Rapport de Projet C++

Yellow Jacket

Etudiant :
Guillaume NGUYEN Main 4
Andrei FLEISER Main 4

Encadrant :
Cécile BRAUNSTEIN

Année 2018-2019

Table des matières

1 Introduction **2**

1.1 Yellow Jacket 2

2 Le Programme : **2**

2.1 Structure générale : 2

2.2 Classe Game : 4

2.2.1 Les conteneurs 4

2.2.2 Les Méthodes 5

3 Diagramme UML **6**

4 Procédures : **6**

4.1 Procédure d'installation : 6

4.2 Procédure d'exéctutions 6

4.2.1 Fenêtre d'introduction 6

4.2.2 Partie principale : 7

4.2.3 Game Over : 8

4.2.4 Victoire : 9

5 Parties dont on est les plus fiers **9**

1 Introduction

1.1 Yellow Jacket

Notre jeu reprend l'histoire des gilets jaunes. Il met en scène les différents partis qui ont pris place durant les manifestations de novembre à janvier. On incarne un personnage entre Andrei et Guillaume. On a en effet préféré concevoir des personnages à notre effigie afin de ne pas avoir à utiliser des images d'autres personnes réelles et respecter ainsi le droit à l'image. L'agent de police, étant non-identifiable grâce à son casque, nous n'avons pas eu à prendre des mesures supplémentaires. Le but est d'esquiver les grenades et les flashball que lance aléatoirement le CRS. Par ailleurs, le joueur peut se déplacer latéralement sur cinq files de voitures. Les voitures roulent une par une sur chaque ligne et le joueur doit arrêter un maximum de voitures qui s'arrêtent lorsqu'elles voient le joueur devant elles. Elles doivent attendre un délai de 3 secondes avant de pouvoir repartir. Le joueur doit ainsi bouger rapidement entre les files pour pouvoir espérer bloquer un maximum de voitures tout en évitant les grenades. À chaque fois qu'il est touché, il perd une vie. Il perd entièrement quand il est touché à trois reprises. Il peut également perdre si le timer (initialisé à 3 minutes) en haut à droite atteint 0 seconde. Pour gagner, il doit bloquer 25 voitures (nombre qui pourra être modifié dans la fonction main()).

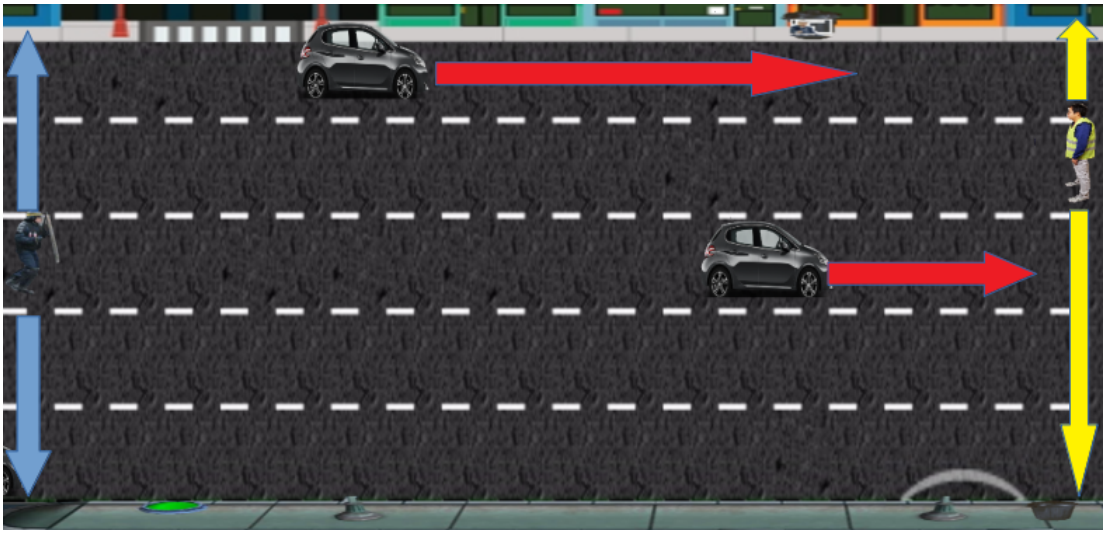

FIGURE 1 – Les divers déplacements

2 Le Programme :

Dans sa conception, *Yellow Jacket* s'inspire du jeu *Astre* que nous avons vus en TP noté. Tout comme ce dernier, il utilise la bibliothèque graphique SFML et possède une classe *Ecran* pour la gérer.

