Rapport projet : Colt Express

Membre du groupe : ETTAYEB Nour & FABREGA Isabel L2 info

Apport du projet :

Nous avons pu à travers ce projet 'une part approfondir nos connaissances dans la programmation d'interface graphique en Java et d'autre part découvrir et mettre en pratique un modèle de conception logiciel : Modèle Vue Contrôleur (MVC).

Travail réalisé & extension :

Nous avons réussi à implémenter toutes les classes nécessaires pour le projet et gérer les deux phases du jeu c'est -à-dire la planification des actions puis leur exécution.
De plus, pour personnaliser notre jeu nous avons affiché les éléments du jeu tel que : les bandits , le marshall , le train et les butins en utilisant des images.
Nous avons également choisi de gérer le démarrage du jeu en ajoutant une fenêtre de début avec un bouton pour démarrer le jeu et un bouton dans la fenêtre principale pour rejouer.
En outre, nous avons ajouté une musique de jeu ainsi qu'un son lorsqu'un tir est effectué, il y a également la possibilité de désactiver le son en appuyant sur un bouton muet présent dans la fenêtre du jeu.
Enfin, nous avons décidé d'afficher l'état du jeu dans la fenêtre principale et non sur la sortie standard (dans le terminal) .

Déroulement/Architecture de notre jeu : (voir également le diagramme de classes)

- ➔ Affichage de la fenêtre principale :

  Nous avons fait le choix de diviser la vue qui implémente l'interface Observer en 3 sous vues :
  1- VueCommande qui contient les boutons nécessaires pour jouer
  2- VueTrain qui contient le train ainsi que ces éléments (les bandits, le marshall , les butins, les wagons)
  3- VueBandit qui contient les informations sur la richesse des bandits (nbr de butins et de balles restantes) construite via une classe interne BanditInfo
  Notons que toutes les sous-vues héritent de la classe JPanel pour pouvoir contenir des JComponent et avoir la possibilité de redéfinir la méthode paintComponent qui gère l'affichage graphique.

  De plus,pour afficher le joueur courant , le numéro de l'action en cours et l'état du jeu après l'exécution d'une action , nous avons ajouté directement à la classe CVue (la vue principale) deux attributs de type JPanel contenant des JLabel avec un texte qui sera mis à jour à chaque modification lorsque le modèle notifie les observateurs .

➔ Gestion des deux phases du jeu planification et action :

Pour pouvoir stocker facilement les actions à exécuter dans la deuxième phase, nous avons créé un type énuméré et non une classe abstraite Action et ajouté un attribut de type ArrayList dans la classe Bandit.

Notons que la gestion du stockage et de l'exécution des actions est réalisée par le Contrôleur qui implémente l'interface ActionListener pour avoir la possibilité de savoir quels boutons ont été cliqués et donc appeler la méthode correspondante en fonction de la phase à laquelle on est.
D'autres part, nous avons choisi de désactiver le bouton Action jusqu'à la fin de la planification et de désactiver les autres boutons lors de la phase d'action(cette gestion des boutons est réalisée dans la vue commande et vue principale).

➔ Déroulement du jeu :

Lorsqu'on exécute notre projet dans un premier temps une fenêtre de début est affichée avec un bouton "start game" sur lequel il faut appuyer pour qu'il créer la fenêtre principale donc démarrer le jeu (cette création de la fenêtre est réalisée par l'appel du constructeur CVue dans une lambda expression).

Ensuite il sera demandé de planifier les actions des bandits (lors de cette phase le bouton action est désactivé). Cette phase de planification implique une communication entre les classes Contrôleur et Bandit ce qui justifie la nécessité que le contrôleur doit accéder au bandit . Notons que lors de cette phase le modèle notifie tous les observateurs car nous avons choisi de modifier la couleur de fond du JPanel du bandit courant.

Enfin, une fois la phase de planification terminée, le bouton action est réactivé (dans la classe CVue) et les autres sont désactivés cela est réalisé par le fait que la vue principale a accès à la sous-vue commande qui a accès au contrôleur.

Remarque : Notre jeu est fini lorsque toutes les rounds ont été joués , nous avons fixé le nombre de round et le nombre d'action réalisée par les bandits avec la même valeur qui est de 3.

