Projet POGL

Paulhiac Kaiji, Pennec Jérémie

L2-INFORMATIQUE POGL 2023-2024

COLT EXPRESS

Présentation générale

Notre projet est découpé en 3 packages : la vue (interface graphique), le contrôleur, et le modèle qui comprend toutes les classes nécessaires à la logique du jeu.

On retrouvera dans le paquet contrôleur :

- Jeu qui lance le programme et gère la bonne instanciation de la partie
- Partie qui gère le déroulement du jeu/partie.Cette classe communique avec Vue pour gérer les interactions avec le modèle. La classe Jeu sera donc celle qui représente la boucle du jeu.

Présentation générale de la boucle du jeu

Le jeu se déroule en appuyant sur les boutons présents sur la fenêtre : lesdits boutons feront avancer le jeu selon la "phase" actuelle du jeu.
Il existe 2 phases:

- la Phase de Préparation(PDP) où seulement les boutons de planification (partie haute de l'interface) sont utilisables
- la Phase D'Action(PDA) où seulement les boutons "action" et "annuler" sont utilisables

La "Phase de Préparation" (PDP) représente la phase où chaque joueur devra indiquer à son bandit quelles actions il va entreprendre. On va empiler un nombre d'actions (En générale jusqu'à 4) sur un buffer. Le buffer fonctionne avec la logique du FIFO(First in, First Out). Quand on veut dépiler le buffer, on regarde l' action la plus anciennement empilée.

La "Phase D'Action" (PDA) représente la phase où nous allons exécuter les actions de chaque bandits tour à tour, avec une possibilité pour le marshall de se déplacer et chasser les bandits.

Il existe 3 types d'actions : Tir, Déplacement, et Braquage.
À ces actions sont associés des directions (il en existe techniquement 5) : Haut, Bas, Droite, Gauche, et Neutre (cette dernière n'est utilisée que pour le Braquage). Dans ce projet, on regroupe l'action et la direction dans une classe : Input.

Par exemple, si on exécute une action de type "Tir" de direction "Droite', le bandit concerné va tirer avec son revolver dans la case à sa droite.

Lors de la PDA, la fonction actionPhase() va être exécutée pour chaque bandit. Une fois ceci fait, derouleTourActionStage() est lancée.

La fonction actionPhase va dépiler 1 action du buffer de bandit et l'exécuter, quant à derouleTourActionStage, elle va calculer le nombre d'action qui reste à tous les bandits, et éventuellement passer la Partie en PDP si il ne reste plus aucune actions à dépiler pour les bandits.
Le cas où l'on retrouverait un bandit avec une action de plus que les autres n'est pas traité en PDA, car la PDP s'assure d'avoir le nombre d'actions nécessaires pour chaque bandit.

En PDP, le retour de clic sur un bouton est traduit par un affichage de la décrémentation de 1 au niveau du nombre d'action du joueur ou du passage au prochain joueur.

En PDA et qu'on appuie sur le bouton "ACTION", celui-ci va dépiler et exécuter 1 action de tous les bandits AVANT de mettre à jour l'affichage.
Dans le code, il existe 2 fonctions qui peuvent êtres associés à ce boutton action, on a décidé de prendre la fonction actionPhase_slow() : la différence entre actionPhase() c'est que actionPhase() dépile 1 fois chaque bandit en 1 seul clic, tandis que actionPhase_slow dépile 1 par 1.

Répartition des tâches

La création du diagramme de classe a été effectuée ensemble et a été modifié avec l'avancement du projet (ajout d'opération, modification de comportement, etc ...).

Les classes représentant les "composantes du jeu de plateau", c'est-à-dire le modèle, sont principalement développées par Kaiji avec l'assistance de Jérémie.

Les classes liées à l'affichage et la classe Partie sont majoritairement développées par Jérémie.

Les tests de chaque classe sont majoritairement développés par les mêmes personnes qui ont créé la classe concernée par ces tests.
À noter que pour le package Vue, n'ayant pas d'outils adéquat pour tester des scénarios avec interaction, les tests ont été fait " à la main " et l'interface a évolué empiriquement.

