Projet LO02

(Automne 2013, livrable 3)

DEPREZ Alexis

LU Rui

I] Introduction

Dans le cadre de l'UV LO02 : Principe et pratique de la programmation objets, nous devons concevoir et développer une version électronique du jeu de cartes « Uno », en nous appuyant sur le langage Java.

Le « Uno » est un jeu de cartes pouvant se jouer à partir de deux personnes. Chacune commence avec sept cartes en main, le but étant de s'en débarrasser avant les autres joueurs. Les cartes ont une valeur en point, allant de zéro à cinquante. A la fin de chaque partie, les cartes restantes de chaque joueur leur accumulent des points. Lors d'une partie classique, si l'un d'eux atteint 500 points, le jeu se termine. Le joueur ayant le moins de points est alors le gagnant. En mode de jeu « challenge », chaque joueur dépassant 500 points est éliminé, jusqu'à ce qu'il ne reste plus qu'un joueur en dessous des 500 points.

Le jeu est composé d'un talon et d'une pioche. Il est possible de poser une carte dès lors que la carte sur le dessus du talon a même couleur, ou le même signe. Les cartes noires peuvent être posées en tout temps.

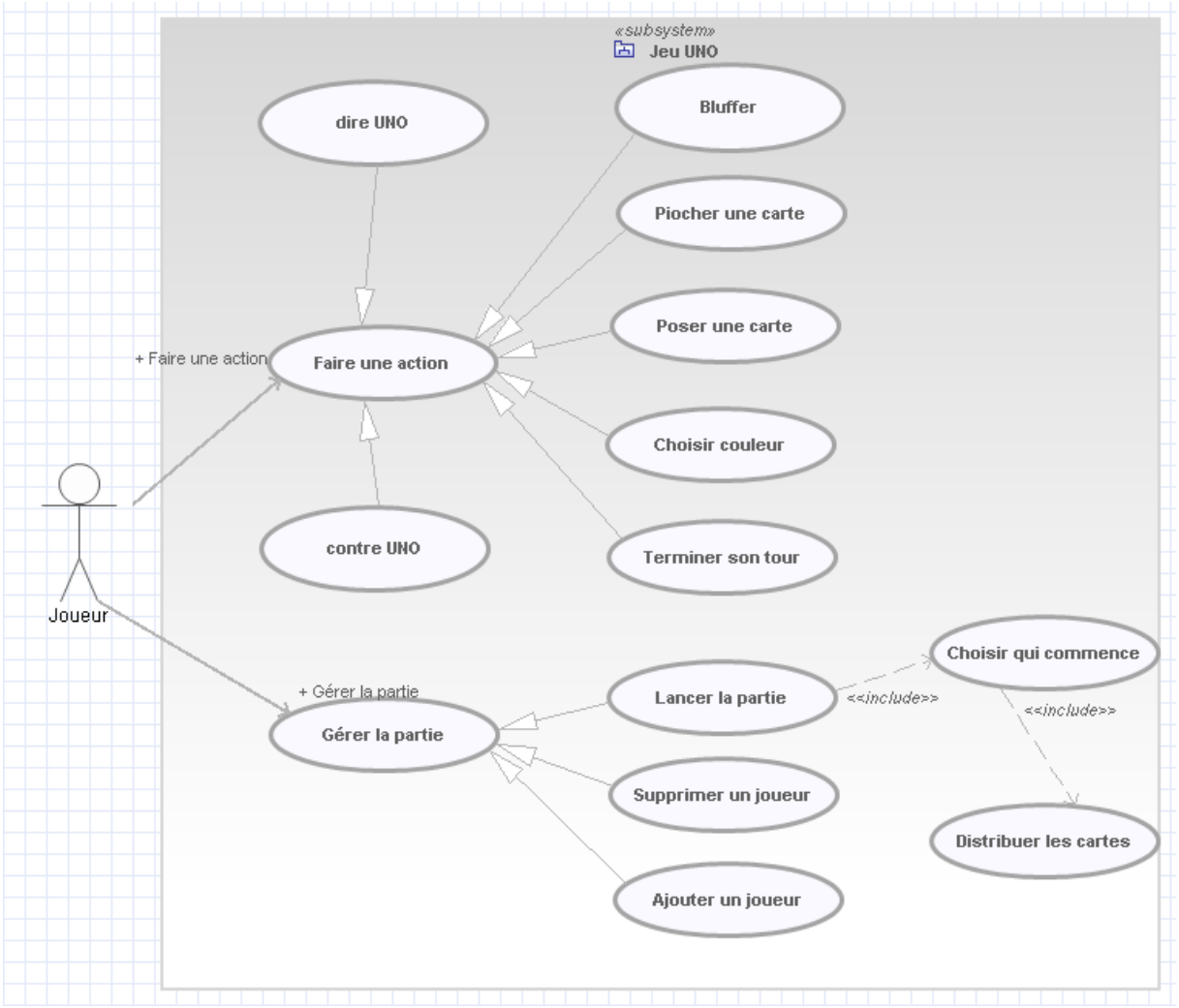
Le jeu est composé de 104 cartes, réparties comme suivant :
- Une carte zéro, deux cartes de un à neuf, deux cartes « +2 », deux cartes « inversion » et deux cartes « passe-tour » pour chaque couleur (rouge, vert, jaune, bleu).
- Quatre cartes « Joker »
- Quatre cartes « +4 »

