Mathématiques appliquées et informatique
MAIN4

Programmation Orientée Objet
Rapport Final
Projet There is no Planet B

24 janvier 2022

Auteurs :
Thomas Rives
Gabin Mazué

Encadré par :
Cécile Braunstein

Sommaire

Introduction et problématique

Dans le cadre de notre projet *There is no Planet B*(mouvement écologique du sauvetage de la planète et de ses océans) de MAIN4, il nous a été demandé de coder une application. Cette dernière pouvait être conçue de manière totalement libre avec pour seules contraintes la liaison avec le domaine de l'Ecologie mais également quelques demandes dans notre code :

- 8 classes
- 3 niveaux de hiérarchie
- 2 fonctions virtuelles
- 2 contenuers différents de la STL
- pas d'erreurs avec Valgrind
- pas de méthodes/fonctions avec plus de 30 lignes
- utilisation d'un Makefile et de tests unitaires

L'application que nous proposons est un jeu vidéo de combat style Arcade nommé *There is no Planet B Fighter*. Le jeu dans son fonctionnement se rapproche de références telles *Street Fighter* ou *Smash Bros* dans lequel des célébrités et personnages forts du monde de l'Ecologie peuvent s'affronter.

PARTIE 1

Description de l'application développée

1.1 Présentation du jeu

Comme dit précédemment, *There is no Planet B Fighter* est un jeu de combat opposant deux joueurs lors d'un affrontement. Chaque joueur a à sa disposition une palette de personnages allant de figures fortes de l'écologie comme Greta Thunberg, de farouches opposants comme Donald Trump, des symboles des changements climatiques avec notamment la bouteille d'eau (figure du gaspillage et de la pollution) ou encore un habitant de la planète de B ... Existerait-elle donc ?

Les combattants une fois choisis sont alors téléportés vers leur lieu de combat. Ce dernier est également choisi par les joueurs au préalable. Des paysages variés sont proposés, dans lesquels les joueurs pourront s'affronter dans un fond de décor post apocalyptique suite à des scénarios en tout genre de dérèglements climatiques : extrême chaleur dans un désert, froid glacial d'une ère glacière, affrontement aquatique suite à l'élévation du niveau de la mer, et pleins d'autres encore !

Comme pour les jeux de combat classiques, l'objectif de chaque joueur est de mettre K.O son adversaire à l'aide d'attaques en tout genre. Les combattants possèdent tout deux une jauge de vie (jauge de couleur verte et plus grande) ainsi qu'une jauge de mana (jauge de couleur bleue, petite et placée en dessous de la vie). Le mana est une energie qui permettra, en fonction de sa quantité, de réaliser divers coups et attaques par notre personnage. Chaque personnage commence avec 100 points de vie (le maximum) et 50 points de mana (la moitié). En effet, les combats se déroulent

principalement au corps à corps avec des séries de coups de poings mais chaque personnage possède en plus une attaque dite **spéciale** et une attaque **ultime** qui se réalisent à distance. Chaque joueur pourra alors infliger des dégâts de trois manières différentes :

- attaque corps à corps qui inflige 5 points de dégats et sans coût en mana
- attaque spéciale qui inflige 10 points de dégâts et coûtant 10 points de mana.
- attaque ultime qui inflige 40 points de dégâts et dont le coût en mana est de 100 (soit la totalité du mana du personnage).

Chaque attaque que l'on reçoit fait gagner en contre partie des points de mana. Dans le cas des attaques corps à corps, 10 points sont obtenus et si l'on subit une attaque ultime, on gagne 60 points de mana.

1.2 Méthode d'utilsation et Déroulement d'une partie

Le déroulement d'une partie ainsi que les commandes à adopter sont présentés ci-dessous.

Lors du lancement du jeu, un premier menu de présentation est affiché sur l'écran qui vous permettra de poursuivre dans le jeu **en appuyant sur la touche de la barre espace**.

