Génie Logiciel et Gestion de Projet

Poneymon

Jonathan CABEZAS 11513280
Rémy NEVEU 11710792

Sommaire

I/ Présentation du projet….....2
II/ Fonctionnalités de l'application….....2
III/ Patterns utilisés….....3
1-Modèle Vue Contrôleur….....3
2-Observateur, Observable ….....4
3-Polymorphisme….....5
4-Forte Cohésion….....5
5-Faible Couplage….....6
6-Stratégie….....6

I/ Présentation du projet

Le projet poneymon avait pour but de partir d'un projet de base et de lui appliquer les concepts et méthodes vus en cours : utilisation de gitlab et de maven pour l'intégration continue et la gestion du projet, refonte pour appliquer le pattern Modèle Vue Contrôleur.

Le concept de base est une course de poneys dont la vitesse change aléatoirement à chaque tour de terrain. Des fonctionnalités ont été rajoutées telles que la possibilité d'accélérer les poneys pour le tour en cours ou encore la gestion de ces derniers par une IA simple.

Des tests unitaires ont été conçus pour quelques fonctions du projet : un test par stratégie et quelques tests sur le PoneyModel ont été faits.

II/ Fonctionnalités de l'application

Les fonctionnalités suivantes ont été intégrées à l'application :

En plus des poneys normaux, il est possible de jouer des NyanPoneys qui voient leur vitesse augmentée pour le tour lors de l'activation de leur pouvoir.

La classe NyanPoneyModel hérite de la classe PoneyModel. Elle est observée par la classe NyanPoneyView qui affiche la bonne animation suivant si le Poney est boosté ou non.

Trois stratégies ont été implémentées pour les NyanPoneys. Elles peuvent respectivement utiliser leur pouvoir lorsque le poney va vite, trop lentement ou lorsque le boost sera suffisant pour dépasser le poney de tête.

Elles dérivent toutes de la classe NyanPoneyStrategy, la super-classe des stratégies associées aux NyanPoneys qui dérive elle même de Strategy. Leur méthode checkPower() est appelée à chaque appel de step(). Si les conditions sont réunies, le pouvoir du poney est utilisé.

Un bouton pause permet d'arrêter temporairement le modèle, les vues ne sont alors plus mises à jour.

Un bouton menu permet de retourner sur la vue du menu lors d'une partie et de l’arrêter.

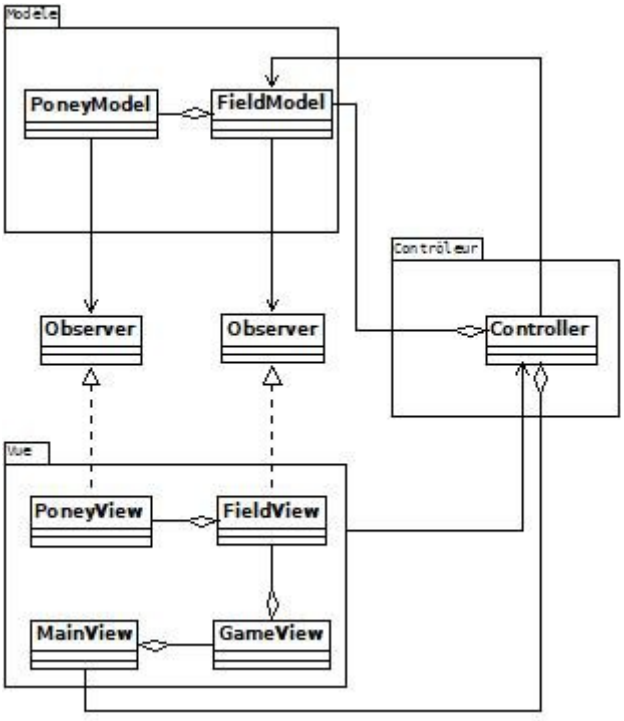
Le menu permet de lancer le jeu, d'accéder aux paramètres et de quitter l'application.

Les vues multiples sont gérées, il est possible d'afficher plusieurs vues pour un même modèle. Cependant il serait possible d'avoir deux vues liées à deux modèles différents.

