CHATIRON Thibault

LAGRANGE Emilien

Automne 2013

Livrable 1 :

Modélisation UML

Sommaire

Introduction

Notre projet est de concevoir avec UML et développer en Java une version du jeu de cartes UNO. Ce projet devra gérer le bon déroulement d'une partie.

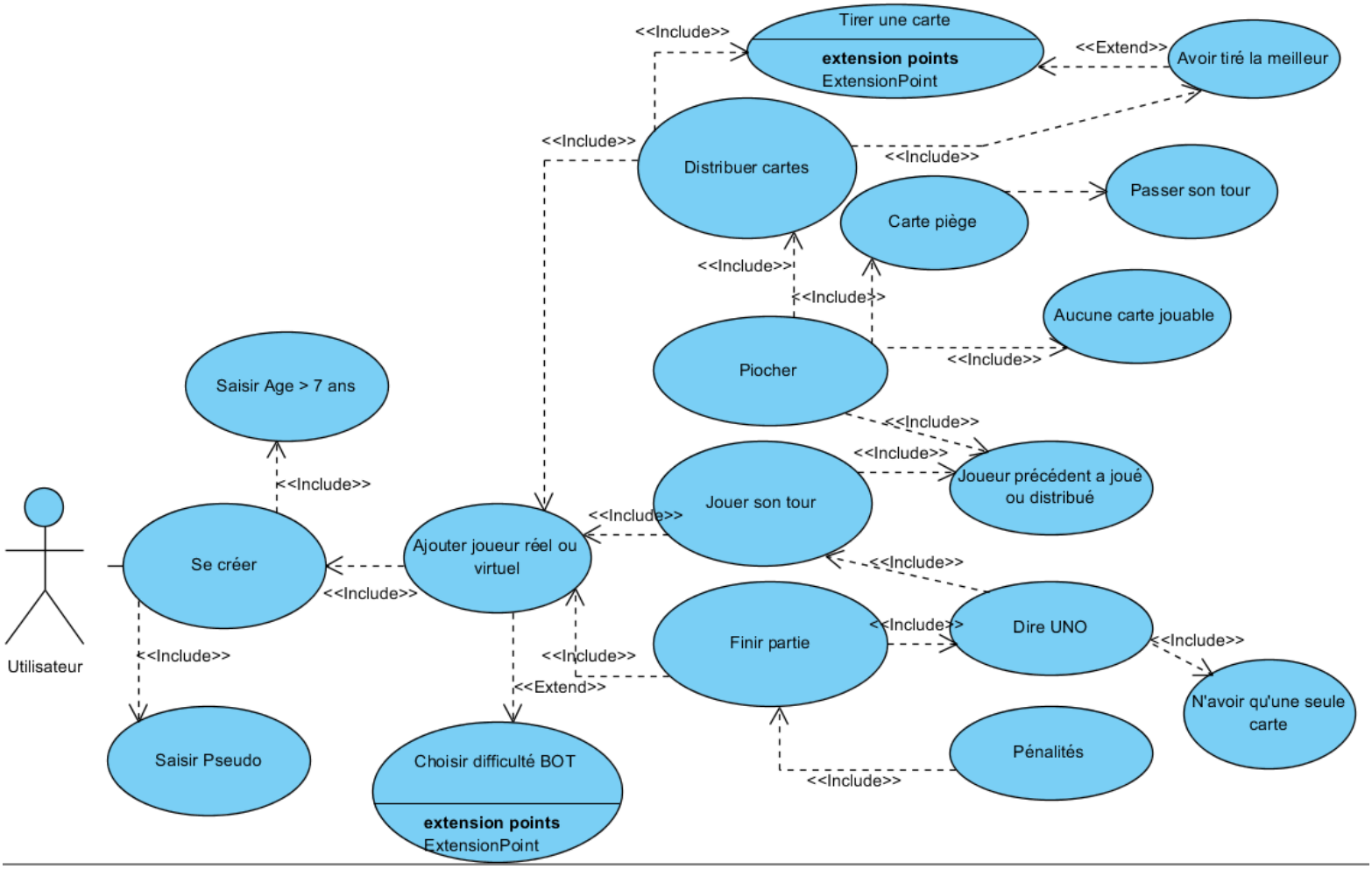
La première partie du projet est donc de représenter avec UML les bases de notre futur programme. Cela va nous permettre d'appréhender de façon claire les différentes étapes de développement au lieu de se mettre à coder différent bout de programmes au hasard. Pour ce fait, nous avons utilisé un logiciel gratuit « **Visual Paradigm for UML 2.0** ». C'est une 'Community Edition', nous permettant d'avoir accès à tous ce dont nous avions vu en cours pour créer nos diagrammes.

