RAPPORT DE PROJET

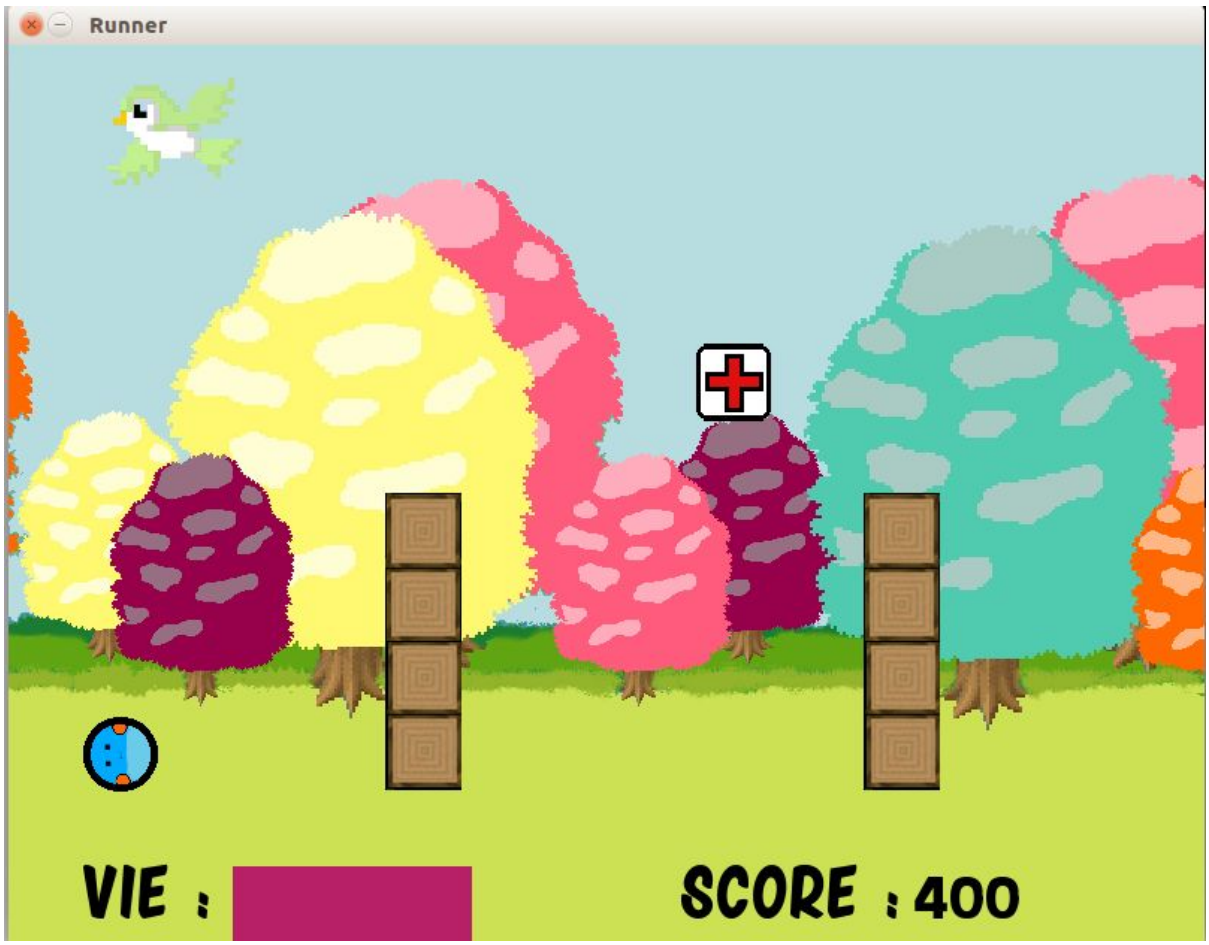
-

RUNNER

Cécile MACAIRE & Claire MARCINIAK

Présentation

Notre projet est un jeu de type Runner.
Le joueur est une balle pouvant se déplacer de droite à gauche, qui peut aussi sauter et dont le but est d'éviter des obstacles, et de récupérer des bonus. On utilise les touches directionnelles pour pouvoir déplacer la balle.
Si la barre de vie du joueur est vide, le jeu est perdu.



