

Lubin Longuepee, Aidan Barouk

22/04/2024

Projet Programmation objet et

Genie Logiciel

1. **Organisation du travail**

Afin de travailler ce projet, nous avons pendant, près de deux semaines, été pendant plusieurs heures à la bibliothèque. Nous travaillions à deux sur les mêmes énoncés pour pouvoir bien comprendre les enjeux du sujet et les contraintes imposés. Nous avons passé environ la moitié de ce temps pour réaliser la partie obligatoire du projet, et l’autre moitié à réfléchir à des fonctionnalités supplémentaires, puis à les implémenter. Nous avons choisi de faire tout le projet ensemble, et de ne pas se répartir le travail car nous estimions que nous serions tout autant efficace à réaliser ce projet conjointement, puisque cela nous a évité de devoir passer du temps à éssayer de comprendre le code de l’autre.

1. **Difficultés rencontrées**

Un projet comme celui-ci ne se fait pas simplement. Nous avons rencontré de nombreuses erreurs, incompréhension et défiance du code qui nous ont retardé dans l’avancer de notre travail.

Le principal problème rencontré était la compréhension des JPanel. Dans un premier temps, nous avions des erreurs d’affichage. En effet, des boutons qui ne s’affichent pas (*ou mal*), une superposition d’image, un bandit qui restant affiché sur le toit (*alors que dans le code il est descendu dans le train*) ou un Marshall qui reste fixe nous ont fait comprendre que les JPanel n’était pas une classe facile à utiliser. Afin de résoudre nos difficultés nous avons effectué des recherches sur Internet qui ont facilité l’utilisation de la classe. Vous trouverez ici 2 liens que nous avons utilisé au cours de nos travaux : <https://www.geeksforgeeks.org/java-jframe/> et <https://zestedesavoir.com/tutoriels/646/apprenez-a-programmer-en-java/558_java-et-la-programmation-evenementielle/2707_positionner-des-boutons/> .

Nous avons finalement opté pour l’utilisation des GridBadLayout pour afficher l’ensemble des vues. En effet, grâce à ce mode d’affichage, nous pouvons définir des zones grâce aux coordonnées, et superposer ces zones entre elles si nous le souhaitons. Nous avons ainsi 3 zones principales : la zone d’affichage du train avec les wagons, les bandits, le marshall ainsi que les butins, la zone d’affichage des butins récoltés par chaque bandit, et enfin la zone des commandes avec tous les boutons permettant de jouer.

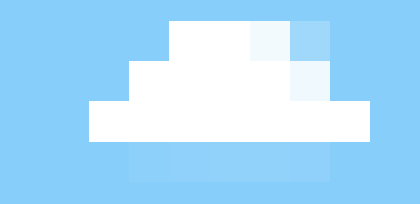
Une des autres difficultés majeures rencontré a été lors de la création de la séparation entre les phases de jeu et d’action. En effet, pour réaliser cette séparation, nous avons choisi d’empecher le joueur d’effectuer des actions qui ne lui sont pas accessible (exemple : un joueur se trouve au début de la phase d’action en haut du dernier wagon, il décide de descendre pour sa première action, alors si il essaye de redescendre à l’action suivante, cette action ne va rien faire parce que le bandit ne peut pas descendre encore.). Nous aurions pu faire le choix de laisser le joueur se tromper et effectuer des actions « dans le vent » mais nous avons choisi de bloquer les actions qui lui sont innaccessibles. Pour cela, nous avons pensé à attribuer un booléen et un entier à chaque bandit (en créant donc respectivement un tableau de booléen et un tableau d’entier), avec le booléen indiquant si le bandit est sur le toit ou non, et l’entier contenant le numéro de son wagon actuel. Cela avait pour objectif de pouvoir avoir accès à la position du bandit, avant d’être dans la phase d’action. Mais le problème rencontré était que nous ne pouvions pas modifier ces variables dans nos méthodes car le compilateur indiquait une erreur à cause d’éventuels conflits. Nous avons donc choisi d’utiliser des AtomicBoolean et AtomicInteger, ce qui permettait de palier à ce problème.

