Jeu Hanabi en réseau

|  |  |
| --- | --- |
| **Nom et Prénom** |  |
| **Classe** | 3INF-IP-M |
| **Enseignant** | A. Vadi |
| **Expert Principal** | D. Antognini |
| **Expert Auditeur** | A. Jeanmaire |
| **Date de début** | 24 avril 2022 |
| **Date de fin** | 19 mai 2022 |
| **Durée Total** | 80 Heures / 110 périodes |

Table des matières

[1. Introduction 3](#_Toc103869916)

[2. Architecture réseau 4](#_Toc103869917)

[3. Technologies Utilisées 5](#_Toc103869918)

[3.1. Networkcomms 5](#_Toc103869919)

[3.2. Sérialisation / Désérialisation 6](#_Toc103869920)

[4. Architecture logicielle 7](#_Toc103869921)

[4.1. Schéma des classes 7](#_Toc103869922)

[4.2. Classe « Partie » 8](#_Toc103869923)

[4.3. Classe « zone de point » 9](#_Toc103869924)

[4.4. Classe « Joueur » 10](#_Toc103869925)

[4.5. Classe « Cartes » 11](#_Toc103869926)

[4.6. Classe « Pioche » 12](#_Toc103869927)

[5. Fonctionnement des méthodes 13](#_Toc103869928)

[5.1. Jouer une carte 13](#_Toc103869929)

[5.2. Défausser une carte 15](#_Toc103869930)

[5.3. Méthode donner un indice 17](#_Toc103869931)

[5.4. Méthode déplacer une carte 20](#_Toc103869932)

[6. Processus de travail 22](#_Toc103869933)

[6.1. Architecture du programme 22](#_Toc103869934)

[6.2. Nomenclature 24](#_Toc103869935)

[6.3. Interface graphique 25](#_Toc103869936)

[6.4. Détail graphique 26](#_Toc103869937)

[6.5. Fin de la partie 27](#_Toc103869938)

[7. Protocole de test 28](#_Toc103869939)

[7.1. Connexion 28](#_Toc103869940)

[7.2. Démarrer une partie 29](#_Toc103869941)

[7.3. Règle de la partie 29](#_Toc103869942)

[7.4. Jouer une partie 30](#_Toc103869943)

[7.5. Déconnexion 32](#_Toc103869944)

[8. Problèmes rencontrés et solutions 33](#_Toc103869945)

[8.1. Connexion client-serveur 33](#_Toc103869946)

[8.2. Affichage des jokers dans la zone point 33](#_Toc103869947)

[9. Améliorations possibles 34](#_Toc103869948)

[9.1. Gérer les déconnexions 34](#_Toc103869949)

[9.2. Ajout d’autre extension 34](#_Toc103869950)

[9.3. Amélioration graphique 35](#_Toc103869951)

[9.4. Indiquer à l’utilisateur si une action n’est pas valide 35](#_Toc103869952)

[10. Conclusion 36](#_Toc103869953)

[11. Annexes 37](#_Toc103869954)

[11.1. Journal de bord 37](#_Toc103869955)

[11.2. Cahier des charges 37](#_Toc103869956)

[11.3. Références 37](#_Toc103869957)

## Introduction

Dans le cadre du TPI de fin de formation, le mandat qui m’est attribué est de recréer en C# Winform le jeu de carte Hanabi. L’objectif du projet est de montrer que l’on est capable de suivre un cahier des charges donné et de réaliser le développement d’une application complexe.

La particularité principale de ce mandat est d’ajouter au jeu Hanabi une dimension réseau et un mode multijoueur. Il s’agit, de pouvoir jouer en suivant les règles de base du jeu de carte de 2 à 5 joueurs. Les possibilités du jeu sont les suivantes : Jouer une carte dans la zone de point, défausser une carte et enfin révéler un indice. Les règles de base d’Hanabi devront être suivies à la lettre. En plus de ces actions, le cahier des charges donne la possibilité à l’utilisateur de déplacer ses cartes et de les marquer.

Pour la partie technique du projet, il est demandé une programmation orienté objet. Un protocole de test précis testant toutes les fonctionnalités du programme est aussi requis.

Le planning du projet est d’environ 100h. Du fait que des délais doivent être respectés, une planification du travail est attendue. Etant donné qu’il peut y avoir des imprévus, les planifications initiale et finale apparaitront dans ce rapport.

## Architecture réseau

Le programme développé utilise une architecture de type client-serveur. Au sein du réseau CPLN, des postes de travail exécuteront le jeu Hanabi qui se connectera sur un serveur.

Le protocole utilisé est TCP et la couche réseau est IP.

Diagram

Description automatically generated

Dans la démonstration, le joueur 1 exécute le programme client et serveur en même temps sur le même poste de travail. Les adresses IP sont automatiquement fournies par le DHCP du CPLN.

Les postes des joueurs 2 et 3 ont uniquement le jeu (client) lancé sur leur machine. Leurs adresses IP leurs sont aussi fourni par le DHCP de l’école.

Au CPLN, les adresses peuvent se trouver sur deux sous-réseau différents le premier 157.26.173.x et le deuxième en 157.26.174.x. L’attribution de ces adresses se fait aléatoirement, il est donc impossible de savoir à l’avance sur quel sous-réseau nous serons.

## Technologies Utilisées

* Visual Studio
* C# Winform
* Librairie Networkcomms
* Librairie NewtonSoft.Json
* Git

### Networkcomms

Cette librairie permet de mettre en place facilement des architectures client-serveur en C#. Elle permet de résoudre le problème du mode multi joueur. La librairie fonctionne ainsi :

Le jeu Hanabi qui s’exécute sur l’ordinateur du joueur est un client qui se connecte à un serveur. Le serveur va abonner la connexion du client a un ou plusieurs évènements identifiés par des identifiants et représenté par une méthode.



Ici le serveur abonne la connexion de chaque client a l’identifiant PlayCard. Ce paquet doit contenir une chaine de caractère et appelle la méthode PlayCard une fois reçu.



Du côté du client, lorsque l’action « jouer une carte » est réalisé par l’utilisateur, un paquet avec comme identifiant PlayCard et qui contient une chaine de caractère est envoyé.

Toute les actions que les clients peuvent faire correspondent à un identifiant différent. Et inversement, le client est abonné à toutes les réponses du serveur.

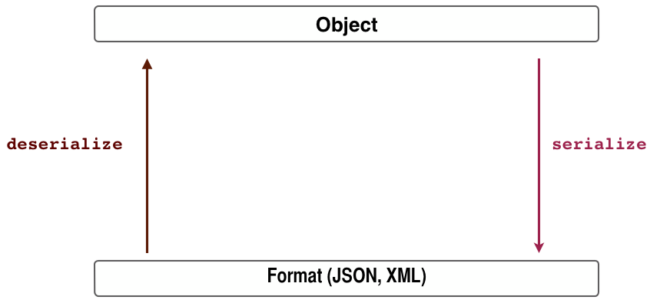


Une fois la partie terminée, il convient de désabonner les connexions pour ne plus recevoir de paquet. Ainsi fonctionnent les connexions client-serveur de la librairie Networkcomms.

### Sérialisation / Désérialisation

La libraire Networkcomms ne permet d’envoyer que des variables basiques (des chaines de caractère, des entiers ou des booléens). Pour les besoins du programme, des types plus complexes comme « Game » ou « Card » doivent d’être envoyé sur le réseau.

