Université de Paris

Rapport:
Projet POOIG 2021:
Les Colons de Catane (jeu de base)

TRAN Henri Bergson
LE NINIVEN Adrien

Janvier 2022

Table des matières

1 Introduction

1.1 Présentation

Notre Projet est une adaptation du jeu de plateau Catan (Les Colons de Catane) uniquement en langage de programmation JAVA.

1.2 Objectif du projet

Notre projet consiste en l'implémentation de l'ensemble des éléments du jeu à travers plusieurs classes JAVA, afin de modéliser les différents aspects du jeu (règles, plateau, cartes etc...) pour ainsi pouvoir jouer une partie de Catan grâce au terminal.

2 Cahier des charges

2.1 Conception du plateau de jeu

Nous avons commencé à nous demander de quel façon allons-nous implémenter les données du plateau. Nous avons pensé tout d'abord à juste faire un tableau de dimension 2 qui ne contiendrait que les biomes et un autre tableau pour les informations de la partie comme les bâtiments crées et les routes. Mais on s'est plutôt dirigé vers un seul plateau de dimension 2 qui contiendrait les biomes , les bâtiments et les routes. Cette méthode nous permettra de ne pas se perdre dans les différents tableaux et de pouvoir séparer les cases logiquement : par exemple pour toutes les cases avec x et y congru a 1 modulo 2 ce sera que des cases biomes. Pour toutes les cases x et y congru a 0 modulo 2 ce sera les cases bâtiments. Et enfin toutes les cases x congru a 0 modulo 2 et y congru à 1 modulo 2, ou l'inverse, ce sera les cases routes.

Ensuite , nous avons séparé le plateau et les cases afin de pouvoir séparer les fonctions qui changent le plateau, et celles qui sont juste là pour récupérer les informations sur chaque case spécifique. Le plateau que nous avons fait est de taille unique : 9x9. Pour nous permettre de définir le nombre exacte de biomes et les numéros de chaque case. Nous aurions pu ajouter d'autres tailles mais le nombre de biomes ne serait pas toujours égaux contrairement au 9x9. L'ordre des numéros assigné aux cases est fixe et suit l'ordre du Catan de base , donc en spirale. Les biomes sont assignés aléatoirement et permettent ainsi des parties qui seront différentes les unes des autres. Chaque case biome possède un attribut 'type' (String) qui nous permet de savoir quel ressource donner. Tandis que les cases routes et bâtiments auront un attribut 'propriétaire' (String) qui nous permet de savoir si une case est possédée par quelqu'un.

2.2 Règles du jeu

Pour l'implémentation des règles, nous avons défini plusieurs méthodes et variables au sein de la classe Jeu pour certaines des règles. Pour les tours, nous avons créé un tableau contenant les joueurs du jeu. Ainsi durant chaque début de tour, une méthode prend en paramètre l'index du tableau des joueurs correspondant à celui qui est censé jouer le tour afin de réaliser les interactions entre le joueur et le plateau. La répartition des ressources à chaque tour est effectuée dès lors que le Joueur lance le dés, une méthode attribut les ressources aux différents propriétaires des colonies adjacentes au numéro de biome obtenu par le dé.

Ainsi pour la plupart des règles du Jeu, en ce qui concerne les "droits" du Joueur, nous avons implémenté un affichage dynamique des différents choix proposé lors d'un tour, en fonction des ressources disponibles du joueur afin que celui-ci ne puisse pas réaliser de coups illégal.

2.3 Vue textuelle

Étant donné l'absence d'interface graphique, nous avons optimisé le plus possible la vue textuelle afin que celle-ci soit la plus compète et plus "agréable" à jouer. Pour cela, nous avons dessiné le plateau dans le terminal à partir du tableau contenant les différentes cases (route, bâtiment, biome).

Pour rendre l'affichage plus claire, nous avons utilisé des séquences d'échappement de couleur ANSI afin de distinguer les différents biomes et ports grâce à une code couleur précis. Pour les ports, nous avons coloré les numéros des cases de chaque axe en fonction du type de ressource. Nous avons également coloré les différentes pièces des joueurs, mais en les soulignant pour d'avantages les distinguer des biomes et ports.

Pour le déroulement du tour des joueurs, ceux-ci sont guidés à travers l'affichage des différents coups qui leur sont possible. Lorsqu'un joueur veut poser une construction, une liste de toutes les coordonnées possibles apparaît ce qui permet au joueur de se repérer par rapport au plateau. il peut également suivre son nombre de ressource qui s'affiche à chaque tour et dès lorsque celui-ci varie.

2.4 Problèmes récurrents

Durant le projet, nous avons été confronté à plusieurs problèmes. Un des plus gros problèmes est l'oubli de certains appel de fonctions qui sont primordiaux au fonctionnement du jeu comme par exemple l'ajout de la route construite par un joueur à son ArrayList qui nous permettait de savoir quelles routes le joueur avait et ainsi obtenir les routes et les colonies construisables avec celles-ci.

