

2021/2022

RAPORT DE PROJET

IMPLÉMENTATION DU JEU CATAN



»» **Racha Nadine DJEGHALI**
»» **Anyes TAFOUGHALT**

TABLE DE MATIÈRES

01 Introduction

02 Cahier des charges

- 2.1** Fonctionnalités implémentées
- 2.2** Fonctionnalités implémentées

03 Extensions

- 3.1** Ce qui a été implémenté
- 3.2** Ce qui n'a pas encore été implémenté

04 Problèmes

05 Conclusion

INTRODUCTION

Catan est un jeu de société constitué principalement d'un plateau, celui ci représente l'île de Catane et ses différents paysages qui apportent différents types de ressources. Selon la position sur le plateau des routes et villages, les joueurs collecteront à chaque tour des ressources qu'ils géreront afin de construire routes et villages. Ces derniers leur permettront de gagner des points de victoire.

L'objectif du jeu est d'être le 1er à atteindre les 10 points de victoire.

Notre projet **consiste** à implémenter les différentes règles de ce jeu de société (écrit dans le langage de programmation orienté objet JAVA).

Nous vous présenterons, dans un premier temps, les parties du cahier des charges qui ont été traitées, les problèmes que nous avons rencontrés, les extensions rajoutées celle qui n'ont pas encore été implémentée et enfin notre représentation graphique du modèle des classes.



CAHIER DES CHARGES



Fonctionnalités implementées :

1. Un environnement de jeu, réalisant l'accueil de l'utilisateur, et permettant de procéder au paramétrage du jeu, celui ci permet au joueur :

- **de choisir le nombre de joueurs (3 ou 4) ;**
- **de sélectionner "humain" ou "IA".**

En effet, pour ce faire nous avons donné le choix à l'utilisateur de choisir le nombre de joueur, le type d'adversaire aussi (IA ou humain) :

- Pour la version textuelle nous avons utilisé un scanner
- Pour version graphique nous avons mis à la disposition de l'utilisateur un formulaire à remplir à l'aide d'un 'JTextField' et de quelques JButton.

2. La création d'un plateau constitué de tuiles carrées.

Plateau:

Pour notre version nous avons décidé de fixer la numérotation des tuiles mais de garder l'emplacement des tuiles indépendant du type de ressources qu'elles produisent (donc le plateau n'est pas fixe car les tuiles sont positionnées aléatoirement pour chaque partie).

Notre plateau est composé de :

- 1.Une matrice à 2 dimensions de type **Tuile** de taille 6*6: représentant l'ensemble des tuiles du plateau ainsi que les ports(présents sur les tuiles précises de coordonnée x/y=0 ou x/y=5).
- 2.Une matrice à 2 dimensions de type **Construction** de taille 6*6: contenant l'ensemble de colonies et de villes construites.
- 3.Un matrice à 3 dimensions de type **Route** de taille 6*6*2: contenant l'ensemble de colonies et de villes construites.
- 4.Une variable de type **Location**: qui contient la position du voleur sur le plateau.

Tuile:

- 1.Une étiquette de type int : car chaque tuile est numérotée
- 2.Un boolean indiquant si le voleur est positionné sur cette tuile
- 3.Une variable de type Location qui indique sa position sur le plateau

Construction:

Est une classe abstraite contenant :

- 1.Un attribut de type **Joueur**: stockant le joueur procédant la construction courante
- 2.Un attribut de type **Location** qui enregistre les coordonnées où est placée la construction courante
- 3.Une variable de type int type : qui peut contenir que 2 valeurs possibles : 0
 - **0** : si la construction courante est une **Colonie**
 - **1** : si il s'agit d'une **Ville**

Colonne:

Cette classe étends la classe Construction

Ville:

Cette classe étends la classe Construction

Route:

Notre classe Route possède :

1. Un attribut de type **Joueur**: stockant le joueur possédant cette route courante
2. Un attribut de type **AreteLocation** qui enregistre les coordonnées où est placée la route courante

Location:

1. Deux variables x et y de type int représentant les coordonnées sur le plateau

AreteLocation:

Cette classe étends la classe Location et possède :

1. Une variable arête de type int : qui peut avoir deux valeurs possibles :
 - **0** : si on parle de l'arête gauche de notre tuile
 - **1** : s'il s'agit de l'arête haute de la tuile

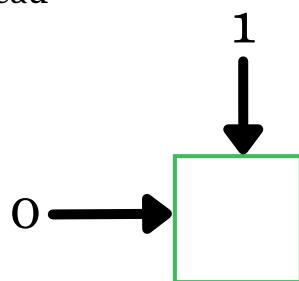
3.L'implémentation de toutes les règles du jeu.