Grâce à librairie NewtonSoft.Json, il est possible de sérialiser un objet, c’est-à-dire de le transformer en chaine de caractère ou inversement de le désérialiser (retrouver l’objet initiale).



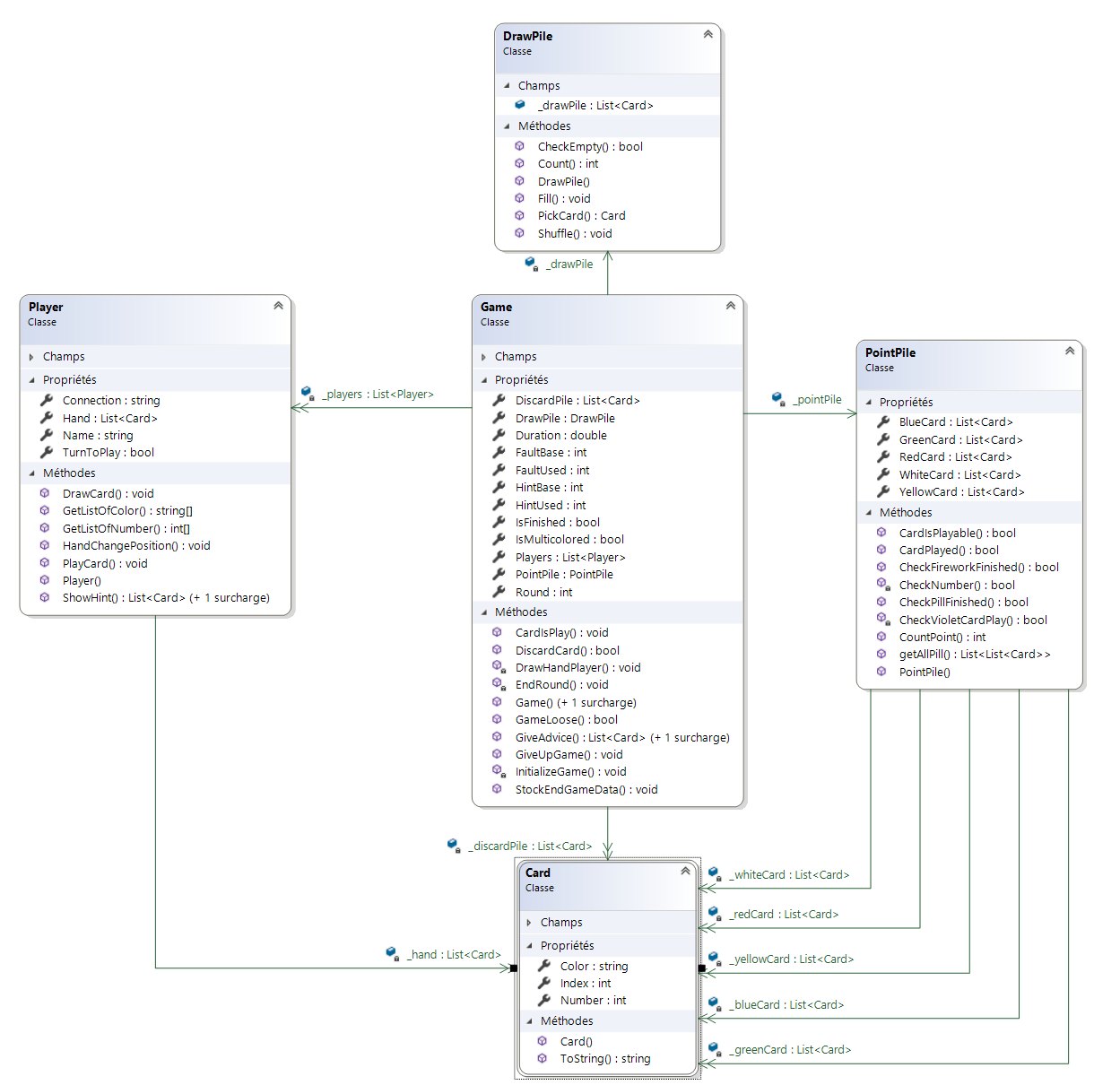
Ci-dessus, un petit schéma pour représenter le fonctionnement de la Sérialisation.

|  |  |
| --- | --- |
| Graphical user interface, text, application, chat or text message  Description automatically generated | {  \"Index\" : 41,  \"Color\" : \"green\",  \"Number\" : 1  } |

Ci-dessus, vous pouvez voir a quoi ressemble une carte sérialiser dans le programme.

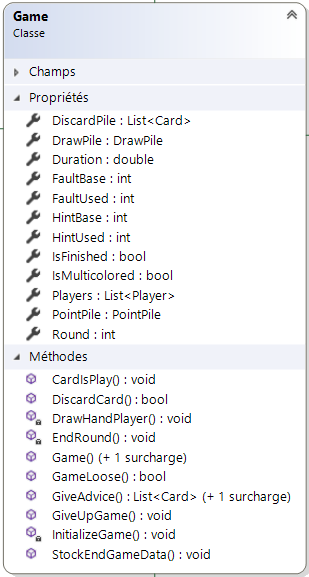
## Architecture logicielle

### Schéma des classes



Voici a quoi ressemble la liste des classes. Vous trouverez ci-dessous, une description plus précise de chaque classe. Pour les besoins de la sérialisation / désérialisation des objets, toutes les propriétés doivent être « public ».

### Classe « Partie »



La classe « Game » a deux constructeurs. Le premier vide est là pour les besoins de la Sérialisation. Le second permet de créer une partie. Pour démarrer une partie, il faut : une liste de joueur, le nombre d’indice a disposition, le nombre de fautes autorisées et l’utilisation ou non des cartes multicolores.

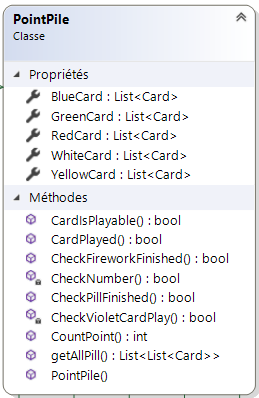
Dans ce deuxième constructeur, la méthode « Initialize Game » est appelée. Cette méthode va d’abord crée les cartes avec lesquelles les joueurs joueront. 50 cartes en mode normal, 55 cartes avec les jokers. Qu’il ajoute à la pioche et il mélange la pioche avec la méthode « Shuffle Card » (méthode statique). La méthode « DrawHandPlayer » qui va ajouter le bon nombre de carte, en fonction du nombre de joueur, à chaque joueur.

Les méthodes CardIsPlay, DiscardCard et GiveAdvice, seront appelé par l’utilisateur lorsque qu’il jouera ces cartes. Pour les deux premières méthodes, il est passé en paramètre la carte qui est joué et le joueur qui a fait l’action. Ces mêmes méthodes ont pour conséquence de faire piocher le joueur. Pour la dernière méthode, il est demandé le joueur qui donne l’indice, le joueur qui reçoit l’indice et quel indice il faut donner. A la fin de chacune de ces méthodes, la fonction « EndRound » est appelé. Cette méthode vérifie si la partie est fini ou non, si la partie n’est pas terminé, elle modifie le joueur qui doit jouer le tour. Sinon elle modifie l’attribut IsFinished.

Il y a quatre moyens de terminer une partie. La première les joueurs décident d’abandonner la partie, méthode GiveUpGame. Les autres moyens de finir la partie sont : les joueurs remplissent l’intégralité des zones de point, les joueurs ont fait trop d’erreurs ou que la pioche est vide.

