Université Pierre et Marie Curie

Polytech Paris UPMC
Electronique et Informatique Systèmes Embarqués

TOURE Souleymane

VU Kévin

EISE4

C++

PROJET

Elections piège à cons

Année universitaire 2016-2017

SOMMAIRE

I) Procédure d'installation

Pour ce projet nous avons utilisé une interface graphique : SFML. Nous avons choisi cette bibliothèque le site officiel de celle-ci possède de nombreux exemples pour la prendre facilement en main, de plus elle possède un forum anglais/français sur lequel il y a une grande communauté active ce qui facilite encore son utilisation en cas de problème.

Installation de la SFML : sudo apt-get install libsfml-dev

Pour compiler, il faut rajouter les options de compilation :

-lsfml-audio -lsfml-graphics -lsfml-window -lsfml-system

Le makefile génère un exécutable qui s'appelle : sysfml

II) Présentation du projet « Election piège à cons »

Le thème du projet étant cette citation de Sarthe, nous avons décidé de réaliser une caricature des élections américaines de 2016 sous forme de jeu. Le joueur contrôle le candidat à la maison blanche Donald Trump. Il peut le déplacer avec les flèches directionnelles dans un carré de 500x500 pixels. Le but étant de ramasser les bulletins qui apparaissent sur le sol pour atteindre un nombre requis de points (100 points) avant un temps imparti (50 secondes), afin d'accéder au dernier tour des présidentiels. Trois choix de difficultés sont proposés au joueur : FACILE, NORMAL, DIFFICILE ; et suivant ces niveaux de difficulté, nous allons faire apparaître toutes les cinq secondes (modes FACILE et NORMAL) ou toutes les dix secondes (mode DIFFICILE), un nombre prédéfini de faux bulletins qui seront des MALUS ou des BONUS. Les MALUS font perdre de la vie et de la vitesse au joueur et les BONUS font le contraire.

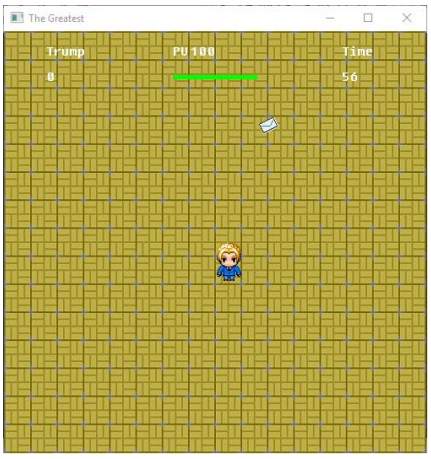

Figure 1 : Extrait du Jeu

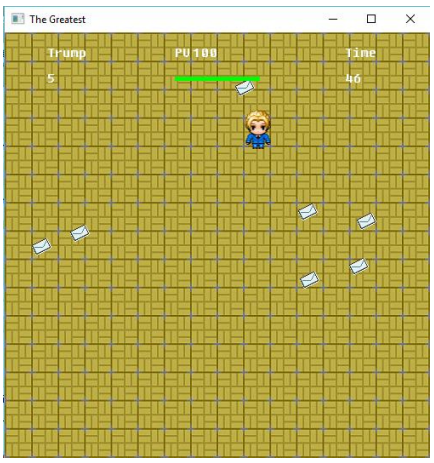

Figure 2: Apparition des bonus/malus

Ainsi il faut pour atteindre le second tour des élections, tenter d'arriver à 100 points tout en essayant de perdre le moins de vie possible.

- Si le joueur arrive à 100 points il devra affronter le candidat démocrate (ici Hillary Clinton)
- Sinon le jeu se termine en affichant un message de défaite récapitulant le nombre de bulletins, de bonus et de malus obtenus.

L'affrontement contre Hillary Clinton se déroule de la manière suivante : le joueur est enfermé entre des murs de bois et de bétons, et il doit échapper à Hillary Clinton qui le poursuit jusqu'à que le temps se soit écoulé. Pour détruire des murs en bois, le joueur devra appuyer sur ESPACE plusieurs fois (3 fois) tout en avançant vers le mur. Une collision du Sprite du joueur avec celui d'Hillary Clinton entraine un dégât de 30 points de vie.