Les cartes « +2 » ont pour effet de faire piocher deux cartes au joueur suivant, et de lui faire passer son tour. Les cartes « inversion » ont pour effet de changer le sens du jeu ». Les cartes « passe-tour » ont pour effet de faire passer le tour du joueur suivant. Les cartes joker et « +4 » permettent de choisir la nouvelle couleur de jeu. En plus de cela, la carte « +4 » fait piocher quatre cartes au joueur suivant. Cependant, si un joueur pose cette carte sachant qu'il peut en jouer une autre, il s'expose à un « contre-bluff ». Si le joueur suivant le devine et l'annonce, alors le détenteur du « +4 » pioche à sa place. Toutefois, s'il annonce à tort, le joueur victime devra piocher deux cartes supplémentaires, pour un total de six cartes.

Une première partie présente un diagramme de l'ensemble des cas d'utilisation du système. Dans un second point, nous présentons le diagramme des classes du cœur de l'application avec la justification de nos choix conceptuels, ainsi que notre utilisation du patron de conception « Strategy » pour la réalisation des joueurs virtuels. Ensuite, nous présenterons le diagramme de séquence du déroulement d'un tour de jeu. Enfin, nous conclurons par la présentation de l'état actuel de l'application.

II] Diagramme des cas d'utilisation

Le diagramme ci-dessous représente les cas d'utilisations du système pour notre jeu de « Uno » :



En premier lieu, le joueur peut gérer la partie. Cela consiste en l'ajout et la suppression de joueurs ; il peut ensuite lancer la partie dès lors qu'il y a au moins deux joueurs. S'en suit le choix du joueur allant commencer, puis la distribution de sept cartes pour chacun des participants.

Une fois la partie lancée, le joueur est en mesure d'effectuer différentes actions lors de son tour de jeu, dépendamment de celles du joueur précédent et de la carte sur le dessus du talon.

Si le joueur précédent n’a plus qu’une carte et n’a pas dit « Uno », alors le joueur courant est en mesure d’annoncer « Contre-Uno ».

Si la carte sur le dessus du talon est une carte permettant de changer la couleur et qu’elle n’a pas encore été définie, le joueur peut choisir sa couleur. Ce cas ne peut se produire que lorsque la première carte du talon est une carte change-couleur, en début de partie.