A la fin d’une partie, le cahier des charges demande de stocker certaines informations dans un fichier csv. La méthode StockEndGameData est donc appelée. Les données sauvegardées sont : les différents joueurs qui ont participé à la partie, le nombre de point final, le temps de la partie, la date de la parte et pour finir les règles choisies au départ.

### Classe « zone de point »



Dans cette classe, le constructeur sert uniquement a instancier les listes, il n’a besoin d’aucun paramètre.

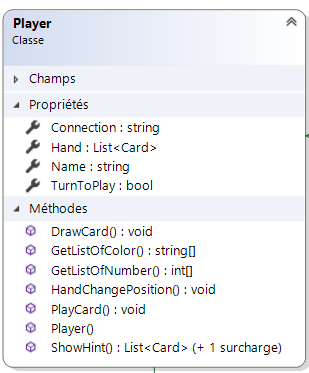
Lorsque l’utilisateur joue une carte dans la zone de point, la méthode CardIsPlayable est appelée. Elle vérifie dans le tas de la bonne couleur si le numéro de la carte est le bon. Par exemple, si la carte 3 bleue est jouée, le jeu récupére la liste BlueCard, et regarde s’il le 2 bleu a déjà été joué. Si c’est le cas, la méthode laisse le coup se jouer. Dans le cas d’une carte violette, la méthode CheckVioletCardPlay est appelée, elle vérifie dans chaque tas 1 par 1 si la carte peut être jouée.La méthode s’arrête au premier tas positif et ne vas pas vérifier dans les tas suivants.

Après avoir eu la confirmation que la carte est jouable, la méthode CardPlayed est appelée, elle ajoute la carte au tas de la bonne couleur. Cette méthode appelle aussi la méthode PillFinished. Elle permet de savoir si le tas est fini à ce tour, c’est-à-dire que la carte numéro 5 est bien posée sur le tas. Si c’est le cas, un indice est ajouté comme le veut les règles.

A chaque fin de tour, la méthode CheckFireworkFinish est appelée, si les 5 tas sont remplis avec les 5 cartes, la partie s’arrête et les joueurs ont gagné.

A la fin d’une partie, il faut afficher le nombre de points réalisés par l’équipe, la méthode CountPill retourne cette information.

### Classe « Joueur »



Le constructeur du joueur demande en paramètre une connexion et un pseudo (chaine de caractère) fourni en début de partie par le joueur. La connexion est utilisée pour savoir quel utilisateur exécute quelle action.

A chaque fin de tour, si l’action le demande, la méthode DrawCard est appelée, elle ajoute une carte à la main (Hand) du joueur.

Les méthodes GetListOfColor et GetListOfNumber, retournent la liste de couleur ou les nombres uniques dans la main du joueur. Elle sont utilisées juste avant l’envoi d’un indice. La liste permet de créer l’indice à envoyer.

Une fois l’indice transmit, la méthode ShowHint est appelée. Elle retourne soit la liste des cartes avec une couleur ou un chiffre spécifique. Cette liste est ensuite affichée chez la personne ayant reçu l’indice.

La méthode HandChangePosition est appelée quand un joueur décide de déplacer une carte de sa main. Cette méthode va permettre de déplacer la carte à l’emplacement voulu par le joueur.

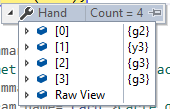
### Classe « Cartes »

Graphical user interface, text, application, chat or text message

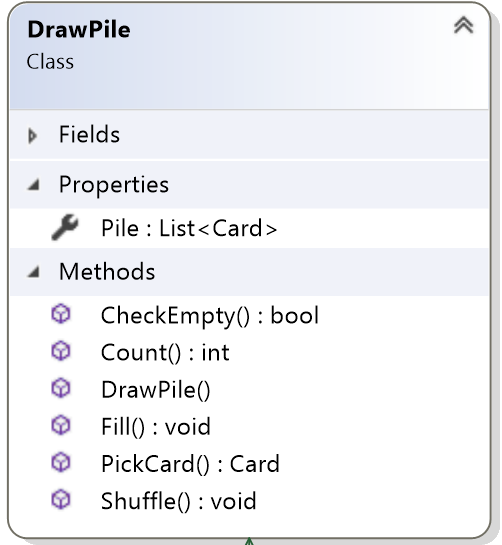
Description automatically generated

Le constructeur de la carte demande de passer en paramètre, la couleur de la carte, le chiffre de la carte ainsi que l’index de la carte qui sera utilisé pour comparer les cartes.

La méthode ToString est surchargée pour faciliter le débogage. En mode débogage, lorsqu’une liste de carte est affichée, la méthode ToString est appelée pour chaque enregistrement. La méthode retourne, la première lettre de la couleur de la carte en anglais et son chiffre. Par exemple a carte 2 verte s’écrit g comme green et le 2 qui correspond à son chiffre.



### Classe « Pioche »



Le constructeur de la classe ne demande aucun paramètre. Il instancie juste la liste de Carte Pile. Il n’existe qu’une seule DrawPile par partie.

Au tout début de la partie, la méthode Fill est appelée. Elle permet de remplir la pioche de carte. Une fois remplie, la méthode Shuffle est appelée. Elle mélange aléatoirement le tas.

La méthode Shuffle utilise l’algorithme de mélange de Fisher-Yates. Qui fonctionne le selon le principe suivant :

Pour i allant de « taille de liste » − 1 à 1 faire :

J ← entier aléatoire entre 0 et i

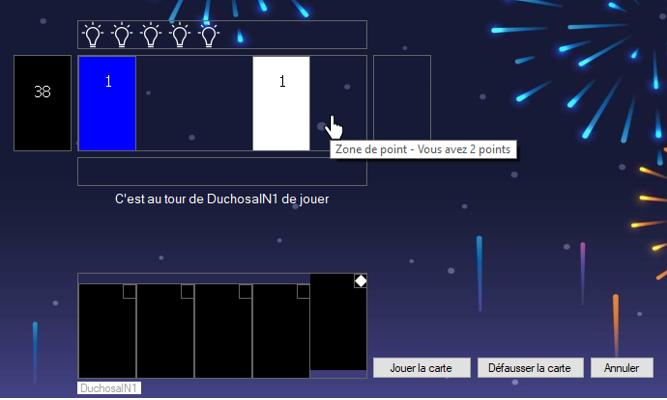
Échanger a[j] et a[i]

Pour afficher la pioche, la méthode count est appelée, elle retourne le nombre de cartes restantes dans la pioche.

A chaque fin du tour, si l’action le demande, la méthode PickCard est appelée. Elle retourne la carte que le joueur pioche. La carte est en même temps ajoutée à la main du joueur et retirée de la pioche.