Pour l'implémentation des règles du jeu nous avons principalement défini 2 classes principales **Joueur** et **Jeu**

1) Joueur:

Notre classe Joueur comprend :

1. Une variable de type String représentant le nom (unique) du joueur.
2. Une variable de type Color pour sa couleur
3. Une HashMap ressources , dans laquelle nous avons associé à chaque type de ressources le nombre de cartes que possède ce joueur
4. Une ArrayList main contenant l'ensemble de ses cartes développement (**CarteDev**)
5. Des variables de type int pour enregistrer le nombre :
 - de colonies
 - de villes
 - de chevaliers
 - de routes
 - de points de victoire
6. Une boolean indiquant si le joueur est celui qui possède la plus grande armée (qui lui fera gagner 2 points de victoire)
7. Un tableau ports de boolean représentant les ports que possède le joueur , alors pour être clair:
Ce tableau est de taille 6 chaque case représente un port spécialisé, par exemple :
 - ports[0]: s'il est à true c'est que le joueur possède un port général
 - ports[1]: s'il est à true c'est qu'on possède une port d'argile



Donc les indices de ce tableau représentent:

1. port général
2. port d'argile
3. port de laine
4. port de minerai
5. port de blé
6. port de bois

Dans cette classe Joueur nous avons implémenté des méthodes basiques dont on aura besoin dans l'implémentation du jeu, notamment des **méthodes**:

- Pour vérifier si un joueur possède un certain nombre de ressources
- Pour permettre à un joueur d'acheter une colonie/ville/route ou encore une carte de développement
- Mettre à jour ses données
- Vérifier si le joueur courant a gagné (s'il a atteint 10 points de victoire)

2) Jeu:

Elle contient:

1. Une ArrayList de Joueur, qui aura au plus 4 joueurs et au moins 3
2. Une plateau de type Plateau
3. Une pioche de type Pioche
4. Une variable de type Joueur stockant le gagnant

Dans notre constructeur lors de l'initialisation du jeu on vérifie que l'utilisateur a bien respecté le nombre de joueur, et la règle qui dit que les noms des joueurs doivent être uniques, on lance donc une exception si ces conditions n'ont pas été satisfaites.

Pour l'implémentation des règles:

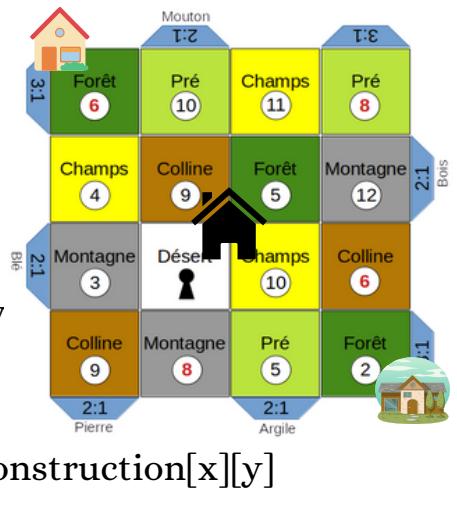
1. La construction des routes, colonies et villes:

Quand un joueur choisit les coordonnées auxquelles il veut positionner sa construction nous mettons à jour notre matrice de construction correspondant à ce choix en affectant ce joueur là au joueur de cette construction (comme précisé en haut chaque construction possède un champ joueur représentant son propriétaire)

Par exemple sur ce plateau si on voulait positionner une colonie  comme sur l'illustration le joueur doit saisir:

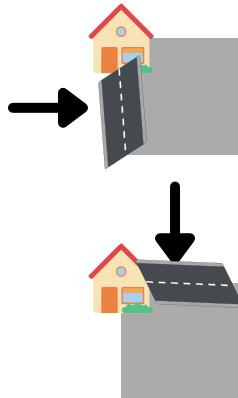
$x=1\ y=1$
pour:  $x=5,y=5$ pour :  $x=3,y=3$

Chaque case de notre matrice construction d'indices x et y représente un sommet possible, c'est à dire :
 $\text{construction}[x][y]$ représente



Et pour les routes :

- `routes[x][y][0]` représente :



- `routes[x][y][1]` représente:



2.Consultation des ressources:

En affichant le nombre de chaque ressource

3.7 aux dés:

- Nous avons créer des méthodes pour défausser la moitié de ses cartes si on a dépassé 7 cartes
- Une méthode pour déplacer le voleur (en mettant à jour la Location du voleur)
- Une méthode pour qu'un joueur J1 vole un Joueur J2 après l'avoir choisi parmi une liste

4.Cartes développement :

Pour répondre à ce problème nous avons créé une classe **Pioche** qui utilise une autre classe :**CarteDev**

CarteDev:

Cette classe est composée de :

