CHATIRON Thibault

LAGRANGE Emilien

Automne 2013

Livrable 3 :

Rapport final

Sommaire

Introduction

Notre projet consistait à concevoir une version du célèbre jeu de UNO en langage JAVA. La version finale de ce projet devait être en mesure de gérer le bon déroulement d'une partie.

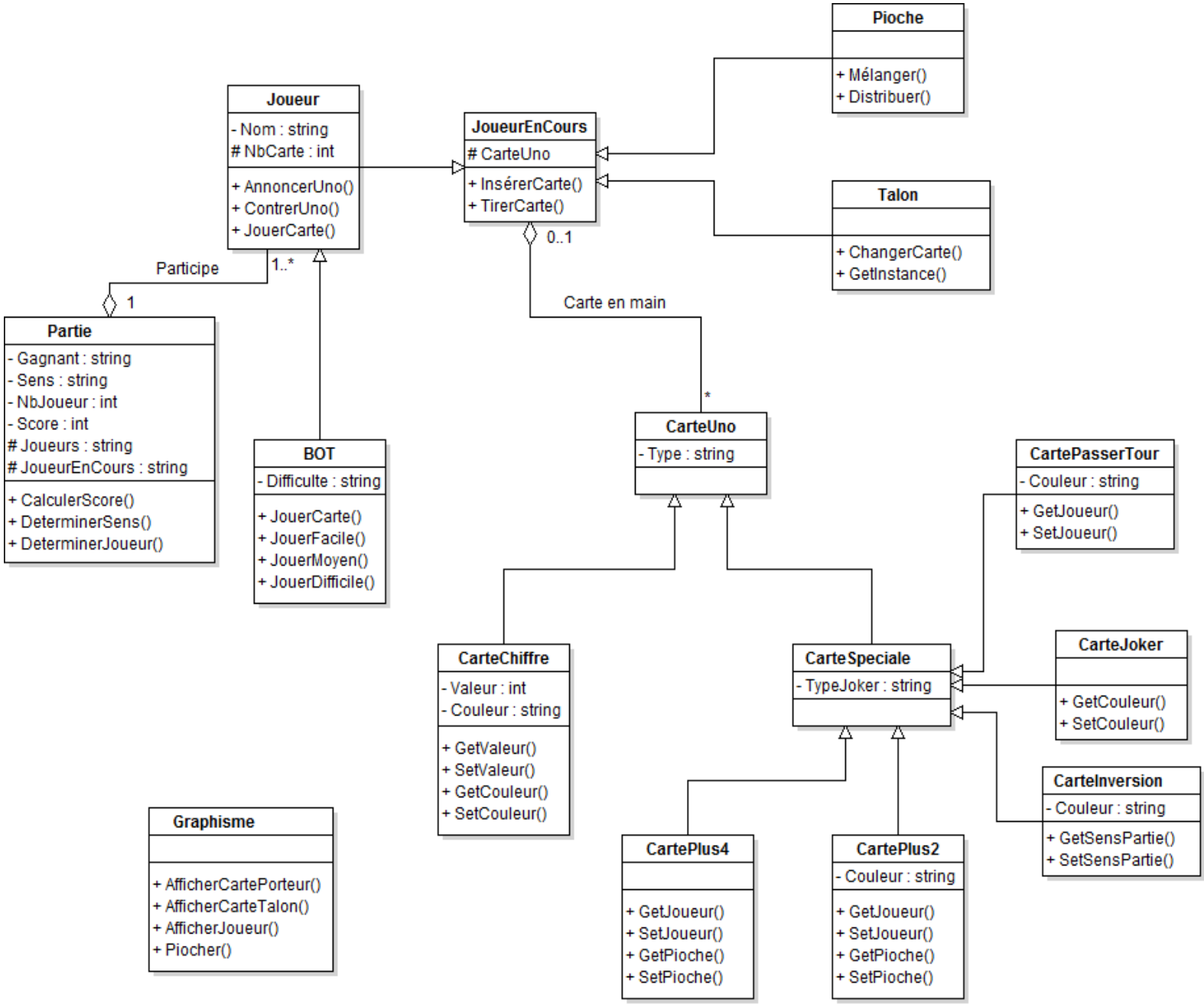
La première partie du projet était de représenter avec UML les bases de notre futur programme. Cela nous a donc permit d'avoir une vue globale de ce qui devait être fait.

La seconde partie fut l'étape « développement ». Nous avons utilisé Eclipse, et avons tous les deux travailler sous Windows pour faciliter les mises à jours des classes. A l'aide de nos diagrammes, une partie du code a été générée, telles que les classes, les relations d'héritage… Nous avons donc codé de manière à ce que l'application se lance, et que l'on soit capable de jouer normalement en ligne de code, c'est à dire avec la console d'Eclipse.