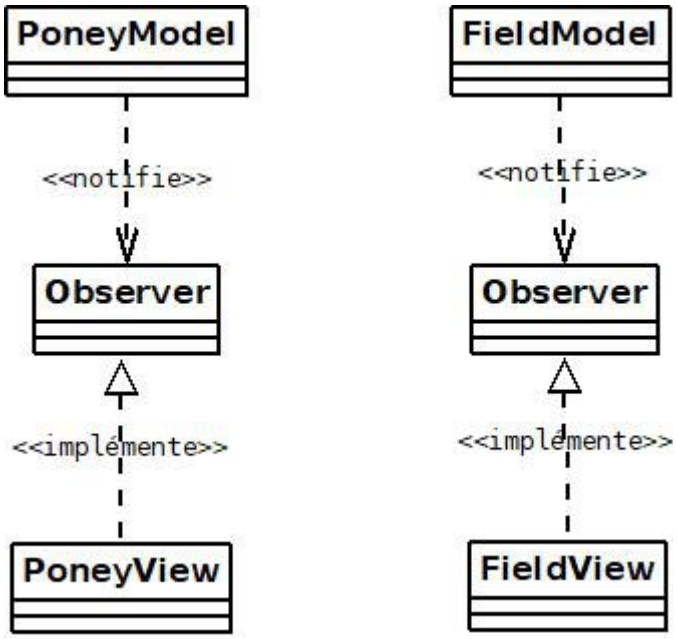
III/ Patterns utilisés

1-Modèle Vue Contrôleur

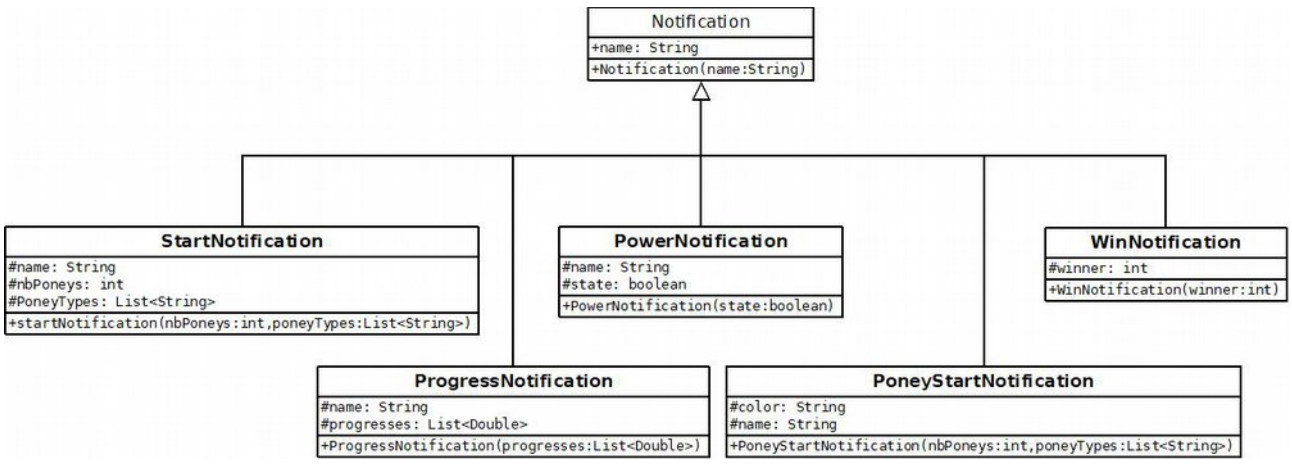
La contrainte du projet était de faire une refonte du code pour mettre en place un pattern MVC. Dans notre cas le contrôleur est une agrégation de modèles et de vues. Les éléments des vues sont mis à jour en étant des observateurs des parties correspondantes du modèle.