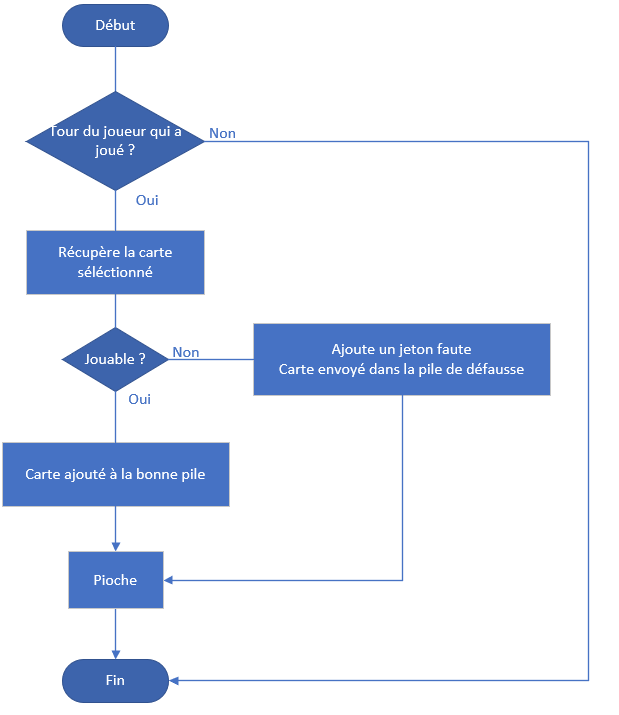
## Fonctionnement des méthodes

### Jouer une carte



Du coté client

Pour jouer une carte, vous sélectionnez une carte de votre main, puis vous appuyez soit dans la zone de point, soit sur le bouton « jouer la carte ». La carte et la méthode « PlayCard » sont envoyés au serveur qui vérifie si l’action est acceptable.



Du coté serveur

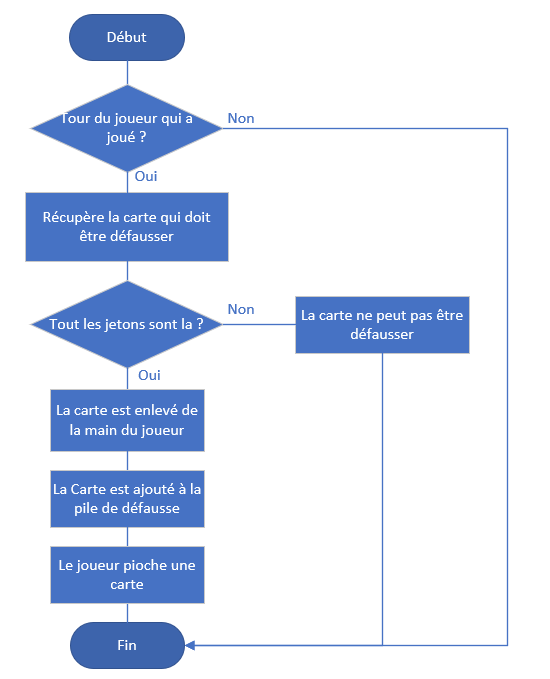
La méthode fonctionne selon la logique suivante : On contrôle si c’est bien le tour du joueur, on récupère la couleur de là de la carte, on regarde si le numéro est bien supérieur de 1 de la carte précédente. Si c’est le cas la carte est jouée, sinon elle est défaussée et une faute est comptabilisée.

### Défausser une carte



Du coté client

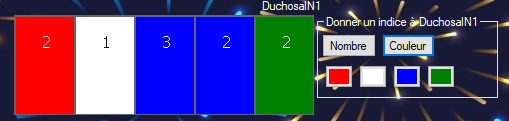
Pour défausser une carte, vous devez sélectionner la carte, puis appuyer soit dans la pile de défausse, soit sur le bouton « défausser la carte ». La carte est sérialisée et envoyée sur au serveur avec la méthode « DiscardCard ».



Du coté serveur

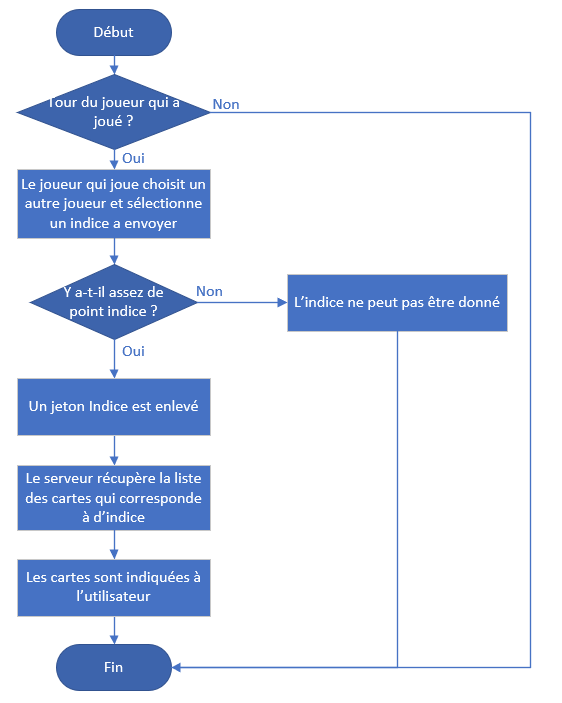
La méthode « DiscardCard » vérifie si au moins un jeton indice a été utilisé, si c’est le cas alors la carte est défaussée sinon il ne se passe rien et le joueur doit faire une autre action.

### Méthode donner un indice



Du coté client (joueur actif)

Pour donner un indice, vous sélectionnez la personne à laquelle vous désirez envoyer votre indice en appuyant sur son nom. Ensuite vous choisissez qu’elle type d’indice (nombre ou couleur) vous voulez envoyer. En fonction du type choisi, une liste de couleurs ou une liste de nombre est affiché. Dans le cas ci-dessous nous envoyons un indice bleu. Toutes les informations de l’envoie d’indices sont sérialiser et envoyer au serveur avec la méthode « SendHint ».



Du coté serveur

Pour le fonctionnement de la méthode, Le serveur regarde s’il y a des indices a dispositions. Il récupère la main du joueur qui reçoit l’indice. Il récupère la liste des cartes qui correspondent à la demande et les retourne au receveur d’indice pour qu’il puisse les afficher. Si aucun indice n’est disponible rien ne se passe et le joueur doit faire une autre action.

A picture containing graphical user interface

Description automatically generated

Du coté client (receveur d’indice)

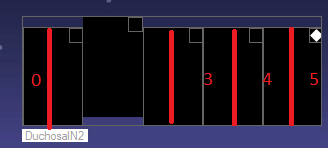
En retour, la personne à qui l’indice est destiné (il ne voit pas ses cartes), les cartes impactées par l’indice seront marquées. Dans l’exemple ci-dessous, les cartes deviennent bleues. Les cartes ne sont marquées que pour un seul tour et les indices disparaissent à la fin du tour.



Du coté client (autres joueurs)

Pour les autres clients, un message indique l’indice est le nom des joueurs concernés.