La troisième et dernière étape était de développer l'interface graphique de notre applet. Partie longue, il a fallu réadapter le code pour que ça marche.

L'intérêt de ce livrable est de voir les différences entre notre premier diagramme de classe, et celui que nous avons fait, qui représente actuellement le projet final. Nous allons également expliciter nos choix, les problèmes rencontrés ainsi que les bugs toujours en cours et connus.

Modélisation UML initiale



Modélisation UML finale

Lors de la production du cœur de l'application et de l'interface en lignes de commandes, nous avons effectué plusieurs changements. Ainsi, on observe des modifications quant à la modélisation UML finale (voir diagramme de classe sur la page suivante).

Nous avons décidé de ne pas y inclure la classe Graphique pour une meilleure lecture du diagramme. A noter qu'il a fallu ajouter dans les classes déjà existantes des fonctions permettant le bon fonctionnement de l'interface graphique.

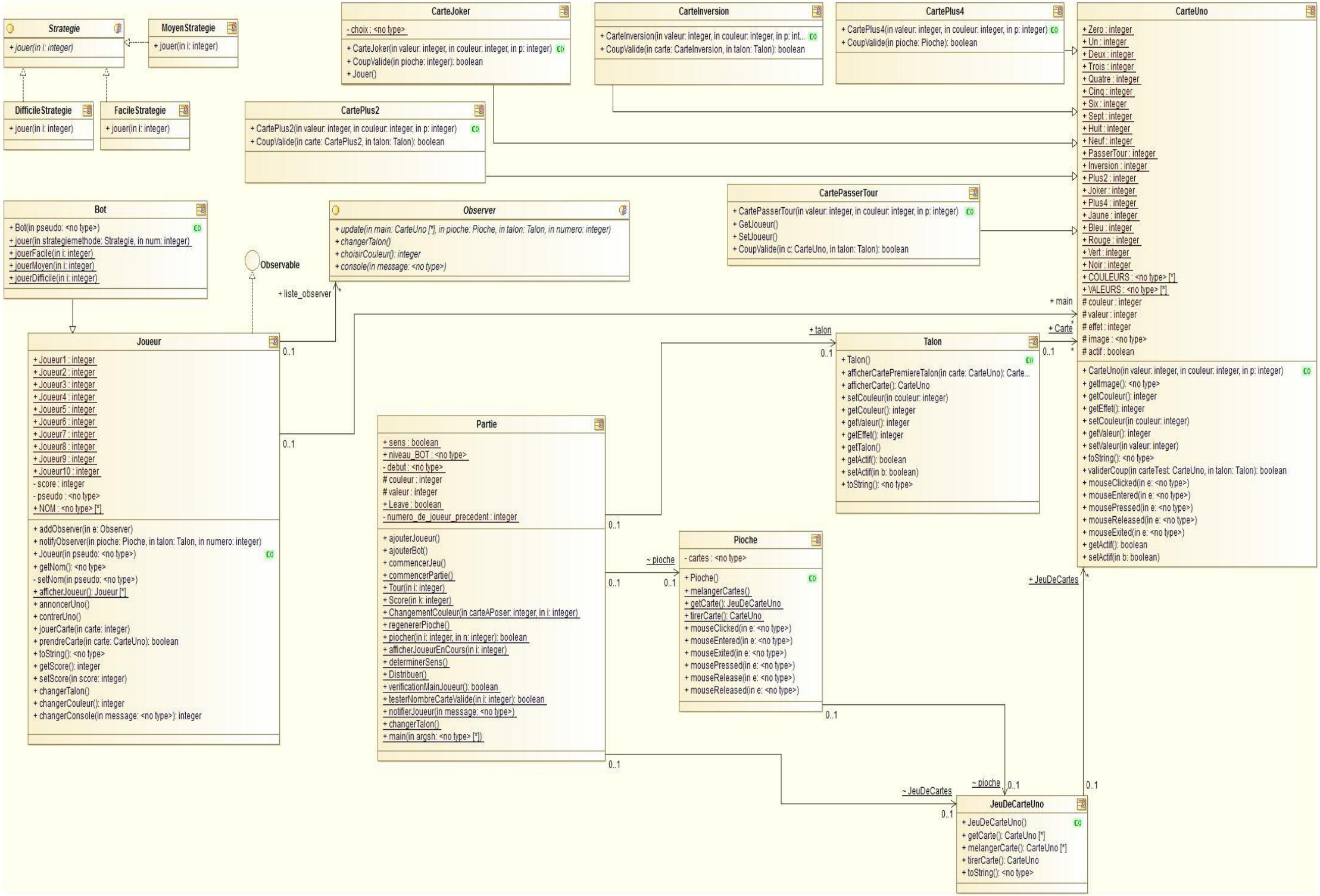
Une partie des diagrammes que nous avons modifiée est celle du comportement des BOTs durant la partie. En effet, nous savons que les BOTs doivent avoir différents niveaux. En difficile, ils devraient préférer une tactique offensive envers ses adversaires, c'est pourquoi il faudra être vigilant et trouver le moyen le plus optimal pour répondre à cette requête. Le moyen utilisé est le patron de conception Strategie. Strategie est une interface, et les différents niveaux implémentent cette interface.