2.1 Structure générale :

En tout, notre programme possède 10 classes avec 3 niveaux hiérarchiques :

— **Ecran :** Tout comme la classe *Screen* du jeu *Astre*, la classe Ecran agit comme un intermédiaire entre la gestion graphique et les autres classes. Si une classe souhaite interagir avec l'écran, elle doit passer par des méthodes publiques de la classe Ecran. C'est en effet elle qui charge et stock l'ensemble des ressources graphique (Textures et Sprite) et audio utiles au jeu.

— **Mobile :** Il s'agit d'une classe abstraite permettant de représenter l'ensemble des éléments mobiles dans le jeu (le joueur, les CRS, les voitures...). Elle est la racine de l'arbre de dérivation. Elle possède quelques attributs protégés propres à l'ensemble des objets mobiles, tels que les coordonnées x-y, la largeur, la hauteur...
Hormis des accesseurs et mutateurs, elle possède une méthode virtuelle pure draw(Ecran &e) qui permet aux objets des classes dérivés de se dessiner eux même à l'écran.

— **Personnage :** Il s'agit là aussi d'une classe abstraite, dérivant de Mobile. Elle donne naissance aux deux personnages antagonistes du jeu : le Gilet jaune et le CRS
La classe possède un attribut protégé *string name* qui va contenir le nom du personnage. Cet attribut va surtout être utile pour le gilet jaune où il indiquera quel personnage jouable a été choisi par l'utilisateur ("Guillaume" ou "Andrei").

— **CRS :** La classe CRS possède un attribut *float LOVE* (Level Of ViolencE)[1] qui représente son niveau de violence. En d'autres termes, il s'agit de l'indicateur qui définira la quantité de grenades de désencerclement que le CRS va devoir lancer par unité de temps. Bien sûr, ce niveau de violence sera amené à augmenter au fur et à mesure que le joueur sera proche de la victoire.

— **Gilet_joueur :** Il s'agit de la classe associé au personnage avec lequel on va jouer. Notre personnage peut avoir trois orientations possibles : de face, de dos et de profil. Elles sont représentées par une énumération nommée *Orientation* et définie dans le header de la classe Gilet_joueur.
Par ailleurs, le personnage a aussi un nombre de vies initialisé à 3, et qui diminue lorsque une grenade ou une flashball rentre en contacte avec le sprite du gilet jaune. Pour ce faire, la classe Gilet_joueur utilise la méthode "bool colision_grenade(float x, float y, Ecran &e)" qui renvoie un booléen indiquant si un projectile de coordonnées x, y est rentré en contacte avec le personnage.

— **Voiture :** Classe dérivant de la classe Mobile, et qui sert à créer les voitures qu'il faudra arrêter. Cette classe possède les deux surcharges d'opérateurs du jeu :

  — virtual int operator-(Mobile const&M) : Cette méthode virtuelle renvoie la distance horizontale entre la voiture et l'objet Mobile passé en paramètre. Dans la pratique cet objet mobile sera soit la voiture placée devant la voiture appelant cette méthode, soit le personnage Gilet jaune.

  — bool operator() (Voiture const& V) ou (Gilet_joueur cont& J) : Cette méthode est surchargée car elle peut prendre en paramètre, soit une autre voiture soit le gilet jaune.
  Dans la pratique, cette méthode va permettre à chaque voiture de prendre de

1. Référence au jeu Undertale

l'information sur l'élément qui est devant elle (autre voiture ou gilet jaune) et de réagir en conséquence.
Par exemple, si l'élément devant elle ralentit ou est à l'arrêt et que la distance qui les sépare (donnée par l'opérateur "-") est inférieur au seuil minimum établi ; elle devra elle-même ralentir, voir s'arrêter. De même, si elle est à l'arrêt et qu'il n y'a aucun obstacle sur sa route, elle doit attendre 3 secondes avant de redémarer.

Par ailleurs, la classe Voiture possède aussi un attribut static nbr_voiture_arret qui est un conteur du nombre total de voitures arrêtées sur la map.

— **Projectile :** Il s'agit d'une classe abstraite héritant de la classe Mobile. Elle sert de support aux deux types de projectiles du jeu : les grenades de désencerclement et les flashball.
Elle possède la méthode virtuelle pure update(float time) qui va servir à mettre à jour la position des projectiles selon une trajectoire qui leur est propre en fonction du temps. Elle renvoie de plus un booléen indiquant si le projectile est encore en train de voler (true), ou s'il a atterrit (false).