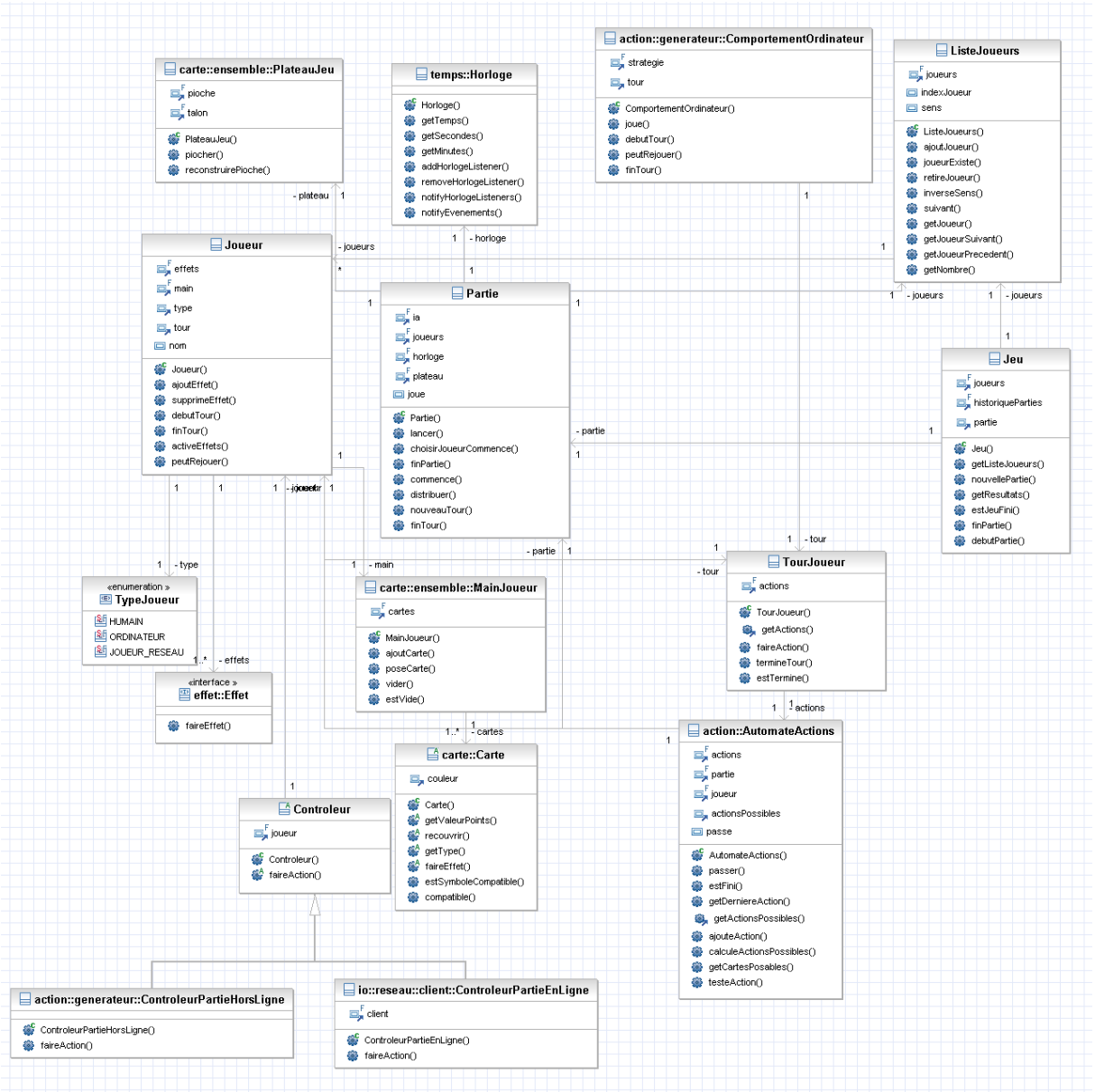
Si le joueur est victime d’un « +4 », il peut alors décider de bluffer ou de piocher. S’il a raison, alors le détenteur du « +4 » pioche à sa place. Toutefois, s’il annonce à tort, le joueur victime devra piocher deux cartes supplémentaires, pour un total de six cartes.