La classe CarteSpeciale a été supprimée, les cartes dites spéciales héritent directement de CarteUno. La classe JoueurEnCours a également été supprimée. En effet, on gère le joueur en cours grâce à la méthode Tour(i) dans la classe Partie, qui parcoure la liste des joueurs et gère le joueur qui doit jouer.

Dans la classe CarteUno, on vérifie si la carte souhaitée peut être jouable.

Une méthode pour changer la couleur est implantée lorsqu'une carte Joker ou Plus4 est jouée. Pour cela, on a ajouté des getters pour obtenir les caractéristiques de la carte présente au-dessus du talon.

Observer/Observable est utilisée pour le bon fonctionnement du moteur du jeu avec l'interface graphique. Cela a permet notamment les notifications et mises à jour lors de la pose d'une nouvelle carte.


Strategie
+ jouer(in i: integer)
MoyenStrategie
+ jouer(in i: integer)
DifficileStrategie
+ jouer(in i: integer)
FacileStrategie
+ jouer(in i: integer)
CarteJoker
- choix : <no type>
+ CarteJoker(in valeur: integer, in couleur: integer, in p: integer)
+ CoupValide(in pioche: integer): boolean
+ Jouer()
CarteInversion
+ CarteInversion(in valeur: integer, in couleur: integer, in p: int...
+ CoupValide(in carte: CarteInversion, in talon: Talon): boolean
CartePlus4
+ CartePlus4(in valeur: integer, in couleur: integer, in p: integer)
+ CoupValide(in pioche: Pioche): boolean
CartePlus2
+ CartePlus2(in valeur: integer, in couleur: integer, in p: integer)
+ CoupValide(in carte: CartePlus2, in talon: Talon): boolean
CarteUno
+ Zero : integer
+ Un : integer
+ Deux : integer
+ Trois : integer
+ Quatre : integer
+ Cinq : integer
+ Six : integer
+ Sept : integer
+ Huit : integer
+ Neuf : integer
+ PasserTour : integer
+ Inversion : integer
+ Plus2 : integer
+ Joker : integer
+ Plus4 : integer
+ Jaune : integer
+ Bleu : integer
+ Rouge : integer
+ Vert : integer
+ Noir : integer
+ COULEURS : <no type> [*]
+ VALEURS : <no type> [*]
# couleur : integer
# valeur : integer
# effet : integer
# image : <no type>
# actif : boolean
+ CarteUno(in valeur: integer, in couleur: integer, in p: integer)
+ getImage(): <no type>
+ getCouleur(): integer
+ getEffet(): integer
+ setCouleur(in couleur: integer)
+ getValeur(): integer
+ setValeur(in valeur: integer)
+ toString(): <no type>
+ validerCoup(in carteTest: CarteUno, in talon: Talon): boolean
+ mouseClicked(in e: <no type>)
+ mouseEntered(in e: <no type>)
+ mousePressed(in e: <no type>)
+ mouseReleased(in e: <no type>)
+ mouseExited(in e: <no type>)
+ getActif(): boolean
+ setActif(in b: boolean)
CartePasserTour
+ CartePasserTour(in valeur: integer, in couleur: integer, in p: integer)
+ GetJoueur()
+ SetJoueur()
+ CoupValide(in c: CarteUno, in talon: Talon): boolean
Bot
+ Bot(in pseudo: <no type>)
+ jouer(in strategiemethode: Strategie, in num: integer)
+ jouerFacile(in i: integer)
+ jouerMoyen(in i: integer)
+ jouerDifficile(in i: integer)
Observer
+ update(in main: CarteUno [*], in pioche: Pioche, in talon: Talon, in numero: integer)
+ changerTalon()
+ choisirCouleur(): integer
+ console(in message: <no type>)
Observable
+ liste_observer
+ main
Joueur
+ Joueur1 : integer
+ Joueur2 : integer
+ Joueur3 : integer
+ Joueur4 : integer
+ Joueur5 : integer
+ Joueur6 : integer
+ Joueur7 : integer
+ Joueur8 : integer
+ Joueur9 : integer
+ Joueur10 : integer
- score : integer
- pseudo : <no type>
+ NOM : <no type> [*]
+ addObserver(in e: Observer)
+ notifyObserver(in pioche: Pioche, in talon: Talon, in numero: integer)
+ Joueur(in pseudo: <no type>)
+ getNom(): <no type>
- setNom(in pseudo: <no type>)
+ afficherJoueur(): Joueur [*]
+ annoncerUno()
+ contrerUno()
+ jouerCarte(in carte: integer)
+ prendreCarte(in carte: CarteUno): boolean
+ toString(): <no type>
+ getScore(): integer
+ setScore(in score: integer)
+ changerTalon()
+ changerCouleur(): integer
+ changerConsole(in message: <no type>): integer
Talon
+ Talon()
+ afficherCartePremiereTalon(in carte: CarteUno): Carte...
+ afficherCarte(): CarteUno
+ setCouleur(in couleur: integer)
+ getCouleur(): integer
+ getValeur(): integer
+ getEffet(): integer
+ getTalon()
+ getActif(): boolean
+ setActif(in b: boolean)
+ toString(): <no type>
+ talon
+ Carte
Partie
+ sens : boolean
+ niveau_BOT : <no type>
- debut : <no type>
couleur : integer
valeur : integer
+ Leave : boolean
- numero_de_joueur_precedent : integer
+ ajouterJoueur()
+ ajouterBot()
+ commencerJeu()
+ commencerPartie()
+ Tour(in i: integer)
+ Score(in k: integer)
+ ChangementCouleur(in carteAPoser: integer, in i: integer)
+ regenererPioche()
+ piocher(in i: integer, in n: integer): boolean
+ afficherJoueurEnCours(in i: integer)
+ determinerSens()
+ Distribuer()
+ verificationMainJoueur(): boolean
+ testerNombreCarteValide(in i: integer): boolean
+ notifierJoueur(in message: <no type>)
+ changerTalon()
+ main(in argsh: <no type> [*])
Pioche
- cartes : <no type>
+ Pioche()
+ melangerCartes()
+ getCarte(): JeuDeCarteUno
+ tirerCarte(): CarteUno
+ mouseClicked(in e: <no type>)
+ mouseEntered(in e: <no type>)
+ mouseExited(in e: <no type>)
+ mousePressed(in e: <no type>)
+ mouseRelease(in e: <no type>)
+ mouseReleased(in e: <no type>)
~ pioche
+ JeuDeCartes
~ JeuDeCartes
~ pioche
JeuDeCarteUno
+ JeuDeCarteUno()
+ getCarte(): CarteUno [*]
+ melangerCarte(): CarteUno [*]
+ tirerCarte(): CarteUno
+ toString(): <no type>
0..1
*