— **Flashball :** Il s'agit d'une classe assez simple. En plus du constructeur et des éléments héritées, elle possède juste un attribut vitesse. Sa trajectoire est, en effet, très simple puisqu'il s'agit d'une ligne droite que la flashball va parcourir assez rapidement.

— **Grenade :** La grenade suit une trajectoire parabolique de coefficient directeur négatif. Elle doit terminer cette trajectoire en un temps donné par l'attribut "_duree". La classe Grenade redéfinit la méthode "update(float time)" de manière à ce qu'elle utilise des équations horaires pour mettre à jour les coordonnées x et y à chaque temps "time".

La classe Game, étant plus complexe, on va l'analyser plus en détail dans la sous-section suivante :

2.2 Classe Game :

Tout comme dans le jeu "Astre", la classe Game est la classe où se déroule le jeu. Elle contient entre autre, des attributs Ecran, Gilet_joueur, CRS et plusieurs timer "sf : :Clock", servant à la gestion temporelle du jeu.

2.2.1 Les conteneurs

C'est dans cette classe que se trouvent les 2 conteneurs du jeu :

— **vector<deque<Voiture» file :** Il s'agit en faite d'un double conteneur qui stock l'ensemble des objets de classe Voiture qui sont présents sur la Map. Le premier conteneur est un "vector" contenant 5 "deque" représentant les 5 voie de circulations présents sur la map. Chaque double queu "deque" contient donc les voitures

présentes sur la voie à laquel il correspond. Nous aurions pu utiliser un conteneur "queu" étant donné que les première voitures à entrer sur une file de circulation sont aussi les premières à en sortir. Toutefois, ce conteneur ne permettait pas d'être parcouru avec un itérateur ; nous avons donc préférer utiliser la double queu "deque".

— **list<Projectile*> projectile :** Il s'agit d'une collection hétérogène de vecteurs d'objets de classe Projectile. Cela est justifié par le fait que les flashball et les grenades ont un fonctionnement similaire. Le choix du conteneur "list", lui est due au fait que cela permet de supprimer plus facilement les éléments, peu importe leur position dans le conteneur. En effet, les flashball ont une vitesse plus grande que les grenades et sont donc suceptibles d'arriver à leur déstination plus rapidement que ces dernières. Une flashball peut donc être supprimée de la liste avant une grenade qui y serait pourtant rentrée plus tôt. Il est donc utile de pouvoir supprimer un projectile quelque soit sa position dans la liste.

2.2.2 Les Méthodes

La classe Game contient plusieurs méthodes dont la plus importante est **bool play()**. Il s'agit la méthode centrale de la classe. C'est à l'intérieur que va se jouer toute la partie via des appels à d'autres méthodes.
Par ailleurs, comme le joueur a la possibilité à la fin d'une partie d'en recommencer indéfiniment une nouvelle ; la méthode play() renvoie un booléen qui indique la décision qui a été retenue par l'utilisateur. Concrètement la méthode play() est appelée à l'intérieur d'une boucle while dans la fonction main(). Cette boucle ne s'arrête que si la méthode play() renvoie un false. Il est ainsi possible de refaire plusieurs parties à partir d'un seul objet de type Game, ce qui permet de ne pas avoir à chaque fois à recharger les ressources graphiques et audio.
C'est là qu'intervient la méthode **void initGame()** qui permet d'initialiser et surtout de réinitialiser les paramètres de l'objet Game à chaque fois que l'on fait ou refait une partie.

Il y a également d'autre méthodes telles qu'entre autres *void car_act(), void crs_act, void grenade_act()* qui se chargent respectivement des actions des voitures, du CRS et des projectiles ; ou bien encore *void evenement()* qui se s'occupe de l'aspect événementiel de la partie principale du jeu.

