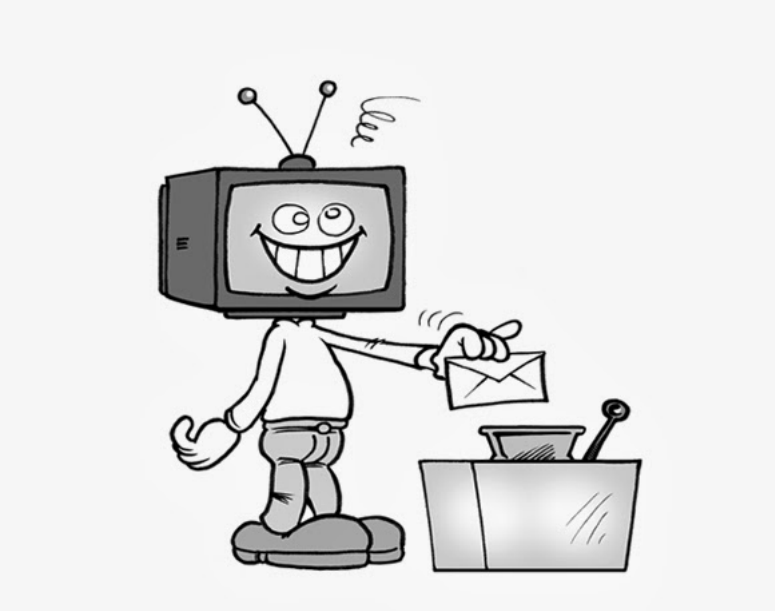
POLYTECH
PARIS-UPMC

PROJET C++

Sujet : Elections, pièges à cons



DANIELI David
DE GIMEL Agnès
MAIN 4

Professeur : Cécile BRAUNSTEIN

Semestre 7

Table des matières

Introduction

Dans le cadre de l'UE de C++, nous avons été amené à créer un jeu minimaliste, s'inscrivant dans le thème : "Election, piège à cons". Le projet était entièrement libre mais avec quelques contraintes techniques à respecter. L'élaboration du code source de ce jeu nous a fait réutiliser les notions vues en cours, mais aussi fait découvrir une bibliothèque graphique commune et accessible : la SDL.
Ce rapport explique le principe du jeu et détaille la structure des codes, en montrant les différentes classes utilisées et les étapes de la simulation.

1 Principe du jeu

1.1 But du jeu

Le joueur contrôle un hélicoptère placé en hauteur de la fenêtre graphique et à l'origine au milieu de la largeur. Il ne peut que le déplacer à gauche ou à droite. En même temps, au sol, des bonshommes apparaissent régulièrement à gauche de l'écran et le parcourent (à la même vitesse). Arrivés à l'autre bout, ils sont arrivés et disparaissent. Certains des bonshommes sont blancs, les autres sont noirs. Ce sont en fait des électeurs allant voter. Le joueur doit en fait faire en sorte qu'un plus grand nombre de ses électeurs (les blancs) arrivent au bureau de vôte que ses opposants (les noirs). Il a donc la possibilité de jeter des projectiles. Si un votant est touché, il est éliminé.
Le joueur a le choix entre deux projectiles : un couteau, ou une bombe qui a une zone d'impact plus importante. Le joueur doit donc éliminer le plus de votants noirs possibles en faisant attention de ne pas blesser trop de ses votants dans le processus, pour qu'à la fin de la simulation, ses alliés soient plus nombreux que ses opposants à l'arrivée.