- Une variable de type String pour le type de la carte développement :
 - 1.Chevalier
 - 2.Progrès
 - 3.Point de victoire
- Une autre variable de type String pour le sous type :

En effet car une carte de type progrés peut être de sous type : Monopole / invention ou encore construction de 2 routes gratuites

Pioche:

Est une classe contenant une liste de 25 cartes à piocher :

- 14 cartes CHEVALIER
- 9 cartes PROGRÈS
- 5 cartes POINT DE VICTOIRE

5.Commerce:

Pour le commerce nous avons utilisé une méthode qui après que le joueur ait choisi la ressource à acheter et celle à sacrifier :

- Vérifie si un joueur possède un port spécialisé , si ce n'est pas le cas elle regarde s'il a un port général , et enfin adopte l'option d'un échange avec la banque qui lui coûtera plus cher(dernière option possible).
- Enfin effectue le commerce si le joueur posséde bien la ressource à sacrifier.

4.IA

Nous avons créé une classe IA qui étends la classe joueur.

Pour celle ci les décisions ont été générées aléatoirement à l'aide de la fonction Math.random().

5. L'implémentation des deux vues.

Vue textuelle:

Cette classe contient principalement :

- Un scanner pour pouvoir communiquer avec les joueurs.
- Un attribut de type Jeu qui nous permettra de faire joueur nos utilisateurs en faisant appel aux différentes méthodes qui ont été implémentées.

Les méthodes de cette classe produisent pas mal d'affichages afin de bien décrire le jeu à l'utilisateur et pouvoir bien le mettre dans le bain.

Vue graphique:

Cette classe utilise d'autre classe comme :

1.CatanPlateau :

- Cette classe est un JPanel, elle permet d'afficher le plateau de notre jeu Catan avec les ports et de le modifier lors de l'étape de construction (ajout d'une route, colonie et ville) ou bien lorsqu'un joueur déplace le voleur.
- Chaque tuile est représentée par un carré avec un type et une étiquette.
- Le voleur est représenté par un cercle noir .

2. PartieBar

- Cette classe représente le JPanel qui assure le fonctionnement du jeu ,
- L'utilisateur commence par saisir le nombre de joueur (3 ou 4 joueurs), ensuite il choisie le type de chaque joueur (humain ou IA) en saisissant les noms des joueurs humains.
- Ensuite il pourra commencer le jeu en cliquant sur le bouton commencer.
- Au cours de chaque tour le joueur peut, après le lancement des dés, construire (une route, une colonie et une ville), faire du commerce(interne ou externe), acheter ou jouer une carte développement en cliquant sur les boutons qui correspondent à chaque action.

3. JoueurBar :

- Cette classe représente le JPanel qui permet d'afficher pour chaque joueur son nombre de ressources, de chaque carte de développement et son nombre de points de victoire