3 Diagramme UML

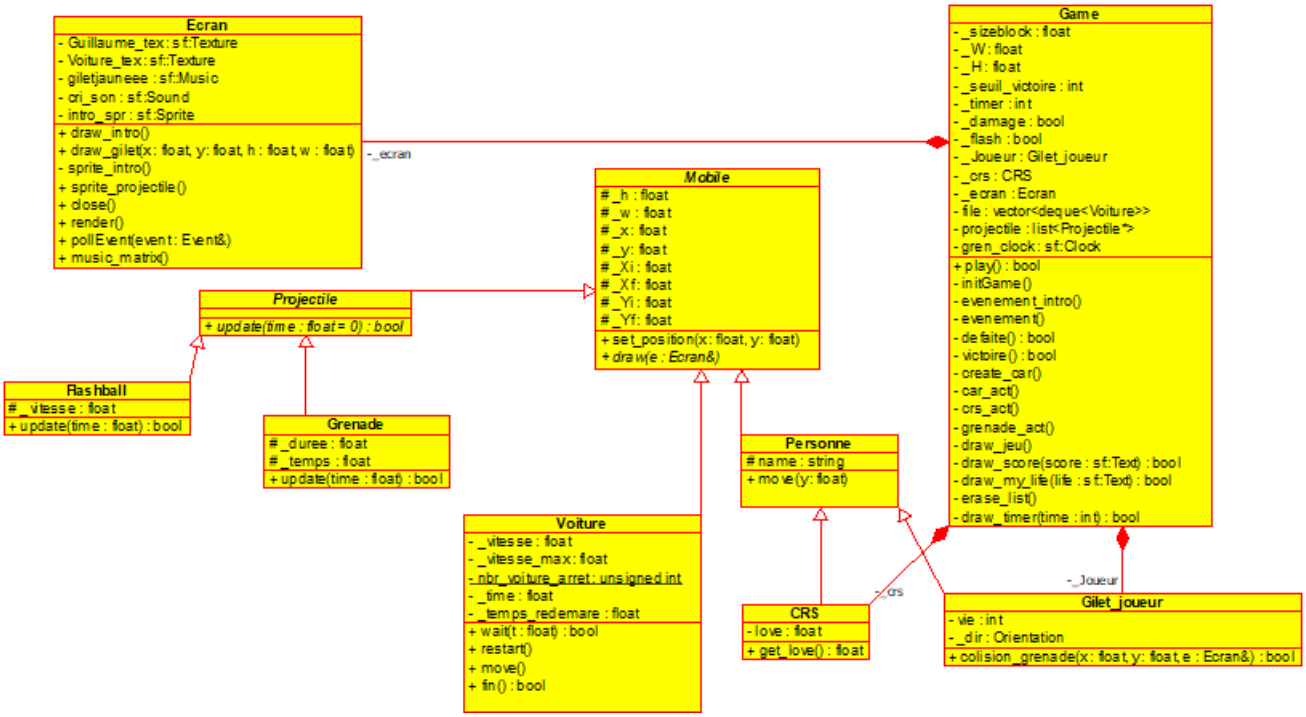

FIGURE 2 – Diagramme uml

4 Procédures :

4.1 Procédure d'installation :

Yellow Jacket utilise la bibliothèque graphique SFML-2.5.1.(la même que celle qui est utilisée par le jeu Astre). Cette bibliothèque étant déjà installée sur les machines de l'école, l'utilisation de Yellow Jacket ne nécessite pas de mesures supplémentaires.

4.2 Procédure d'exéctutions

4.2.1 Fenêtre d'introduction

Une fois que le programme est lancé dans le terminal, le joueur est placé devant une fenêtre d'introduction dans laquelle il lui est demandé de choisir son personnage parmis 2 choix possible : Guillaume et Andrei (les deux se jouent de la même façon)

FIGURE 3 – Ecran d'accueil

Pour choisir, il suffit de cliquer sur son personnage préféré et le jeu commence directement.

4.2.2 Partie principale :

C'est là que le jeu commence véritablement. Comme présenté dans l'introduction de ce rapport, le joueur va devoir se déplacer avec les touches directionnelles verticales du clavier afin d'arrêter les voitures qui viennent vers lui tout en évitant de se faire toucher par les projectiles du CRS.