L'étape 'développement' se fera sous « Eclipse », logiciel gratuit que nous avons l'habitude d'utiliser depuis longtemps maintenant. Grâce à nos diagrammes, une partie du corps du programme sera générée, telles que les classes, les relations d'héritage et les interfaces.

Au final, le programme devra gérer la valeur et la couleur des différentes cartes, mais également de celles qui sont spéciales comme la carte « inversion » par exemple. Toutes ces cartes ayant un impact important sur le déroulement de la partie, nous sommes conscients qu'un travail important devra être réalisé pour la réalisation du projet.

Diagramme de cas d'utilisation

Pour notre projet, nous avons donc commencé par préparer le diagramme de cas d'utilisation afin de décomposer le jeu pour avoir une meilleure vision de son fonctionnement.



Nous commençons donc par un utilisateur Lambda qui sera le joueur Humain. Une fois que cette personne lance le programme, il faut qu'il puisse être identifié/créé. Pour cela il doit saisir un Pseudonyme. Les règles du jeu imposent aux joueurs d'avoir plus de 7ans, nous avons donc ajouté une option où l'utilisateur devra également indiquer son âge pour pouvoir continuer.

Une fois cette première étape fini, cet utilisateur doit pouvoir jouer avec des amis et ainsi créer de nouveaux joueurs, ou alors ajouter des BOTs, puisqu'il faut minimum 2 joueurs pour que la partie commence. S'il y a ajout d'un BOT, il a le choix entre plusieurs niveaux de difficultés. Dans le cadre de notre projet, le niveau de difficulté sera réglé sur « hard ». Les BOTs ainsi créés auront une attitude offensive envers les autres joueurs, et privilégierons une carte piège à une carte normale. Ici,

nous avons donc mis un « extend » puisqu'aucun BOT obligatoire est nécessaire pour débuter une partie.

Une fois les joueurs ou les BOTs ajoutés, la partie peut démarrer, il faut donc que les cartes soient distribuées. Cela se fait donc par la relation « include », puisque la distribution ne peut se faire sans être minimum 2 joueurs. Distribuer une carte nécessite au préalable que chaque joueur ait tiré une carte. Le reste des cartes ne sera distribué uniquement par le joueur qui aura tiré la meilleure. On utilise la relation « extend » entre 'tirer une carte ' et 'Avoir tiré la meilleure' puisqu'on est pas obligé de tirer la meilleur carte pour pouvoir tirer une carte, néanmoins, on est obligé d'avoir tiré la plus forte pour pouvoir Distribuer les cartes.

Ensuite, viens la question de la pioche. Ainsi un joueur pioche une carte seulement lorsque les 7 cartes initiales ont été distribuées, lorsqu'aucune de ses cartes ne lui permette de jouer, ou lorsqu'il reçoit une carte piège. Ceci par le biais de relation « include », puisque la pioche dépend de ces cas-là. Mais pour piocher il faut que ce soit à son tour, c'est pourquoi il y a une relation « include » entre 'Piocher' et 'Joueur précédent a joué ou distribué', qui oblige que le joueur de droite ait joué ou distribué les cartes pour pouvoir jouer son tour.

Toutes les cartes « pièges » (« +2 », « +4 », « inversion », « passer ») provoquent l'annulation du tour de celui recevant la pénalité. Nous avons donc mit une relation de généralisation entre 'Carte piège' et 'Passer son tour', puisque ce sera systématique lors de la pose d'une de ces cartes.

Tout comme 'Piocher', 'Jouer son tour' ne peut se faire que si le joueur précédent a fini son tour ou a distribué les cartes. Ce cas d'utilisation ne peut exister que si il y a 2 joueurs minium, d'où la relation « include » avec 'Ajouter joueur réel ou virtuel'.

