M2203 - Projet de POO

Runner

Jules Despret -- Romain Cremery
S2A

2015 / 2016

Enseignant Responsable : M. Rubi

Table des matières

1. Introduction

1.1. Contexte

Au cours de notre second semestre en DUT Informatique notre tâche était de réaliser un projet commun aux modules Conception et Programmation Orientée Objet nous a été proposé.

Ce projet servait à nous faire manier la programmation objet et la librairie standard du C++ (STL), tout en nous faisant découvrir la libraire SFML.

1.2. Objectifs

Le but de ce projet est de produire un jeu vidéo “Runner”; c’est-à-dire un personnage qui court sans arrêt et doit sauter pour éviter des obstacles et attraper des bonus pour faciliter le jeu et faire grimper le score.

1.3. Outils utilisés

Pour réaliser ce projet, nous nous sommes servi du langage C++ couplé à la libraire SFML 2.0, permettant un affichage graphique grâce à de nouvelles classes. Pour ce qui est du versionning, nous avons opté pour GitHub.

2. Fonctionnalités demandées

Les fonctionnalités suivantes ont été intégrées avec succès :

Fonctionnalité minimales :

Écran d'introduction du jeu ✓

1. Écran de menu permettant via des boutons :
    1. de lancer le jeu ✓
    2. de quitter ✓
2. Écran de jeu permettant d'afficher :
    1. un arrière plan qui défile ✓
    2. la balle (joueur), des obstacles à éviter, des points à ramasser ✓
    3. le score et le niveau de vie du joueur
3. Pendant le déroulement du jeu : ✓
    1. la balle peut se déplacer sur un axe des abscisses et sauter pour éviter les obstacles ✓
    2. Arrivée aléatoire des points et des obstacles ✓
    3. 3 sortes d'obstacles ✓
    4. gestion du score ✓
    5. gestion du niveau de vie ✓
4. Un écran de transition lorsque la partie est terminée ✓

Fonctionnalités supplémentaires :

1. Gestion de bonus pour le joueur (vie, invincibilité, capacité de voler, ralentissement du jeu, etc...)

↳ **Nous nous sommes contentés d'introduire le bonus de soins, que nous avons appelé "Médikit".**

Les fonctionnalités suivantes n'ont pas pu être intégrées :

2. Gestion des meilleurs scores avec affichage d'un écran des meilleurs scores depuis le menu
3. Écran d'options accessible depuis le menu permettant de :
    1. Régler la difficulté du jeu (vitesse du jeu, nombre d'obstacles à éviter)
    2. Support multi-langues
4. Changement de fond défilant avec le temps (au bout d'un certain temps, le personnage atteint un nouvel "endroit")
5. Ajout de son (fond sonore, bruitage)

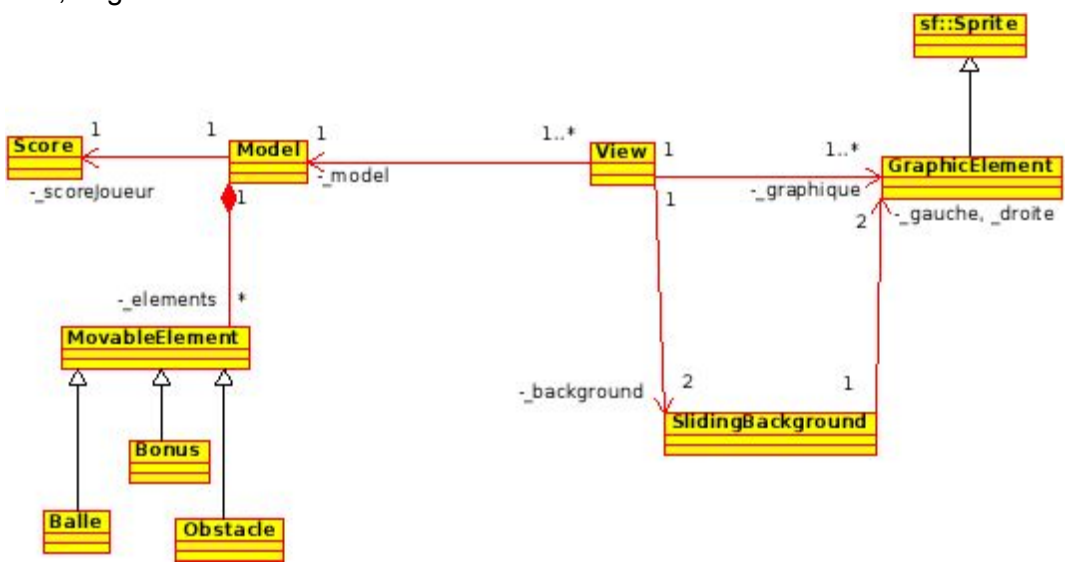
Fonctionnalités bonus :

