GENIE LOGICIEL

Rapport projet de développement : Puerto Rico

TEAM ROCKET

JANIN Rémi
SARIGUZEL Begüm
DIALLO Elhadj Mamadou Foula
PAJANY CARPIN CAOUNDIN Allan

Encadré par
Mr. **RENEVIER Phillipe**
Mr. **COLLET Phillipe**

11 Novembre 2019

SOMMAIRE

INTRODUCTION 2
1. ORGANISATION DU CODE 3
1.1 Les classes et packages 3
1.2 Interaction entre les classes 4
1.3 Hierarchie d'héritage 5
2. PATRONS DE CONCEPTION 5
2.1 Utilisation potentielle 5
2.2 Utilisation prévu dans la suite du projet (TER) 6
CONCLUSION 6

INTRODUCTION

Faisant suite à notre projet de développement du jeu de société Puerto Rico, nous présenterons dans ce rapport les parties concernant le génie logiciel. Nous allons dans un premier temps parler de l'organisation générale du code qui nous a permis de modéliser les concepts du jeu et ensuite des éventuelles solutions que nous aurions pu implémenter afin d'améliorer la structure de notre code.

1. ORGANISATION DU CODE

Ci-dessous un schéma représentatif du code avec les packages et les liens d'interactions entres les classes principales :