Lorsqu'un joueur est sur le point de finir, il doit dire 'UNO' seulement lorsqu'il possède une seule carte et lorsque c'est à son tour de jouer. Nous avons donc mit une relation « include » entre les deux cas d'utilisations.

Lorsque la partie est finie, et que tous les cas ont été satisfaits, c'est à ce moment-là que le comptage des pénalités commence. Donc 'Pénalités' est relié avec une relation « include » avec 'Finir partie' puisque la partie doit être finie pour compter les points. Une fois la comptage fait, la distribution des cartes recommences, et ce, jusqu'à ce qu'un joueur, selon la variante utilisé, atteigne le nombre de point requis pour gagner.

Diagramme de classe

Ensuite, nous avons élaboré le diagramme de classe correspondant au jeu de carte UNO.

Préparation du jeu

** Au début d'une partie, il faut mélanger les cartes et distribuer 7 cartes à chaque joueur.*

On va donc créer une première classe « Partie ». Cette classe doit gérer le nombre de joueurs, le score, le nom du gagnant, le nom des joueurs et le sens (voir explication dans *Déroulement de la partie*). Il faut donc y associer des méthodes telles que « CalculerScore », « DeterminerSens ». « DeterminerJoueur » permettra de connaitre le nom du joueur en cours et ainsi de déterminer le joueur suivant en fonction du sens de la partie.

** Pour commencer une partie, il faut retourner la première carte de la pioche pour commencer le talon.*

On va donc créer deux autres classes qui seront « Pioche » et « Talon ».
A la classe « Pioche », on utilisera les méthodes « Mélanger » et « Distribuer » 7 cartes à chaque joueur.
Les méthodes de la classe « Talon », seront « GetInstance » afin de déterminer la carte qui se trouve en haut du talon, et « ChangerCarte » lorsqu'on joueur qu'il soit réel ou virtuel, joue une carte.

On ajoutera une classe « Porteur » dont héritent les classes « Joueur » et « Pioche », celle-ci mémorise la liste des cartes et définie les opérations « InsérerCarte » et « TirerCarte ».

Déroulement de la partie

** Le joueur situé à gauche de celui qui a distribué les cartes commence à jouer.*

Il faut gérer les joueurs, et notamment le joueur suivant. C'est pourquoi, il faut déterminer le sens afin de pouvoir de dire qui est le joueur suivant et la valeur pas défaut est le sens horaire.

Pour le bon fonctionnement du jeu, il faut bien sûr des joueurs d'où la création d'une classe « Joueur ». Pour chaque joueur, il faut un « nom », le « nombre de cartes » qu'il leur reste. Leurs cartes sont disponibles dans la mémoire de la classe « Porteur ».
A noter, une classe « BOT » héritant de « Joueur » qui a plusieurs niveaux de difficulté. En effet, il peut tout simplement commencer par jouer des cartes chiffres ou alors être plus agressif et donc jouer ses cartes Joker et notamment les cartes +2 et +4.

Dans le jeu de Uno, il faut gérer les valeurs, les couleurs et les spécialités de « CarteUno ». On va donc créer cette classe globale qui comprendra les cartes dites classiques composées d'un chiffre et d'une couleur, et les « CarteSpeciale ».

Les attributs de « CarteChiffre » sont tout simplement la valeur et la couleur de la carte. Les opérations liées à cette classe sont un getter de valeur et de couleur.

Les cartes spéciales sont constituées de « CartePlus2 », « CartePlus4 », « CartePasserTour », « CarteInversion» et « CarteJoker ».

Puisque les cartes Inversion, Passe-Tour et +2 sont en couleur, la classe correspondant à ces cartes aura comme attribut couleur.

La carte Inversion change le sens de la partie, il faudra donc récupérer le sens de la partie et le modifier, ceci se traduit avec les getter et setter de « SensPartie ».

La carte Joker n'a pour rôle que de changer la couleur de la prochaine carte qui va être jouée, on va donc opérer avec un « GetCouleur » et un « Setcouleur ».

