Programmation concurrente et interfaces interactives

Rapport

BORISSOV PIOTR | FERREIRA ALVES PATRICIA

Introduction : présentation de l'objectif du projet

Le but de ce projet est de réaliser un jeu vidéo où l'utilisateur contrôle un vaisseau pouvant se déplacer dans les quatre directions. Le jeu est une course où le joueur doit atteindre les points de contrôles apparaissant sur la piste en un certain temps, tout en évitant les obstacles et les adversaires générés par le programme.

Le programme doit donc générer une piste, ainsi que des obstacles et des adversaires aléatoirement. Il crée également un vaisseau, dont les mouvements sont contrôlables par le joueur et dont la vitesse augmente lorsqu'il se rapproche de la piste. On doit avoir une détection de collision qui fait perdre la partie si le vaisseau touche un obstacle ou un adversaire, et le joueur perd aussi si le temps imparti, qui augmente lorsqu'il dépasse un point de contrôle, est écoulé.

L'interface doit afficher la piste, avec un décors au dessus, le véhicule, les obstacles et les adversaires, ainsi que le score du joueur, la vitesse du véhicule et le temps restant avant la fin de la partie et les différents éléments devront être animés.

Nous avons réalisé ce projet en Java, en utilisant l'API Swing afin de créer l'interface interactive

Analyse globale : comment est structuré le projet et quelles sont les principales fonctionnalités

Nous avons structuré ce projet en quatre packages différents : le premier contenant une classe Main qui crée les objets nécessaires et qui lance le jeu, et les trois autres correspondant aux trois parties du modèles MVC : le modèle *model*, la vue *view* et le contrôleur *control*.

Au lancement du programme, celui-ci crée une piste et un véhicule, et les affiche sur l'interface, avec un décors au dessus de l'horizon, qui est la limite de la piste. Le véhicule est animé : il clignote de couleurs différentes et il est accompagné de petites lumières vertes qui tournent automatiquement.

On détecte si l'utilisateur appuie sur les flèches du clavier avec un keyListener, ce qui permet de lancer le jeu et de déterminer si le véhicule bouge ainsi que le sens dans lequel il doit se déplacer. Un autre thread utilise ensuite ces valeurs pour modifier le véhicule est la piste : on déplace le véhicule dans la direction dans laquelle il doit avancer et on déplace les points de la piste vers la gauche, si le véhicule va à droite, à droite, si le véhicule va à gauche. Si le véhicule descend, on éloigne les points de la piste pour donner une impression de zoom, et si le véhicule monte, on les rapproche.

Un thread différent est utilisé pour donner l'impression que le vaisseau avance. Il diminue la vitesse si le vaisseau s'éloigne de la piste et l'augmente si il s'en rapproche, et il appelle des fonctions permettant d'augmenter la hauteur des points de la piste (ce qui, sur l'affichage, donne l'impression qu'ils descendent de l'écran et partent vers l'arrière du vaisseau). On crée de nouveaux points à l'horizon, si besoin, afin que la piste soit générée à l'infini.

La valeur d'une variable représentant le score est augmentée lors de l'avance. La vitesse indique à quel point la piste et le score sont modifiés.

On a également des points de contrôles créés lorsque le vaisseau avance d'une certaine distance. Ils sont représentés par une ligne horizontale sur l'affichage et avancent de la même façon que la piste.

Tous les threads modifiant les modèles doivent également notifier la vue, afin qu'elle puisse prendre en compte les changements.

Organisation du travail : comment vous avez organisé le travail dans le temps, qui a fait quoi, etc.

Un diagramme de Gantt est toujours le bienvenu

1.Organisation du travail

Nous nous sommes mis en binôme quelque peu après le début de la séance et Patricia avait déjà implémenté un simple squelette du code basé sur le modèle fait lors du tutoriel Flappy Bird.

Notre premier réflexe fut de créer une repo sur GitHub, cela facilite grandement le travail et fluidifie l'avancée du projet : https://github.com/upsudghosts/MVNI.git.

Dans un second temps nous avons mis sur papier, après avoir relu le sujet, les éventuelles classes dont nous aurions besoin. Plus généralement nous nous sommes mis d'accord sur la direction globale qu'allait prendre notre code afin d'éviter tout soucis d'uniformisation pendant l'avancement du programme.

Enfin la répartition des tâches se fait assez naturellement. Nous suivons l'ordre des choses comme indiqué dans le sujet. Chacun choisit rapidement ce qu'il souhaite développer et sans plus attendre commence à travailler.

