BETTAN
Etienne
EL KOURI Malak
MASCARO Lauriane
TAKI Moinesha

Projet UNO
Groupe 44

UNO

I. Fonctionnalités proposées

Lors de ce mini projet, nous avons souhaité développer un code permettant à quatre personnes de jouer ensemble au UNO. Ce jeu reprend les mêmes règles que n'importe quel jeu classique de UNO, à une exception près : la partie verbale du jeu, où chaque joueur sur le point de gagner doit l'annoncer en criant "Uno!", et où tous les autres joueurs peuvent le bloquer en s'écriant par anticipation "Contre-Uno!" ne sera pas implémentée.

Au lancement du jeu, chaque joueur rentre son nom (ou un pseudo) dans une fenêtre dessinée pour. Après appui sur le bouton "**Enregistrer**", ces noms sont mémorisés : ils serviront à identifier le joueur en cours, puis à désigner le vainqueur. Il est également possible de sauter cette étape, auquel cas les joueurs seront identifiés par des numéros.

La partie se déroule tour par tour. Le joueur peut visualiser son jeu grâce à l'interface graphique, ainsi que la carte supérieure de la défausse. Il peut ensuite choisir la carte qu'il désire jouer (si celle-ci peut être jouée) en rentrant son numéro dans le cadre prévu pour. La carte jouée est alors retirée du paquet du joueur, pour s'ajouter à la défausse.

Une carte peut être jouée si:

- elle est de la même couleur que la précédente
- elle porte le même numéro que la précédente
- c'est une carte +4 ou changement de couleur

Les cartes spéciales (+2, +4, inverser, passer, changement de couleur) ont été implémentées, les joueurs entraînent l'activation de leurs "pouvoirs" : rajouter des cartes dans le paquet des joueurs, changer le sens du jeu, faire passer le tour du joueur suivant ou changer la couleur de jeu.

Si un joueur n'a pas de carte jouable, il pioche puis passe au joueur suivant. Un écran gris s'affiche à chaque transition entre deux joueurs, demandant à passer l'appareil au joueur suivant: cela évite qu'un joueur puisse voir le paquet de cartes d'un autre.

La partie se termine lorsqu'un joueur n'a plus de cartes. En fin de jeu, le vainqueur est annoncé, ainsi que son score.

II. Problèmes rencontrés et solutions employées

Durant l'élaboration du jeu nous avons été confrontés à plusieurs problèmes, voici les plus gros bugs que nous avons rencontrés :

- **Affichage des cartes** :

Nous avons tout d'abord essayé de créer chaque carte et de lui associer un bouton à l'aide d'une succession de **if**. Mais cela s'avérait trop lourd et long. Nous avons donc préféré utiliser la méthode **paint** dans **FenetreJeu** pour dessiner les images. Pour cela nous avons modifié la classe **Carte**, en ajoutant un attribut "**image**", qui permet d'accéder au nom du fichier de l'image dans le dossier.

- **Ajout de cartes :**

Pour les cartes spéciales "**+2**" et "**+4**", la carte supérieure s'actualisant immédiatement, c'était en effet le joueur ayant posé cette carte qui récupérait le tas. Nous avons ensuite essayé de déplacer la condition, les cartes s'ajoutait à la main du bon joueur, mais celui-ci pouvait tout de même jouer une carte. La solution trouvée a donc été de créer un attribut booléen, qui devient "**true**" lorsque l'on passe au joueur suivant.

- **Choix couleur :**

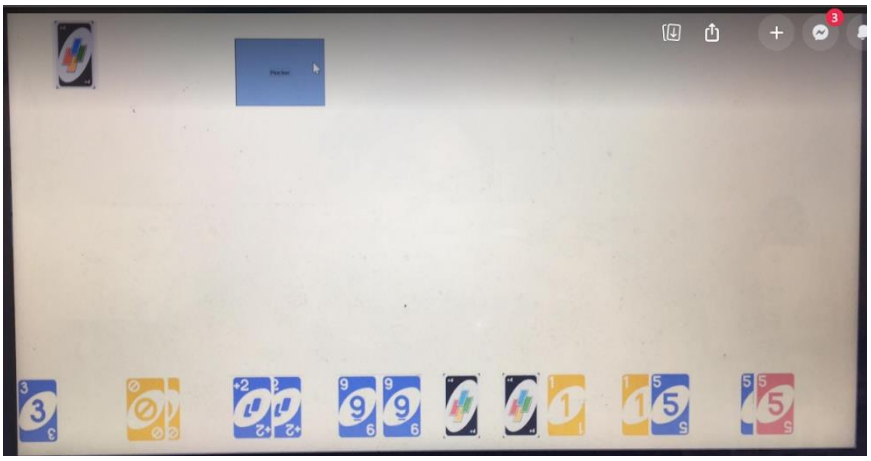
Pour les cartes "**choixCouleur**" et "**+4**", il fallait que le joueur puisse choisir la couleur de la défausse. Pour cela nous avons choisi de créer un **JTextField** pour pouvoir entrer la couleur voulue, et ainsi changer l'attribut couleur de la carte supérieure. Le fait est que la couleur n'était pas prise en compte. Pour régler ce problème nous avons écrit une méthode "**getCouleur**" dans la classe **Carte** pour pouvoir récupérer la couleur en tant que String, ainsi la couleur entrée par le joueur était bien prise en compte et le joueur suivant est obligé de jouer une carte de la couleur choisie.