Par la suite, le menu de sélection des personnages est à l'écran. Le joueur 1 et le joueur 2 ont alors le choix parmi tous les combattants et s'ils souhaitent jouer un même personnage cela reste tout à fait possible. Les commandes de navigation au travers des menus sont propres aux deux joueurs. Elles reprennent des touches de clavier classiques et souvent utlisées sur ordinateur pour les jeux vidéos. Ainsi nous avons :

Joueur	Curseur vers le haut	Curseur vers le bas	Curseur vers la droite
1	Appuyer sur **Z**	Appuyer sur **S**	Appuyer sur **D**
2	Appuyer sur ↑	Appuyer sur ↓	Appuyer sur →

Joueur	Curseur vers la gauche	Valider	Annuler
1	Appuyer sur **Q**	Appuyer sur **E**	Appuyer sur **R**
2	Appuyer sur ←	Appuyer sur **K**	Appuyer sur **L**

Une fois les personnages respectifs choisis, les joueurs vont alors décider du lieu de l'affrontement suite à l'affichage à l'écran du menu de sélection des cartes. C'est à l'aide des commandes du Joueur 1, identiques au menu précédent, que l'on réalise cette action. Il n'y a donc qu'un joueur qui choisit la "map" où le combat se déroule.

Dès lors le combat peut commencer et débute à la fin du compte à rebours lancé après les cinématiques de combat. Joueur 1 et Joueur 2 reprennent alors les précédentes commandes avec pour seules différences que ces dernières permettent ici les déplacements des personnages et leurs attaques. En effet nous avons les commandes suivantes pour chaque joueur :

Joueur	haut	bas	droite	gauche
1	Appuyer sur **Z**	Appuyer sur **S**	Appuyer sur **D**	Appuyer sur **Q**
2	Appuyer sur ↑	Appuyer sur ↓	Appuyer sur →	Appuyer sur ←

Joueur	Corps à Corps	Special	Ultime
1	Appuyer sur **E**	Appuyer sur **R**	Appuyer sur **T**
2	Appuyer sur **K**	Appuyer sur **L**	Appuyer sur **M**

Le combat s'achève seulement lors du K.O d'un des joueurs (la barre de vie atteint 0 points) mais il est possible de mettre la partie en cours en **pause en appuyant sur la touche Tab**, un menu s'affichera alors vous permettant en outre de recommencer le combat, de changer les personnages, l'environnement du combat voire de repartir au menu de présentation du jeu (si un joueur se sent en mauvaise posture bien évidemment).
Lors du combat, les informations du Joueur 1 sont situées en haut à gauche et celles du Joueur 2 en haut à droite. Durant le combat si la vie d'un joueur diminue à 50 points, celles-ci deviendra orange et si le seul des 20 points est atteint alors elle passera au rouge. Concernant la jauge de mana, si elle atteint sa valeur maximale de 100 points, elle sera entourée dès lors d'une bordure dorée et un son avertissant le joueur sera retentit.
Si l'un des joueurs finit sans vie, alors le combat s'achève comme annoncé précédemment et le vainqueur est affiché avec toute la gloire qu'il mérite. On offre alors la

possibilité aux joueurs de recommencer le combat, de changer de personnages ou de retourner au menu de présentation (menu de lancement du jeu).

PARTIE 2

Mise en valeur de l'utilisation des contraintes

Durant ce projet nous avons tenté de répondre au mieux aux contraintes imposées que nous avons au préalable citées en **Introduction**.

- Le nombre de classes supérieur à 8 :
  Notre code possède un total de 13 classes, observables avec l'**UML** (qui est présenté par la suite).
- 3 niveaux de hiérarchie :
  A mesure que nous avancions dans ce projet, nous avons réalisé l'importante utilité des hierarchies de classes nous facilitant grandement notre programmation. Nous souhaitions respecter au mieux le fondement de la programmation orienté objet. Ainsi, la classe **Attaque ultime** hérite de la classe **Attaque spéciale** qui elle même descend de la classe **Attaque**. Ou encore nous avons les classes des menus du jeu avec **Menu pause** héritant de **Menu Selection** héritant lui-même de **Menu**.
- 2 fonctions virtuelles :
  Possédant des grandes hiérarchies (comme vue ci-dessus) mais également des classes abstraites, les fonctions virtuelles ont été d'un usage fortement important. Prenons l'exemple de la fonction *gestion_ collisions* dont nous faisons appel afin de traiter les collisions entre **Personnages**, les collisions entre **Attaque** (qu'elles soient **speciale** ou **ultime**) mais aussi les collisions entre Personnage et Attaque. Nous pouvons citer également la fonction *show_ menu* définie pour chaque classe de **Menu**, affichant toutes les différentes classes issues de cette derniière.