Le joueur peut choisir de poser une carte. Si la carte posée ne requiert aucune action, il peut alors terminer son tour. S’il ne lui reste plus qu’une carte après cela, il peut annoncer « Uno ».

Enfin, le joueur peut choisir de piocher une carte. Il pourra ensuite en poser une, ou terminer son tour.

III] Diagramme de classes

Le diagramme ci-dessous présente les classes du cœur de l'application. Celui-ci n'a souffert d'aucune modification conceptuelle depuis la version précédente, cette dernière étant déjà stable et complète :



La classe jeu défini le jeu du « Uno ». Elle contient la partie courante, et mémorise les résultats des parties précédentes, ainsi que la liste des participants.

La classe partie permet de gérer une manche du jeu du « Uno ». Une fois lancée, elle notifie le premier joueur du début de son tour, et, une fois qu'il a terminé, en informe le joueur suivant. Ceci continue jusqu'à ce qu'un joueur n'ait plus de cartes. La partie encapsule le plateau de jeu, lequel contient deux tas de cartes : le talon et

la pioche. Cet objet met à disposition des méthodes pour piocher ou poser une carte, permettant de reconstruire la pioche dans le cas où celle-ci serait vide.

La liste des joueurs mémorise le joueur courant, et permet d'avancer vers le suivant en fonction du sens de jeu. Chaque joueur possède une main, regroupant les cartes qu'il possède. Un joueur est défini par son type, selon qu'il soit humain s'il est joué par un joueur physique, ordinateur s'il s'agit d'une intelligence artificielle ou encore joueur réseau s'il est contrôlé par une personne présente sur une autre machine.

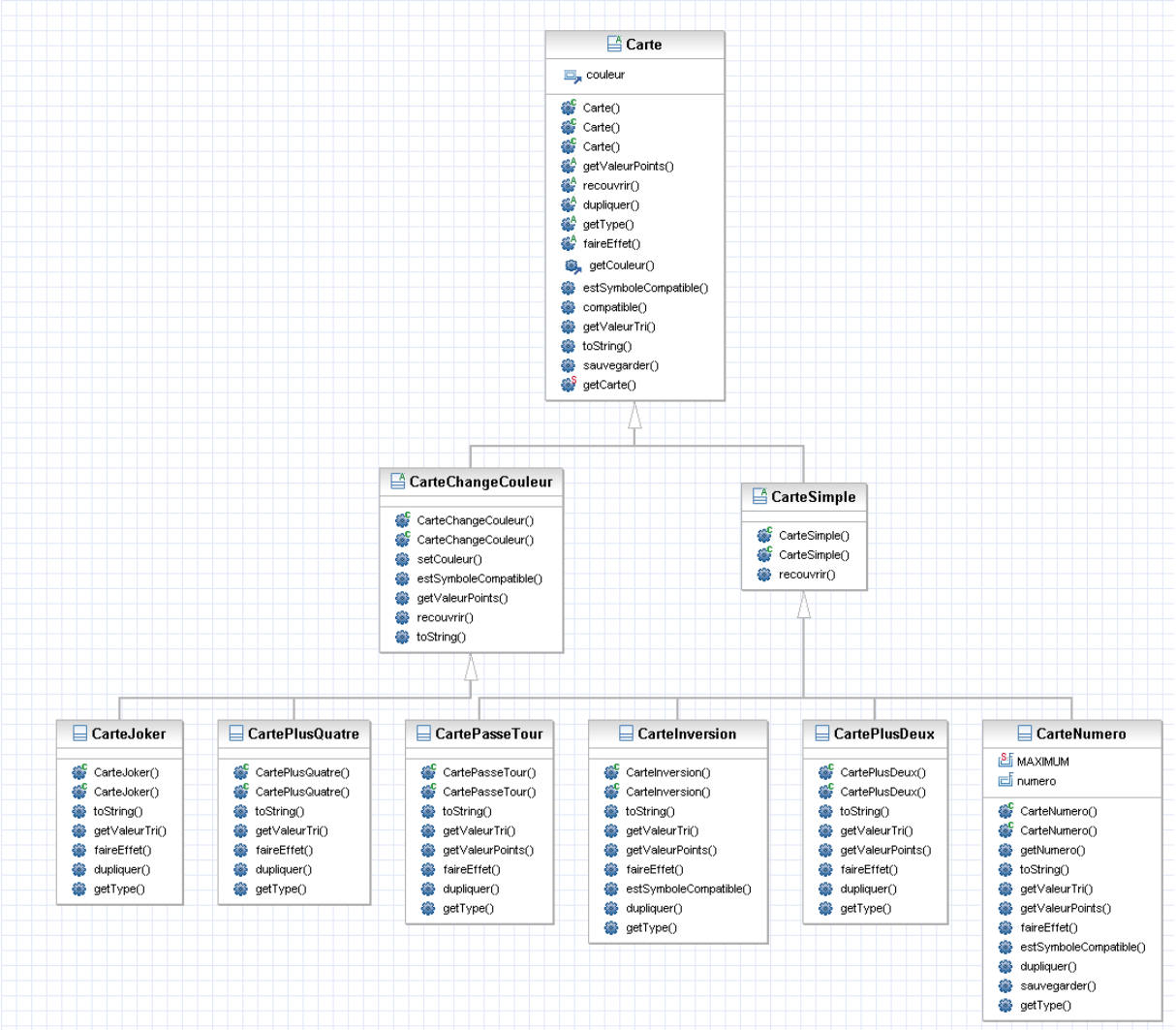
Lorsqu'un joueur commence son tour, une instance de « TourJoueur » est créée, lui permettant d'effectuer des actions en fonction de ses possibilités. Cet objet contient un automate présentant toutes les actions réalisables par le joueur. Après chaque action, celui-ci se met à jour en recalculant les actions possibles. Les écouteurs des tours des joueurs peuvent accéder au tour créé, leur permettant de choisir quelle action effectuer parmi celles proposées.