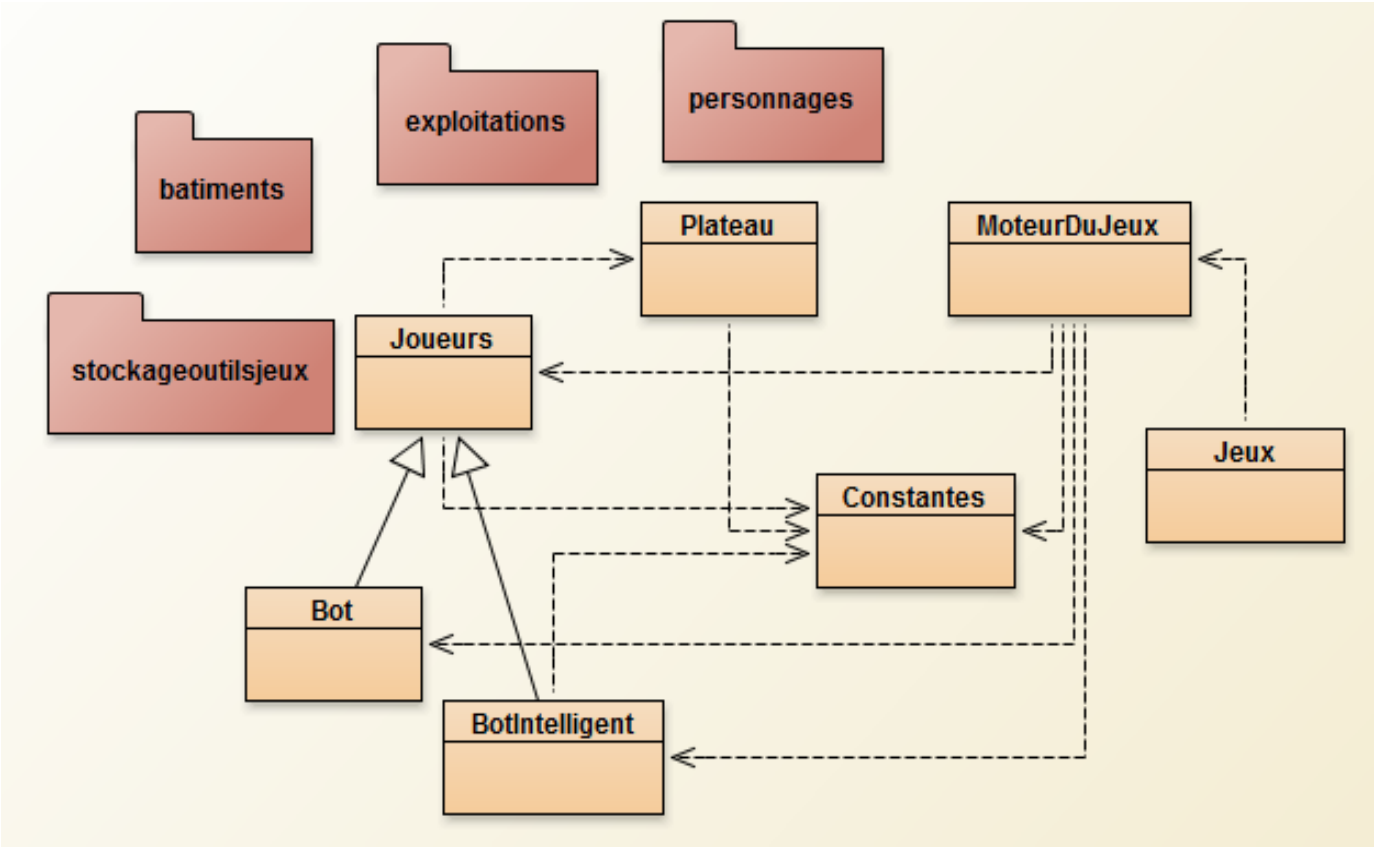

Image 1 : Schéma UML du code

1.1 Les classes et packages

Pour améliorer la lisibilité de notre code ainsi que sa transportabilité, nous avons décidé de mettre en place différents packages, en prenant en considération les différents concepts du jeu.
Les sept classes ainsi que les quatre packages représentés sur le schéma de notre code ci-dessus, se trouvent dans le package principal, « puertricotr » et ont été définis, selon les objectifs suivants :

Les classes :

- *Jeux*, permet l'initialisation du moteur du jeu et le lancement d'une ou plusieurs parties.
- *MoteurDuJeux*, s'occupe du déroulement d'une partie.
- *Joueurs*, *Bot* et *BotIntelligent* gèrent les actions du joueur/bot (choix des plantations/bâtiments/tonneaux/navires, placement des colons, une hashmap permet de stocker les noms et nombres de tonneaux possédé).
- *Plateau*, contient les éléments du plateau d'un joueur (cité, ile…)
- *Constantes*, stocke les constantes des noms de plantations ainsi que des personnages.

Les packages :

Image 2 : Représentation des packages

- 1. stockageoutilsjeux : Créé pour regrouper les classes *Navires*, *Réserve*, *Banque* et *Magasin*.
- 2. personnages : Contient les classes des sept rôles du jeu.
- 3. exploitations : Contient la classe *Exploitation* permettant de définir les plantations et la carrière.
- 4. bâtiments : Contient la classe *Batiment* et trois autres packages, « productions », « petits » et « grands » dans lesquels sont définies les classes qui y sont liées.

1.2 Interaction entre les classes

Notre code est articulé autour de la classe *MoteurDuJeu*. La majorité des interactions se passent donc entre cette classe *MoteurDuJeu* et les autres classes, car c'est elle qui organise entre eux les différents objets (Joueurs, Bâtiments, Plantations, etc…) nécessaires au bon déroulement de la partie. Il existe cependant quelques particularités au niveau de certaines classes :

- *Plateau* qui gère les bâtiments et plantations du joueur, n'est accessible qu'a travers celui-ci.
- *Magasin*, *Navires* et *Reserve*, initialisées dans le moteur du jeu, mais utilisées que par les classes *Marchand*, *Capitaine* et *Producteur*. Même principe avec la classe *Banque*, utilisée par *CherucheurOr*, *Batisseur* et *Maire* à l'exception que *MoteurDuJeux* l'utilise aussi, par exemple lors de l'ajout d'un doublon à un personnage non sélectionné durant le tour ou bien en fin de partie durant le calcul des points de victoires pour les bonus des grands bâtiments.

Cependant, pour favoriser la modularité du code, nous avons essayé de limiter le nombre d'interactions entre les classes, cela a permis notamment de bien définir les responsabilités de chacune d'entre elles et donc d'éviter de rajouter de la complexité au code.