La carte Passer un tour empêche le joueur suivant de jouer, il faut donc récupérer le joueur et le sens de la partie pour déterminer le joueur qui va jouer : GetJoueur(), et SetJoueur() permettra de modifier le joueur qui devait jouer.

La carte +2 et +4 doivent opérer sur Pioche et Joueur. En effet, il faut déterminer le joueur suivant et celui-ci devra piocher deux ou quatre cartes selon la carte jouée. On a donc un get/set Joueur et Pioche.

** Si le joueur ne possède pas de cartes lui offrant une de ces possibilités, il doit alors tirer une carte de la pioche. Si compatible, pose la carte sinon garde la carte*

Le joueur doit donc pouvoir tirer une carte, opération visible dans les méthodes de la classe du joueur en cours, soit « Porteur ».

** Si plus qu'une carte, dire « uno » sinon risque de pénalités*

Un joueur doit pouvoir annoncer Uno, contrer un Uno et pouvoir jouer ses cartes. Les opérations associées à la classe « Joueur » seront donc « AnnoncerUno », « ContrerUno » et « JouerCarte ».

** Du point de vue du graphisme*

En prévision de la modélisation du jeu, on va créer une classe « Graphisme » qui aura comme opération d'afficher les cartes du joueur en cours, du talon et les joueurs.

