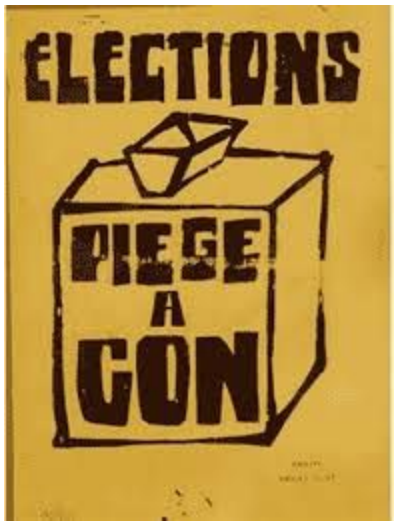
ASGARALY Sakina
LE BRAS Guillaume

Projet C++ EI-SE 4
POTUS ATTACK

Thème : élection, piège à con.



Année 2016/2017

Sommaire

I - Le jeu

Concept :

Nous avons décidé d'implémenter un jeu de type running, inspiré par exemple du jeu Line Runner sur android.
Le but étant que le personnage (candidats sur-caricaturés à des élections présidentielles impossibles) arrive au bout du niveau en ayant évité tous les obstacles (qui sont de forme rectangulaire, soit en les sautant soit éliminant avec un lancé de balle).

Les blocs rouges sont les obstacles à éviter, les bleus sont les bonus de vie et les roses les bonus d'invincibilité.

Chaque collision avec un obstacle fait perdre au personnage un nombre précis de points de vie .
Nous avons aussi 2 sortes de bonus. En rentrant en collision avec ces derniers, soit nous regagnons des points de vie, soit nous devenons invincible pendant une période de 10s.

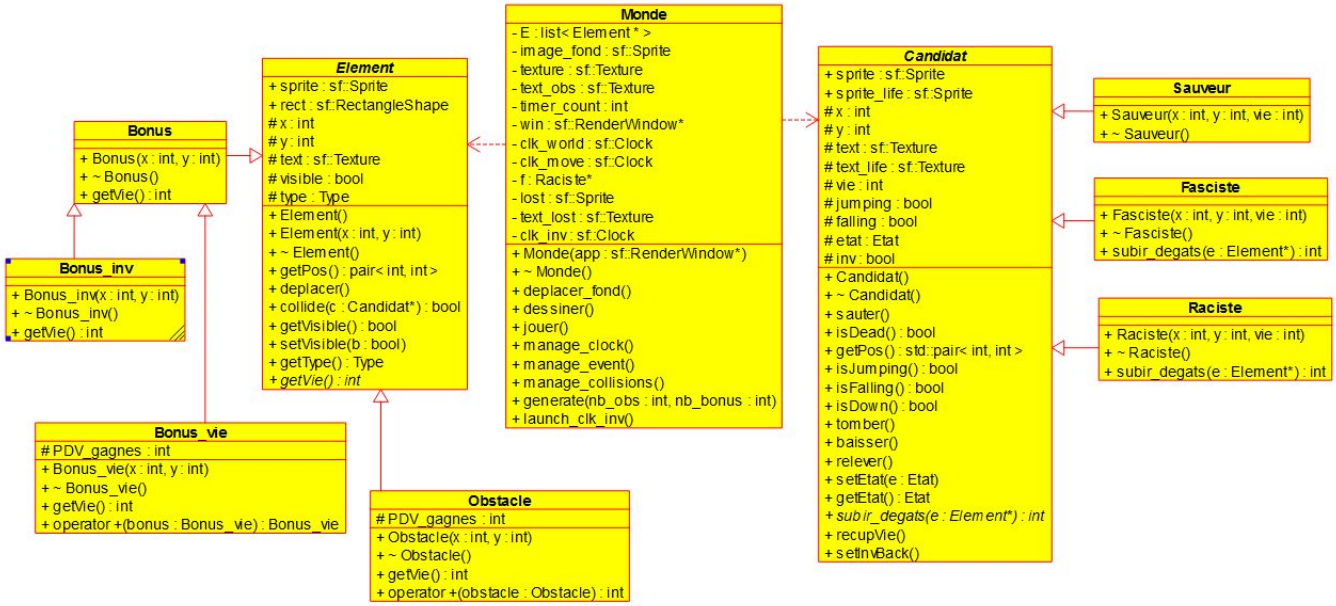
Le jeu se termine quand on arrive à la fin du niveau si le personnage est encore vivant (nombre de points de vie>0).
Il devient alors le nouveau président du monde.

Dans le cas contraire, si pendant le jeu le personnage meurt (nombres de points de vie = 0) alors le jeu s'arrête et nous n'avons pas de président (l'anarchie nous guettera alors).
La vie est représentée par un coeur qui se situe en haut à gauche de l'écran, il grossit ou rétrécit selon le nombre de point de vie que l'on possède.

Instruction :

Pour sauter SPACE, pour se baisser b.

II - Diagramme UML de l'application



Le diagramme de classes au format .png est fourni dans le répertoire.

III - Procédure d'installation et d'exécution

La rédaction du code a été fait à l'aide d'un éditeur de texte classique (gedit). Pour le côté graphique, nous avons utilisé la bibliothèque SFML en l'installant via le terminal avec la commande suivante : sudo apt-get install libsfml-dev .