Enfin, nous avons cherché longtemps dans quelle structure nous pouvions stocker les actions des bandits avant de les exécuter. Nous avons d’abord pensé à des tableaux, puis à des listes chainées. Mais en faisant des recherches sur internet, nous avons trouvé le type java dont nous avions besoin et qui était le plus adapté à nos problématiques : le type Queue (<action>). En effet, quand on ajoute un élément dans une Queue, il est directement placé à la fin de celle-ci, tandis que si on essaye d’accéder à un élément d’une Queue, on accède au premier élément (on accède aux éléments dans l’ordre dans lequel on les a placé). Ce type était parfait pour réaliser notre « file d’attente » d’actions.

1. **Partie Bonus**

Au cours de la réalisation de ce projet, la partie bonus a été, de loin, la partie que nous avons préféré travailler. Des multitudes idées fusaient dans la bibliothèque. Cependant, par manque de temps, de moyen ou de difficulté de réalisation, nous n’avons pas pu toutes les écrire. Voici celles que nous avons alors gardé.

* **Modification de l’affichage**. Nous nous sommes rendu compte que modifié l’affichage donne une meilleure expérience de jeu au joueur. C’est pourquoi, nous avons décidé de modifier la couleur du background à l’aide de la fonction setColor(int r, int g, int b) que nous avons placé dans une nouvelle classe (VueColor). De plus, nous avons ajouté un style cartoon avec les cactus et le nuage (cf image) que nous avons nous même réalisé sur Photoshop.



*Cactus réalisé par Aidan*

*Cactus et nuage réalisés par Lubin*

* **Des nouvelles fenêtres et des paramètres au jeu**. Au lancement du programme, nous avons ajouté une nouvelle fenêtre qui explique le but et règle du jeu pour rendre l’expérience plus claire. De même, nous avons implémenté différentes possibilités de jeu avec le choix du nombre de joueur et les niveaux de difficulté (facile, difficile moyen). Ce choix de difficulté change plusieurs paramètres du jeu, comme la nervosité du marshall, le nombre de butins à récolter pour gagner, le nombre de wagons, le nombre de balles… Encore dans une nouvelle fenêtre, l’utilisateur choisi le nombre de joueur (entre 1 et 4) à l’aide d’un Jslider. Une fois le nombre choisi, l’utilisateur à la possibilité de nommer ses joueurs, s’il ne souhaite pas nommer son personnage, un nom par défaut lui sera attribué (« *Joueur n »*). A noter, que si l’utilisateur ne valide pas ses choix, le programme s’arrêta.
* **Un début aléatoire**. Pour que ça ne soit pas toujours le Joueur 1 qui commence et qui puisse gagner plus facilement, nous avons décidé de rendre l’ordre du jeu aléatoire. En effet, à chaque début de partie, la fonction shuffle modifie l’ordre de jeu des joueurs.
* **Un nouveau butin**. En plus, du magot, des bourses et des bijoux, nous avons ajouté un nouveau butin qui est la Montre de Luxe. Celle-ci apparait qu’une fois par partie dans le deuxième wagon (en première classe). La Montre de Luxe a la particularité de se déplacer (*elle possède à peu près la même nervosité que le Marshall*). Il existe 4 différents types de montre. Cependant, la Montre de Luxe est attachée au poignet d’un des passagers. Pour la récupérer, il faudra alors intimider ce passager en utilisant une de ses balles.



*Differents Types de Montre de Luxe disponibles*

* **Une fin au jeu**. Comme dans chaque jeu, il faut un gagnant. Dans le nôtre, il y a différents types de victoire. La première : garder la montre de luxe 3 ou 4 tours (*le nombre de tour change en fonction de la difficulté*). Un tour est égal à toutes les actions réalisé par tous les joueurs. Alors, une fois que quelqu’un a pris la montre, les autres joueurs peuvent partir en chasse à l’homme, pouvant le faire perdre la montre avec les tirs. La deuxième option pour gagner est d’avoir en sa possession un certain pourcentage de la somme totale des butins (la montre ne compte pas dans cette somme totale). Ce pourcentage de la valeur totale, est défini en fonction de la difficulté. En pillant les passagers, les joueurs se rapprochent de cette somme pour gagner la partie. Une fois que l’une des deux possibilités de victoire est atteinte, une nouvelle fenêtre apparait annonçant la gagnant et le jeu s’arrête. Si durant le même tour, et durant la même action, deux joueurs gagnent tous les deux, l’un grâce à la montre, et l’autre grâce à la détention d’une assez grande quantité de butins, le joueur qui gagne la partie est celui qui a la montre en sa possession.