2-Observateur, Observable

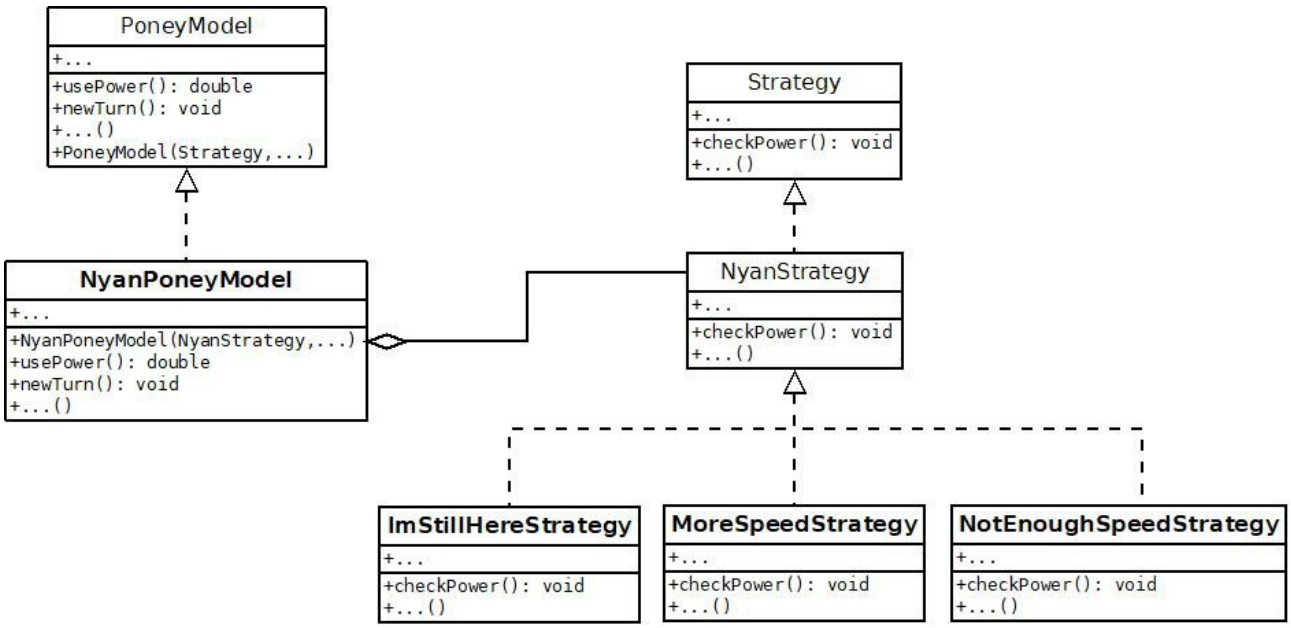


Les classes PoneyView et FieldView sont mises à jour grâce au pattern observateur, observable. Des classes dérivées de la super-classe Notification donnent des informations aux observateurs lorsque ceux ci sont notifiés.



3-Polymorphisme

Les classes dérivées de Strategy et de PoneyModel utilisent du polymorphisme. Notamment pour masquer les fonctions checkPower() de Strategy et newTurn() et usePower() de PoneyModel.



4-Forte Cohésion

On peut vérifier que toutes les classes de notre programme ont un fonctionnement découpé en petites fonctions de quelques lignes. De plus le fonctionnement de chacune de ces classes peut être décrit en une phrase courte et concise.

5-Faible Couplage

Chaque classe prise isolément est clairement compréhensible et est indépendante de la structure interne des classes qu'elle référence.

Par exemple le modèle ne connaît pas les vues et leur fournit des informations via des Notification. Les vues reçoivent des événements qui déclenchent le traitement associé dans le contrôleur, contrôleur qui à son tour appelle des fonctions du modèle pour le mettre à jour, ou des fonctions des vues principales.

Le contrôleur ne connaît que les vues principales, ce sont ces vues qui contiennent les sous-vues correspondant au menu, à la partie en cours, aux paramètres, etc.

De plus notre modèle MVC est plutôt générique.

On peut imaginer changer le modèle et utiliser notre MVC pour un jeu tout autre sans avoir à changer beaucoup de code (seulement l'aspect graphique des vues).

6-Stratégie

Ce pattern est employé pour spécifier un comportement différent à une IA. Chaque PoneyModel fait référence à une stratégie. Il est donc très facile de rajouter une stratégie sans avoir à toucher à une autre classe.

Voir le schéma de 3-Polymorphisme