Nous avons choisi la bibliothèque SFML écrite en C++, qui permet les mêmes libertés que la SDL, tout en étant plus haut niveau. Elle est donc plus adaptée pour débuter, en partie grâce aux abstractions qu'elle offre.

Afin de lancer le jeu, il suffit de compiler nos fichiers (make), puis taper la commande "main -h" afin d'afficher les options concernant le choix du personnage.
Pour finir, on exécute notre programme grâce à la commande "./main x" où "x" correspond à votre choix de personnage (1, 2 ou 3). Exemple : "./main 2é.

IV - Fonctions importantes

La fonction Generate:

```
void Monde::generate(int nb_obs, int nb_bonus) //generation des obstacles et des bonus
{
	int prev_val_x = 400;
	int prev_val_y = 300;
	for (int i = 0; i < nb_obs; i++) //
	{
		//nouvel obstacle : position en x comprise entre position en x de l'obstacle précédent et position en x precedente +200
		prev_val_x = prev_val_x+(rand()%12+4)*10 ;
		prev_val_y = 350 - 50*(rand()%3); //position aléatoire en y compris entre 350 et 250
		E.push_back(new Obstacle(prev_val_x, prev_val_y)); //on ajoute à la fin de la liste le nouvel obstacle
		//cout << "obstacle : " << prev_val_x << " ; " << prev_val_y << endl;

	}
	//cout <<"fin obs " << endl;
	prev_val_x = 400;
	for (int i = 0; i < nb_bonus; i++)
	{
		//nouveau bonus : position en x comprise entre position en x de le bonus précédent et position en x precedente +200
		prev_val_x = prev_val_x+(rand()%120+4)*10 ;
		prev_val_y = 350 - 50*(rand()%3); //position aléatoire en y compris entre 350 et 250
		//cout <<"fin obs " << endl;
```

```
                E.push_back(new Bonus_vie(prev_val_x, prev_val_y)); //on ajoute à la fin de la liste le nouveau bonus
                //cout << "Bonus vie : " << prev_val_x << " ; " << prev_val_y << endl;
        }
        E.push_back(new Bonus_inv(rand()%1000+2000, 350 - 50*(rand()%3)));
}
```

Cette fonction est importante car elle génère le niveau de jeu, et sans cette fonction, le jeu n'en est pas un. Le niveau est généré aléatoirement à chaque fois, chaque fois que l'on jouera, le challenge sera différent. Cette fonction nous rend fiers car la formule qui permet de générer le niveau paraît simple mais nous a demandé un travail assez conséquent. Nous avons par ailleurs bloqué un certain temps sur des bugs liés à l'utilisation de l'héritage sur les Bonus. Nous avons également utilisé le conteneur list<> de la STL pour contenir l'ensemble des éléments du niveau.

La fonction subir_degats(Element *e) qui suit est également une fonction dont nous sommes fiers car c'est celle qui nous a causé le plus soucis liés à l'utilisation d'une méthode virtuelle pure et à l'héritage. De plus, il a fallu utiliser l'astuce du type d'élément renvoyé par getType() pour pouvoir appliquer la bonne action sur le candidat. Cette fonction contient aussi les actions sur l'image du coeur qui représente la vie (augmentation ou réduction de la taille de l'image en fonction de l'élément avec lequel on rentre en contact).

```
int Fasciste ::subir_degats(Element *e)
{
        switch (e->getType())
        {
                case OBSTACLE :
                        if (!inv)
                        {
                                vie += e->getVie();
                                if (vie > 0) //si vie > 0 on diminue la taille de l'image
                                        sprite_life.setScale(0.01f*vie,0.01f*vie);
                        }
                        return 1;
                        break;

                case INV :

                        inv = true;
                        sprite.setScale(3, 3);
                        return 2;
                        break;
                case VIE :
                        vie += e->getVie();
                        if (vie > 0) //si vie > 0 on diminue la taille de l'image
                                sprite_life.setScale(0.01f*vie,0.01f*vie);
                        return 3;
                        break;
                case NONE :
                return 0;
```

```
                    break;
            }
            return 0;
}
```

V - Difficultées rencontrées

Lors de ce projet, nous avons rencontré plusieurs difficultés que nous pouvons regrouper en deux parties.

La première concerne l'organisation de notre groupe. En effet, notre emploi du temps, les échéances de fin de semestre ou encore l'arrivée de vacances nous ont demandé un certain effort pour parvenir à travailler efficacement sans se voir. La communication efficace au sein de l'équipe a toutefois aidé à surmonter ce problème.

La seconde partie se concentre sur la conception même de notre application. D'une part, il nous a fallu nous familiariser avec la bibliothèque graphique SFML (la gestion de la fenêtre, des évènements, gestion des objets…).
Ensuite, il nous a fallu gérer certaines erreur de compilation que nous ne maîtrisions pas encore, telles que des erreurs concernant les constructeurs des classes filles (ces constructeurs étaient déclarés mais non-initialisés).

D'autres erreurs de compilation sont survenue sans pour autant nous bloquer outre mesure.

VI - Améliorations possibles

- Animations du personnage (multiplier les sprites lors d'un mouvement/d'une action),
- Implémentation d'ennemis avec lesquels le candidat pourrait interagir,
- Diversification des obstacles pour rendre le jeu plus dynamique,
- Implémenter plusieurs niveaux, ou modifier la difficulté (gérer l'apparition d'obstacles, d'ennemis ou de bonus).