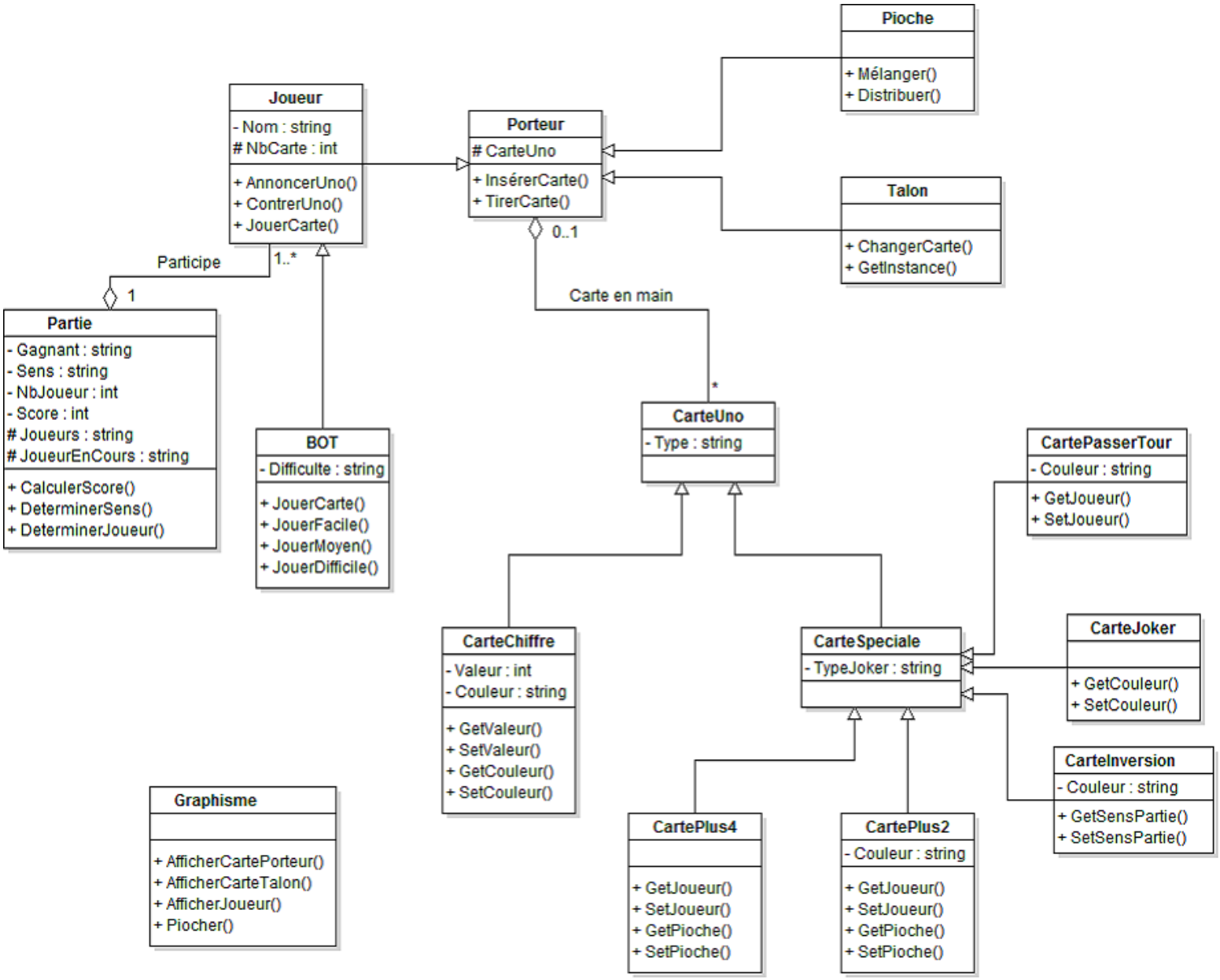

Pioche
+ Mélanger()
+ Distribuer()
Joueur
- Nom : string
NbCarte : int
+ AnnoncerUno()
+ ContrerUno()
+ JouerCarte()
Porteur
CarteUno
+ InsérerCarte()
+ TirerCarte()
Talon
+ ChangerCarte()
+ GetInstance()
Participe
1..*
1
0..1
Carte en main
Partie
- Gagnant : string
- Sens : string
- NbJoueur : int
- Score : int
Joueurs : string
JoueurEnCours : string
+ CalculerScore()
+ DeterminerSens()
+ DeterminerJoueur()
BOT
- Difficulte : string
+ JouerCarte()
+ JouerFacile()
+ JouerMoyen()
+ JouerDifficile()
*
CarteUno
- Type : string
CartePasserTour
- Couleur : string
+ GetJoueur()
+ SetJoueur()
CarteChiffre
- Valeur : int
- Couleur : string
+ GetValeur()
+ SetValeur()
+ GetCouleur()
+ SetCouleur()
CarteSpeciale
- TypeJoker : string
CarteJoker
+ GetCouleur()
+ SetCouleur()
CarteInversion
- Couleur : string
+ GetSensPartie()
+ SetSensPartie()
Graphisme
+ AfficherCartePorteur()
+ AfficherCarteTalon()
+ AfficherJoueur()
+ Piocher()
CartePlus4
+ GetJoueur()
+ SetJoueur()
+ GetPioche()
+ SetPioche()
CartePlus2
- Couleur : string
+ GetJoueur()
+ SetJoueur()
+ GetPioche()
+ SetPioche()

Diagramme de séquence

Nous finirons par modéliser le diagramme de séquence pour un tour de jeu. Pour simplifier la modélisation, on va supposer qu'il n'y a que deux joueurs. De plus, le premier joueur à jouer sera un Bot.

Pour commencer une partie, il faut d'abord déterminer un joueur (1) et un sens (2) afin de savoir qui jouera le premier. Ensuite, nous allons mélanger le jeu (3) et distribuer les cartes à chaque joueur (4). A ce moment, nous pouvons retourner la première carte de la pioche pour la mettre dans le talon (5). Le Bot commence à jouer (6).

S'il joue une carte Inversion, il faut récupérer le sens de la partie (7) et le modifier (8). C'est alors au Bot de rejouer (9).

S'il joue une carte Passer un tour, il faut récupérer le nom du joueur suivant (10). On peut alors modifier le nom du joueur (11). C'est donc au Bot de rejouer (idem à 9).

Les conditions suivantes ne seront pas représentées dans le diagramme pour un souci de place :

S'il joue une carte +2/+4, il faut récupérer le nom du joueur suivant (idem à 10). On peut alors modifier le nom du joueur (11). Ce joueur doit piocher 2/4 cartes(5.1). (Dans le cas de +4, le joueur doit annoncer une nouvelle couleur get/setCouleur). C'est donc au Bot de rejouer (idem à 9).