- 2 conteneurs différents de la STL :
  Pour leur caractéristiques comme leur taille non fixée et la facilité d'obtenir une informartion grâce aux clés, nous avons utilisé principalement des **map** et des **vector**.
- méthodes et fonctions avec moins de 30 lignes et commentées :
  Nos fonctions font moins de 30 lignes, et nous avons fait en sorte de factoriser dès le début de ce projet toutes nos fonctions. Elles sont également commentées.
- Erreurs Valgrind :
  L'outil Valgrind nous a permis d'obtenir nos informations concernant nos éventuelles fuites de mémoire. Nous possédons certaines fuites (bien que celles-ci soient en quantité très inférieure comparée à la quantité totale allouée de mémoire).
- Makefile et tests unitaires :
  Le projet est comme demandé, accompagné d'un fichier Makefile ainsi que de tests unitaires.
- 2 surcharges d'opérateur :
  Dans le but de mettre à jour les valeurs des jauges de vie et mana de chacun des personnages, nous utilisons des opérateurs *jauge_ baisse* ainsi que *jauge_ augmente*.

```
==40975== HEAP SUMMARY:
==40975==     in use at exit: 319,023 bytes in 3,236 blocks
==40975==   total heap usage: 483,369 allocs, 480,133 frees, 8,728,098,760 bytes
 allocated
==40975==
==40975== LEAK SUMMARY:
==40975==    definitely lost: 17,176 bytes in 6 blocks
==40975==    indirectly lost: 16,224 bytes in 91 blocks
==40975==      possibly lost: 0 bytes in 0 blocks
==40975==    still reachable: 285,623 bytes in 3,139 blocks
==40975==         suppressed: 0 bytes in 0 blocks
==40975== Rerun with --leak-check=full to see details of leaked memory
==40975==
==40975== Use --track-origins=yes to see where uninitialised values come from
==40975== For lists of detected and suppressed errors, rerun with: -s
==40975== ERROR SUMMARY: 69204 errors from 2 contexts (suppressed: 2 from 2)
```

Figure 1 - Capture d'écran des résultats de Valgrind

PARTIE 3

Diagramme UML de l'application

Le diagramme se trouve dans le dossier en meilleure qualité et sous le nom de **UML.png**.

PARTIE 4

Installation et exécution du code

- Environnement MacOS :
  Une fois rendu dans le dossier *There_is_no_Planet_B_MacOS*, depuis le terminal, exécuter les commandes suivantes :

  ```
  make
  ./jeu
  ```

- Environnement Linux :
  Il est nécessaire au préalable de télecharger la bibilothèque graphique et audio de **SFML**. Ainsi il faut éxecuter les commandes suivantes afin de lancer le jeu depuis le dossier *There_is_no_Planet_B_Linux* , ou exécuter seulement les deux dernières si on a déjà en possession la bibilothèque SFML.Il est possible égalament de se rendre sur le site de la SFML afin de télécharger les librairies ou obtenir plus d'informations (https://www.sfml-dev.org)

  ```
  sudo apt-get install libsfml-dev
  make
  ./jeu
  ```

PARTIE 5

Fiertés

De manière générale, nous sommes plutôt satisfaits du rendu que nous proposons bien que déçus également car nous aurions voulu peaufiner au maximum notre jeu. Néanmoins, nous pensons présenter un contenu de bonne qualité et nous sommes fiers de détails présents dans le jeu :

- Nous avons respecter l'essence même de la programmation objet en créant des classes fidèles à la réalité et en respectant les inclusions et dépendances.
- toutes les images que nous utilisons sont dans un format 1920x1080 mais nous avons pu mettre à l'échelle toutes nos fenêtres et images en fonction de la résolution de chaque ordinateur qui lance le jeu. Il n'y aura donc jamais d'images déformées ou coupées.
- Chaque personnage possède ses propres bruitages et caratéristiques. Ainsi le bruit du saut, les bruits d'attaques, les vitesses de saut et vitesses de déplacement sont propres à chaque personnage.
- les montages photos et gif ont été fait ou retouché image par image afin d'obtenir le meilleur affichage possible.

PARTIE 6

Conclusion

Comme les parties précédentes le laissent présager, nous avons pris énormément de plaisir à mener ce projet. Le fait de réaliser un projet en étant quasi libre dans toutes nos décisions a été très plaisant et nous a permis de réaliser un projet qui était non seulement scolaire mais également très intéressant du point de vue personnel (nous avions déjà évoqué la possibilité d'une collaboration de coder un jeu vidéo dans un contexte hors scolaire). Par conséquent, nos connaissances du langage C++ ont été fortement consolidées et plus précisément nos connaissances en interfcae graphique avec SFML.
Nous restons néanmoins décus de ne pas avoir pu ajouter plus de personnages et peaufiner certains détails du jeu par manque de temps.