Ingénierie Mathématiques et Data Science 4A

Rapport de Projet 4A

Logiciel Arbitrage de Tennis de Table

Lisa NAPIERALA et Derya KOCAK

Résumé

Durant ce projet, nous avons développé un logiciel d'arbitrage de Tennis de Table. Pour ce faire, nous avons créé un diagramme UML : diagramme de classe technique. Ce dernier nous a permis de savoir à quoi devait ressembler le logiciel et de quelle manière le développer. Nous l'avons développé dans un premier temps sous Linux puis sur QT Creator en utilisant le langage C++. Le logiciel possède une page d'accueil, une page avec toutes les fonctionnalités pour le déroulement du tournoi, plusieurs pages pour la saisie et des pages affichant des messages selon l'action réalisée. Tout au long du Tournoi, le logiciel enregistre des fichiers. Ces derniers permettent de sauvegarder un tournoi, et de le reprendre à tout moment.

Mots-clés *:* tennis de table, diagramme de classe technique, QT Creator, C++, Linux, sauvegarde informatique.

Abstract

During this project, we developed a table tennis judge software. To do so, we created a UML diagram: Class Diagram. This diagram allowed us to know what the software should look like and how to develop it. We developed it first under Linux and then on QT Creator using the C++ language. The software has a home page, a main function page, several input pages and message pages to be displayed. Throughout the tournament, the software saves files. These files allow you to save a tournament, and to continu it at any time.

Keys-words: Table Tennis, Class Diagram, QT Creator, C++, Linux, Backup

Sommaire

Introduction

Notre projet de 4A consiste en la création d'un logiciel de Juge arbitrage automatique pour les compétitions de Tennis de Table.

Pour cela, le client souhaite pouvoir utiliser ce logiciel de manière fluide et « facile ». Ce logiciel doit pouvoir sauvegarder une liste de participants à partir d'un fichier, générer des poules et des matchs éliminatoires. De plus, le client souhaite pouvoir imprimer des feuilles de rencontres et reconnaître automatiquement via une caméra les scores remplis sur la feuille de rencontre.

Nous avons aussi voulu développer un logiciel qui est « stupid proof », c'est-à-dire un logiciel qui indiquerait si l'utilisateur ne rentre pas les bonnes informations et demanderait ainsi à l'utilisateur de réitérer sa saisie. De même, nous allons ajouter la possibilité de reprendre le tournois à n'importe quel moment pour que l'utilisateur puisse fermer le logiciel s'il le souhaite pour reprendre le lendemain par exemple.

Nous verrons d'abord, dans ce rapport, nos recherches sur le projet, puis l'architecture de notre logiciel. Ce dernier ayant été programmé sous Linux et Qt, nous allons, dans un second temps, analyser ces deux versions et leurs utilisations. Enfin, nous ferons le bilan de notre projet avant de conclure.

Recherches

Pour nous conformer aux attentes du client, nous avons fait de nombreuses recherches sur le déroulement en compétition d'un tournoi de Tennis de Table. Dans un premier temps nous avons réalisé nos recherches sur le site de la fédération française de Tennis de Table et y avons trouvé un manuel de déroulement d'un tournoi.

Un tournoi se déroule en plusieurs étapes. Il y a en premier les phases de poules. Selon le nombre de joueurs il peut y avoir 4 ou 8 poules. Tous les joueurs d'une poule se rencontrent, pour les différencier, chaque joueur possède un numéro de licence individuel et unique, un nom, un prénom. Pour gagner un match, un joueur doit obtenir onze points, cependant il faut que son adversaire ait au moins deux points d'écart. Par exemple, si les deux joueurs sont à 10-11 le match n'est pas gagné, il faudrait un score de 10-12. Ainsi le match continue tant que les deux joueurs n'ont pas deux points d'écart.

A la suite d'une rencontre les joueurs obtiennent des points : 3 points si le joueur à gagner, 2 points pour un match nul, 1 points pour la défaite et 0pt pour un forfait ou par pénalité (décision de la commission). Ces points de rencontres permettent de réaliser un classement par poules. Seuls les deux premiers de la poule sont sélectionnés. S'il y a égalité, il faut regarder le nombre de points d'écart lors des matchs. Par exemple, si un joueur gagne 8-11, il y a trois points d'écart, on cumule ces points d'écart sur tous les matchs de poules. Celui qui a le plus petit point d'écart cumulé est sélectionné.