1. Ajout d'argent, sauvegarde et chargement de l'argent gagné lors des parties précédentes
2. Ajout d'un écran d'achat d'améliorations (plus de vie, saut plus élevé, invincibilité plus longue, etc...) et sauvegarde/chargement de ces améliorations
3. changement de charte graphique (modifiable depuis l'écran d'options)

3. Implémentation

3.1. Architecture générale

Le projet adopte une architecture Modèle-Vue. Toutes les données sont stockées et gérées dans un Modèle. La Vue affiche le jeu à partir de ces données sans pouvoir les modifier directement, et gère les entrées clavier et souris.



3.2. Détail des classes

➔ Model : Gère les élément et le joueur, et calcule chaque étape du jeu.

➔ Score : Enregistre un pseudonyme et calcule le score total du joueur à partir du temps et du nombre de pièces et de bonus ramassés.

➔ MovableElement : Définit un élément mobile, avec des coordonnées, une taille, et des méthodes pour le manipuler. Il constitue le coeur du jeu en étant à l'origine de la plupart des objets manipulés dans le Modèle.

➔ Balle : Définit l'élément mobile correspondant au joueur. A la différence des autres éléments mobiles, la balle possède des points de vie et est le seul élément pouvant être manuellement manipulé durant la partie.

➔ Bonus : Définit un élément mobile générique servant à accorder des bénéfices au joueur s'il le ramasse. Les bonus sont différenciés par une chaîne de caractère, le "type", qui permet de définir comment l'afficher et quel bénéfice attribuer au joueur s'il le ramasse. Il existe deux sortes de bonus : les pièces et les médikits.

➔ Obstacle : Définit un élément mobile générique dont le rôle sera d'infliger des dommages au joueur si celui-ci rentre en collision avec. Comme pour les bonus, les obstacles sont différenciés par un "type" qui définit leur apparence. Il existe trois sortes d'obstacle : les basiques, les aériens et les grands.

➔ View : Gère l'affichage d'images et de texte à partir de la classe Model et détecte puis gère les évènements utilisateurs, c'est-à-dire les entrées clavier et souris.

➔ GraphicElement : Définit un élément graphique, dont les coordonnées, les informations et l'apparence sont directement liés à un MovableElement. Cette classe hérite de la classe Sprite, qui appartient à l'API SFML.

➔ SlidingBackground : Cette classe permet de gérer les images affichées en arrière-plan du jeu. Le SlidingBackground est constitué de deux GraphicElement joints à partir desquels ont produit un fond défilant.

4. Algorithmes

4.1. Algorithme de calcul de coordonnées d'un nouvel élément

```
void Model::rajouterElement(bool debutJeu)
{
    calculerEcart();
    int xAgauche = 0;
    for(auto e : _elements) {
        if(e->getX() > xAgauche)
            xAgauche = e->getX();
    }
    if(debutJeu)
        ajouterElementAleatoire(xAgauche + _ecart + LARGEUR_JEU);
    else
        ajouterElementAleatoire(xAgauche + _ecart);
}
```

Cet algorithme est appelé dans le modèle lorsqu'il faut ajouter un nouvel élément au jeu, soit à son début pour l'initialiser, soit lorsqu'un autre élément a été supprimé et doit donc être remplacé. Son but est de calculer une coordonnée X et d'appeler une méthode qui va créé un élément aléatoire placé à cette cordonnée X calculée.

L'algorithme commence par appeler la méthode "calculerEcart" du modèle qui va calculer un écart aléatoire et le stocker dans l'attribut "_ecart" du modèle. La suite va, au moyen d'une boucle, trouver la coordonnée X de l'élément le plus à droite dans la collection d'éléments du modèle.

A partir de cette coordonnée X trouvée et de l'écart calculé, un autre algorithme est appelé pour déterminer le type de l'élément, le construire et l'insérer dans la collection d'éléments du modèle.