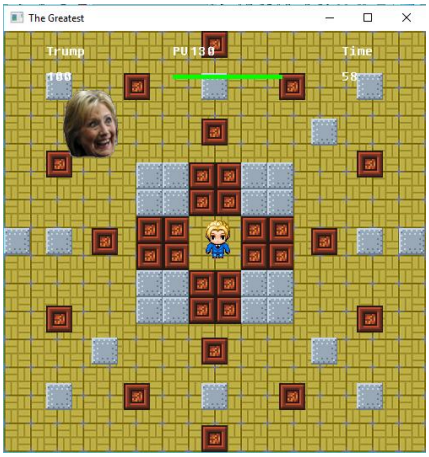

Figure 3: Affrontement du boss

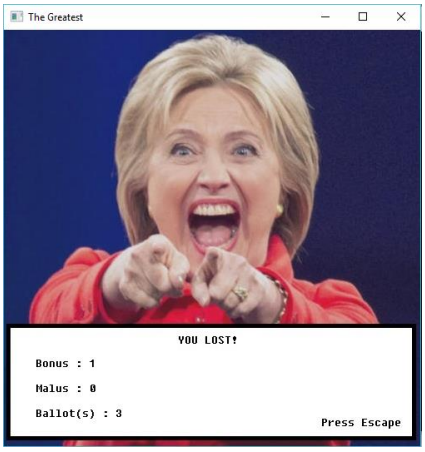

Figure 1: Défaite du joueur

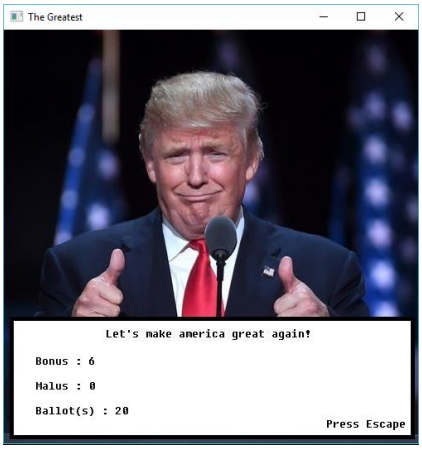

Figure 5 : Victoire du joueur

- Si le joueur reste en vie à la fin du temps il aura un message de victoire récapitulant le nombre de bulletins, de bonus et de malus obtenus.
- Sinon il aura le message de défaite.

Par l'absurdité du jeu, nous montrons que les élections sont des « pièges à cons ».

(Note : Dans un jeu il faut différencier que ce l'on voit (**les textures**) et ce que le jeu considère quand des tests de collisions sont faits (**les sprites**). Par exemple l'image du personnage de Donald Trump fait 48x48 pixels, cependant à l'écran les zones autour de l'image affichée de Trump sont transparentes mais bien présentes dans l'affichage. Donc quand deux objets entrent en collision, on peut avoir l'impression qu'ils ne se touchent pas alors que celle-ci a bien eu lieu, c'est ce que l'on appelle plus communément la **hitbox**)