Les joueurs sélectionnés, après les phases de poules, passent ainsi à l'étape suivante : les matchs éliminatoires. Dans ce cas, un joueur qui perd se fait directement éliminer et arrête le tournoi. Les joueurs étant éliminés au fur et à mesure, le nombre de matchs diminue. La finale a lieu lorsqu'il ne reste plus qu'une seule rencontre entre 2 joueurs.

Ainsi nous allons développer un logiciel qui reste conforme à ces règles.

Nous avons pu constater qu'il existait beaucoup de logiciels concurrents. Cependant, ils sont souvent payants pour pouvoir accéder à toutes les fonctionnalités, de même pour la reconnaissance par webcam. Le plus gros problème que nous avons retrouvé chez nos concurrents est le fait que l'utilisateur doit enregistrer à la main tous les joueurs participant à la compétition. Pour pallier ce problème nous allons proposer d'importer un fichier .txt avec déjà le numéro de licence, le nom et le prénom des joueurs. De plus, nous allons permettre d'imprimer 4 rencontres en une seule feuille, contre 2 rencontres pour les concurrents, afin de minimiser notre impact écologique. Enfin, notre logiciel ne nécessitera pas d'avoir accès à internet : encore un point positif par rapport à la majorité des logiciels du marché.

Architecture du logiciel

Fonctionnalités

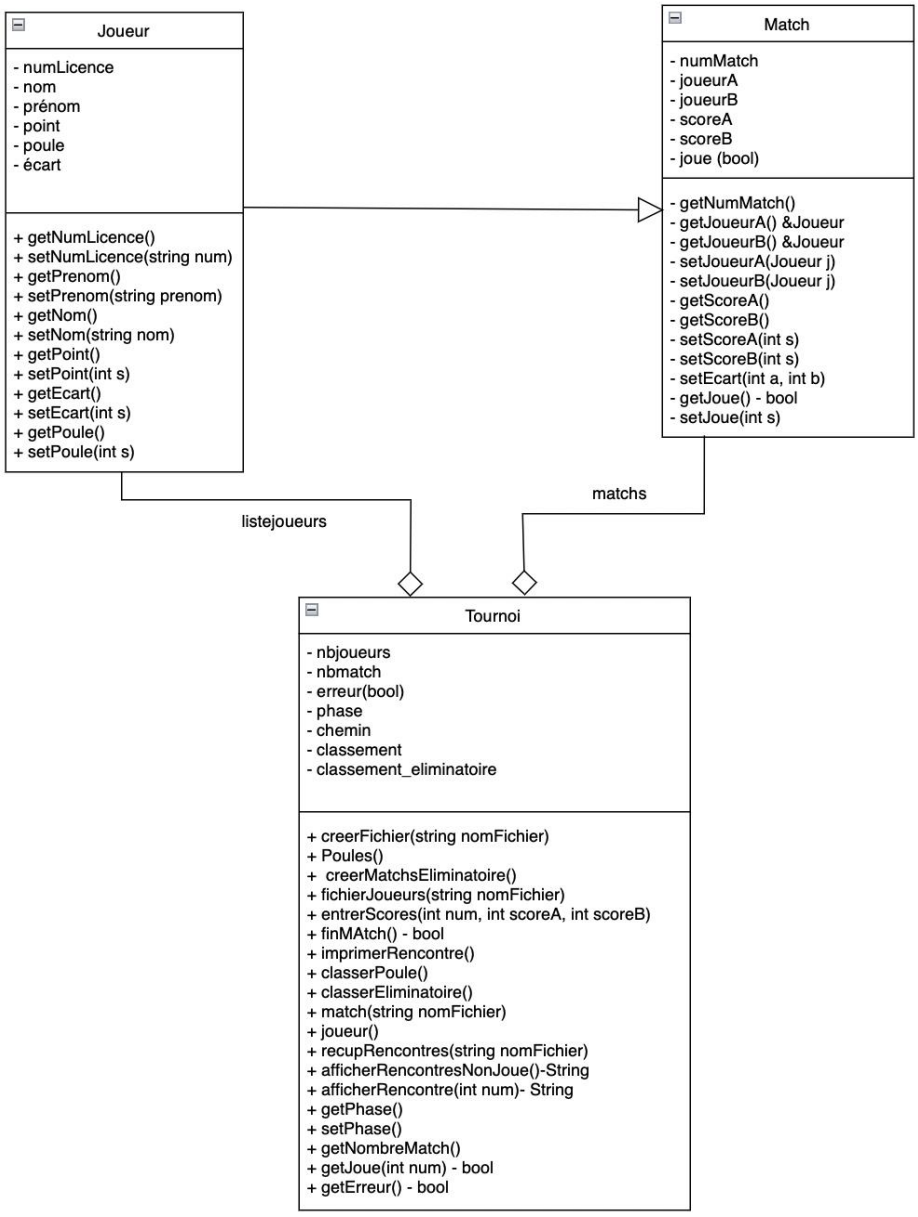


Figure I - Diagramme d'analyse technique

Pour la conception du logiciel, nous avons commencé par un diagramme d'analyse technique *[Figure A]*. Ce dernier est composé de 3 classes : **Joueur, Match** et **Tournoi.**

La classe Match utilise la classe Joueur puisqu'un match est une rencontre entre deux joueurs pour se réaliser. Les classes Joueur et Match, elles, composent la classe Tournoi. En effet, sans joueurs et sans matchs le Tournoi ne peut pas avoir lieu. Nous allons maintenant expliquer en détail ces classes.

Tout d'abord, la **classe Joueur :**

Les attributs de cette classe permettent de définir les caractéristiques d'un joueur : un numéro de licence (numLicence) unique à chaque joueur, un nom, un prénom, un point qui correspond à la somme des points gagnés par matchs (+2 si match gagné, +1 nul , +0 si perdu), un écart qui correspond à la somme des écarts de points lorsque le joueur gagne (ex: joueur gagne 11-7 on ajoute + 4 à l'écart) qui permet de faire la différence si pendant un classement il y a égalité de points entre deux joueurs. Enfin, le joueur possède en attribut son numéro de Poule qui sera initialisé au début du tournoi.