Introduction

Ce projet a été réalisé en binôme dans le cadre des cours de POO et de COO (programmation orientée objet et conception orientée objet). Nous avons travaillé à l'aide du logiciel QT Creator pour la partie codage et GIMP pour créer les sprites qui constituent le jeu. Nous avons utilisé la bibliothèque externe SFML (Simple and Fast Multimedia Library) et la STL (Standard Template Library). Nous allons vous expliquer notre projet avec les objectifs qui étaient demandés, son architecture, les classes présentes, et quelques algorithmes qui nous semblent pertinent de détailler.

Sommaire

I. Objectifs

1. Les fonctionnalités minimales

a. Écran d'introduction du jeu
b. Écran de menu permettant via des boutons :
    - de lancer le jeu
    - de quitter le jeu
c. Écran de jeu permettant d'afficher :
    - un arrière plan qui défile.
    - la balle, des obstacles à éviter, des points à ramasser.
    - le score et le niveau de vie du joueur.
d. Pendant le déroulement du jeu :
    - la balle peut se déplacer sur un axe des abscisses et sauter pour éviter les obstacles.
    - arrivée aléatoire des points et des obstacles.
    - 3 sortes d'obstacles.
    - gestion du score.
    - gestion du niveau de vie.
e. Un écran de transition lorsque la partie est terminée

2. Les fonctionnalités supplémentaires

1. Gestion de bonus pour le joueur (vie, invincibilité, capacité de voler, ralentissement du jeu, etc...).
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 :
    - Régler la difficulté du jeu (vitesse du jeu, nombre d'obstacles à éviter).
    - 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).

3. Fonctionnalités bonus

1. Ajout d'argent, sauvegarde et chargement de l'argent gagné lors des partie 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), ...

II. Les tâches

1. La répartition des tâches

Pour réaliser ces fonctionnalités, nous nous sommes réparties les tâches de cette façon:

Claire	Cécile
● gestion du score. ● gestion de la barre de vie. ● affichage du score et de la barre de vie. ● réalisation des méthodes gérant la collision entre la balle et les obstacles. ● réalisation des sprites du jeu.	● saut de la balle. ● affichage de l'écran. d'introduction, du menu et de l'écran de fin. ● gestion des obstacles et de leur affichage dans le jeu. ● gestion de la musique. ● gestion des bonus et de leur affichage dans le jeu.

Les ateliers du jeu ont été réalisé en commun lors des cours de suivi de projet. Cela correspond à l'affichage du Sliding Background, à la gestion du mouvement de la balle, à la classe AnimatedGraphicElement pour animer les objets graphiques et à l'affichage de texte dans le jeu.

2. Les tâches réalisées

A l'issu de ce projet, nous avons réussi à réaliser toutes les fonctionnalités minimales demandées. Nous avons donc pu nous lancer dans la réalisation des fonctionnalités supplémentaires.
Nous avons implémenté deux types de bonus:

- des pièces qui permettent d'augmenter le score de 100.
- des bonus vies qui permettent d'ajouter à la barre de vie 50pv.

Également, la difficulté du jeu augmente avec le temps. En effet, le temps qui gère l'affichage des obstacles diminue petit à petit.
Nous avons réussi à changer de fond défilant en fonction du temps.
En effet, si le temps est inférieur ou égal à 20 secondes, le joueur voit apparaître le fond numéro 1, qui se caractérise par un fond plus sombre. Puis, dépassé ce temps, le fond change en des couleurs plus claires.

Le jeu se compose d'une musique qui se lance tout au long du jeu grâce à l'utilisation de la bibliothèque sonore de la SFML.
Nous avons implémenter les fonctions permettant d'enregistrer les meilleurs scores dans un fichier texte. Malheureusement, les scores s'enregistrent mais le fichier se remplit d'une multitude de valeurs dont on ne sait même pas à quoi elles correspondent.
La dernière fonctionnalité supplémentaires que nous avons implémenté est l'affichage des règles du jeu à partir du bouton RULES du menu.

Nous n'avons pas réalisé les fonctionnalités bonus.