III) UML

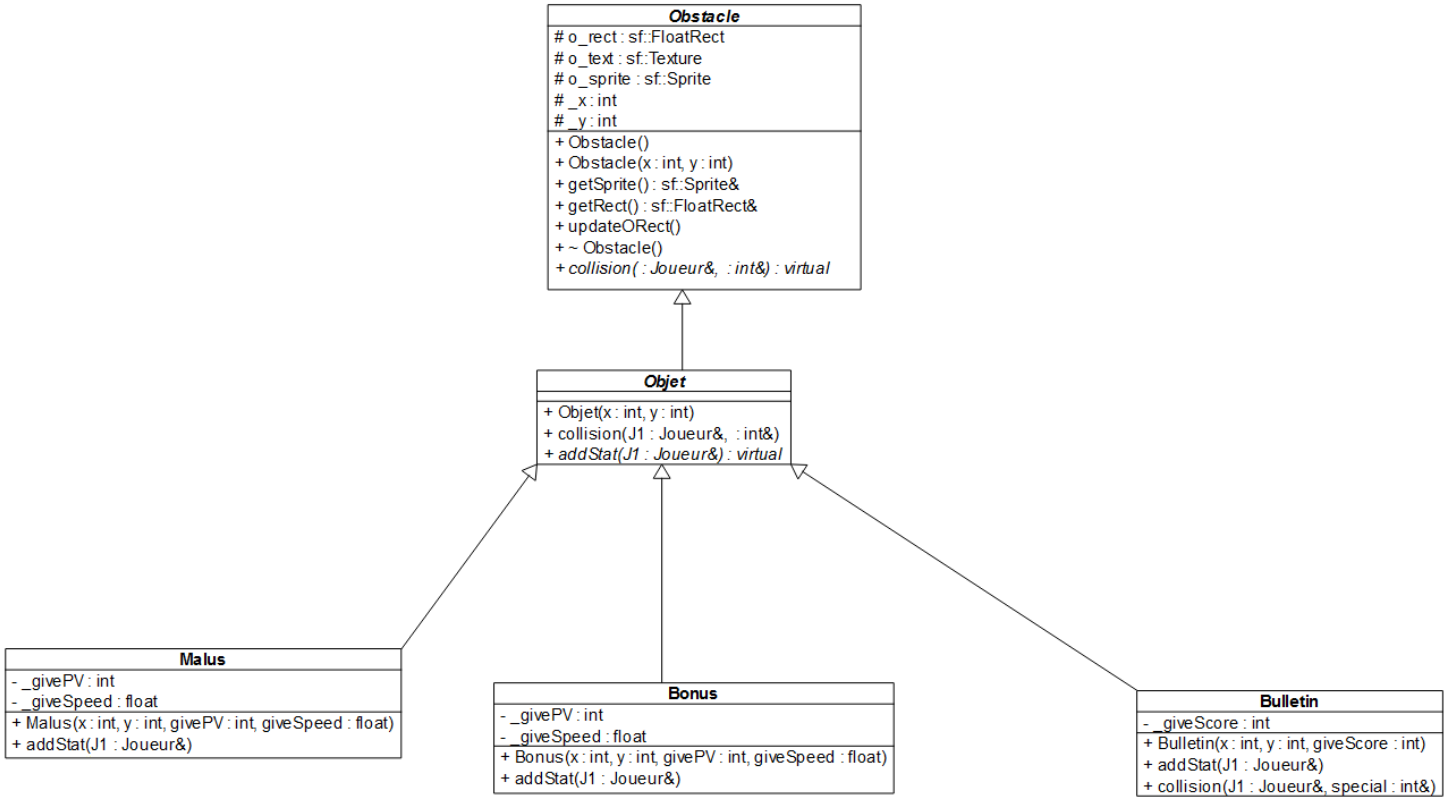

Figure 2: UML partie 1

Notre programme est tout d'abord composé d'objets. Ce sont les objets décrits dans la présentation ci-dessus qui permettent au joueur d'avancer avec la fonction virtuelle ***addStat***. Il y a les **bulletins**, les **bonus** et les **malus** qui héritent de la classe **objet**. La classe Objet hérite de la classe **Obstacle** puisque ces objets rentrent en collision avec le personnage d'où la nécessité d'effectuer cet héritage.

Nous avons aussi des **murs** (cf. page 4) qui sont fixes et à l'image de la classe objet, les murs dérivent de la classe Obstacle car ils rentrent en collision avec le joueur pendant la partie. Il y a aussi une classe **Bois** qui hérite des murs. Cependant, elle a une résistance ce qui permet de détruire les murs en bois pendant une partie.
Durant la présentation du programme au début du rapport, nous avons parlé du Boss du jeu Hillary, c'est aussi un obstacle.
Pour gérer tous les obstacles différents, nous avons mis en place une classe **ListObstacle** qui contient un vecteur d'Obstacle. Cela simplifiera la gestion des collisions et le placement des différents obstacles sur la carte.
Les classes **Joueur** et **Fenêtre** héritent d'aucune classe.