1.3 Hierarchie d'héritage

L'héritage a été appliqué à trois classes :

- **Batiment** : classe mère définissant les informations concernant les différents bâtiments. Tous nos bâtiments de productions, petits et grands héritent d'elles.
- **Personnage** : classe mère abstraite dans laquelle est gérée les actions et privilèges de chaque personnage. Elle possède des méthodes pour ajouter et récupérer les doublons d'un personnage ainsi qu'une méthode action que l'on redéfinie dans chacune de ses classes filles, nous permettant ainsi de limiter la complexité du code et de réduire à une seule fois l'appel à cette méthode dans la classe *MoteurDuJeux* lorsque l'on effectue l'action du personnage choisi.
- **Joueurs :** classe mère où l'on définie les propriétés d'un joueur "réel". Les classes *Bot* et *BotIntelligent* en hérite et nous redéfinissons à l'interieur de ces deux classes toutes les actions pouvant être faites par un joueur, comme par exemple le choix d'une plantation ou d'un bâtiment.

2. PATRONS DE CONCEPTION

2.1 Utilisation potentielle

Pour cette version du projet nous n'avons pas utilisé un patron de conception précis même s'il faut noter que l'on a quand même un code généralement bien structuré avec une bonne encapsulation. Par ailleurs, nous aurions pu utiliser quelques-uns, dont entre autres :

- **Le pattern Singleton** : Dans notre cas nous aurions pu utiliser ce motif de conception sur les classes *Banque*, *Réserve* et *Magasin*, afin de garantir que tous les joueurs utilisent la même instance de ses derniers tel qu'indiqué dans les règles du jeu.
- **Le pattern Observateur :** Ce patron de conception aurait pu être utilisé pour la gestion des interactions entre les classes *Joueurs* et les trois autres classes citées ci-dessus. Par exemple, lorsque l'on prend des doublons de la banque, on doit d'abord décrémenter l'attribut concerné pour ensuite incrémenter en retour celui du joueur, cela permettrait donc de faire observer aux joueurs, la banque, afin qu'ils soient notifiés d'un éventuel changement.

2.2 Utilisation prévu dans la suite du projet (TER)

Dans le cadre de l'implémentation d'une intelligence artificielle pour le jeu, nous projetons d'utiliser le patron de conception Strategy.

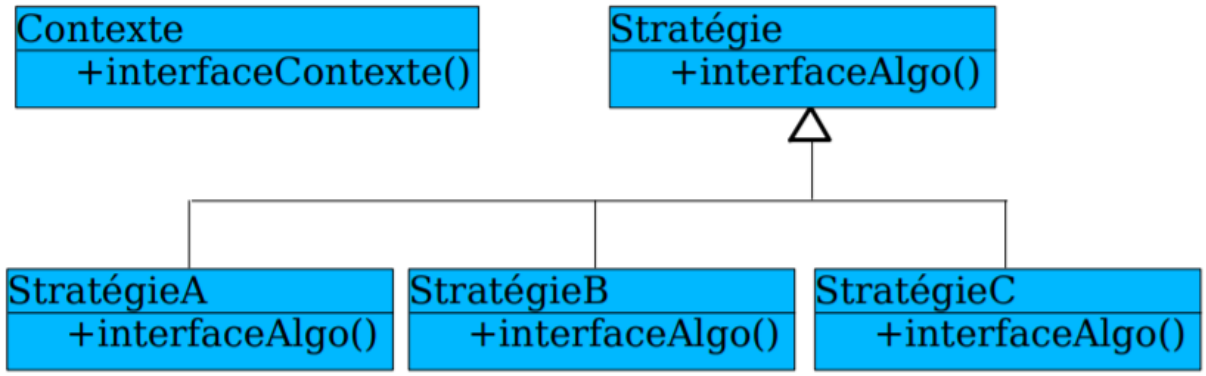

Image 3 : Schéma patron de conception Strategy

Ce patron de conception nous permettrait de définir plusieurs stratégies pour nos différents bots de façon très abstraites et de les utiliser en fonction de la stratégie que l'on voudra adopter selon les bots.

CONCLUSION

Nous pouvons dire que ce projet nous a donné une expérience très enrichissante dans la gestion de projet en groupe avec GitHub ainsi que dans l'utilisation de bonnes pratiques du développement logiciel en Java.

Il nous a ainsi permis de nous rendre compte de l'importance d'avoir une bonne organisation au niveau de l'équipe pour la définition et la répartition des tâches, d'avoir un code bien structuré et d'effectuer des tests unitaires afin d'augmenter la robustesse du programme face aux bugs et aux erreurs.