Étant donné que cet algorithme se base sur des coordonnées et non sur une gestion du temps, il faut commencer par initialiser le tout premier élément du jeu. C'est ici qu'intervient le booléen "debutJeu", qui si activé va ajouter à la coordonnée X la largeur de l'espace de jeu, afin que le tout premier élément apparaisse en dehors de l'écran.

4.2. Algorithme de déterminisation et d'insertion d'un nouvel élément

```
void Model::ajouterElementAleatoire(int xCourant)
{
    MovableElement *nouvelElement = nullptr;
    int determination_element = rand()%100;

    if(determination_element < CAP_OBSTACLE) {
        int determination_obstacle = rand()%100;
        if(determination_obstacle < CAP_BASE) {
            nouvelElement = new Obstacle(xCourant, HAUTEUR_SOL-TAILLE_ELEMENTS, TAILLE_ELEMENTS, TAILLE_ELEMENTS, -_vitesseJeu, 0, OBSTACLE_BASE);
        }
        else if(determination_obstacle < CAP_AERIEN) {
            nouvelElement = new Obstacle(xCourant, HAUTEUR_ELEMENTS_AERIENS, TAILLE_ELEMENTS, TAILLE_ELEMENTS, -_vitesseJeu, 0, OBSTACLE_AIR);
        }
        else {
            nouvelElement = new Obstacle(xCourant, HAUTEUR_SOL-2*TAILLE_ELEMENTS, 2*TAILLE_ELEMENTS, 2*TAILLE_ELEMENTS, -_vitesseJeu, 0, OBSTACLE_GRAND);
        }
    }
    else {
        int determination_bonus = rand()%100;

        if(determination_bonus < CAP_PIECE) {
            nouvelElement = new Bonus(xCourant,HAUTEUR_ELEMENTS_AERIENS, TAILLE_ELEMENTS, _vitesseJeu, PIECE);
        }
        else {
            nouvelElement = new Bonus(xCourant,HAUTEUR_ELEMENTS_AERIENS, TAILLE_ELEMENTS, _vitesseJeu, MEDIKIT);
        }
    }
    if(nouvelElement != nullptr)
        _elements.insert(nouvelElement);
    if(VERBOSE)
        nouvelElement->verbose();
}
```

Cet algorithme est appelé par l'algorithme précédent. Il va créer un MovableElement vide et calculer un premier nombre aléatoire compris entre 0 et 100. Selon la plage de valeurs dans laquelle se situe ce nombre, l'élément sera soit un obstacle soit un bonus.

Quand cette étape est passée, un second nombre aléatoire compris entre 0 et 100 est calculé, et à partir de ce nombre l'algorithme va déterminer le type de l'obstacle ou du bonus. Selon les conditions atteintes, l'élément sera construit à partir du type calculé et de la coordonnée X passée en paramètre de la méthode. L'élément est ensuite inséré dans la collection d'éléments.

La méthode "verbose" permet, si le booléen "VERBOSE" est actif dans le modèle, d'afficher dans le terminal les informations de l'élément nouvellement créé.

5. Difficultés rencontrées

Durant la réalisation du projet, nous avons surtout rencontré quelques problèmes mineurs que nous avons contourné sans perdre trop de temps.

La plus grande difficulté que nous avons eu découle d'un parti pris adopté pendant la conception : nous voulions regrouper les éléments en paquets, les "chunks". Ce choix nous a fait poser de nombreux problèmes lors de l'affichage, puisque la Vue devait passer par une classe supplémentaire pour obtenir les informations dont elle avait besoin. De plus, nous exploitions mal les possibilités offertes par cette techniques, le meilleur exemple étant que la taille des chunks et le nombre d'éléments les composants étaient fixes. Par soucis de simplicité, nous avons abandonné cette méthode pour revenir à une génération d'éléments aléatoires indépendants.

6. Conclusion

La réalisation du projet était intense et demandait un réel investissement personnel à long terme. Il nous a fallu travailler régulièrement et efficacement pour parvenir à nos objectifs.

Pour ce qui est du résultat, nous sommes plutôt satisfaits du résultat. Malgré un début de réalisation plus tardif que la majeure partie des autres équipes, nous sommes arrivés à atteindre nos objectifs et à fournir un programme conforme aux fonctionnalités minimales demandées.

Le sujet du projet est très intéressant, puisqu'il requiert d'exploiter au maximum nos connaissances en C++ tout en nous faisant travailler avec une librairie externe inconnue, la SFML, que nous avons du apprendre sur le tas.