### Méthode déplacer une carte



Du coté client

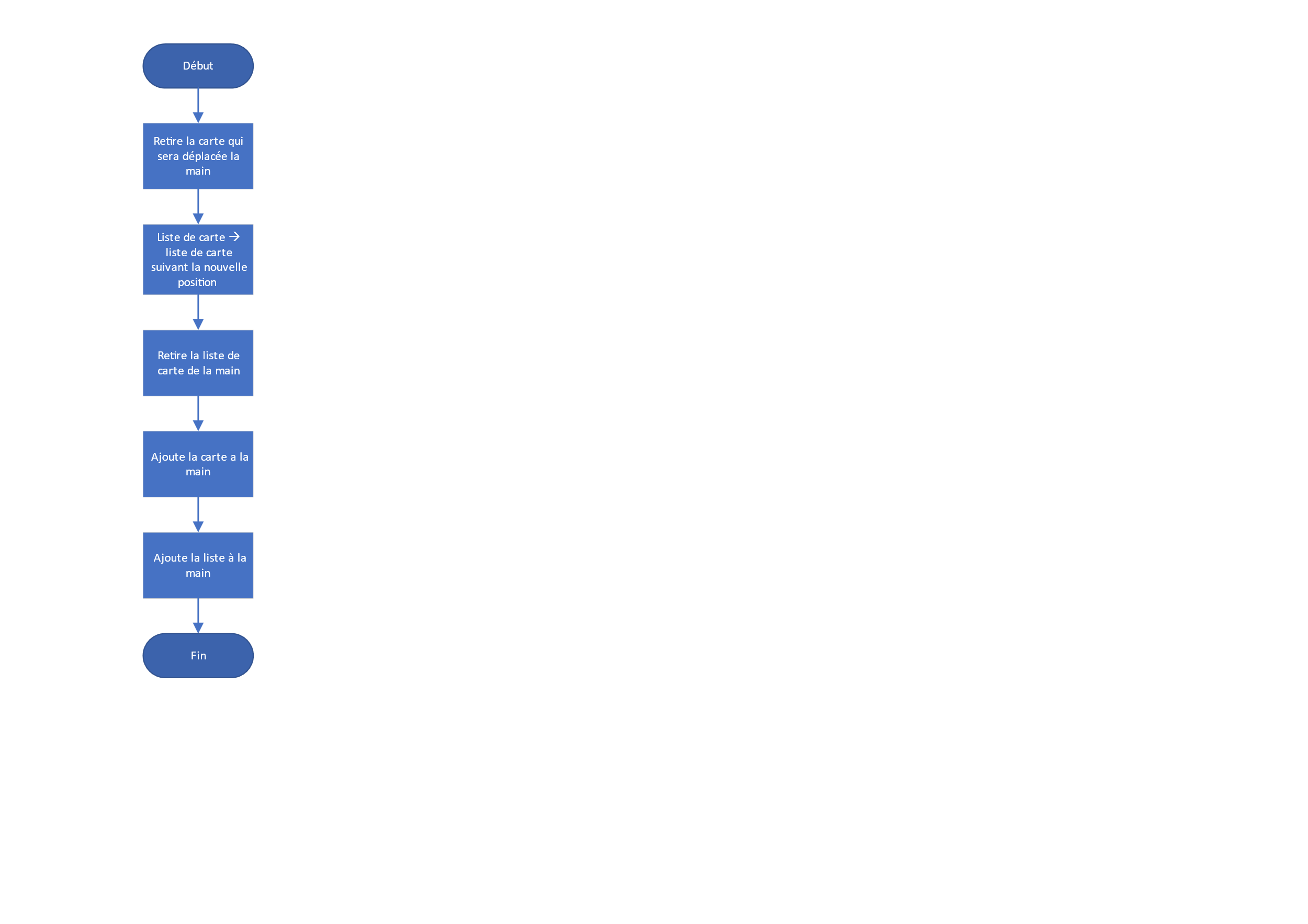
La méthode de déplacement d’une carte fonctionne selon le principe suivant. Vous sélectionner votre carte, elle est mise en évidence. A l’aide du clic droit de la souris, vous pouvez sélectionner la zone dans laquelle la carte ira se placer. Dans ce cas, si l’utilisateur décide d’appuyer dans la zone 0, la carte se placera à gauche de la carte.

Text, letter

Description automatically generated

Dans le code

La position de la carte est calculée en fonction de la position du curseur de la souris dans le panel contenant les cartes du joueur. La nouvelle position de la carte est ensuite envoyée au serveur aves la méthode « HandChangePosition ».



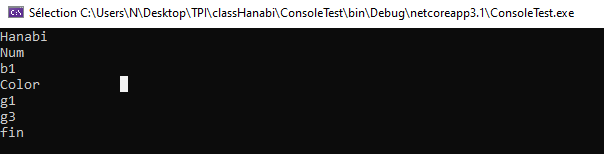
Du coté serveur

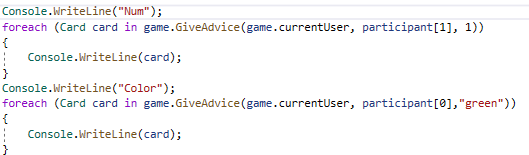
La méthode fonctionne ensuite par la logique suivante, la carte qui est déplacée est retiré de la main. Une liste temporaire contenant la liste des cartes allant de la carte qui se trouve à la nouvelle position de la carte, jusqu’à la dernière carte de la main est créé. Cette liste est ensuite retirée de la main. La carte qui est déplacé est rajouté à la main et pour finir la liste temporaire est aussi rajouté à la main.

## Processus de travail

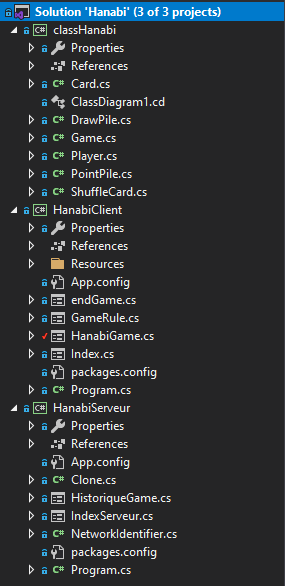
### Architecture du programme

Comme demandé dans le cahier des charges, une implémentation orienter objet a été mise en place. Au début de la programmation, les classes ont été créer dans un projet séparé. Cela a permis de mettre en place les différents attributs et méthodes. Toute les classes ont ensuite été implémenté et tester dans un projet console pour que les classes ne soit pas lié à une interface graphique.





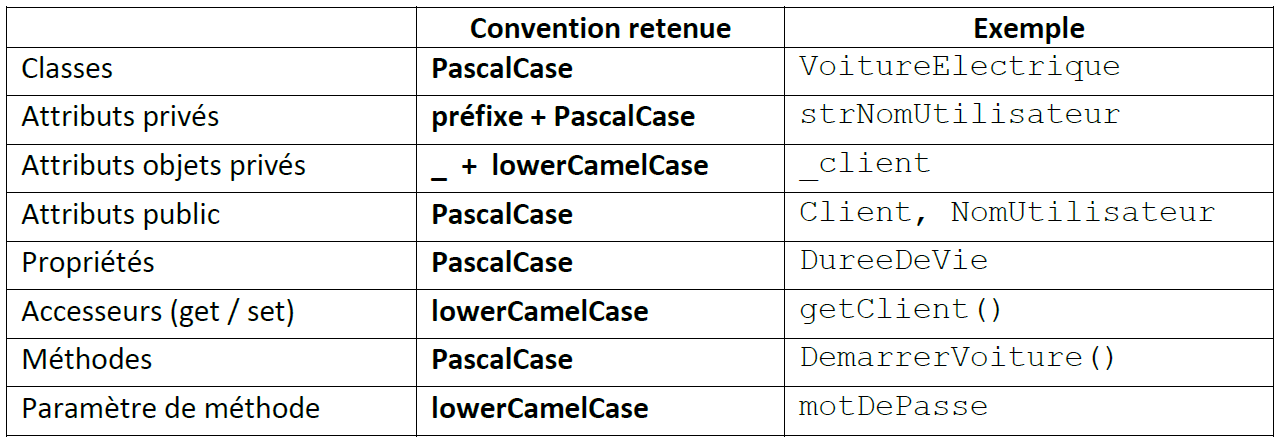
Dans le cas si dessus, le programme console affiche la liste des cartes numéro 1 dans la main du joueur 2 et la liste des cartes vertes du joueur 1.