Nous avons utilisé le patron de conception « Strategy » pour représenter les différents comportements d'un joueur. Cela se concrétise par l'implémentation d'une interface, proposant les différentes actions possibles, et demandant le rang de l'action à effectuer. Dans le cas d'un joueur physique, cela met à jour son interface graphique pour lui proposer les différentes actions réalisables. S'il s'agit d'un joueur virtuel, celui-ci devra choisir quelle action il souhaite effectuer.

Le contrôleur est une classe abstraite, se définissant selon que la partie se joue en ligne ou hors ligne. Il suit le patron de conception MVC, pour permettre l'ajout d'une interface graphique. Cet objet fait le lien entre un joueur physique et le modèle, représenté par la partie. Dans le cas d'une partie hors ligne, l'action est réalisée directement. S'il s'agit d'une partie en ligne, l'action est envoyée au serveur, qui la traite et en informe tous les clients. Ainsi, le serveur reste maitre de la partie.

La classe horloge permet d'introduire une notion de temporalité. Ainsi, nous pouvons restreindre le temps maximum qu'un joueur peut prendre pour réaliser son tour, et exécuter des actions par défaut. Cela permet également de laisser un délai entre chaque tour, pour permettre aux joueurs d'observer le jeu. Cela se concrétise par l'ajout d'évènements, se produisant après un temps donné.

Le diagramme ci-dessous présente les classes associées à une carte :



Chaque carte possède une couleur : rouge, bleu, vert ou jaune. Les cartes permettant de changer de couleur ont une couleur non définie par défaut, pouvant être choisie via l'action de choix de couleur. Les autres sont considérées comme des cartes simples, dont la couleur ne varie pas.

Toute carte possède une méthode permettant d'obtenir sa valeur en points, calculée en fonction des spécificités du jeu du « Uno ». Elle possède également une méthode permettant de déterminer si elle est compatible avec une autre carte, en comparant leur couleur et leur symbole. Une méthode permet également d'obtenir une valeur entière afin de trier les cartes par couleur et par type.