Etat actuel de l'application

Nous finirons par conclure quant à l'état actuel de l'application, et donc si le cahier des charges est respecté.

Nous pouvons gérer le score des joueurs. Cependant, il est dit que la partie s'arrête lorsqu'un joueur a obtenu 500 points alors que dans notre cas, nous terminons la partie lorsque la manche est terminée. A la fin de cette manche est affiché le gagnant et son score.

Nous avons traité la gestion de la première carte du talon. C'est-à-dire lorsqu'une carte spéciale est jouée, et selon cette carte, le joueur devra soit piocher, passer son tour, choisir une couleur, et le sens de la partie pourrait changer.

Nous gérons automatiquement la distribution alors que le cahier des charges demandait à ce que soit celui qui a la carte avec la valeur la plus petite, et donc que le joueur suivant est celui se positionnant à gauche de celui qui distribue.

Nous n'avons pas réussi à créer un fichier .jar exécutable, qui fonctionne correctement. Notre fichier .jar affiche bien les informations demandées, comme par exemple la fenêtre « ajouterJoueur », mais ne démarre pas la partie. Nous sommes encore en en développement sur ce sujet.

Conclusion

Le cours, les TD, les TP ainsi que les nombreux tutoriaux sur internet nous ont permit d'achever ce projet sur une note positif. Actuellement sous Eclipse le jeu se déroule correctement. On peut jouer, et même s'entrainer à jouer contre des Bots en Difficile avant d'affronter des amis, ou encore sa famille.

Au stade actuel, il reste quelques modifications, et des patchs sont à prévoir pour apporter au jeu une meilleur stabilité, et optimisé les tours de jeu. sssIl faut encore résoudre le problème avec le fichier .jar, qui nous pose à l'heure actuelle des difficultés, celui-ci ne gérant pas tout a fait nos attentes.