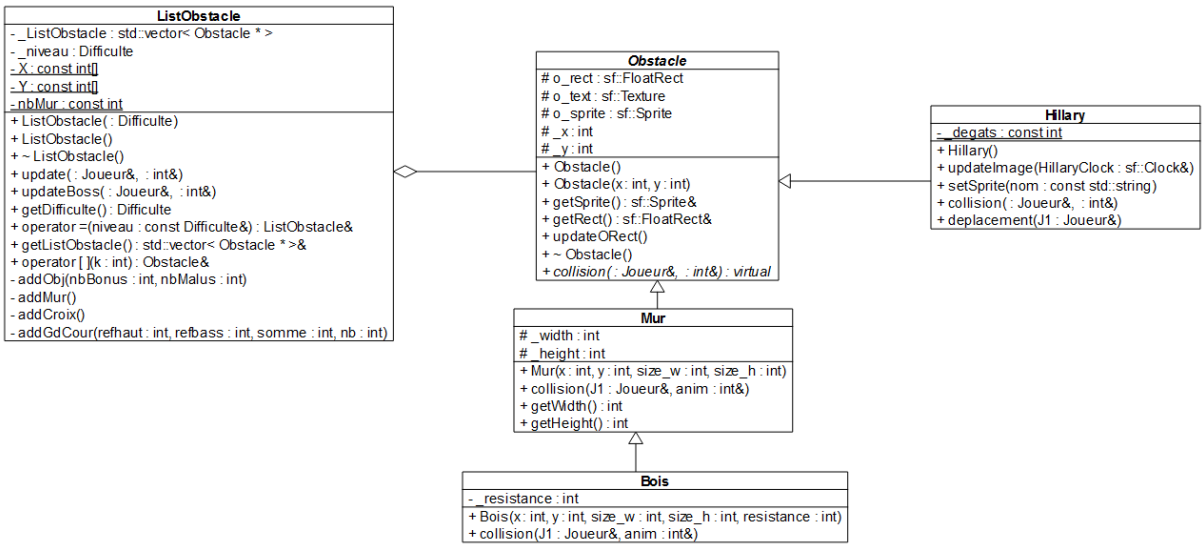

Figure 3: UML Partie 2

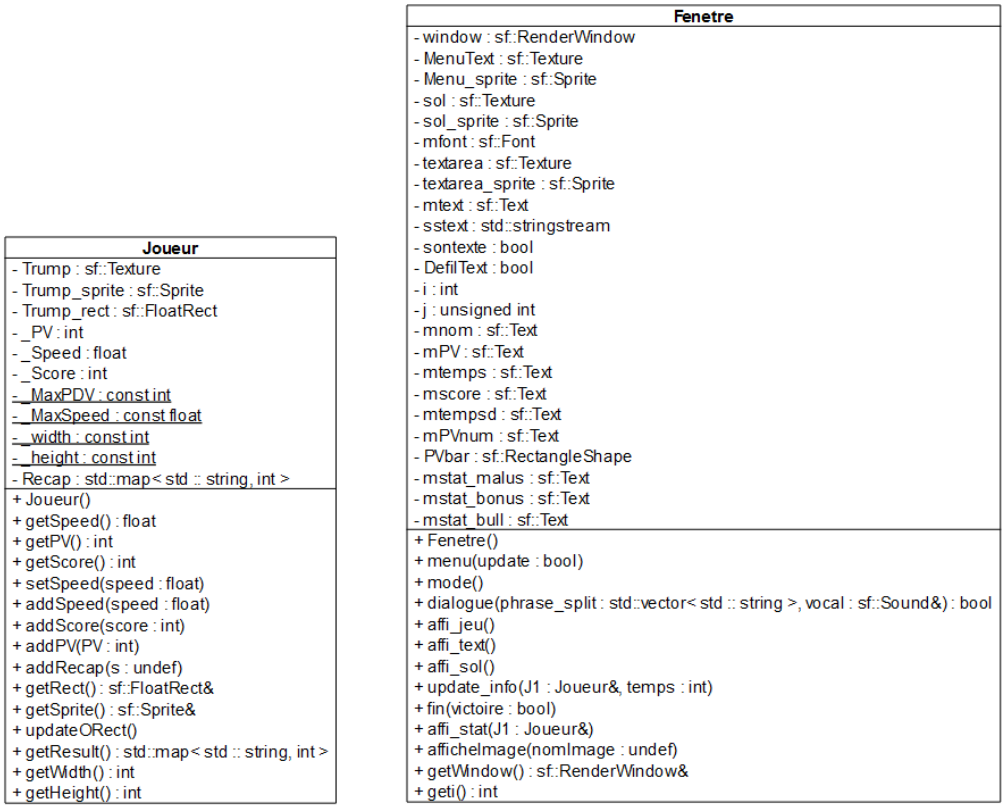

Figure 4: UML partie 3

IV) Description de l'application

A) Modélisation des données

a) Représentation des classes

Afin de réaliser le jeu, il nous fallait afficher une fenêtre, comme nous l'avons vu il s'agit celle-ci fait 500x500. Nous avons pour cela créé une classe éponyme ayant tous les champs correspondant aux différents affichages que nous allions faire. Par exemple, comme montré dans l'UML, la classe fenêtre contient des champs qui sont des textures, des Sprites, des polygones ou encore du texte. Par conséquent dès qu'une fenêtre est créée, son constructeur aura chargé toutes les textures, les aura appliquées sur les Sprites, puis il aura chargé les zones de texte avec leur position, leur taille, leur couleur et ce qu'ils contiennent.
Ensuite il ne suffira plus qu'à appeler les différentes fonctions d'affichage de la classe fenêtre pour montrer ce que l'on souhaite suivant les différents moments du jeu.

Comme montrés dans la partie III), beaucoup de classes héritent de la classe Obstacle. En effet tout ce qui est affiché à l'écran est en majorité du Sprite, sauf pour les zones de texte et le sol. Si Joueur n'hérite pas d'Obstacle, cela vient du fait qu'il ne possède pas la fonction ***collision***.

b) Avancement dans le jeu

La mise à jour des données dans l'affichage se fait dans le main à travers un ***while***. Tout d'abord avec la fonction ***afficheImage*** de la classe Fenêtre, le jeu affichera une fenêtre d'avertissement qui durera 5 secondes. La gestion du temps se fait à l'aide de la classe ***Clock*** de la SFML. Suite au panneau d'avertissement le jeu affichera un menu (toujours avec afficheImage) permettant au joueur de choisir son niveau de difficulté démarrera une musique d'introduction, pour cela nous avons utilisé les classes ***Keyboard*** et ***Music*** de la SFML.