III. Architecture du projet

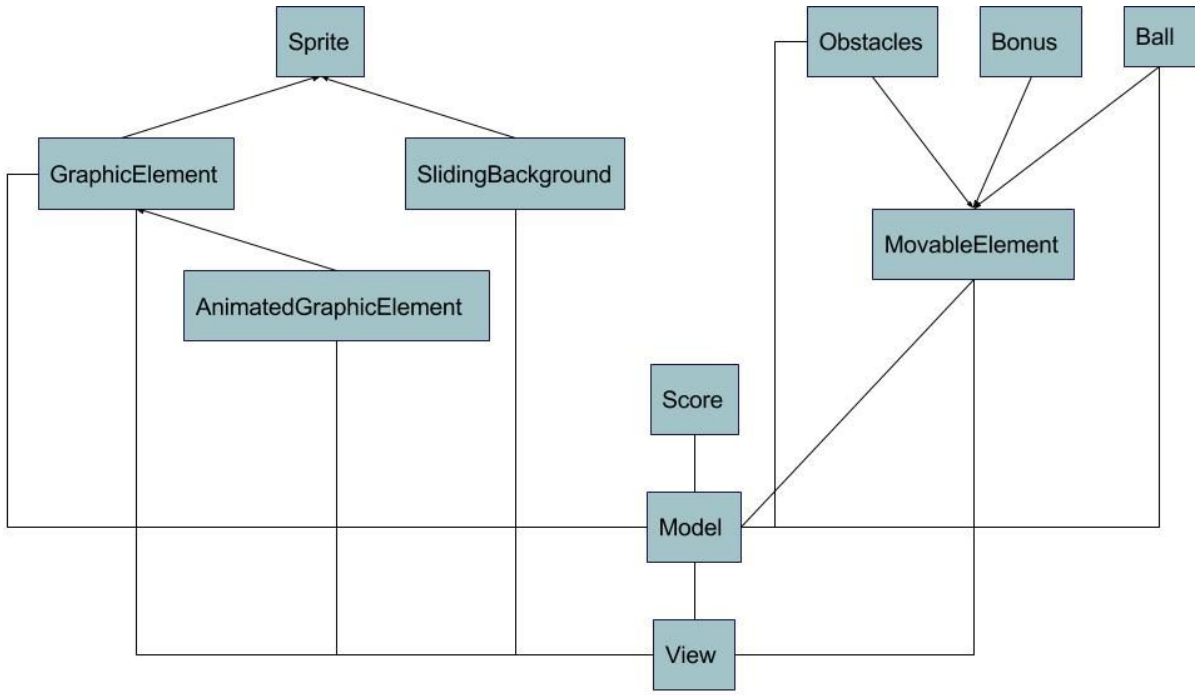


Diagramme très simplifié

Voici un diagramme très simplifié de l'architecture des classes du Runner. L'héritage est représenté par une flèche. Les associations entre classe sont représentées avec des traits.

IV. Les Classes

Le projet se décompose en 10 classes.

1. **GraphicElement**

La classe GraphicElement hérite de la classe fournie par la SFML, Sprite. Cela permet de créer un élément graphique qui a une texture et des coordonnées en x, y, w et h. Cette classe comprend une fonction dessin, pour dessiner l'élément graphique dans la View, une fonction setPosition pour définir la position du sprite en x et en y, une fonction updatePosition qui met à jour la position du sprite et enfin, une fonction getPosition qui permet de retourner la position du sprite.

2. **AnimatedGraphicElement**

La classe AnimatedGraphicElement permet d'animer un élément graphique. Dans le jeu, nous avons juste décider d'animer la balle. Cette classe hérite de la classe GraphicElement. Elle se compose d'une fonction qui permet de dessiner l'élément animé du jeu en fonction du temps.

3. **Ball**

La classe Ball hérite de la classe MovableElement. Cette classe se compose uniquement d'une fonction move() qui permet à la balle de ne pas sortir de l'écran.