- **Temps de latence entre chaque joueur :**

Nous voulions créer un temps de latence entre chaque joueur pour éviter la triche et que les joueurs n'aient pas accès au jeu des uns et des autres. Mais nous avons connu beaucoup trop de difficultés à effectuer cette fonction, nous avons donc décidé de créer une fenêtre qui se ferme lorsque le joueur appuie sur un bouton « **Passer au joueur suivant** ». Pour cela, nous avons créé un **JFrame** dans **FenetreJeu**, qui s'affiche lorsqu'on appuie sur le bouton "**joueurSuivant**".

- **Mise à jour de l'affichage :**

Lorsque le joueur pioche ou dépose une carte, ses cartes et donc l'affichage doivent être mis à jour. Le problème rencontré est visible sur la photo suivante :



Après l'ajout d'une carte dans la main du joueur, celle-ci se redessinait en ajoutant la carte, mais on voyait toujours les anciennes cartes superposées en arrière, la main se dessinait plusieurs fois. Nous avons essayé de vider la **LinkedList** "**cartesEnCours**" pour faire un **repaint()**, mais nous avions toujours le même problème. Après des recherches sur des blogs, nous avons trouvé la solution. Dans

la méthode "**paint()**", nous faisons donc appel à la méthode **super.paint(g)**. Cette solution nous permet alors d'actualiser les cartes affichées sans superposition avec l'ancien affichage.

III. Principe de l'algorithme

Le programme s'articule principalement autour de deux classes importantes : **Uno** et **FenetreJeu.**

La première permet d'initialiser le jeu, en créant le paquet de 104 cartes et en distribuant 7 cartes par joueur. Cette classe implémente aussi la méthode **piocher**, qui renvoie une carte de la pioche tout en supprimant celle-ci de la **LinkedList pioche**, on l'utilise dans **Uno** pour initier les mains initiales des joueurs. Les cartes sont des objets de type **Carte,** se sont des photos de cartes au format png, et les joueurs des objets de type **Joueur**, les 4 joueurs appartiennent au tableau **joueur**, de type **Joueur**. La méthode **main** de la classe **Uno** va aussi créer une **FenetreMenu**, en lui envoyant le tableau de joueurs et la pioche. Cette fenêtre a pour but de permettre de lancer la partie. C'est la première fenêtre apparaissant lorsqu'on lance le programme. Elle va ensuite créer une fenêtre **NomJoueurs** en lui envoyant la pioche et le tableau de joueurs, cette classe sert à taper et enregistrer le pseudo des joueurs. La fenêtre **FenetreJeu** s'ouvre et la partie commence lorsqu'on appuie sur le bouton **Commencer** de **NomJoueurs**, qui crée une **FenetreJeu**.

La classe **FenetreJeu** est la classe contenant une grande partie de la logique du jeu, ainsi que son déroulement. Ainsi le constructeur **FentreJeu** permet de créer ce qui sera l'interface principal du joueur, c'est-à-dire la pioche, la défausse ainsi que le menu qui lui permet d'interagir. L'attribut LinkedList **cartesEnCours** stocke les cartes du joueur en cours puis se vide avant de passer au joueur suivant. Au début de la partie la méthode **partie** permet de mettre une première carte sur la défausse de telle manière à ce qu'elle ne soit pas une carte de type "**+2**", "**+4**", "**choixCouleur**", c'est ensuite au tour du premier joueur. La méthode **paint()** permet d'afficher son jeu sur le plateau. Le joueur a alors plusieurs possibilités :

- piocher : on fait appel à la méthode **piocher**, ce qui donne une carte au joueur et enlève la carte supérieure de la pioche. Le joueur ne peut pas rejouer car le bouton **jouer** devient invisible.
- jouer : le joueur choisit le numéro de la carte qu'il souhaite jouer dans un **JTextField num**. La méthode **jouableVoid()** vérifie si cette carte peut être jouée (compatibilité avec la carte supérieure de la défausse). Si elle ne peut être jouée il s'affiche un message d'erreur, demandant au joueur de piocher. Dans le cas contraire, la méthode **carteSuperieure** enlève la carte sélectionnée de la main du joueur et l'ajoute sur le haut de la défausse. Si cette carte était une carte spéciale, le déroulement du jeu se fait dans le **actionPerformed**.