Au final le programme contient 3 projets dans sa solution. Le premier projet contient toutes les classes où la logique du jeu est mise en place. Le deuxième projet contient le client qui affiche la partie et permet à l’utilisateur de faire des actions. Le dernier projet contient lui le serveur qui va faire le lien entre les différents clients et la logique de jeu.

### Nomenclature

La nomenclature pour toutes les classes a été suivit selon les nomenclatures vues dans le module 226A.

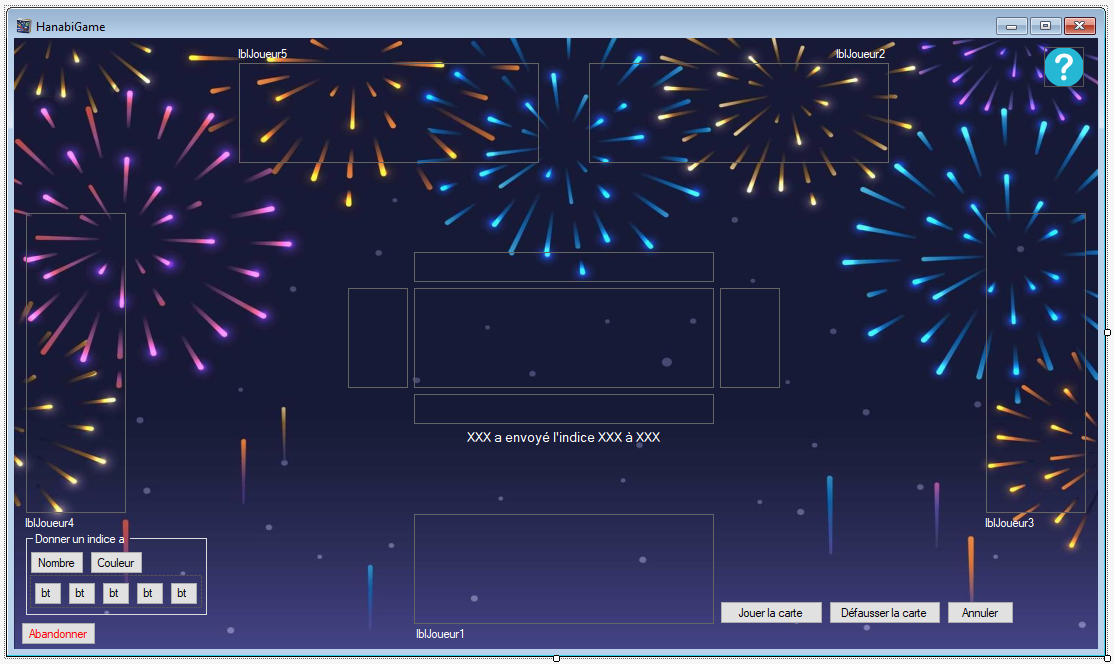


Pour ce qui est de la nomenclature suivit dans le programme en lui-même, toutes les variables sont typées et nommées en fonction de leur utilité. Utilisé la même convention de nommage dans le code aurait été un travail énorme qui prend énormément de temps. Comme le temps pour se projet est assez

Dans le code, l’anglais est utilisé comme langue principale. Toute les variables et méthodes sont nommé en anglais. Pour le texte qui est affiché au client et les commentaires du code, ils sont eux en français.

### Interface graphique

Au niveau de l’interface graphique, La liste de cartes qui se situe en bas de l’écran est toujours celle de l’utilisateur qui peut interagir sur le client. L’ordre d’affichage se fait ensuite dans l’ordre inverse des aiguilles d’une montre. La personne qui joue après vous, se situe toujours à votre droite. Pour mettre en place cette mécanique, un monde personnel est envoyé à chaque client. Dans ce monde personnel, la liste d’utilisateur est réorganisée pour que le joueur qui l’a reçoit soit toujours à l’emplacement 0 de la liste.



Tous les conteneurs de Control sont affichés au préalable. Tous les Conteneurs qui ne sont pas utilisé lors d’une partie son cachés. Au lancement de la partie et a chaque fin de tour, le monde entier est envoyé et redessiner complétement sur chaque client.

Le nombre de joueur peut varier entre 2 et 5 personne. L’affichage est différent en fonction du nombre de personne. Par exemple, durant une partie à 4 joueurs, un conteneur est toujours placé en bas d’écran. Deux conteneurs sont placés verticalement, le premier a droite et le second à gauche. Et le dernier conteneur est alignées avec le premier mais en haut de la fenêtre. A 5 joueurs la partie du haut accueille 2 joueurs.

### Détail graphique

Il est indiqué sur la pioche le nombre de carte qui y reste.



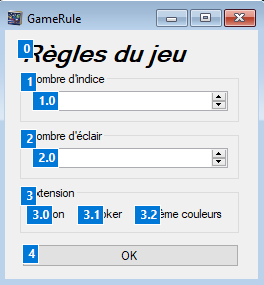
Quand une carte est sélectionnée dans la main du joueur, elle est un peu surélevée pour monter de quelle carte il s’agit



Le pseudo de la personne qui peut jouer durant le tour est surligné d’une autre couleur.

|  |  |
| --- | --- |
| Joueur actif | Joueur autre joueur |
|  |  |

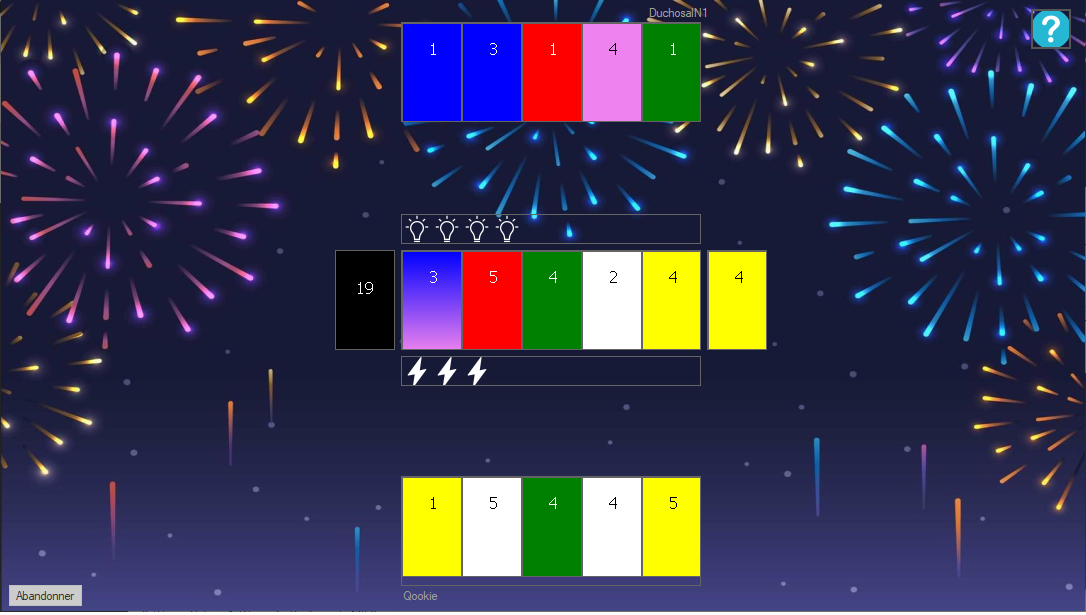
Et pour finir, il est important de noter que l’ordre de tabulation est respecté sur tous les formulaires. Ci-dessous, vous pouvez voir un exemple avec le formulaire pour envoyer un indice au serveur.



