Projet “There is no Planet B”

Auteurs : AMADY Naysel
HETTAL Mélissa
EISE-4 2021-2022

Encadrants : Mme Braunstein
Mr. Huber

Ce document est un compte rendu du projet « There is no planet B » dans le cadre du cours de « Langage objet C++ ».

Ce projet a pour but de mettre en application les notions abordées et acquises dans ce cours. Le projet doit comporter et/ou remplir les conditions suivantes :

- 8 classes ;
- 3 niveaux de hiérarchie ;
- 2 fonctions virtuelles différentes et utilisées à bon escient ;
- 2 surcharges d'opérateurs ;
- 2 conteneurs différents de la STL ;
- Diagramme de classe UML complet ;
- Commentaire du code ;
- Pas d'erreurs avec Valgrind ;
- Rendu par dépôt git, adresse à envoyer par mail avec dans le sujet le motif ` [EISE/MAIN 4 C++ Projet] ;
- Pas de méthodes/fonctions de plus de 30 lignes (hors commentaires, lignes vides et assert) ;
- Utilisation d'un Makefile avec une règle "all" et une règle "clean" ou autre outil de build automatique
- Utilisation de tests unitaires

Table des matières

1. Description de l'application développée :

Pour ce projet, nous avons décidé de modéliser un jeu, que l'on a nommé « No planet B ».

1.1. But du jeu :

« No planet B » est un jeu où le joueur a pour objectif principal d'obtenir un score de 80 points. Pour ce faire, il doit détruire et éviter d'être touché par certains objets, les « EnergiePolluantes ». Aussi, il doit aussi faire en sorte d'attraper le maximum d'objets « EnergieRenouvelable », car elles sont ses meilleurs alliés et vont lui permettre d'augmenter ses points de vie.
Lorsque le joueur est touché par une « EnergiePolluantes », il perd des points de vie. Le jeu s'arrête lorsqu'il n'a plus de point de vie ou lorsqu'il a atteint le score souhaité.

1.2. Présentation de l'implémentation

Lors de la modélisation du jeu, 12 classes ont été utilisées.

1.2.1. Eléments du jeu :

Le jeu se décompose en 10 classes permettant de représenter les différents éléments du jeu. (cf. diagramme UML en fin de compte rendu)

On peut retrouver deux types de classes qui ressortent de cette modélisation :

- La classe **Personnage**, classe mère et abstraite, regroupant tous les personnages du jeu qui peuvent se déplacer dans la fenêtre graphique.
- La classe **Panpan**, classe permettant de représenter les projectiles utilisés par le héros dans sa lutte contre les EnergiePolluantes. Lors de leurs collisions avec celles-ci, le score du joueur augmente de 5 points. Cependant, en cas de collision avec des EnergieRenouvelable, le joueur perds des points de vie.

La classe Personnage :
C'est une classe qui regroupe les personnages se déplaçant dans le jeu. En guise de personnages, nous avons :
- Le personnage principale instance de la classe Joueur : « player »
- Les méchants, instances des classes filles de la classe EnergiePolluantes : Nucléaire et Pétrole
- Les gentils, instances des classes filles de la classe EnergieRenouvelable : Solaire et Eolienne

Nucléaire et Pétrole ainsi que Solaire et Eolienne n'ont pas le même comportement, il a donc fallu effectuer un héritage pour distinguer ces différents types de méchants/gentils.

Les principales différences à observer entre ces deux types de méchants/gentils sont :
- La valeur accordée à leurs points d'attaques/points de soins
- Leur vitesse de déplacement dans la fenêtre graphique

La classe Panpan :
Elle correspond à la modélisation des projectiles lancés par le joueur pour exterminer les EnergiePolluantes.
Deux types de comportement sont à observer avec ses projectiles suivant leur cible :
- S'ils touchent une EnergiePolluantes alors elle est exterminée et le joueur gagne 5 points au niveau de son score
- S'ils touchent une EnergieRenouvelable alors elle est exterminée et le joueur perds 5 points de vie lorsque c'est une Eolienne et 25 points de vie dans le cas où c'est un Solaire.

1.2.2 Structure de la fenêtre graphique :

3 classes permettent quant à elle de composer la fenêtre graphique.
Il s'agit des classes :
- Jeu, gérant le fonctionnement et l'interface graphique du jeu. La bibliothèque graphique utilisée est la QT4
- Affichage, gérant l'affichage des informations liées au personnage (score et points de vie)
- Bouton, permettant de générer les boutons du menu ainsi que celui pour rejouer.