4. VueGraphique :

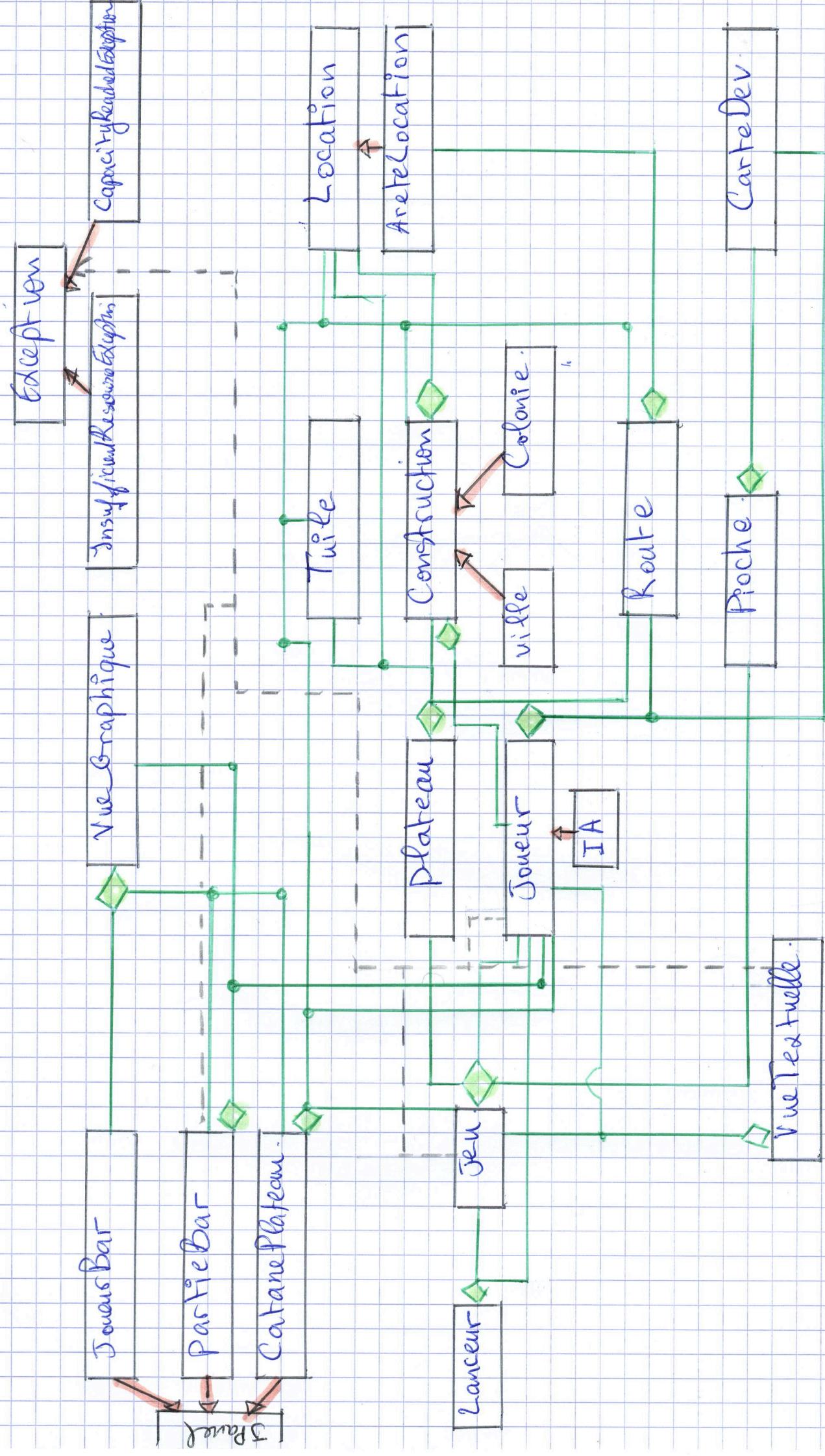
Enfin:

Cette classe crée le JFrame qui contient les 3 JPanel (CatanPlateau, PartieBar, JoueurBar).



Fonctionnalités non implementées :

- Nous n'avons pas pu implémenté de méthode pour la route la plus longue.



EXTENSIONS:

**Ce qui a été implémenté :*

1. Négoce:

En effet nous avons implémenté la négoce :

- En créant une méthode qui permet à l'utilisateur de choisir un adversaire avec qui il va échanger des ressources, celle ci vérifie que toutes les conditions sont bien satisfaites et effectue l'échange.

2. Ajout d'une carte développement "DEPLACER ROUTE":

En effet cette carte permet au joueur de replacer une route :

- En créant une méthode qui permet à l'utilisateur saisir les coordonnées de la route concernée puis si tout vas bien celles du nouveau emplacement et déplace la route sélectionnée.

**Ce qui a pas encore été implémenté :*

1. Ajouter la possibilité lors du paramétrage du jeu de créer un plateau avec des tuiles triangulaires:

Nous avons pensé a rajouté une troisième valeur a notre variable présente dans la classe AreteLocation :

- En effet celle ci pouvait avoir '0' ou '1' comme valeur pour designer s'il s'agit de l'arête gauche ou haute (comme expliquer en haut) on aurait pu rajouté la valeur '2' pour designer la route qui coupera le carré en 2 .



2. Ajout d'une carte développement "RELANCER DÉS":

En effet cette carte aurait permis au joueur de relancer le dés.

PROBLÉMES :



1.Nous n'avons pas pu résoudre le problème de la route la plus longue.



2.Lors du deuxième tour le joueur peut positionné une route collée à une autre colonie autre que celle qu'il vient de créer ce qui peut laisser une colonie seule(sans routes)



3.Nous avons eu un problème de temps aussi, nous pensons que nous pouvons améliorer considérablement notre code et réorganiser pas mal de choses (rajoutées dernièrement)

CONCLUSION :

L'objectif d'implémenter le jeu de Catan avec ces différentes règles a bien été atteint.

Ce projet nous a permis tout d'abord de bien appliquer nos connaissances que nous avons acquises avec vous durant ce semestre, telle que la conception ,la modélisation,la gestion de projet mais aussi la bonne communication .

Nous avons pu grâce a celui-ci de profiter du savoir et des connaissances de nos coéquipiers.

À travers les recherches faites afin de répondre au besoin nous avons aussi approfondi et renforcer considérablement nos connaissances.

Certes nous n'avons pas pu finir le projet comme on l'avait souhaité , mais nous pensons que nous sommes en mesure de bien l'améliorer ,et sommes aussi malgré tout satisfait du résultat.

Fin