Pour faire fonctionner le jeu et l'affichage, la méthode **majCartesEnCours** permet d'actualiser la liste qui correspond aux cartes du joueur dont c'est le tour. La méthode **avancer** permet de passer au joueur suivant. La méthode **actionPerformed** permet de détecter quand le joueur effectue une action mais intègre aussi le déroulement du tour selon le schéma vu précédemment.

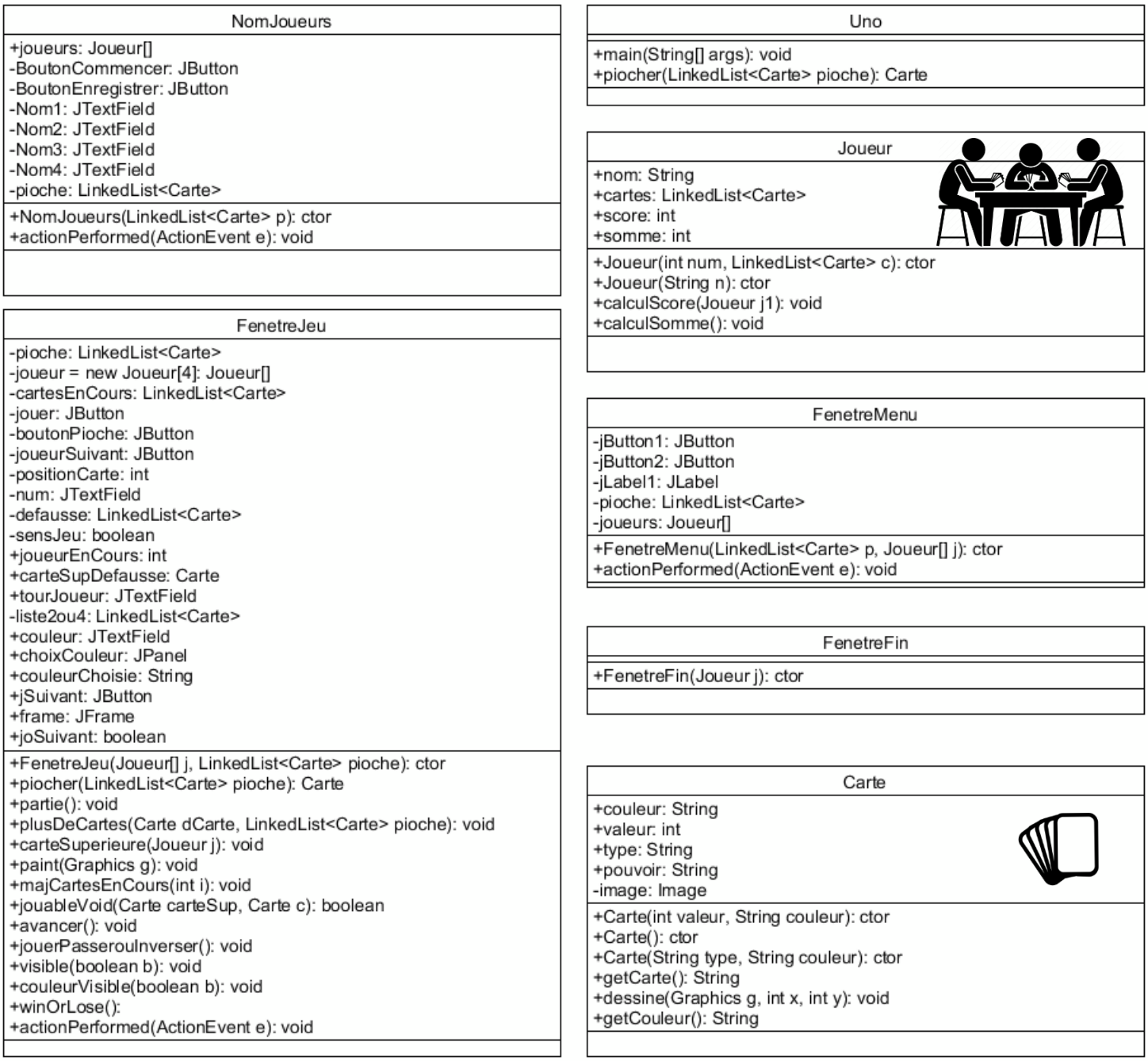


Figure 1. diagramme UML du UNO

IV. Suggestion d'améliorations

Nous avons au cours du projet rencontré un certain nombre de difficultés techniques, de ce fait nous avons, parfois à cause d'elles, souvent par manque de temps, été contraints de supprimer certaines fonctionnalités que nous souhaitions initialement mettre en place.