4. **Bonus / Obstacles**

La classe Bonus et la classe Obstacles sont identiques mais j'ai préféré en créer deux distinctes car un bonus n'est pas un obstacle.
Ces deux classes possèdent une fonction getType() qui retourne le type de l'obstacle ou du bonus. C'est une fonction virtuelle constante.
Ces deux classes héritent de la classe MovableElement, car les bonus et les obstacles sont des éléments mobiles.

5. **Score**

La classe Score gère le score. Nous retrouvons dans cette classe deux fonctions getScore() et getVie() qui retourne le score, et la vie (barre de vie du joueur).

Deux autres fonctions setScore() et setVie() définissent le score et la vie. Cette classe possèdent également les fonctions permettant d'enregistrer les meilleurs scores dans un fichier texte.

6. SlidingBackground

La classe SlidingBackground gère l'affichage des fonds défilant. Cette classe hérite de la classe Sprite fournie par la SFML.
Cette classe se décompose en 3 fonctions: une première fonction permet de dessiner les fonds défilants. Une deuxième permet de donner un mouvement au fond. Et une dernière fonction permet de définir la vitesse de défilement.

7. Model

La partie Model du jeu gère tous ce qui ne concerne pas l'affichage. Cela gère les données de l'état du jeu (les bonus, les obstacles, le score, …).
Voici quelques éléments clés qui constituent le model:

- un constructeur qui instancie la balle et le score.
- un destructeur qui détruit la balle et le score créés précédemment.
- nextStep() qui calcule la prochaine étape. Cette méthode gère le moment ou les obstacles et les bonus vont s'afficher grâce au temps. Elle gère aussi le déplacement des éléments, et le moment ou il y a une collision entre un élément et un obstacle ou un bonus. Enfin, elle gère aussi le saut de la balle.
- l'ensemble des méthodes qui permettent de retourner la position de la balle, sa dimension.
- moveBall() : qui se déplace de 5 ou de -5 en abscisse selon le choix de la flèche directionnelle.
- un vector de MovableElement getNewMovableElements() qui retourne les nouveaux éléments et getMovableElements() qui retourne les éléments.
- une méthode addElement() qui prend en paramètre une chaîne de caractère. Cette méthode permet, selon le type de l'élément de créer un nouvel obstacle ou un nouveau bonus, et de les ajouter au vector des nouveaux éléments et des éléments. Cette fonction définie l'ensemble des bonus et des obstacles que nous avons implémenté dans le jeu.
- jumpBall() permet de donner à la balle une impulsion de -60 en ordonnée.
- collision() gère la collision entre deux movableElement.
- resultatCollision() gère le score en fonction du type de l'obstacle (le score diminuera ainsi que la barre de vie).
- resultatCollisionBonus() gère le score en fonction du type de bonus (le score ou la barre de vie augmenteront).
- writeScore() permet d'écrire le score, de le mettre en string au lieu d'un int.
- les méthodes qui retourne la vie et le score (setScore() et setVie()).

- bestScores() gère les meilleurs scores.

8. View

La partie view gère tout ce qui concerne l'affichage. On affiche sur la fenêtre en fonction du model, duquel on récupère toutes les données du jeu.

- Un constructeur qui instancie tous les objets en leurs rajoutant leurs sprites et leurs positions.
- Un destructeur qui permet de faire des delete pour éviter toutes fuites mémoires.
- Des fonctions draw qui permettent de dessiner à l'écran l'étape que l'on veut :
  + drawMenu() qui permet de dessiner l'écran de menu.
  + drawGame() qui permet de dessiner l'écran de jeu.
  + drawEnd() qui permet de dessiner l'écran de fin de jeu (game over).
  + drawRules() qui permet d'afficher les règles.
- La fonction treatEvents() qui gère la gestion des touches et du clic de la souris: en fonction de la touche appuyée, on fait bouger la balle à droite ou à gauche, on la fait sauter.
- La fonction synchronize() qui gère plusieurs choses : en fonction d'un timer, on ajoute du score, ou on fait apparaître des obstacles à l'écran. Elle permet aussi de se débarrasser des obstacles qui on eu une collision avec la balle, ou ceux qui sortent de l'écran. Enfin, elle permet aussi d'afficher le GAME OVER si la vie est inférieure ou égale à 0.