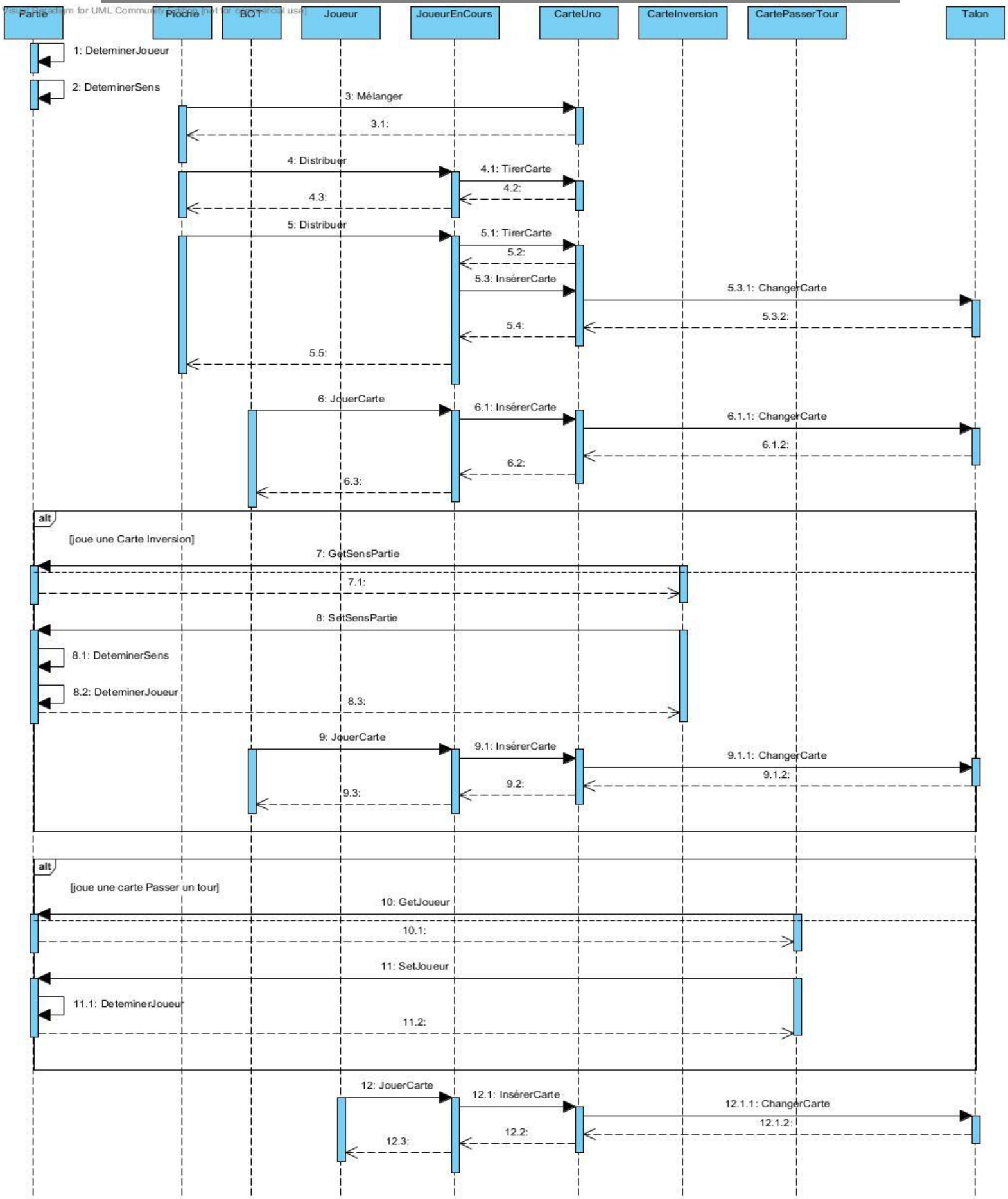
S'il joue une carte Joker, il faut modifier la couleur. Le joueur suivant joue.

S'il joue une carte chiffre, il faut modifier la carte du talon. On utilisera donc un get/set Valeur s'il joue une carte de la même couleur sinon un get/set Couleur si carte de même valeur.

Pour finir un tour de jeu :

Quand le Bot a joué une carte, c'est à l'autre joueur de poser une carte (12). S'il ne peut pas jouer, il doit piocher une carte (5.1). Si elle est jouable, il la joue (5.3,5.3.1) sinon il passe son tour. C'est donc à l'autre joueur de jouer (9).

