Projet de C++ : Paradise Paper

Geng Ren
Lancelot Satge

Main 4

POLYTECH
PARIS-UPMC

Sommaire

I- Présentation du jeu

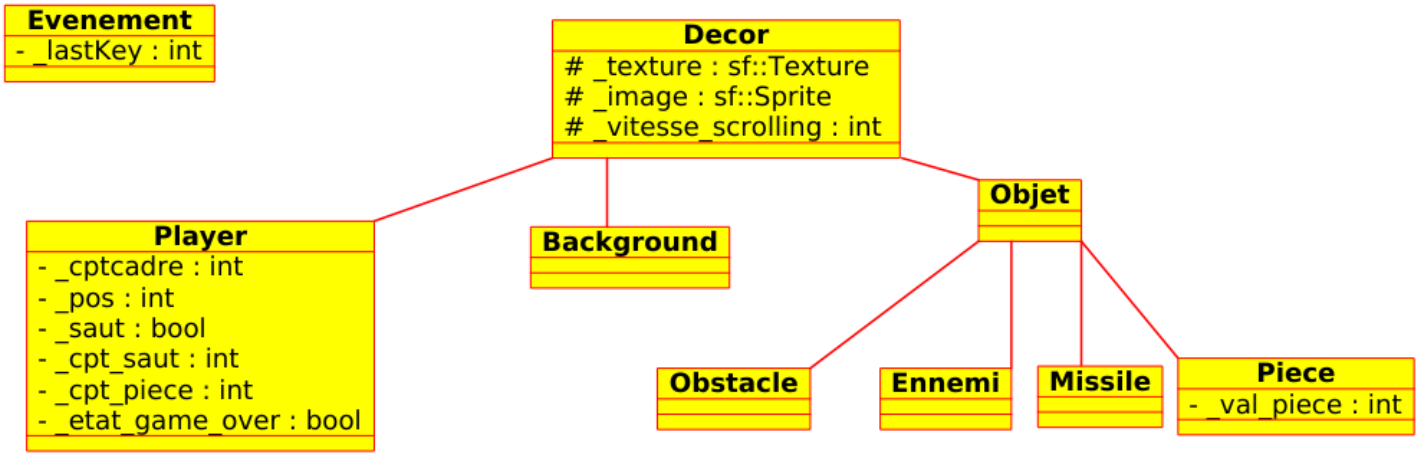
L'objectif initial était de s'inspirer du jeu Temple Run afin de recréer un jeu similaire adapter à notre convenance. Le résultat est proche dans certaine mesure mais tout de même assez personnel. L'univers Star Wars nous a semblé adapté pour raconter une histoire qui parlerai au plus grand nombre, notamment avec la sortie du huitième opus en décembre 2017.

Le joueur incarne un stormtrooper qui doit fuir les jedis envoyés par le fisc intergalactique. Le but du jeu est donc de faire le score le plus important possible. Pour cela il faut ramasser des pièces et parcourir la plus grande distance possible en évitant pièges et ennemis. **Lorsque vous avez récupérés 20 pièces, un vaisseau apparaît le décor se transforme, on vous laisse découvrir ce qu'il s'y passe !**

II- Les classes

1. Diagramme de classes UML

Ceci est le diagramme UML sans les méthodes, voir l'image du diagramme UML complet fourni pour plus d'informations. Le jeu ayant évolué après la rédaction du rapport il se peut qu'il y ait des différences minimes avec le code réelle, cependant le corps du jeu est ici bien défini.



2. Description des classes

Nous avons 9 classes, avec trois niveaux de hiérarchies.

- **Evenement :** elle nous permet de gérer la position des différents décors et objets mais également d'influer sur le jeu en appuyant sur les touches. L'attribut _lastKey nous sert à connaître la dernière touche que l'on a appuyé. Le conteneur map sert à stocker les différentes musics jouées lors des phases du jeu.

- **Decor :** elle regroupe tout ce qui va être affiché à l'écran. Dans SFML pour afficher une image il faut déclarer une texture et un sprite, d'où nos deux attributs. De plus, pour faire « vivre » l'écran il faut faire bouger les décors tous ensemble, c'est ici que la vitesse de scrolling intervient.
De la classe Décor hérite trois autres classes :

- **Player :** elle permet de gérer le personnage du joueur. Ce personnage est statique il ne peut bouger que sur l'axe horizontal, il ne peut pas avancer c'est le décor qui bouge. L'attribut _cptcadre permet de gérer le mouvement du cadre que l'on affiche au sein de l'image du personnage, ce qui permet de lui donner l'impression de bouger. _pos permet de savoir si le joueur est à gauche, droite ou milieu. _saut de savoir s'il en état de saut, et cpt_saut permet de savoir où il en est dans le saut pour gérer la différence de taille et la rendre fluide au possible. Et cpt_piece et etat_game_over de connaître le nombre de pièce et de savoir si la partie est fini ou pas.

- **Background :** elle est la classe qui gère l'affichage du fond qui est en scrolling permanent et infini.