V. Algorithmiques

Voici quelques parties de code que nous avons décidé d'expliquer plus précisément
.

```
std::vector< MovableElement *> i = _model->getMovableElements();
for (auto element: i)
{
    //Suivant le type de l'obstacle, on ajoute la bonne texture avec les bonnes coordonées.
        if(element->getType() == 0){
            GraphicElement *obstacle = new GraphicElement(_obstacleTexture, 100, 100, 100, 50);
            _elementToGraphicElement[element] = obstacle;
        }
        else if (element->getType() == 1){
            GraphicElement *obstacle1 = new GraphicElement(_ennemies, 100, 400, 100, 150);
            _elementToGraphicElement[element] = obstacle1;
        }
```

Pour chaque movableElement, si le type de l'élément est 0, on va donc instancier un nouveau graphicElement, puis dans la map _elementToGraphicElement, à l'entrée de l'élément, on associe un GraphicElement.

```
for(auto elm :_elements)
{
    elm->move();
    if(elm->getX()+elm->getW() < 0) {
        _elementsTruck.push_back(elm);   //
    }
    else if(collision(_ball, elm) == true)
    {
        _elementsTruck.push_back(elm);
        resultatCollision(elm, C);
        resultatCollisionBonus(elm, B);
    }

}
```

Pour chaque éléments du vector, on donne un déplacement. Puis, si l'élément sors de la fenêtre à gauche, on ajoute l'élément dans le vector _elementsTruck (qui sera delete). Si le jeu détecte une collision entre la balle et l'élément, alors on ajoute l'élément dans le vector _elementsTruck(qui sera delete) et on appelle les fonctions résultatCollision(résultat d'une collision si c'est un obstacle) et résultatBonus(résultat d'une collision si c'est un bonus).

```
timeS = clock.getElapsedTime();
if(timeS.asMilliseconds()%50 == 0 && timeS.asMilliseconds() > 3000 && play != 2)
{
    _model->setScore(_model->getScore()+100);
}
```

Cette partie du code permet d'affecter à timeS la valeur du temps passé. Toutes les 50 millisecondes (c'est-à-dire 0.05 secondes) et si le temps est supérieur à 3000 millisecondes (c'est-à-dire 3 secondes) et que play n'est pas égal à 2 (c'est-à-dire que l'on est pas dans l'écran de menu), on rajoute au score +100.

VI. Conclusion

1. Améliorations possibles ?

Nous nous sommes rendues compte que les évènements liés à la souris sont valables tout le long du jeu. De ce fait, lorsque la partie se lance, si l'on clique à l'emplacement des boutons du menu, les évènements liés aux boutons vont se lancer (quitter le jeu, afficher les règles, …). Également, nous aurions aimé améliorer l'enregistrement des meilleurs scores et gérer leur affichage. De plus, nous aurions aimé implémenter plus de bonus, par exemple, un bonus qui permet de donner à la balle de l'invincibilité.
Il aurait aussi été intéressant de rajouter des bruitages, par exemple, lorsque la balle saute, lorsque la balle attrape des bonus, lorsqu'il y a une collision entre la balle et les obstacles,...
Enfin, nous aurions pu améliorer la qualité de certaines fonctions, mieux les implémenter.

2. Quelques points intéressants !

Nous avons réussi à implémenter un Runner qui possède les fonctionnalités principales du jeu. L'apparition des obstacles et des bonus ainsi que le mouvement de la balle (gauche, droite, saut) se font de manière fluide.
Nous avons réussi à implémenter un menu et un écran de fin.
Les heures de suivi de projet ont été un moyen de mettre en commun et de régler les problèmes que l'on a rencontré tout au long de l'implémentation (problèmes de fuites mémoires, d'affichage de certains sprites,...).

3. Gestion du projet et des tâches.

La gestion de projet s'est plutôt bien passé pour notre binôme. Nous n'avons pas utilisé d'outils tels que git, mais nous nous envoyions les dernières versions à chaque fois, et il n'y a eu aucuns problèmes liés à cette méthode de partage de fichiers. Les taches ont bien été réparties, nous avons pu chacune travailler sur une partie du jeu pendant que l'autre travaillait sur une autre partie, ce qui a permis un travail efficace.