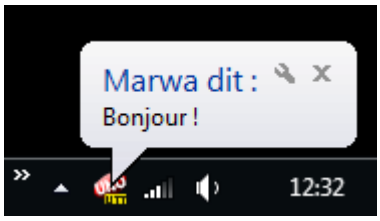
Une méthode permet de dupliquer une carte, afin d'obtenir une copie reflétant l'état de la carte à un instant donné, permettant de les ajouter dans l'historique des cartes pour chaque joueur en fin de partie, en conservant leur état au moment de la fin de la partie.

Une méthode statique de la classe « Carte » permet de créer une carte à partir d'un flux, provenant soit d'un fichier, soit du réseau.

Afin qu'il soit possible de jouer en réseau, nous avons prévu cette fonctionnalité dès le début. La recherche et la diffusion d'une partie s'effectue en multicast UDP, afin de détecter et partager toutes les parties sur le même réseau. Il est également possible, à partir de l'adresse d'une machine et d'un numéro de port, de la questionner sur la présence ou non d'une partie hébergée sur celle-ci. L'utilisateur peut ainsi entrer les adresses sur lesquelles il souhaite rechercher des parties, via une interface graphique se voulant la plus ergonomique possible. Les parties en réseau se jouent de la même façon que les jeux hors ligne, à ceci près qu'un chat permet aux joueurs de communiquer entre eux, et que le serveur est maître du déroulement de la partie. Notre conception respecte de patron de conception Client-Serveur TCP.

Notre application s'adapte également à la langue de l'utilisateur. Celle-ci est détectée au lancement de l'application en fonction de la configuration de son ordinateur, ou il lui est également possible de la modifier depuis le menu ou l'icône présente dans la barre des tâches. Nous avons écrit un fichier contenant la traduction des textes dans cinq langues : Français, Anglais, Chinois, Espagnol et Portugais. Lorsqu'une phrase doit être affichée dans l'application, nous faisons appel à un singleton permettant de retourner la phrase dans la langue actuelle de l'application.

Une icône s'ajoute désormais à la barre des tâches lors de l'exécution de l'application, afin d'accéder à certaines fonctionnalités et de permettre l'affichage de notifications lorsque la fenêtre n'est pas visible. Il s'agit là aussi d'un singleton, une seule icône étant nécessaire par processus.



De nombreuses animations apparaissent également tout au long du jeu et sur les différents écrans, afin de rendre l'application plus attrayante. On retrouve notamment le déplacement des cartes lors du jeu, permettant également de suivre plus aisément le déroulement du jeu. Dans la même optique, des sons viennent enjoliver le jeu.

Nous avons également implémenté plusieurs variantes du jeu :

- Le jeu en équipes. Les joueurs sont divisés en deux équipes. A la fin de chaque partie, les points d'une équipe sont la somme des points de chaque joueur de cette équipe.

