer eiffel_game2.

Semaine 13: Création d'un serveur qui nous permettra de sauvegarder des données avec des connaissances de réseau. On intègre le serveur dans le thread.

Semaine 14: Documentation améliorée et le joueur est maintenant centré dans l'écran. Ajout d'une classe "ennemy" qui gère les ennemis d'une liste liée.

Semaine 15: Grosse semaine. Ajout de collisions entre les ennemis, les objets, le joueur et deckard_cain (un npc qui redonne le plein de vie en dans le village). Améliorations des mouvements du joueur et de la caméra centrée. Le joueur peut maintenant attaquer et se faire attaquer. Le boss final a été ajouté ainsi que des leviers qui le font apparaître une fois tous activés. Changement côté réseau. On peut afficher le highscore au lieu de le "print". Beaucoup d'images ajoutés pour améliorer l'expérience de jeu.

Diagramme UML

Diagramme du jeu:

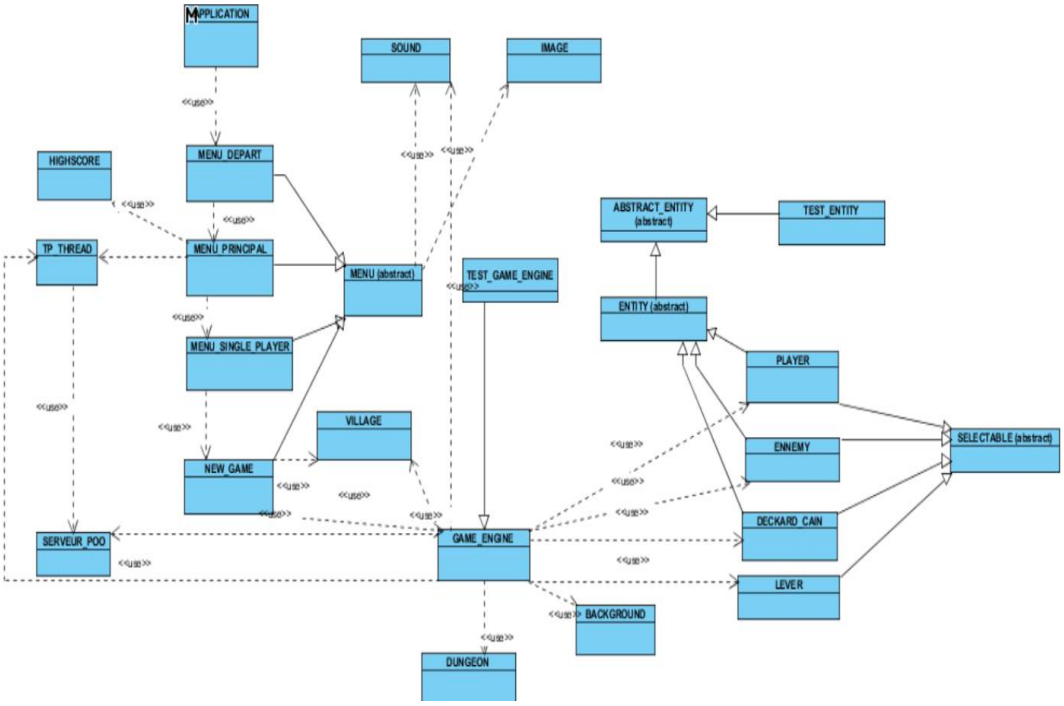

Diagramme du serveur:

Autocritique du projet:

Lorsque Marc et moi nous sommes mis en équipe, nous avons imaginé un genre de jeu qui serait dans le même genre que Diablo (vue de haut et le personnage qui bouge en cliquant avec la souris). J'avais une bonne idée des étapes à suivre pour réussir à créer un jeu dans le genre et j'ai tout de suite donné confiance à Marc. Plus les semaines avançaient, plus je perdais l'enthousiasme que j'avais envers le projet. Marc continuait a avancé le projet alors que je manquais des cours et que je portais peu d'attention au progrès du projet. J'aidais Marc du mieux que je pouvais lorsque j'étais en classe et je faisais les images, car Marc déteste faire des images. À un point assez tard dans la session, j'ai décidé de me reprendre et de recommencer à travailler dans le projet avec plus d'efforts. Les dernières semaines ont été les plus grosses semaines de programmation pour Marc et moi, car nous avions beaucoup de retard et nous voulions un bon résultat. Nous y sommes enfin arrivés. Si ce n'était pas de Marc, j'aurais peut-être laissé tombé le projet, même si c'est le projet qui m'intéressait le plus. Malgré le retard, nous avons un projet jouable et amusant. Je dois dire que nous n'avions pas beaucoup de connaissances en objet, car le premier cours était relativement facile à passer sans même faire d'objet, mais nous voyons maintenant la programmation objet sous un tout autre angle grâce à Eiffel et la merveilleuse librairie de Louis!

Les différents 'designs' objet

Constructeur: le constructeur de la classe 'deckard_cain' crée une nouvelle entité avec un x et un y, ainsi que plusieurs attributs incluant des surfaces tournées pour faire tourner l'entité lors de l'affichage.

Héritage simple: La classe 'deckard_cain' hérite des fonctions de 'entity' et, par le fait même, de 'abstact_entity' pour avoir accès aux fonctions et attributs de tous les entités.

Héritage multiple: Les ennemis de la classe 'ennemy' hérite de 'entity' et de 'selectable', car ils sont à la fois des entités et des objets sélectionnables. De cette façon, l'ennemi peut agir comme les autres entités et être sélectionné comme les autres objets sélectionnables.

Classe abstaite: La classe abstraite 'abstract_entity' sert à définir les attributs de bases de tous les entités comme par exemple: le x, le y, le hp, etc.

Méthode redéfinie : Dans les menus, la redéfinition a été utilisée pour différencier le menu_action de la classe 'menu' qui itère les menus et le menu_action des autres menus qui font une boucle pour gérer les événements dans le menu en cours.