Les méthodes de la classe Joueur permettent d'accéder à ses attributs, ainsi que de les modifier.

Ensuite, la **classe Match :**

Elle permet de définir une rencontre entre deux joueurs à l'aide de ses attributs. Ces derniers sont : un numéro de Match unique à chaque phase (numMatch), un premier joueur (joueurA), puis un deuxième (joueurB), ainsi que leurs scores (scoreA, scoreB). Enfin la classe Match possède un attribut joue de type booléen pour savoir si le match a été joué ou non.
Les méthodes de cette classe permettent de récupérer ou modifier les attributs de la classe Match ou encore des joueurA et joueurB. SetEcart(int a ,int b) par exemple, prend en paramètre les scores des deux joueurs qui ont effectué le match, l'écart des points du match est mis à jour dans l'attribut point des deux joueurs. Si le joueur a gagné cette valeur sera négatif et s'il a perdu cette valeur sera positive.

Enfin, la **classe Tournoi :**

Cette classe possède en attribut un vecteur (listejoueurs) qui possède tous les joueurs du tournoi, le vecteur 'classement' est utilisé pour classer tous les joueurs par poules selon leurs points, le vecteur 'classement_eliminatoire' fonctionne avec le même principe que classement mais pour les phases éliminatoires. Nous avons aussi en attribut un vecteur matchs composés de tous les matchs à jouer pour une certaine phase. Pour savoir si nous sommes en phase de poule ou non, nous avons initialisé un entier 'phase'. De plus, un booléen erreur qui sera vrai si une erreur apparaît lors de l'importation des fichiers est un attribut de cette classe. Enfin, dans cette dernière, il est nécessaire d'avoir un attribut chemin sous forme de chaîne de caractère : il s'agit du chemin récupéré lors de l'importation du premier fichier des joueurs. Cet attribut permet d'enregistrer les fichiers "joueurs_maj.txt", "Rencontre_Poule.txt", etc. dans le même dossier que le fichiers de joueurs.
Les méthodes de cette classe permettent de gérer un Tournoi du début à la fin, ainsi que d'en récupérer un en cours de route.

La méthode **créerFichier(string nomFichier)** permet de créer un fichier au nom de nomFichier dans le dossier du tournoi (où se situe le fichiers joueurs.txt). Elle nous est utile dans plusieurs méthodes qui ont besoin de créer un fichier.