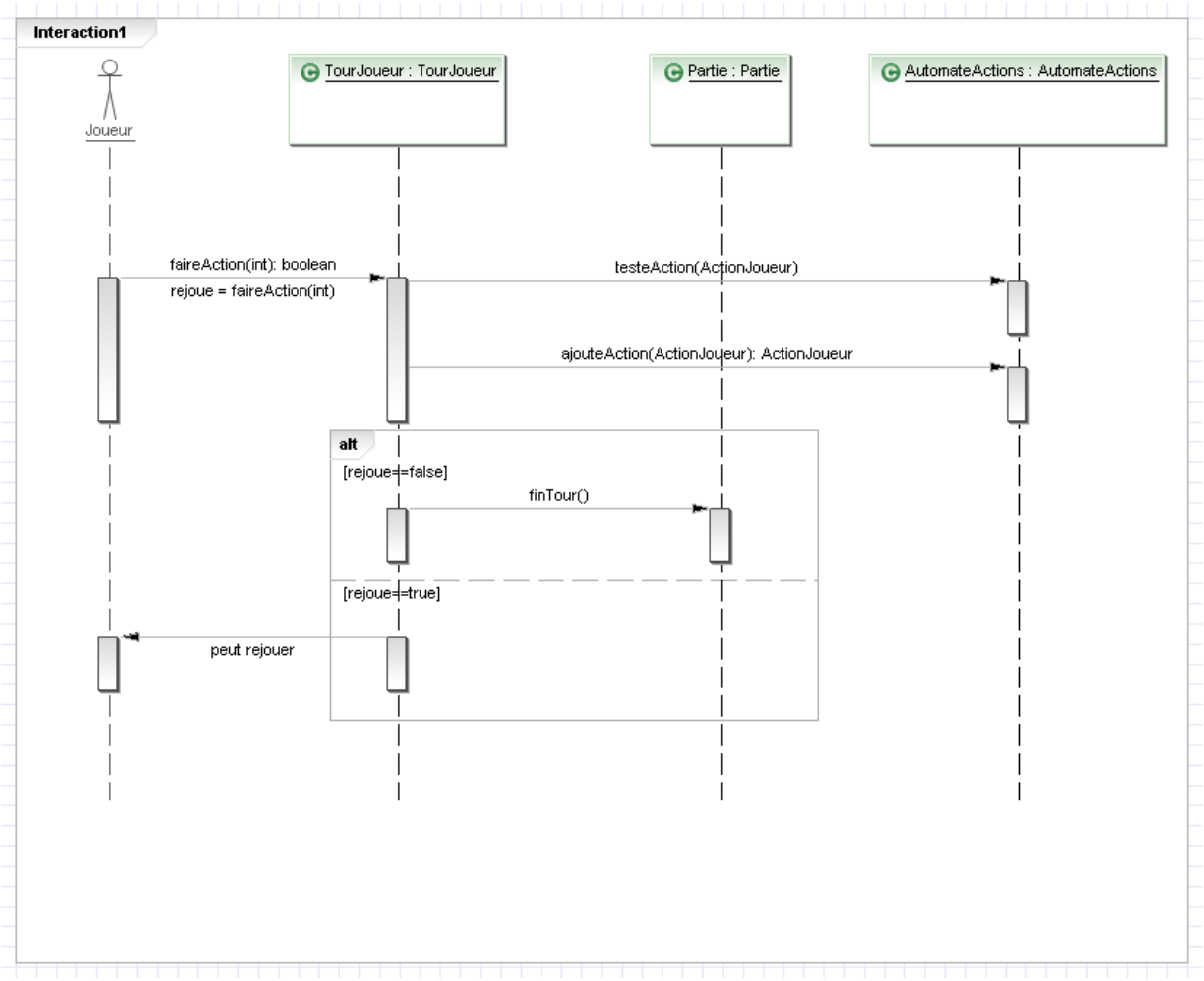
- Le duel. Deux joueurs s'affrontent, selon les mêmes règles que le mode classique, à ceci près que la carte « inversion » permet de rejouer automatiquement.

- Uno challenge. Lorsqu'un joueur dépasse 500 points, il est éliminé. Les manches se succèdent ainsi jusqu'à ce qu'il n'en reste plus qu'un.

La conception et la réalisation de ces modes de jeu fut aisée, étant donné qu'il nous suffisait d'étendre le mode de jeu classique de redéfinir certaines méthodes selon les spécificités de chaque variante.

IV] Diagramme de séquence

Le diagramme de séquence ci-dessous présente le déroulement d'un tour de jeu :



Au commencement de son tour de jeu, une liste des actions possibles est présentée au joueur. Il peut alors choisir de réaliser une action, en donnant le rang de celle-ci dans la liste des actions possibles. L'automate des actions vérifie alors que l'action est réalisable. Si elle est valide, cette action est ajoutée à la liste des actions réalisées au cours du tour de jeu, et les possibilités sont recalculées. Dans le cas contraire, une exception est levée, interrompant l'action du joueur. Lorsque le joueur réalise une action, la méthode lui indique s'il est en mesure de rejouer. Sinon, le tour signale à la partie qu'il est terminé, laquelle pourra vérifier si le joueur a gagné, et commencer le tour du joueur suivant si le jeu n'est pas terminé.