Obtenir les routes et les colonies construisables pour chaque joueur a aussi été un problème que nous avons rencontré car il fallait qu'on prennent en compte les cases non construisables, toutes les différentes possibilités de construction de route par rapport à une route, et comment

ne pas récupérer deux fois la même case. Un autre des problème était les échanges car si un joueur avait, par exemple un port de bois et a deux bois ,mais fait l'échange avec 2 bois mais un seul à la fois. Cela ne le comptait pas comme un échange avec le port de bois valide car nous n'avons pas gardé en mémoire les ressources déjà choisies pour l'échange. C'est le même problème que pour le cas où le joueur commence à choisir des ressources a échanger, puis quitte l'échange en plein milieu. Nous avons donc décidé d'autoriser les échanges avec port spéciaux que si le joueur échange les deux ressources nécessaires en même temps. Et nous avons bloqué le joueur dans l'échange dès qu'il commence à choisir des ressources pour faire l'échange.

Nous avons aussi essayé d'implémenter la route la plus longue mais nous n'y somme pas arrivé. Notre idée pour faire cela était d'utiliser une fonction récursive qui garderait en mémoire les routes qu'on ne doit pas parcourir dans chaque appel récursif de façon a ne pas revenir sur des routes que nous avons déjà parcouru ou des routes qui appartiennent à d'autres branches de routes. On vérifie pendant ces appel récursives si la direction de la route n'est pas bloquée par un bâtiment adverse puis on fait des appel récursifs sur chaque route en ajoutant la longueur actuelle. Puis on récupérerait à la fin le maximum entre chaque résultat des appels. Nous nous sommes rendu compte au cours du projet qu'un autre des problèmes que nous avions eu était le fait de bien séparer les fonctions et les classes en plusieurs plus petites classes, ce qui nous aurait pu être bénéfique pour distinguer les différentes parties comme par exemple le voleur, les actions sur les routes/colonies, les cartes, et les échanges. Concernant l'interface graphique, nous avions clairement sous-estimé la charge de travaille que celle-ci nous imposerait et de plus, nous nous sommes rendu compte que l'architecture de notre code ne serait pas suffisamment modulable afin de pouvoir y greffer l'interface graphique. Nous avons donc abandonné l'implémentation de interface graphique.

2.5 Pistes d'extensions

Nous aurions voulu améliorer l'IA du jeu afin que celle-ci réalise des coups plus "intelligents" plutôt que de choisir un choix au hasard à chaque tour. Par exemple limiter la construction des routes à une route maximum par tour, poser le voleur en parcourant le tableau de façon à ce que l'IA détecte le ou les biomes ayant le plus de colonies adverse à proximité. Il pourrait aussi par exemple chercher les cases les plus proches des numéros 6 et 8 ou celles avec des ressources qu'il n'a pas et chercher a créer des routes pour pouvoir faire une colonie sur la case trouvée. Une autre façon de l'améliorer aurait été par exemple de lui donner une variable qui est initialisé à 3 ou 5 , et qui force l'IA à garder ses ressources et ne pas construire de route si il ne peut pas créer une colonie, créer une ville, ou acheter une carte développement. Cette variable serait donc décrémentée chaque tour jusqu'à 0 puis se remettrait à sa valeur initiale après ce tour.

Une des variantes que nous avons pensé à ajouter serait une "carte guerre" qui serait qu'en une seule exemplaire dans la pile de cartes. Elle permettrait au joueur de choisir une colonie qui est liée à une route du joueur avec la carte, ou une ville, et permettrait de lancer une guerre contre celui-ci. Les deux joueurs devront lancer deux dés chacun leur tour et le

plus grand total gagne la manche, jusqu'à ce qu'un joueur gagne 3 fois au total. Chaque joueur peut additionner son nombre de chevaliers de son armée à l'un des résultats des dés une seule fois pour pouvoir l'aider a gagner. Si le joueur qui utilise la carte gagne, il récupère la colonie adverse mais doit lui donner 3 ressources aléatoires si possible. Si c'est une ville, elle redevient une colonie et le joueur qui utilise la carte vole une ressource aléatoire. Si le joueur qui a utilisé la carte perd, le défenseur récupère toutes les ressources du joueur qui a utilisé la carte guerre, et peut-être lui vole un point de victoire.

Une autre variante que nous aurions voulu ajouter serait une fonctionnalité en rapport avec le désert afin de lui donner de la valeur en parallèle au fait qu'il ne donne pas de ressources. En effet, on pourrait donner une règle au désert : le premier joueur à créer une ville adjacente au désert obtiendra l'effet suivant. Tout les 5 tours du joueur, il aura le choix de défausser 4 ressources au choix pour pouvoir gagner un point de victoire. Il ne pourra gagner que 2 points de victoire maximales avec cette méthode.

3 Conclusion

Ce projet nous a permis d'améliorer notre rigueur au niveau de la conception des différents algorithmes, afin de ne pas surcharger le code inutilement et améliorer la lisibilité de celui-ci. Il nous a également appris à mieux évaluer la proportion de travail face à une tâche donnée. De plus, nous avons donc été obligés d'organiser notre code grâce notamment à de la documentation étant donné le nombre de méthodes de certaines classes. Le nombre d'informations à prendre en compte en même temps nous a permis de comprendre par nous même à quel point un bonne hiérarchie des classes et des fonctions, ainsi qu'une très bonne organisation est nécessaire entre les membres de chaque groupe pour ne pas se perdre dans les nombreuses fonctions et variables du code. Nous avons aussi compris l'importance de ne pas reporter des tâches à trop tard et de les commencer dès que l'on peut pour ne pas être débordé par le travail et respecter la date de rendu.

4 Modèle des classes