Problème majeure rencontré lors du développement

Nous n'avions pas eu de problème majeur rencontré dans le développement des "composantes du jeu de plateau", hormis quelques bugs mineurs qui ont vite étés

corrigés.
Nous avions eu quelques problèmes de compréhension dans la manipulation des JUnits, mais cela n'a pas été très compliqué à résoudre.
Nous avions eu cependant pas mal de conflits au début lors de la conception du diagramme de classe, et nous avions dû parfois un peu les altérer lors du développement du projet.

Ce qui a été retenu (Optionnelle)

Nous avons appris à concevoir de 0 un diagramme de classe, et de travailler sur une programmation qui s'appuie sur celui-ci, dont l'importance du diagramme dans le cas de communication entre les classes pour garder le tout logique.
Nous avons également appris à factoriser plus de code avec l'héritage, découvert l'intérêt d'avoir un modèle pour faire communiquer les différentes parties du code , avec ici MVC.
puis enfin nous avons pu prendre plus en main JUnit et bien sûr maîtriser un peu plus le langage JAVA.

Présentation des certaines parties du code

Classe abstraite : Entité

C'est une classe qui permet de représenter tous les objets du jeu du plateau qui peuvent être représentés par des coordonnées, en sachant que les coordonnées sont définis par les "Wagon/Train" du jeu du plateau. Donc, un ordonnées représente sur quelle wagon se trouve "L'entité", est si il se trouve sur "Le toit", ou "Dans le wagon" (Ainsi, x désigne dans quelle wagon se trouve l'entité tel que 0 représente le wagon le plus à gauche, et y désigne si l'entité se trouve sur le toit ou non tel que 1 représente le toit du wagon). La classe abstraite Personnage, les Objets, sont des classes qui héritent d'Entité.

Classe abstraite : Personnage

Représente les entités qui peuvent se déplacer de manière autonome. Bandit et Marshall héritent de cette classe abstraite. Cette classe implémente la méthode "deplace()".

Les classe utilisés pour le positionnement des entités : Scene et Plateau

Classe Plateau désigne le jeu de plateau lui même, et contient toutes les instances de Scene et Entité (Voir en bas pour Scène)

Scene désigne les "pièces/wagons” du jeu : il ne possède pas de système de coordonnées car Scene agit comme un système de coordonnées pour aider les Entités à s'orienter. À la place, Plateau crée une arrayList de Scène en 2 dimensions, et l’index de cette d’une scène permet d'obtenir la position de celle-ci sur le Plateau.

Par conséquent, chaque Entité garde une trace du Plateau auquel il appartient, et peuvent consulter où se trouve les autres entités ou où il se trouve en demandant à Plateau de leur donner la Scene qu'il occupe. Comme chaque Entité ne garde pas directement quelle Scene il occupe, il demande à Plateau de retrouver la Scene qu'il occupe en lui fournissant leurs coordonnées (x, y), ces coordonnées décrivent où se trouvent donc la Scene dans la arrayList de Scene de Plateau.

Pour terminer, nous allons justifier pourquoi il est utile de créer une classe Scene à part pour définir les coordonnées du jeu de plateau, au lieu de juste utiliser les coordonnées en int de (x,y) : Nous avons 2 raisons principale pour cela :
- 1 : C'est pour pouvoir facilement avoir accès aux autres entités, par exemple quand un Bandit veut braquer un wagon, il suffit juste de regarder s' il y a des Objets dans la Scene qui contient le bandit.
-2 : C'est plus facile de coder l'affichage, car nous n'avons pas à calculer en temps réel les entités se trouvant dans une même Scene pour les afficher.

Conclusion

Ce projet a nécessité approximativement plus de 10H de contribution au code pour chacun de nous 2. La conception du diagramme de classe a pris environ 3h.

Hormis quelques bugs, nous ne sommes pas tombés sur des problèmes de sémantique/ d'algorithme particulièrement alarmants. Nous pensons qu' avoir passé un certain temps sur le diagramme de classe nous a permis d'économiser un temps considérable.
A noter qu’il est compliqué de savoir si nos tests couvrent correctement notre code , sans outils dédiés à cela.

Nous avons aussi réalisé qu’il était compliqué de s’aligner sur la même direction et façon de programmer à deux. Il existe de multiples façons de visualiser et conceptualiser la problématique à traiter.