Difficulté rencontrées :

Remarque : Tout au long du projet nous avons rencontrée un certain nombre de difficultés qu'on a finalement pu résoudre

- ➔ Affichage Train :

  La première difficulté qu'on a rencontrée est à propos de l'affichage du train. Avant d'utiliser une image pour le train, on voulait tester le projet dans un affichage similaire à celui vu en classe. Au début, le problème se trouvait dans les coordonnées des wagons, quelque chose qu'on a réussi à résoudre rapidement, mais un nouveau problème est apparu dans les ordinateurs windows. Le train s'affiche deux fois, mais il apparaît correctement dans les ordinateurs Mac. Après avoir fait des recherches sur internet, on a trouvé que ce problème apparaît à cause d'un oubli de l'appel super.paintComponent().

- ➔ Affichage des buttons dans le panel de commande :

  Au moment de faire l'affichage des boutons de commande, on avait une idée de comment on voulait qu'ils soient afficher. On a rencontré des difficultés au moment de trouver une manière de réaliser cette idée. On a décidé de séparer les différents types de boutons en différents panels, pour pouvoir contrôler leur position plus facilement. Notre plus grande difficulté rencontrée a été de trouver une manière de placer les boutons de tir et déplacement. On a essayé différents types de layouts, mais à la fin on a trouvé que faire différents panels de BoxLayout était facile et efficace pour ce qu'on voulait.

- ➔ Mise à jour des butins dans l'affichage

  Une fois qu'on avait fini avec l'affichage des trois sous-vues de notre JFrame (le train, le panneau de contrôle et les informations des bandits), on s'est rendu compte que le nombre de butins pour chaque bandit n'est pas mis à jour après chaque braquage. En analysant notre code, on a trouvé que le problème était dans la méthode qu'on avait pour ajouter des informations du bandit au JPanel VueBandit. Originalement, on avait naïvement marqué que pour mettre à jour les informations du bandit, il fallait faire un appel à la méthode d'ajout des informations une deuxième fois pour chaque bandit. Mais cette dernière crée juste un nouveau JPanel pour le bandit donné et ne modifié pas celui qui existé déjà. Alors, pour pouvoir avoir un JPanel associé à un bandit, qu'on pouvait modifier directement, on a décidé de créer une sous-classe de VueBandit qui hérite de JPanel. Celle-ci compilait toutes les informations du bandits d'une manière qui les rendait accessibles et nous a facilité énormément la tâche de mettre à jour les JLabels associé a les butins.

- ➔ Trouver une manière d'exécuter les actions une par une

  Une partie essentielle dans le projet est d'exécuter lors de la phase d'action de la manière demandée (action par action). Celle-ci n'a pas été aussi évidente qu'elle ne

le paraît. En effet, nous avons pu facilement exécuter simultanément toutes les actions planifiées à chaque round par chaque bandit en utilisant les mêmes variables utiles pour la phase de planification (exemple : le joueur courant). Cependant, ces variables jouaient un rôle essentiel pour plusieurs méthodes d'affichage tel que la méthode update qui a besoin d'avoir le joueur courant en fonction de la phase du jeu. Il était donc nécessaire de déclarer d'autres variables spécifiques à la phase d'action pour résoudre le problème rencontré.

➔ Test du marshall :

Lorsqu'on voulait tester que tout allait correctement avec le marshall, on s'est trouvé face à une question importante : comment tester que le mouvement du marshall marche correctement s'il fait tout aléatoirement. Après un moment de contemplation, on a décidé sur une manière de tester son mouvement semi-correctement. On a fait un appel à la méthode de déplacement du marshall, on a testé qu'il restait dans son wagon ou se déplaçait dans un des wagons situation à côté puis on le remet à son wagon original. On a mis tout ça dans une boucle for qui va exécuter ces trois actions 20 fois. En plus des tests, on a fait un appel à une méthode d'affichage pour vérifier que le mouvement est vraiment aléatoire et qu'il ne reste pas dans un seul wagon pendant tous les tours de boucle. Tout ça nous a été suffisant pour conclure que le marshall se déplace aléatoirement et dans les directions qu'on veut.