Toutefois, nous gardons à l'esprit que c'est un travail commun.

Ainsi, chaque semaine à lieu un débriefing de la part de chacun des membres afin d'expliquer sa partie à l'autre. (si il n'y avait aucune corrélation entre les tâches. Ex: mettre des images pour le background *vs* animer le vaisseau) Aussi nous n'hésitons pas à nous entraider lors d'éventuels blocages.

Le Projet avance ainsi sans encombres et les résultats sont très satisfaisants.

2. Diagramme de Gantt

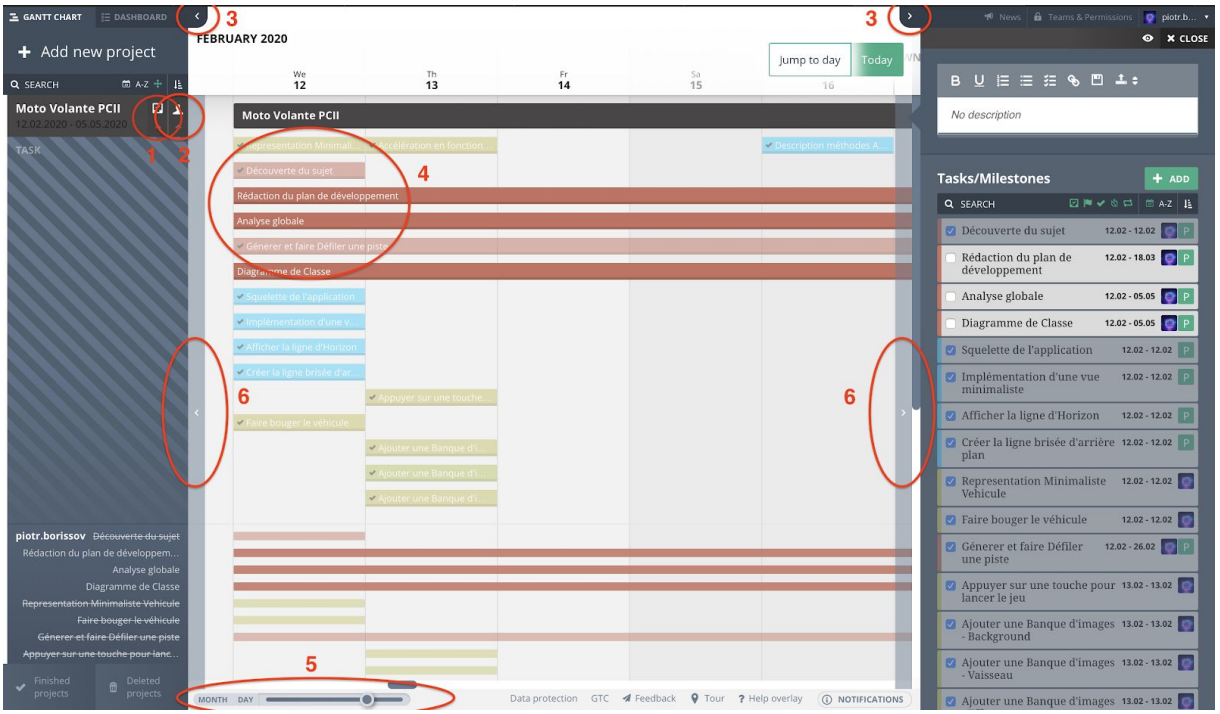
Afin de modéliser le diagramme de Gantt, nous avons choisi d'utiliser un outil en ligne permettant une manipulation rapide, claire et efficace pour notre travail.

Nous allons donc fournir de simples explications en annexe pour faciliter la navigation dans le diagramme ainsi que le lien d'accès.

Voici le lien pour le visualiser :

https://app.agantty.com/#/sharing/35d9e4264b64620b27492a37fb14ab26

ANNEXE



MENU LATÉRAL GAUCHE

1 - Montrer/Masquer le point de vue général des tâches. (Celui dans lequel se trouve la zone 5)

2 - Montrer/Masquer les tâches pour chaque collaborateur.

GENERAL

3 - Masquer les menus latéraux

4 - Appuyer sur une tâche montre ses informations dans le menu latéral droit. SUrvoler au moins 2 secondes avec la souris fera sortir une petite fenêtre pop Up avec les infos de la tâche.

5 - Permet d'ajuster le Zoom sur les jours / Les semaines.

6 - Étends la vue dans le calendrier. On utilise ensuite les barres coulissantes pour faire défiler la timeline.