Par exemple, dans le cahier des charges proposé au début du projet, nous suggérions de coder une interface graphique beaucoup plus interactive, à travers laquelle le joueur aurait pu jouer en cliquant directement sur la carte souhaitée. Nous avons tenté de mettre en place cette solution, en

superposant aux cartes affichées des **JButton** pour les rendre cliquables. N'y étant pas parvenus, nous avons remplacé cela par le système que vous avez pu expérimenter, où chaque joueur rentre le numéro de la carte qu'il souhaite jouer. Cette solution est simple et fonctionnelle, mais moins agréable pour l'utilisateur. Il serait donc pertinent de retravailler notre code afin d'implémenter la solution envisagée initialement.

Nous pourrions également ajouter la possibilité de choisir le nombre de joueurs souhaité (dans la fenêtre dans laquelle on entre le nom des joueurs par exemple). En effet, nous avons initialement choisi 4 car c'est une configuration courante, et que nous voulions nous concentrer d'abord sur le déroulement du jeu. Cependant, ce choix est une fonctionnalité assez simple et utile à ajouter.

Enfin, il faudrait améliorer encore l'interface graphique pour afficher la taille des paquets de cartes de tous les joueurs, afin que chaque personne puisse se rendre compte du nombre de cartes restant à ses adversaires.

V. Carnet de route

- **08/03: séance 1**
  - réflexion sur la structure globale du code, mise en place d'un diagramme UML
  - implémentation des classes **Carte**, **CarteChiffre**, **CarteSpeciale**, **Joueurs**
  - début d'implémentation de la classe nommée **Uno** contenant le **main**. Cette classe contenait initialement toutes les méthodes nécessaires au déroulement du jeu, qui ont par la suite été déplacées dans **FenetreJeu**

- **15/03: séance 2**
  - Nous avons finalement abandonné les classes héritières de **Carte** (**CarteChiffre** et **CarteSpeciale**) : cette séparation aurait compliqué inutilement notre code, il nous a semblé plus pertinent de ne conserver que la classe ancêtre **Carte** en y définissant deux constructeurs distincts
  - Initialisation des paquets de carte des joueurs, de la pioche et de la défausse dans **Uno**
  - Suite du travail sur les méthodes nécessaires au déroulement du jeu dans **Uno**
  - Début du travail sur l'IHM: création des classes **FenetreJeu** et **Deck** (finalement abandonnée)

- **22/03: séance 3**
  - Création de la classe **NomJoueurs**, permettant aux joueurs de s'identifier en début de chaque partie
  - Modification des méthodes gérant l'utilisation des cartes spéciales, pour les adapter à toutes les situations prévues par les règles du jeu (par exemple, si un joueur reçoit un +2, il ne pioche deux cartes que s'il ne joue pas un +2 derrière)

- **29/03: séance 4**
  - Modification des constructeurs de **Carte** pour associer à chaque carte son image, et ainsi pouvoir les afficher dans **FenetreJeu**
  - Début de modification et simplification de la structure du code pour résoudre les problèmes rencontrés avec l'IHM

- **05/04: séance 5**
  - Résolution de bugs sur les actions spéciales (+2 et +4 notamment)
  - Mise en œuvre des changements de couleur, jusqu'alors laissés de côté

  - Poursuite des modifications de l'IHM

- **26/04: séance 6**
  - Implémentation de la classe **FenetreFin**, permettant l'affichage du vainqueur et de son score
  - Amélioration de l'aspect esthétique de l'interface graphique
  - Dernière tentative pour rendre les cartes cliquables
  - Résolution de bugs

VI. Bibliographie

- JavaDoc
- Images des cartes, et règles du jeu: https://www.regledujeu.fr/uno/
- Accés aux images dans la classe Carte: https://www.tutorialspoint.com/java_dip/java_buffered_image.htm
- Fenêtre entre chaque joueur : Fermer une JFrame en Java avec un bouton - WayToLearnX

VII. Implication de chaque membre du groupe

Nom	Implication
Etienne BETTAN	20
Malak EL KOURI	30
Lauriane MASCARO	20
Moinesha TAKI	30