Application informatique

Another Side-scrolling Platformer Engine

Réalisé par : Séverine Delaplace
Amaury Fauvel
Quentin Godart
Aurélien Villette

Table des matières

1 Introduction

1.1 Présentation

Another Side-scrolling Platformer Engine (ASPE) est la démonstration d'un jeu-vidéo en 2 dimensions à défilement horizontal (la caméra se déplace latéralement avec le joueur). Le joueur s'y déplace librement avec les flèches du clavier et la barre d'espace.

[PLUS DE DETAIL SUR LES FONCTIONNALITE]

Ce type de jeu figure parmi les plus représenté dans l'histoire du média interactif, avec de nombreux titres marquant comme Super Mario, Sonic the Hedgehog ou Rayman. Le but d'un tel jeu est de parcourir des niveaux (souvent de gauche à droite) parsemés d'obstacle et ennemis.

[IMAGE SUPER MARIO]

1.2 Objectifs du projet

Le but du projet est de réaliser un moteur graphique adapté aux jeux de plateforme. Il doit être doté d'une architecture robuste et extensible qui pourrait être utilisée dans le cadre d'un jeu complet. Le nombre de fonctionnalités est minimal car l'objectif est de fournir la base d'un jeu, et non un jeu complet.

2 Implémentation

2.1 Choix de programmation et workflow

Langage

Le langage utilisé pour l'intégralité du projet est le C++. Le C++est un langage orienté objet influencé par le C. Un de ses principaux atouts est la gestion fine des ressources et ses performances. C'est un langage de choix pour toute application en temps réel employant des calculs graphiqueset est couramment utilisé par les studios de développement de jeux. C'est un langage que nous n'avons pas appris au cours de notre scolarité, c'était donc l'occasion pour nous de nous familiariser avec un nouveau langage.

Bibliothèque graphique

La bibliothèque graphique employée est le Simple and Fast Multimedia Library (SFML). C'est une bibliothèque assez bas niveau, elle nous permet de gérer efficacement les ressources (images affichées). Elle a l'avantage d'être conçue pour les langages orientés objet, c'est pourquoi nous l'avons préféré à la Simple DirectMedia Layer (SDL).

La SFMl est séparée en différents modules :

— Window : gère l'ouverture de la fenêtre basée sur OpenGL. C'est dans ce contexte que se déroule le jeu. Sert aussi à récupérer les inputs du clavier.
— Graphics : implémente le chargement de textures, l'affichage de sprites et la gestion de la caméra.
— System : pour les configurations diveres (taux de rafraîchissement notamment)
— Sound : gère les sons et pistes audio (malheureusement non utilisé ici)

— Network : pour les communications en réseau (non utilisé ici)

Méthodes de travail

Nous avons choisi l'environnement de développement CLion (de la suite JetBrains) pour ses fonctionnalités utiles comme l'intégration de Git, notre gestionnaire de versions.

Comme outil de communication, nous avons utilisé Slack. Slack est une application web de chat adapté au monde du travail grâce à son système de salon et à la centralisation d'autre application comme Trello.

Trello se présente sous la forme d'un tableau où l'on affiche les différentes tâches à faire, en cours et terminées, assignables aux membres du groupe. Finalement, nous avons rarement utilisé Trello car nous n'étions que 4 et nous côtoyons tous les jours.

2.2 Architecture

La qualité de l'architecture était notre principale préoccupation. Elle permet d'avoir une base de jeu largement extensible, car la quasi-totalité des fonctionnalités d'un jeu complet ne concernent que des ajouts au modèle du jeu.

Nous avons choisi une architecture Modèle-Vue-Contrôleur. Elle permet une grande maintenabilité et une clarté du code. De plus elle connaît son lot de bonne pratique qui assure la robustesse du programme.

[SCHEMA DE L'ARCHITECTURE]

Le jeu étant en temps réel, une boucle principale tourne jusqu'à la fermeture de la fenêtre. Dans cette boucle, la classe principale `Game` appelle tour à tour le `EventHandler` qui récupère les inputs clavier, le `GameModel` qui met à jour tout le modèle à partir des inputs récupérés, et le `GraphicRenderer` qui affiche le niveau et les entités qui y évoluent.

Contrôleur

La partie contrôle est la plus simple : le `EventHandler` récupère l'état de chaque touches associée à un contrôle avec la méthode SFML :

```
sf::KeyBoard::isKeyPressed()
```

L'association entre les contrôles (directions, saut) et les touches du clavier (espace, flèches) est faite par l'espace de noms `Controls`.

> Note : en C++, pour `Nom::fonction()`, `Nom` est un espace de nom pour une collection de fonctions utilitaires.
> `sf` est l'espace de nom de la SFML.

3 Répartition des tâches

Une fois notre sujet validé nous avons commencé par effectuer une petite réunion pour discuter de l'approche de ce projet. Nous avions décidé d'utiliser Git dès le départ avant de voir que chaque groupe de projet allait avoir son propre dépôt Git. En outre, pour faciliter la communication entre les différents membre du groupe nous avons décidé d'utiliser également Slack.
Lorsque nous avons commencé à coder nous avions besoin d'une base à partir de laquelle nous allions développer nos classes c'est donc Amaury qui s'est chargé de coder cette base. Ensuite nous nous sommes répartis les tâches en fonction des désirs de chacun et des besoins du projet.

4 Difficultés rencontrées

En faisant le choix de développer ce projet en C++ avec la bibliothèque SFML nous nous sommes imposé une première difficulté, celle d'apprendre le langage et son fonctionnement avec la bibliothèue graphique.
Une seconde difficulté a été d'apprendre à bien utiliser Git afin de limiter les problèmes liés à son utilisation au cours du développement.
La principale difficulté a été d'écrire le code car puisque nous partions de rien nous ne savions pas vraiment ce dont nous allions avoir besoin donc avant de réellement coder le jeu nous avons un peu tâtonné. Nous nous sommes renseignés sur internet pour savoir un peu quelles méthodes nous pouvions utiliser pour réaliser ce projet et nous avons donc opté pour plusieurs solutions techniques dont certaines ont évolué au cours du projet comme par exemple la méthode utilisée pour afficher un niveau. Au départ nous voulions utiliser des chunks (tronçons de blocs générés lorsque le joueur se trouve à une certaine distance) mais finalement nous avons choisi d'opter pour une autre solution puisque notre niveau est connu à l'avance donc nul besoin de le charger/décharger au fur et à mesure de la progression du joueur.
Nous avons également dû gérer le temps en raison des nombreux projets à réaliser ce semestre.

5 Bugs

5.1 Les bugs connus

5.2 Les solutions qui auraient pu être apportées

6 Pistes d'amélioration

A ce stade le projet ressemble plus à une ébauche d'un jeu qui peut être améliorée qu'à un vrai jeu abouti. Différentes pistes d'améliorations sont d'ailleurs envisageables :

—
—
—

—
—
—
—
—