2. Utilisation des contraintes :

2.1. 3 niveaux de hiérarchies et 8 classes différentes :

L'organisation en 3 niveaux de hiérarchie, ainsi que l'implémentation de 8 classes minimum nous ont permis de différencier les classes EnergieRenouvelable et EnergiePolluante, et d'effectuer des héritages pour avoir les classes filles Nucléaire, Pétrole et Solaire et Eolienne. Cette contrainte nous a donc permis d'avoir plus de diversité concernant les différents acteurs de notre jeu.

2.2. Méthodes virtuelles :

Les méthodes virtuelles utilisées sont les suivantes :

- Dans Personnages la méthode afficher () étant appelé dans la surcharge d'opérateur « << ». Cette méthode selon si elle est appelée par la classe EnergieRenouvelable ou EnergiePolluante ne va pas contenir le même code et surtout pas « afficher » les mêmes informations
- Dans EnergieRenouvelable la méthode virtuelle soins () est importante car suivant l'EnergieRenouvelable (Eolienne ou Solaire) mise en paramètre dans cette fonction les points de vie reçues par le personnage seront différents
- Dans EnergiePolluante la méthode virtuelle degats() est importante car suivant l'EnergiePolluante (Nucléaire ou Pétrole) mise en paramètre dans cette fonction les dégâts subis par le joueur ne seront pas les mêmes

2.3. Conteneurs de la STL :

Concernant les conteneurs de la STL nous avons choisi d'utiliser des vectors et une Map, dans la classe Jeu.
Nous utilisons deux instances de la classe vector étant :

- _tabPollu , étant comme son nom l'indique un vector d'EnergiePolluante que l'on remplit dans la méthode creer_mechant()
- _tabRenouv , étant comme son nom l'indique un vector d'EnergieRenouvelable que l'on remplit dans la méthode creer_gentil()

Nous avons choisi d'utiliser des vectors pour ces « tableaux » car l'allocation de mémoire se fait de manière dynamique ; nous n'avons donc pas à nous soucier de la taille de notre tableau.

Concernant l'autre type de conteneurs de la STL, nous utilisons la map « _mapPollu ». Elle nous permet de classer les EnergiePolluante par catégorie, Nucleaire ou Pétrole, et d'avoir le nombre exact d'occurrence de chaque catégorie se trouvant dans le vector _tabPollu. La map est remplie dans la méthode play() à chaque vague de méchants créé.

Elle est de type <EnergiePolluante,int> avec int qui représente le nombre d'occurrences de l'EnergiePolluante à laquelle elle est associée.

2.4. Surcharges d'opérateur :

L'opérateur « << » a été surchargé pour permettre l'affichage des informations sur la console, ainsi que pour nous aider à débugger notre code :

- Il a été surchargé dans la classe EnergieRenouvelable pour permettre d'afficher les différents attributs de chaque Energie
- Il a été aussi surchargé dans la classe EnergiePolluante pour permettre la même chose

L'opérateur « < » a été surchargé dans la classe EnergiePolluante car il été indispensable pour l'implémentation de la fonction Maj_map() ; où la comparaison entre deux EnergiePolluante permet de les classer et de mettre à jour leur occurrence dans la map .

3. Fierté :

3.1. La classe Jeu, implémentation de la partie graphique :

La partie concernant l'implémentation du graphisme du jeu est une des parties dont nous sommes les plus fières. En effet, c'était une première pour nous de réaliser une interface graphique pour un jeu, ça représentait donc une certaine satisfaction de voir nos lignes de codes se transformer en une fenêtre graphique que l'on pouvait modifier à notre guise. C'est aussi une partie qui nous a pris beaucoup de temps pour plusieurs raisons.
D'une part, il nous a fallu nous documenter et comprendre le fonctionnement de Qt et de ses libraires pour pouvoir implémenter la majorité de nos idées.

D'autre part, il nous a fallu réadapter certaines parties du code pour pouvoir les gérer via l'interface graphique. Par exemple, les déplacements des joueurs et leur position sont maintenant géré grâce au librairie Qt, indispensable pour notre interface graphique, ce qui n'était pas le cas au début.