Au commencement de son tour, plusieurs actions s’offrent au joueur :

S’il est victime d’un « +4 », il doit choisir entre piocher quatre cartes ou contre-bluffer. Dans le cas où son contre-bluff se révèle correct, il peut jouer son tour normalement, sans quoi il pioche les cartes sans pouvoir jouer.

Si le joueur précédent n’a plus qu’une carte et n’a pas annoncé « Uno », le joueur peut alors annoncer « Contre-Uno », obligeant l’autre personne à piocher deux cartes.

Le joueur peut choisir de poser une seule de ses cartes, si elle s’avère être compatible avec la carte sur le dessus du talon.

Il peut également choisir de piocher une carte, une seule fois. Il pourra ensuite choisir de terminer son tour ou de poser une carte.

Si la carte sur le dessus du talon est une carte permettant de changer la couleur et qu’elle n’a pas encore été définie, le joueur peut choisir sa couleur. Ce cas peut également se produire lorsque la première carte du talon est une carte change-couleur, en début de partie.

V] Etat actuel de l'application

Au regard du cahier des charges, nous voyons que notre application atteint les objectifs demandés. En plus de réaliser tous les points requis, nous avons également développé toutes les fonctionnalités optionnelles, et toutes celles qui permettaient de l'enrichir sans s'éloigner du thème principal.

Après de nombreux essais en situation et la réalisation de tests unitaires, notre application semble également résistante aux « bugs » et autres comportements non prévus.

Notre conception permet également l'ajout de fonctionnalités futures sans difficulté, grâce notamment à l'utilisation du patron de conception Observer-Observable, enrichi par l'utilisation de l'interface « EventListener » combinée à « EventListenerList ».

La liste ci-dessous récapitule les objectifs validés par notre application :

- ✓ Modèle de jeu fonctionnel et complet de 2 à 10 joueurs humains ou virtuels.
- ✓ Intelligence artificielle programmée selon le patron de conception « Strategy ».
- ✓ Application jouable en ligne de commande.
- ✓ Interface utilisateur composée d'enchainement d'écrans : préparation de partie, affichage des scores, configuration du jeu, lecture des règles, etc…
- ✓ Interface graphique de jeu, avec animations et possibilité d'avoir plusieurs joueurs humains sur la même fenêtre, via l'utilisation d'onglets.
- ✓ Utilisation du patron de conception MVC.
- ✓ Documentation des classes, interfaces, méthodes, constantes et packages.
- ✓ Respect des conventions d'écriture spécifiées par Sun Microsystems.
- ✓ Jouable en réseau avec un chat pour communiquer entre les joueurs.
- ✓ Présence de l'application dans la barre des tâches pour affichage de notifications (si supporté par le système).
- ✓ Possibilité d'utilisation dans 5 langues : Français, Anglais, Chinois, Espagnol et Portugais.
- ✓ Propriété sur les images et sons utilisés.

VI] Conclusion

Les diagrammes dans les pages précédentes présentent un aperçu simplifié de notre réalisation, le programme complet été composé de 230 classes et interfaces, et de 14405 lignes de code. Nous avons trois pôles principaux : le modèle, l'interface utilisateur et les entrées-sorties. La réalisation de ce projet nous a permis de développer nos connaissances et notre technique dans la conception, la modélisation ainsi que la programmation orientée objet. Nous avons également eu l'opportunité de travailler en binôme.

Lors de la réalisation de l'interface graphique, nous avons cherché à la rendre la plus ergonomique possible, ainsi qu'attractive visuellement. Des animations et sons permettent de renforcer cette tendance. Les images, sons et ressources utilisées dans l'application sont également de notre réalisation. Les icônes du menu principal proviennent d'internet et sont libres de droit.