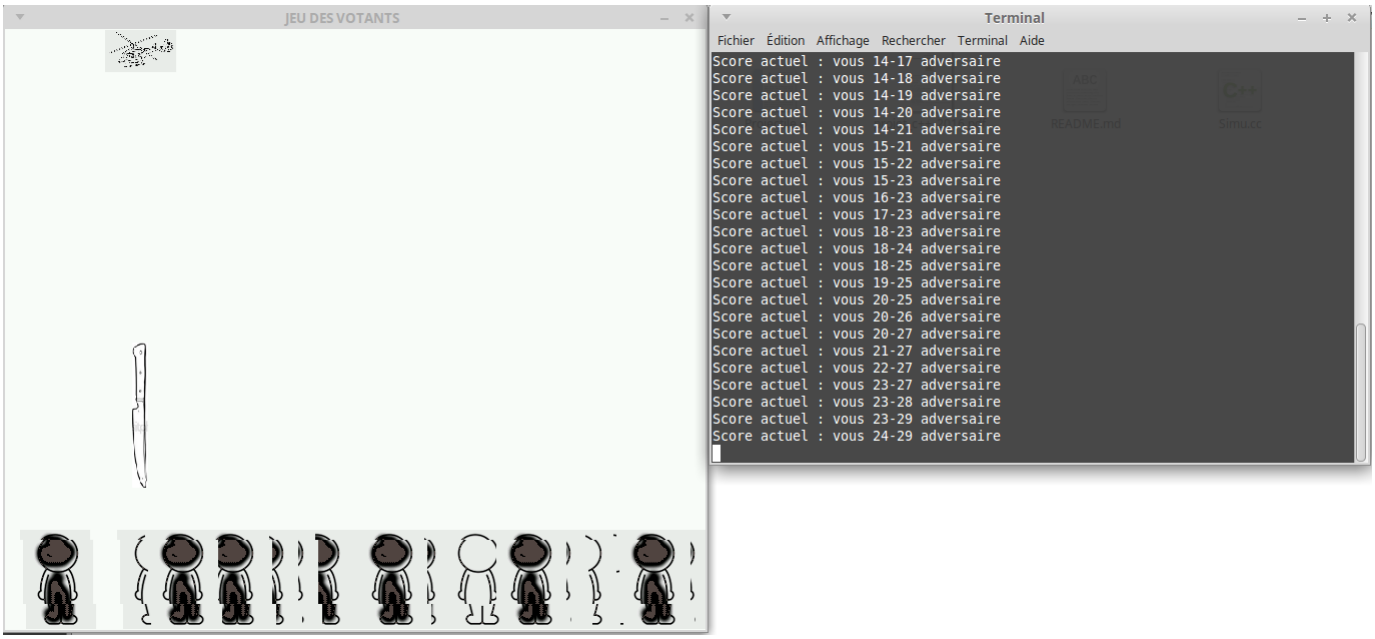


FIGURE 1 – Exemple de partie

1.2 Les classes

1.2.1 Diagramme de classe des figures

La classe figure est la classe mère d'un certains nombres de classes, la hiérarchie est détaillée dans le digramme suivant de la figure 2.

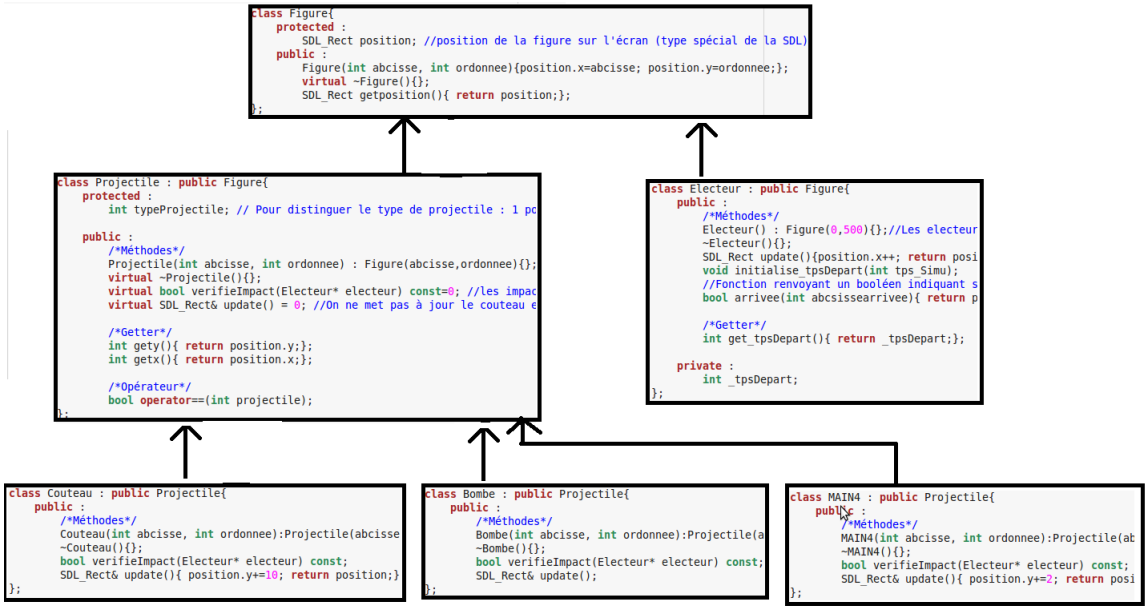


FIGURE 2 – Diagramme de classe des figures

1.2.2 La simulation

La simulation est une classe qui est appelée pour faire fonctionner le jeu. Sa structure est présentée en Figure 3. Elle contient les attributs suivants :