La méthode **fichierJoueurs(string nomFichier)** sert à lire le fichier, contenant les joueurs, importé au début du tournoi. Pour ce faire, la méthode utilise la classe **QFile** pour la version Qt.

La méthode **Poules()** est une méthode qui définit le nombre de poules du tournoi. Dans un premier temps, 4 poules s'il y a moins de 24 joueurs, 8 sinon. Ensuite, la méthode attribue à chaque joueur sa poule de façon aléatoire, dans chaque poule il y a le même nombre de joueur à un joueur près si le nombre de joueur n'est pas un multiple de 4. Ensuite, les joueurs sont mis à jour dans le fichier "joueurs_maj.txt" à l'aide de la méthode **joueur().** Cette dernière permet d'enregistrer les joueurs et leurs attributs mis à jour dans un fichier sous le nom "joueurs_maj.txt". Enfin, dans la méthode Poules(), le vecteur matchs est initié avec tous les matchs de poules avant d'être enregistrés sous le nom "Rencontre_Poule.txt" à l'aide de la méthode **match(string nomFichier)**. Cette dernière permet d'enregistrer les matchs et leurs attributs mis à jour dans un fichier nommé nomFichier.txt.

La méthode **entrerScores(int num, int scoreA, int scoreB)** est utilisée pour permettre à l'utilisateur de saisir les scores d'une rencontre. Selon si le tournoi est en phase de poule ou non, le fichier de rencontres est mis à jour à l'aide de la méthode match("Rencontre_Matchs.txt") ou match("Rencontre_Poule.txt").

Une fois tous les scores saisies, la méthode **classerPoule()** permet de créer un classement à l'aide des résultats des rencontres. Le classement est enregistré sous le nom "Classement_Poules.txt".

Une fois la phase poule terminée et classée, la méthode **creerMatchsFinPoule()** est utilisée. Cette dernière récupère du vecteur classement les deux premiers de chaque poule et crée une rencontre entre le 1er de la poule 1 et le 2e de la poule 2, le 1er de la poule 2 et le 2e de la poule 3 etc. On efface le contenu de l'ancien vecteur de matchs, pour le remplir avec les nouvelles rencontres. Enfin, la méthode fait appel aux méthodes joueur() et match("Rencontre_Matchs.txt") pour mettre à jour ces fichiers.

Une fois que les matchs d'après poule sont terminés, le tournoi passe à l'étape suivante a l'aide de la méthode **creerMatchsEliminatoire().** Dès qu'un joueur perd, il est ajouté dans le vecteur classement_eliminatoire, tandis que le gagnant prend place dans une rencontre contre un autre gagnant. Ces rencontres forment le nouveau vecteur matchs. Enfin, la méthode met à jour le fichier de rencontres avec la méthode match("Rencontre_Matchs.txt").

Généralement, en début de tournoi, l'utilisateur a besoin d'imprimer les feuilles de rencontres. De ce fait, le tournoi utilisera la méthode **imprimerRencontre().** Cette méthode crée d'abord le fichier "RencontreAImprimer.txt", puis le remplit avec les rencontres du vecteur matchs. L'objectif est que l'utilisateur imprime cette feuille et écrive les résultats dessus puis grâce à l'intelligence artificielle nous pourrions lire les scores et les rentrer automatiquement dans le logiciel.

Pour ceux qui ne voudraient pas imprimer sur papier les matchs, la méthode **afficherRencontre(int num)** existe. Elle permet d'afficher à l'écran la rencontre ayant pour numéro : num.

Nous proposons aussi de pouvoir afficher sur l'ordinateur tous les matchs qui n'ont pas été joués. C'est la méthode **afficherRencontresNonJoue()** qui est utilisée. Cette dernière fait appel à la méthode afficherRencontre(int num) pour chaque rencontre qui a son attribut joue égale "false".

La méthode **classerEliminatoire()** permet de classer les joueurs qui ont fait les matchs post poules.

Pour savoir à quel moment classer, ou passer une étape etc. la méthode **finMatchs()** renvoie true ou false selon si toutes les rencontres sont jouées ou non.