N'est pas représentée la fin de la partie qui détermine qui est le gagnant grâce à CalculerScore de la classe Partie.


Partie
Pioche
BOT
Joueur
JoueurEnCours
CarteUno
CarteInversion
CartePasserTour
Talon
1: DeterminerJoueur
2: DeterminerSens
3: Mélanger
3.1:
4: Distribuer
4.1: TirerCarte
4.2:
4.3:
5: Distribuer
5.1: TirerCarte
5.2:
5.3: InsérerCarte
5.3.1: ChangerCarte
5.3.2:
5.4:
5.5:
6: JouerCarte
6.1: InsérerCarte
6.1.1: ChangerCarte
6.1.2:
6.2:
6.3:
alt
[joue une Carte Inversion]
7: GetSensPartie
7.1:
8: SetSensPartie
8.1: DeterminerSens
8.2: DeterminerJoueur
8.3:
9: JouerCarte
9.1: InsérerCarte
9.1.1: ChangerCarte
9.1.2:
9.2:
9.3:
alt
[joue une carte Passer un tour]
10: GetJoueur
10.1:
11: SetJoueur
11.1: DeterminerJoueur
11.2:
12: JouerCarte
12.1: InsérerCarte
12.1.1: ChangerCarte
12.1.2:
12.2:
12.3:

Conclusion

Pour conclure, les différents diagrammes ayant fini d'être crée, il ne nous reste plus qu'à continuer à coder le jeu à l'aide d'Eclipse. C'est une partie délicate puisque nous ne sommes pas des programmateurs nés. Le cours ainsi que la Java Doc nous sera utile, ainsi que les tutoriels sur internet pour nous permettre de finaliser et rendre réalisable ce projet.

Concernant les diagrammes, le cours nous a été très utile, ainsi qu'un livre empreinté au SCD intitulé « Modélisation et conception orientées objet avec UML 2 ». C'est d'ailleurs pour ça que nous sommes confiants sur le début de la modélisation, concernant la création de l'utilisateur principal du jeu. Tout ce qui comprend le mélange aléatoire des cartes, et la distribution de celles-ci est assez clair pour nous. Au vus des prochains TP, et du travail personnel à fournir, nous pensons commencer par cette étape. L'idée serait de créer dans un premier temps, sans passer par l'interface graphique, un jeu de cartes que l'on puisse manipuler, mélanger, distribuer. La suite serait de pouvoir visualiser le talon et de tirer une carte. C'est pourquoi ça viendrait dans un second temps.

Si ces deux étapes se déroulent correctement, nous nous attendons par la suite à coder le fait de jouer chacun son tour, et donc de gérer les priorités. Celle par exemple de celui qui distribue, et donc que c'est au joueur se positionnant à sa gauche qui jouera en premier. Cela ne devrait pas nous poser de gros soucis dans le code. Mais il faudra par la suite que nous insérions les cartes pièges qui sont à même de modifier le comportement du jeu, et donc du sens de déroulement de la partie.

La partie des diagrammes que nous pensons modifier sera celle du comportement des BOTs durant la partie. En effet, nous savons que les BOTs doivent préférer une tactique offensive envers ses adversaires, c'est pourquoi il faudra être vigilant et trouver le moyen le plus optimal pour répondre à cette requête sans pénaliser le BOTs en quesiton. Nous repensons au super ordinateur « Deep Blue », qui lors qu'une partie de jeu d'échec contre le champion du monde en titre, avait sacrifié sa reine pour pouvoir gagner. C'est un peu pareil ici, doit-il réellement déposer les cartes pièges en premier, ou sommes-nous supposé le rendre capable de choisir, comme par exemple, déposer une carte normal (s'il le peux) en avant dernier, dire UNO, et finir avec une carte « +2 » ou « +4 », pénalisant davantage son adversaire.

Le reste du diagramme nous semble assez clair. S'il y a des changements, ce sera quand nous taperons les lignes de codes et qu'une erreur nous semblera évidente, ce qui n'est pas le cas actuellement.