- _tpsCourant qui est initialisé à 0 et est incrémenté à chaque appel de run().
- _tpsSimulation : choisi à la construction
- cpt_vote_pour_1 : à chaque appel de run, regarde si de nouveaux alliés sont arrivés et incrémente en conséquence.
- cpt_vote_pour_2 : même chose pour les votants opposants.
- joueur1 : joueur qui joue (structure expliquée plus loin)
- _listProjectiles : à chaque appel du run, si le joueur lance un couteau ou une bombe, l'élément s'ajoute dans la liste.
- _listElecteur1 : contient les électeurs alliés déjà partis et pas encore arrivés (et toujours en vie !) et devant être affichés dans la fenêtre graphique.
- _listElecteur2 : même chose pour les opposants.

Les méthodes sont pour la plupart appelées par la fonction run() qui est détaillée plus loin.

1.2.3 La classe Joueur

La classe joueur a pour structure celle présentée sur la figure 4. Elle est représentée par l'image de l'hélicoptère.

```
    private :
        int _tpsCourant;
        int _tpsSimulation;
        int cpt_vote_pour_1;
        int cpt_vote_pour_2;
        Joueur joueur1;//Le joueur qui joue est le joueur 1
        Joueur joueur2;
        std::vector<Projectile*> _listProjectiles;
        std::vector<Electeur> _listElecteur1; //votent pour 1
        std::vector<Electeur> _listElecteur2; //votent pour 2
    public:
        Simu(int tpsSimulation): _tpsSimulation(tpsSimulation){cpt_vote_pour_1=0; cpt_vote_pour_
        ~Simu(){};
        void initialise();
        int run(SDL_Surface *ecran,SDL_Surface* helico,SDL_Surface *couteau,SDL_Surface *bombe,SD
positionHelico,SDL_Event& event, int &continuer);
        void gererEvenementsClavier(SDL_Event& event,int &continuer,SDL_Rect& positionHelico);
        void MajAffichageProjectiles(SDL_Surface* ecran,SDL_Surface *couteau,SDL_Surface *bombe)
        void checkEntrants();
        void checkArrivees();
        void MajAffichageElecteurs(SDL_Surface* ecran,SDL_Surface *votant1,SDL_Surface *votant2)
        void supprimer_bonhomme1();
        void supprimer_bonhomme2();
        /*Getter*/
        int getcpt1(){ return cpt_vote_pour_1;};
        int getcpt2(){ return cpt_vote_pour_2;};
        int get_tpsSimulation(){ return _tpsSimulation;};
        std::vector<Electeur>& getListElecteur1(){return _listElecteur1;};
        std::vector<Electeur>& getListElecteur2(){return _listElecteur2;};
};
```

FIGURE 3 – Classe Simulation

```
class Joueur{
    public :
        Joueur(){};
        ~Joueur(){};

        /*Getter*/
        std::vector<Electeur>&  getlistVotants(){

    private :
        int _abcisse;
        std::vector<Electeur> listVotants; //List
};
```

FIGURE 4 – Classe Joueur

2 Structure du code

2.1 Le main

Dans le main, on télécharge notamment les images nécessaires à la fenêtre graphique (ce sont des formats .bmp). On a donc une image d'un votant blanc, une image d'un votant noir, d'un couteau, d'une bombe et une image d'hélicoptère. On crée ensuite la simulation qui est le coeur du projet. Le main comprend un while qui va appeler la méthode run de la simulation pour mettre à jour le jeu et la fenêtre graphique.
C'est également dans le main que se fait tout l'affichage. Pendant le jeu, il est donc important de laisser le terminal ouvert à côté pour suivre le décompte des points. En effet, il affiche le nombre d'alliés et le nombre d'opposants arrivés, et donne ainsi de précieuses indications sur la situation du joueur.