- **Objet :** elle regroupe toutes les classes qui sont propre aux entités pouvant influer le cours de la partie et qui se superpose au décor :

- **Obstacle :** elle représente des objets qui sont fixe mais qui bouge avec le scrolling de l'écran et qui font perdre le joueur s'il les touche ;

- **Ennemi :** elle est similaire à la classe obstacle seulement les ennemis peuvent avancer en plus de bouger avec le scrolling et ils évitent les obstacles en se décalant.

- **Missile :** elle regroupe les projectiles qui partent du joueur et qui peuvent tuer les Ennemis.

- **Pièce :** elle représente tout simplement les pièces que doit prendre le joueur.

Toutes les variables importantes (délais apparitions, score, liste objet, etc) sont définies dans variable_global.hh et initialisées avec la méthode Init() de la classe Evenement.

III- Particularités du code

1. Gestion de l'apparition des objets

Le délais d'apparition et l'emplacement des objets tels que les obstacles, les ennemis et les pièces sont générés aléatoirement.

Pour générer des objets à des intervalles de temps différents, nous avons eu recourt à la classe sf::Clock de SFML. En effet, cette classe démarre un compteur de temps lorsqu'elle est crée et se remet à 0 quand on appelle la méthode restart. Ainsi, nous avons crée 4 clocks différentes pour chacun des 4 classes filles de l'Objet. Puis un nombre aléatoire (compris entre les bornes inférieures et supérieures de l'objet en question) est généré, on l'appelle le délais d'apparition. Cela correspond au temps d'attente entre deux apparitions d'un même objet. Nous comparons alors ce délais à l'horloge de l'objet, et nous ajoutons l'objet dans sa liste lorsque la clock dépasse ce délais.
Durant la partie, le temps d'apparitions des objets sera diminué en fonction du score avec la méthode gestion_difficulte de la classe Evenement. Plus le score augmente plus les ennemis et obstacles apparaissent souvent.

Pour l'apparition spatiale, nous générons un nombre en 0 et 2 qui correspondra à la position de l'objet, 0 à gauche, 1 au milieu et 2 à droite.

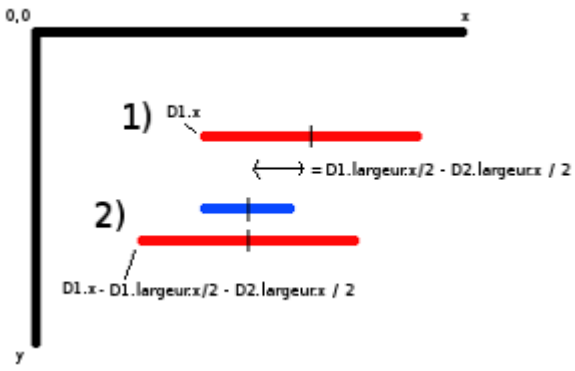
2. Centrer les objets

Centrer les objets par rapport au joueur peut paraître simple à première vue, cependant ce n'est si facile à gérer. Nous avons donc créé une méthode Centrer() dans la classe Decor qui permet de centrer un décor D1 par rapport un autre décor D2. La difficulté provient du fait que l'origine des coordonnées x et y se trouve en haut à gauche de l'écran. Si l'image de D1 que l'on veut centrer est plus large en x que l'image de D2, alors sa nouvelle position en x sera :

New_D1.x = D1.x -(D1.largeur.x / 2 – D2.largeur.x/2)

Et si la largeur de D1 est inférieur à largeur de D2 :

New_D1.x = D1.x + (D1.largeur.x / 2 – D2.largeur.x/2)



Centrer les objets était très important pour créer une harmonie dans le jeu.

3. Changement de décor

Changer de décor est simple et apporte une réelle plus-value au jeu. Nous modifions simplement les images des décors, et les bruits associés aux images. Ce qui prend le plus de temps est de trouver les images et les sons pouvant représenter l'ambiance et l'histoire que nous voulons raconter. La bibliothèque SFML étant bien écrite, toute la gestion de modification des tailles et des coordonnées est prise en charge.

IV- Procédure d'installation

Afin de pouvoir compiler notre programme, assurer vous que la bibliothèque SFML soit bien installée sur votre machine. Puis lancé le makefile, puis un programme nommé main sera créé ce qui correspond à notre jeu. La version de SFML que nous avons utilisé est 2.3.2, une version antérieur peut avoir des fuites mémoires au niveau de la création de la fenêtre et peut parfois provoquer des erreurs Valgrind, mais cela n'altère en aucun cas l'expérience du jeu. En utilisant la version 3.11.0 de Valgrind et la version 5.4.0 de g++, aucune fuites mémoire et erreurs sont détectés.

V- Améliorations Futures

Avec plus de temps les améliorations que nous aurions voulu apporter aurait été:

- Ajouter des bonus qui permettrait au joueur d'avoir des états temporaires et qui lui permettrait des actions impossible sans, ou d'augmenter la valeur des pièces un instant.

- Ajouter des ennemis pouvant réagir aux lasers, les renvoyer par exemple comme le font les jedis normalement.

- Pouvoir sélectionner son personnage au début de la partie.

Malheureusement le temps nous manque pour réaliser toutes ces fonctionnalités. De plus, tous ces ajouts représentent uniquement du front et non pas du back, l'objectif pour nous était de réaliser la structure de déroulement du jeu en priorité avec les particularités de chacune en rendant l'univers plaisant.