### Fin de la partie



A la fin de la partie, il était précisé dans le cahier charges que le nombre de point fait par l’utilisateur. Cette information est affichée avec un message qui change en fonction du nombre de point que vous avez fait. Une petite animation de feux d’artifice y est aussi ajoutée. Si les joueurs font moins de 10 points durant la partie, un seul feu d’artifice est affiché.



Quand la partie est fini, un bouton « ? » apparait, il permet de réafficher le terrain. A ce moment la nos cartes, qui était jusqu’à la caché, sont maintenant visible. En réappuyant sur le bouton « ? » le formulaire est réaffiché.

## 

## Protocole de test

### Connexion

|  |  |  |  |
| --- | --- | --- | --- |
| Test à effectuer | Résultat escompté | Résultat obtenu | Commentaires |
| L’utilisateur entre une adresse IP valide qui n’est pas un serveur valide | Le programme indique à l’utilisateur que le serveur n’existe pas | OK |  |
| L’utilisateur entre une adresse IP invalide | Un message indique à l’utilisateur qu’il ne s’agit pas d’une adresse IP valide | OK |  |
| L’utilisateur une adresse IP valide et il n’entre pas de pseudo | Un message demande à l’utilisateur d’entrer un pseudo | OK |  |
| L’utilisateur entre partiellement une adresse IP | Un message lui indique que son adresse est incomplète | OK |  |
| L’utilisateur n’entre pas d’adresse IP | Un message lui demandant d’entrer une adresse IP apparait | OK |  |
| L’utilisateur entre le caractère ; dans son pseudo | Lorsque le client essaie de se connecter au serveur, le caractère ; est enlevé du pseudo pour être envoyé | OK | Le fichier CSV qui affiche l’historique utilise le caractère ; pour séparer les champs |
| L’utilisateur entre l’adresse IP valide d’un serveur valide et entre un pseudo Mais la partie contient déjà 5 joueurs | Un message apparait pour indiquer à l’utilisateur pour lui indiquer que la partie est pleine | OK |  |
| L’utilisateur entre l’adresse IP valide d’un serveur valide et entre un pseudo valide | Le client se connecte au serveur | OK |  |

### Démarrer une partie

|  |  |  |  |
| --- | --- | --- | --- |
| Test à effectuer | Résultat escompté | Résultat obtenu | Commentaires |
| L’utilisateur démarre la partie alors qu’il est seul dans la partie | Le client indique qu’il ne faut plus de personne pour démarrer une partie. | OK |  |
| L’utilisateur lance la partie, il y a entre 2 et 5 personnes dans la partie | La partie démarre, une nouvelle fenêtre est ouverte sur chaque client. | OK |  |

### Règle de la partie

|  |  |  |  |
| --- | --- | --- | --- |
| Test à effectuer | Résultat escompté | Résultat obtenu | Commentaires |
| L’utilisateur indique que le nombre d’indice maximal est de 8 | 8 indices sont disponibles pour la partie, 8 ampoules sont affichées au début. | OK | Idem pour tous les nombre de 3-8 |
| L’utilisateur indique que le nombre d’erreur est 5 | Durant la partie les joueurs auront le droit de faire 5 erreurs, au maximum 5 éclaires seront affichées | OK | Idem pour tous les nombre de 3-5 |
| L’utilisateur entre un nombre en dehors des limites du nombre d’indice | Le programme remet automatiquement le nombre valide le plus proche | OK | Identique pour le nombre d’erreur |
| L’utilisateur ne veut pas ajouter d’extension | Il y a au total 50 cartes et aucunes des cartes n’est violette | OK |  |
| L’utilisateur veut ajouter l’extension Joker | Il y a au total 55 cartes et 5 d’entre elles sont violette | OK |  |

### Jouer une partie

|  |  |  |  |
| --- | --- | --- | --- |
| Test à effectuer | Résultat escompté | Résultat obtenu | Commentaires |
| L’utilisateur sélectionne une carte dans sa main | La carte est mise en évidence en se surélevant | OK |  |
| L’utilisateur joue la carte sélectionner qui est une carte qui peut être jouer | La carte est retirée de la main du joueur et placé dans la pile correspondante. Le joueur pioche automatiquement une carte | OK |  |
| L’utilisateur joue la carte sélectionner qui n’est pas une carte qui peut être jouer | La carte est retirée de la main du jouer et placé de la zone de défausse, un éclair est ajouté. Le joueur pioche automatiquement une carte | OK |  |
| L’utilisateur appuie dans la zone de point sans avoir sélectionner de carte | Rien ne se passe | OK |  |
| L’utilisateur joue une carte alors que ce n’est pas son tour | Rien ne se passe | OK |  |
| L’utilisateur appuie dans le petit carré pour marquer une carte | Le symbole dans le carré change | OK |  |
| L’utilisateur Sélectionné une carte et veut la déplacer entre deux autres cartes de la main à l’aide du clic droit | La carte se déplace au bonne endroit | OK |  |
| Le joueur veut défausser une carte sélectionner et tous les indices sont encore disponible | Rien ne se passe | OK |  |
| Le joueur veut défausser une carte sélectionner et il manque des indices | La carte est ajoutée à la pile de défausse, un indice est ajouté. Le joueur pioche automatiquement une carte. | OK |  |
| Test à effectuer | Résultat escompté | Résultat obtenu | Commentaires |
| Le joueur veut défausser une carte, ce n’est pas son tour | Rien ne se passe | OK |  |
| Le joueur veut défausser une carte, aucune carte n’est sélectionnée | Rien ne se passe | OK |  |
| Un joueur donne un indice de couleur rouge a un autre joueur, le nombre d’indice a disposition est suffisant | Chez le joueur qui reçoit l’indice, la couleur rouge apparait sur chaque carte rouge de sa main, un indice est enlevé | OK | Même test pour les autres couleurs |
| Un joueur donne un indice du chiffre 3 a un autre joueur, le nombre d’indice a disposition est suffisant | Chez le joueur qui reçoit l’indice, le nombre 3 est écrit sur les cartes ayant un 3 un indice est enlevé. | OK | Même test pour les autres chiffres |
| Un joueur donne un indice a un autre joueur, le nombre d’indice est insuffisant | Rien ne se passe | OK |  |
| Un joueur donne un indice à un autre joueur, ce n’est pas à son tour de jouer | Rien ne se passe | OK |  |
| Après avoir fait une action jouer, défausser ou donner un indice, le joueur ayant le droit de jouer change | Le label du joueur qui peut jouer change de couleur. | OK |  |
| Un joueur a réussi à compléter une pile d’une couleur de 1 à 5 il manque des indices | Un indice s’ajoute. | OK |  |
| Un joueur a réussi à compléter une pile d’une couleur de 1 à 5 tous les indices sont encore à disposition | La carte se déplace au bonne endroit | OK |  |
| Un joueur a appuyé sur le bouton abandonner | La partie s’arrête, le formulaire de fin de partie s’affiche | OK |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test à effectuer | Résultat escompté | Résultat obtenu | Commentaires |
| Il n’y a plus de carte dans la pioche et une action demande au joueur de piocher | La partie s’arrête, le formulaire de fin de partie s’affiche | OK |  |
| Il n’y a plus de carte dans la pioche et une action ne demande pas au joueur de piocher | L’action se fait sans poser de problème | OK |  |
| Après avoir joué une carte qui n’est pas valide, le nombre max de faute a été dépassé | La partie s’arrête, le formulaire de fin de partie s’affiche | OK |  |
| Toute les piles ont été rempli avec les chiffres de 1-5 | La partie s’arrête, le formulaire de fin de partie s’affiche | OK |  |
| Le joueur essaie de jouer une carte alors que la partie est terminé | Rien ne se passe | OK |  |