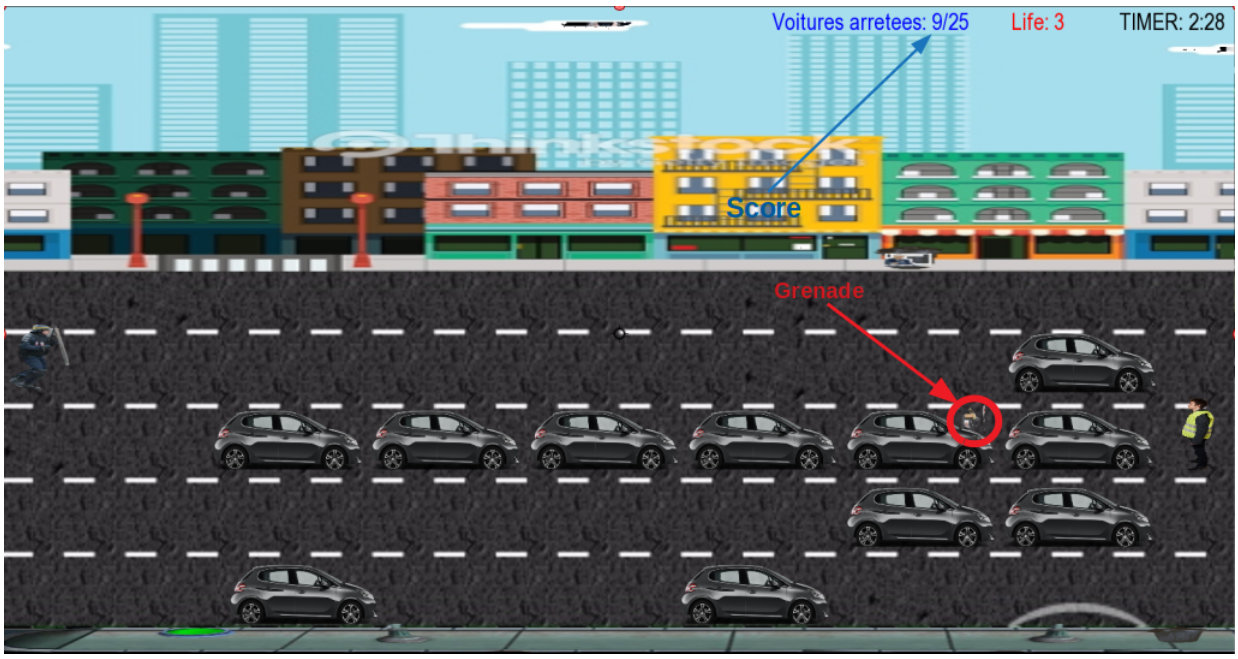

FIGURE 4 – Partie principale du jeu

Attention, il faut bien rester devant la voiture jusqu'à ce qu'elle s'arrête complètement, autrement elle va juste ralentir puis réaccélérer juste après que le joueur ne soit plus devant elle. Il faut aussi périodiquement repasser devant les voitures arrêtées sinon elles redémarrent au bout de 3 secondes.

4.2.3 Game Over :

S'il perd ses 3 vies ou que les 3 minutes qu'il avait pour gagner sont écoulées, le joueur se fait taper et embarquer par les forces de l'ordre. Il lui est alors proposé de revenir tenter sa chance le samedi suivant en cliquant sur le bouton "Try again" et en recommençant une partie.

FIGURE 5 – écran de game over

4.2.4 Victoire :

Si le joueur parvient à arrêter 25 voitures, il a droit à la victoire. Pour accéder à cette phase de jeu sans avoir à arrêter 25 voitures, il suffit d'aller dans la fonction main() et de modifier le second argument du constructeur de Game (initialisé à 25) par un nombre de voitures plus petit (5 par exemple).

```
int main(int argc, char const *argv[])
{
    float sizeblock=100;//Taille des blocs
    bool try_again(true);//vrai tantque le joueur
    Game jeu(sizeblock, 25, 180); // 2ème argument:

    while(try_again){//Tant que le joueur souhaite
        try_again=jeu.play();
    }
```

FIGURE 6 – Comment hacker le jeu

5 Parties dont on est les plus fiers

Il y a plusieurs aspects de l'implémentation dont nous sommes fiers.
Le codage de la classe Voiture et des méthodes de Game qui l'utilisent a sans aucun doute été la partie la plus complexe du projet.
Coder la trajectoire parabolique des grenades ne fut également pas une mince affaire et nécessita de prendre un papier et un crayon pour trouver les équations horaires en nous inspirant des souvenirs des cours de physique mécanique.
Prendre en main les divers aspects de la programmation graphiques (textures, sprite, temps...) fût aussi intéressant.
Toutefois, ce qui nous rend le plus fier, c'est le jeu en lui-même. Il s'agit du premier jeu vidéo que l'on crée de A à Z. Au delà des contraintes techniques imposées, nous avons voulus réaliser un jeu fluide, agréable à jouer, avec des sons et musiques sympathiques ; et qui reprenne des constantes du jeu vidéo telles que les timers, le score, les vies, les easter eggs...