Pour toute ces raisons, la classe Jeu a été assez laborieuse à écrire, et nous n'avons pas pu satisfaire l'intégralité de nos envies concernant le graphisme, par manque de temps.
La classe Jeu se décompose en plusieurs parties :

- Celle qui gère le début de la partie : avec l'initialisation de la fenêtre graphique ainsi que l'affichage du menu
- Celle qui gère le déroulement du jeu : avec la création des méchants et des gentils via deux méthodes créer_mechants() et créer_gentil().
- Celle qui gère la fin de la partie en fonction des signaux émis. Cette dernière partie, nous a causé plusieurs soucis à la fin du projet, elle sera détaillée par la suite.

3.1.1 Gérer la fin de la partie :

Pour gérer la fin de la partie nous avons dû penser aux deux manières de « finir » le jeu :

- Soit le joueur gagne en atteignant le score souhaité.
- Soit il perd car il a perdu l'intégralité de ces points de vie.

Le joueur gagne : atteinte du score souhaité :

Pour implémenter cette version de la fin du jeu, une méthode winner () a été implémenté dans la classe Joueur .

```
void Joueur::winner()
{
    if(this->_score>= 80){ // si on atteint le score objectif alors on a gagné
        emit win();// émission du signal win()
    }

}
```

-Méthode winner()-

Lorsque le joueur atteint le score de 80 points on émets le signal win(), étant par la suite interprété par la méthode play() dans la classe Jeu.

Dans cette méthode on lie le signal win () au public slot gagner() qui part la suite va dispatcher le message apprenant la nouvelle au joueur .

```
QObject::connect(player,SIGNAL(win()),this, SLOT(gagner()));
```

-Connexion entre le slots gagner () et le signal win() dans la méthode play()-

Le joueur perd : l'intégralité de ces points de vies ont été perdu :

Grâce à la méthode maj (), se trouvant dans la classe Affichage, nous allons émettre un signal dead() lorsque les points de vie sont nulle. On émets le signal dead () puis dans la classe Jeu on effectue le même mécanisme qu'avec le signal win().

```
else if (grille->player->getpV()<(VIE_JOUEUR/4) && grille->player->getpV()>=0){
    setPlainText("Pv: " + QString::number(grille->player->getpV())+"\nScore : "+QString::number(grille->player->getScore()));
    setDefaultTextColor(Qt::red);
    setFont(QFont("times",16));
    if(grille->player->getpV()==0){
        emit dead();
    }
}
```

-Partie du code de maj () permettant l'émission du signal dead()-

```
QObject::connect(affichage,SIGNAL(dead()),this, SLOT(gameOver()) );
```

-Connexion entre le slots gameOver () et le signal dead()-

4. Procédure d'installation et d'exécution du code :

4.1. Installation générale et exécution du code :

Pour pouvoir faire tourner proprement les programmes Qt sur votre ordinateur, il vous faut ajouter des bibliothèques logicielles supplémentaire à l'aide de la commande suivante :

$ sudo apt-get install qt4-dev-tools libqt4-dev libqt4-core libqt4-gui

Afin de compiler le projet entrer la commande :

$ make all

Pour lancer le jeu entrer la commande :

$ ./no_planetB

4.2. Utilisation du jeu :

Une fois le jeu lancé, un menu apparaît ; 3 choix-vous sont proposés :

- Jouer, en cliquant dessus vous lancez le jeu
- Quitter, en cliquant dessus vous quittez le jeu
- Instructions, en cliquant dessus des instructions apparaissent avec les règles générales du jeu (exterminer les EnergiePolluante et épargner les EnergieRenouvelable)

Lors du jeu, le déplacement du joueur ne se fait qu'à l'aide des flèches droite et gauche de votre clavier.

5. Conclusion :

En guise de conclusion, ce projet a été très instructif et nous a permis de mettre à profit l'enseignement reçu lors des cours et TP de C++.

Il nous a aussi permis de gérer un petit projet de C++ et d'acquérir de nouvelle connaissance concernant les fenêtres graphiques en C++ en utilisant les bibliothèques fournies par Qt.

Cependant, ayant dû réaliser ce projet dans un délai imparti nous avons dû faire quelques concessions, nous avons cependant penser à certains points que nous aurions pu améliorer, ou que nous pourrons améliorer, tel que la mise à disposition de de plusieurs niveaux dans le jeu, ou la mise en place de plus de diversité concernant les personnages, autre que le joueur principal. Nous sommes tout de même satisfaites de proposer un jeu fonctionnel à l'image de nos nouvelles connaissances en programmation orientée objets.

6. Diagramme UML :

Diagramme UML de No Planet B

Amady Naysel Hettal Melissa | January 23, 2022