### Déconnexion

|  |  |  |  |
| --- | --- | --- | --- |
| Test à effectuer | Résultat escompté | Résultat obtenu | Commentaires |
| Le serveur s’arrête | Un message est affiché sur les clients, la partie s’arrête | OK |  |
| Un client quitte la partie, il revient au bout de 30 seconde | Un Timer de 30 sec apparait, dès que l’utilisateur se connecte, la partie reprend | KO | La gestion des reconnexions n’a pas pu être mise en place |
| Un client quitte la partie, il ne revient pas au bout de 30 seconde | Un Timer de 30 sec apparait, au bout de 30sec, le formulaire de fin de partie apparait | KO | La gestion des reconnexions n’a pas pu être mise en place |
|  |  |  |  |

## Problèmes rencontrés et solutions

### Connexion client-serveur

Ce qui était prévu pour la connexion client-serveur, était que lorsque qu’un client se lance, un Broadcast est fait sur le réseau. Une liste de serveur qu’il peut rejoindre lui est donné. Il peut ensuite décider lui-même quel serveur il veut rejoindre.

Au CPLN, cette méthode n’est pas applicable, car le réseau est séparé en plusieurs sous-réseau. Et malheureusement le broadcast ne passe pas à travers les sous réseaux. Donc si le client ne se trouve dans le même réseau que le serveur, ils ne pourront pas se voir.

De mon point de vue, la meilleure solution à ce problème, a été de demander au client d’entrer l’adresse IP du serveur, à la manière d’un mot de passe. Une autre solution, aurait été d’avoir un serveur avec une adresse IP fixe dans le réseau et le programme se connecte automatiquement au serveur.

### Affichage des jokers dans la zone point

Avec l’extension « Joker », la couleur violette est ajoutée. Les cartes violettes peuvent être joué comme une carte normale, mais à la place de remplir un tas de couleur violette, elle se positionne sur le premier tas dans le quelle la carte est valide.

Quand une carte violette était jouée soit elle prenait la couleur du tas et à ce moment-là on ne savait pas qu’un joker avait été joué. Ou soit elle gardait la couleur violette mais à ce moment-là, on ne savait plus de quelle couleur était initialement le tas.

La solution utilisée pour résoudre ce problème, a été de faire que la couleur de la carte soit dégradée de la couleur initiale du tas vers le violet, comme cela, on peut bien distinguer le fait qu’une carte joker aie été joué et on sait toujours de quelle couleur est le tas.



Ci-dessus, vous voyez la zone de point d’une partie ou un joker 3 est venu se placer dans la pile de carte bleu.

## Améliorations possibles

### Gérer les déconnexions

Lors d’un appel avec l’expert, il m’a conseillé de mettre en place un système permettant de gérer les déconnexions. Le principe aurait été que si un utilisateur se déconnecte, tous les autres joueurs reçoivent un message et un Timer démarre. Si le joueur se reconnecte, la partie reprend. Si au bout de 30 seconde, la personne ne s’est pas reconnecté la partie s’arrête.

Pour mettre en place cette fonctionnalité, le mieux est dès qu’un client se déconnecte, le serveur envoie un message a tous les clients pour les prévenir que la partie se met en pause. Le serveur se remet à écouter sur le port 60000 et si une connexion est faite depuis n’importe où, le serveur le considère comme l’ancien joueur. Cette méthode n’est pas sécurisée. Pour plus la sécurisé, il faudrait mettre en place un système de mot de passe. Si le mot de passe entrez au début et à la reconnexion, alors nous savons que c’est le même joueur.

La raison principale pour laquelle cette fonctionnalité n’a pas été mise en place est à cause du manque de temps, et qu’elle n’était pas demandé dans le cahier des charges, elle n’était donc pas primordiale. Cependant, une petite gestion a été mise en place, si le serveur n’est plus atteignable, les clients se ferme. Si la connexion avec un client est perdue, la partie est considéré comme terminé.

### Ajout d’autre extension

Dans le cahier des charges, il m’était demandé d’ajouter une extension avec la couleur violette qui est utilisé comme joker.

Dans le jeu Hanabi, il existe plusieurs autres extensions qui pourrait être mis en place. Dans le choix des règles, un Bouton radio est déjà mis en place si plus tard quelqu’un veut reprendre le projet et ajouter la couleur violette comme une 6ème couleur, il ne lui reste plus qu’à ajouter la logique de cette extension dans les classes. Le reste de la mécanique est déjà mise en place car elle est identique au 5 autres couleurs.

Une autre extension qui pourrait être mise en place est l’extension poudre noir, quia ajoute des cartes de couleurs noir. La particularité de cette extension est qu’il faut remplir le tas a l’envers. C’est-à-dire en commençant à 5 et terminant à 1. Et quand on compte les points à la fin de la partie, elle n’augmente pas le nombre de point mais elle fait perdre 1 points par carte noir qui n’ont pas été posé.

### Amélioration graphique

Dans l’idéale, une charte graphique aurait été mise en place, elle aurait été utile pour avoir une palette de couleur pour les cartes et ne pas utiliser les couleurs automatiques en C# qui ne sont pas forcément très belle.

L’expérience utilisateur n’est pas non plus toujours super intuitive, notamment dans pour la partie déplacer une carte dans sa main. Sans explication, les utilisateurs il n’est pas certains qu’il trouve la fonctionnalité par eux-mêmes.

### Indiquer à l’utilisateur si une action n’est pas valide

Actuellement, lorsque le joueur fait une action alors qui n’as le droit, par exemple, s’il essaye de jouer une carte alors que ce n’est pas son tour, rien ne l’empêche. Il y a tout de même des sécurités sur le serveur qui vont empêcher ce genre d’action. Mais il ne se passe rien du coté, l’utilisateur ne comprend pas forcément pourquoi il ne peut pas faire cette action. Il faudrait alors afficher sur le client un petit message qui lui indique d’où vient l’erreur.

## Conclusion

Je finirai enfin par mon opinion personnelle, j’ai beaucoup apprécié faire ce projet, il m’a permis d’apprendre énormément de chose au niveau de l’informatique, de comment planifier un projet et de tenir compte du peu de temps que nous avons pour tout faire. Au final, je suis fière du produit final qu’y ai rendu, il ressemble presque parfaitement à ce que je voulais dès le début du projet. Ce qui est tout de même frustrant, c’est de ne pas avoir pu finir certains petits détails du projet, notamment a cause du temps mis a disposition.

## Annexes

### Références

Image de titre :

<https://www.cocktailgames.com/jeu/hanabi/>

Image de fond du programme client :

<https://www.vecteezy.com/vector-art/1446173-colorful-fireworks-background>

Sérialisation

<https://www.tutorialexample.com/serialize-python-object-to-string-and-deserialize-it-to-object-for-python-beginners-python-tutorial/>

Algorithme de mélange

<https://fr.wikipedia.org/wiki/M%C3%A9lange_de_Fisher-Yates>