Après le choix de la difficulté, le jeu va afficher le personnage, l'environnement et les zones de textes pour le score et l'explication du jeu. Les explications de jeu et autres dialogues font notamment appel à la fonction ***FenDial*** (dans le main) ainsi qu'à la classe **utility** utilisée dans le TP sur les réactions moléculaires, nous détaillerons cela dans la partie V). Il est important de souligner que l'affichage des éléments dans la fenêtre doit se faire dans un ordre précis pour éviter que des éléments viennent se superposer à d'autres éléments.

Exemple : *ordre d'affichage lorsque le joueur peut déplacer le personnage*
Sol -> Obstacle -> Joueur -> Informations sur le score, temps…
Dans ce cas-là, les informations seront toujours visibles, même si le joueur se déplace sur celles-ci.

La gestion des mises à jour des données, de la fin du temps ou encore du passage au Boss lors de la phase de jeu est réalisée dans la fonction ***FenUpInfo*** que nous détaillerons dans la partie B). Lorsque la phase de jeu est totalement terminée (victoire ou défaite : cf. II)), nous refaisons appel à la fonction ***FenDial*** pour afficher un dialogue correspondant à la fin.
Enfin nous affichons l'image de victoire/défaite avec les statistiques en refaisant appel à la fonction ***afficheImage***. Il sera alors demandé au joueur d'appuyer sur la touche ECHAP qui fermera la fenêtre et fera sortir le programme du ***while***.

B) Description de quelques fonctions importantes

a. Fonctions Collisions

L'une des fonctions primordiales pour notre programme est une fonction de gestion des collisions. Nous devons gérer les collisions du Joueur avec son environnement. Tous les sprites que nous avons utilisés sont rectangulaires bien qu'on ne le voit pas forcément à l'image. Puisqu'ils sont tous rectangulaires, nous avons utilisé une fonction comprise dans la bibliothèque de la SFML : « intersects ». Cette fonction retourne ***true*** si un rectangle touche un autre rectangle et ***false*** sinon.

Le fait de gérer les collisions nous permet alors de contrôler le placement de tous nos obstacles, pour éviter qu'il y en ait deux au même endroit. De plus, si le joueur ne pouvait pas faire de collisions, il ne pourrait pas gagner de points ou jouer au jeu.

b. Mise à jour des informations (FenUpInfo dans main.cc)

Afin de pouvoir jouer correctement au jeu, nous devons constamment mettre à jour les informations et cette fonction s'occupe de faire cela.