2.2 La simulation

Le constructueur de simulation prend en paramètre un entier qui définira la durée de la simulation. La classe simulation a une méthode update() qui à chaque pas de temps va :

— Interpréter la commande de l'utilisateur. À chaque pas de temps, l'utilisateur peut appuyer sur une flèche (gauche ou droite) du clavier pour diriger l'hélicoptère ou appuyer sur la flèche du bas pour lâcher un couteau et sur la flèche du haut pour lâcher une bombe (qui a une zone d'impact plus importante). S'il ne fait rien, il faut malgré tout que les votants avance et le

temps ne doit pas se "figer" en attendant que l'utilisateur appuie sur une touche. Pour cela, nous avons utilisé la fonction PollEvent de la SDL.

— Mettre à jour les projectiles. Les projectiles qui ont été lancés et n'ont pas encore été détruits sont stockés dans une liste dans la classe Simulation. On mets leurs coordonnées à jour et on les affiche.

— Vérifier si des votants entrent dans la fenêtre graphique. Pour cela on parcourt les deux listes des électeurs (alliés et adversaires) et on regarde pour chacun la durée de départ, définie aléatoirement entre le temps de la simulation moins une certaines valeur (pour que les derniers aient les temps d'arriver).

— Vérifier si certains sont arrivés : même chose, on parcourt les listes pour voir l'abscisse courante, au delà d'une certaine valeur, on supprime l'électeur de la liste et on incrémente le bon compteur (en fonction de la liste à laquelle appartient l'électeur, soit opposant ou allié).
— Mise à jour des électeurs, on incrémente l'abscisse des électeurs des deux listes.

— On vérifie les impacts des projectiles sur les électeurs. On vérifie d'abord leurs ordonnées. Au delà d'une certaine valeur, on regarde pour les couteaux s'ils touchent un ou plusieurs électeurs grâce à la fonction verifieImpact() dans la classe Projectile. Si c'est le cas, ils sont détruits ainsi que leurs cibles. Même chose pour les bombes qui elles doivent toucher le sol (vont plus loin que les couteaux) et couvrent une zone plus large pour verifieImpact().

3 La fonction Poll Event

Un des enjeux principaux pour réussir la création du jeu : maîtriser la gestion des événements. En effet, il était important qu'à chaque pas de temps, le joueur puisse agir directement sur la fenêtre graphique en déplaçant son personnage ou en jetant un projectile. Nous nous sommes donc penchés sur les différentes façons de gérer tout cela avec la SDL.

Nous avons trouver deux façons de gérer des évènements de ce type, les deux méthodes étant assez similaires. Il s'agit des fonctions SDL_WaitEvent() et SDL_PollEvent(). Leur utilisation est simple : elles prenent en paramètre un événement de type SDL_Event. Il faut ensuite regarder la sous-variable event.type et faire un test sur sa valeur. Généralement on utilise un switch pour tester l'événement.

La différence entre les deux fonctions sont que SDL_WaitEvent() est bloquante : elle attend un événement, ce qui est inconvénient puisque l'on souhaite que si le joueur ne fait rien, les bonshommes continuent d'avancer et les projectiles lancés de descendre. A l'inverse, la fonction SDL_PollEvent() n'est pas bloquante, c'est pour cette raison qu'elle a été retenue.

Conclusion

Ce projet nous a permis de mettre en application les notions que nous avons vues en cours de C++ comme l'héritage, les méthodes virtuelles etc... mais aussi de découvrir une bibliothèque graphique opensource et très intéressante avec laquelle nous avons du nous familiariser pour réussir à gérer un bon affichage.

Nous avons pu mener le développement d'un petit jeu vidéo minimaliste de bout en bout, projet qui nous a beaucoup apporté sur le plan technique, et qui de surcroit nous a beaucoup plu à coder et une fois réalisé, à utiliser.

Des améliorations pourraient être apportées à la simulation et notamment l'affichage, les bons-hommes très proches se couvrant entre eux et ne sont donc pas tous visibles. De plus, le score peut être suivi sur le terminal mais l'on pourrait imaginer l'écrire sur la fenêtre graphique.