Tout d'abord nous mettons à jour les différents paramètres tel que la vie, le nombre de points du joueur, le temps et on vérifie s'il y a une collision entre le joueur et les différents obstacles sur la carte.

Ensuite, nous vérifions que le temps est positif pour continuer la partie, que nous soyons dans la partie normale du jeu ou contre le boss.
Le cas échéant, nous diminuons la variable de temps toutes les secondes à l'aide de la classe Clock de la SFML et nous avons une variable « ***spécial*** » qui est incrémentée toutes les 5 ou 10 secondes selon le niveau afin de mettre les bonus/malus sur la carte.
Si le personnage n'a plus de vie, nous mettons le booléen de fin du jeu à ***true*** et de victoire à ***false***. Cela nous servira à choisir l'une des deux fins possibles par le joueur.

Si le temps est nul nous vérifions les points du joueur, s'il a assez de points pour continuer la partie et s'il n'est pas contre le boss. S'il était dans le mode du boss, c'est la fin de la partie puisqu'il a survécu.

On a aussi des mises à jour secondaires comme la mise à jour de l'image d'Hillary Clinton qui change toutes les secondes.

A la fin de ces mises à jour, le main gère le comportement futur avec les différentes variables que nous avons modifié, par exemple si on est à la fin du jeu ou dans le mode du boss.

V) Parties de l'implémentation dont nous sommes fiers

De manière général nous sommes satisfaits de l'ensemble du programme que nous avons réalisé. Nous souhaitions réaliser un jeu qui caricaturait les élections, en particulier les présidentiels, et nous avons pu le faire avec l'interface graphique. Nous avions pour objectif de réaliser ce jeu dans l'esprit des jeux vidéo rétro, notamment dans celui de la Gameboy Color.

Ainsi dans le jeu, les phases de dialogues ont été reproduits de manière à coller le plus possible avec un dialogue de jeu Gameboy, c'est-à-dire qui s'afficherait au fur et à mesure avec un son de bip semblable à du morse.

Pour cela nous avons écrit nos dialogues dans des fichiers textes, et nous avons réutilisé la classe utility vue dans le TP sur les réactions moléculaires. Nous avons pu ainsi découper nos textes en morceaux uniformes qui étaient stockés dans un vecteur de string. Chaque élément du vecteur est ainsi lu caractère par caractère et ajouté au fur et à mesure dans un sf :: texte (classe de la SFML pour l'affichage de texte), et pendant que les caractères sont ajoutés dans ce sf :: texte, nous démarrons une musique qui reproduit nos bips. (Cf. fonction ***dialogue*** dans la classe ***Fenêtre***)

Nous sommes aussi fiers de l'implémentation des bonus et malus. En effet, nous avons réussi à afficher les enveloppes à un intervalle de temps voulu et de façon aléatoire. De cette façon, il est plus difficile pour le joueur de savoir s'il va tomber sur un malus, un bonus ou une enveloppe ce qui montre l'absurdité de la chose puisque la victoire se joue sur la chance.

CONCLUSION

Ce projet nous a permis de mettre en pratique les connaissances en C++ que nous avons acquis au cours de l'année. A travers ce dernier, nous avons pu utiliser des notions telles que la surcharge d'opérateur dans des contextes où son utilisation nous simplifiait la tâche. Ainsi il fallait réfléchir par nous-même aux endroits où une surcharge serait utile contrairement aux TP qui étaient guidés.

En outre, nous avons pu apprendre à utiliser une bibliothèque graphique. Avec la SFML nous avons mieux compris les concepts d'affichage dans une fenêtre graphique, avec notamment la gestion des collisions entre les objets ou des problématiques sur le taux de rafraichissement d'une image. En effet, sur un des ordinateurs, l'affichage allait trop rapidement donc nous avons configuré le nombre d'images par seconde afin d'obtenir la même vitesse pour tous les ordinateurs.

Par ailleurs il est a noté que nous avons eu des avertissements de fuites mémoires indiquées par Valgrind. Nous avons compris par la suite que ces dernières étaient dues à la bibliothèque SFML, par exemple il y avait des fuites mémoires lors de la création d'une fenêtre.