Comme dit précédemment, on utilise ces méthodes pour créer et faire un tournoi. Mais on peut également les utiliser lorsque l'on reprend un tournoi. Dans ce cas, la classe Tournoi fait appel à la méthode **recupRencontres(string nomFichier)**. Cette dernière récupère le fichier "joueurs_maj.txt" pour récupérer les joueurs avec les points où ils se sont arrêtés.
Ainsi sont récupérés les matchs selon la phase dans laquelle est le Tournoi repris.
Pour le cas où le tournoi serait repris à la phase d'après poule, le fichier match ne comporte plus que les joueurs qui ont passé certaines étapes. Donc le fichier ne comprend pas tous les matchs post poule. Pour pouvoir classer les joueurs éliminés, la méthode récupère le fichier de classement_Eliminatoire.txt qui est mis à jour régulièrement. De ce fait, même les matchs avant l'étape actuelle seront pris en compte, et le classement final sera donc significatif.

Enfin, pour le bon déroulement du tournoi, la classe possède des méthodes qui permettent d'accéder à ses attributs (**getNombreMatch()** , **getErreur()**, **getPhase()**) ou encore **setPhase()** pour passer de la phase poule à la phase post poule et **getJoue(int num)** pour accéder à l'attribut joue du match numéro num.

Conditions à respecter

Pour le bon déroulement du tournoi il y a plusieurs conditions à respecter notamment avec le format du fichier joueur.

Fichier liste de joueurs

Ce fichier doit être sous format .txt et respecter une certaine syntaxe. Dans un premier temps il nous faut le nombre de joueurs qui participent au tournoi. Puis ligne par ligne, il faut le numéro de licence, le nom, le prénom, les points (=0 au début), l'écart (=0 au début) et la poule (=0 au début) de chaque joueur *[Figure B].*

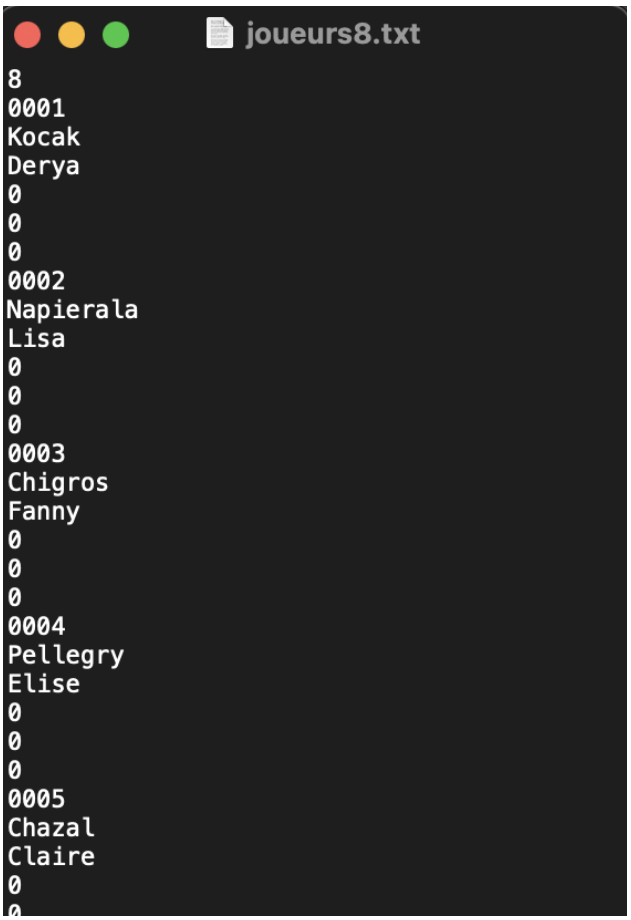


Figure II - Exemple de fichier joueurs.txt

Emplacement fichiers sous Qt

Notre logiciel enregistre les fichiers de sauvegarde, les fichiers à imprimer ainsi que les classements au même endroit que le fichier joueurs.txt importé au début.
De ce fait, si l'utilisateur commence un tournoi, le quitte, et en recommence un, son premier tournoi sera complètement effacé. Car les fichiers de sauvegarde remplaceront les précédents.

Afin de pouvoir commencer plusieurs tournois, et d'en reprendre un sans perdre les sauvegardes, il faut faire un dossier pour chaque tournoi.
Lorsqu'un tournoi débutera avec l'importation d'un fichier de joueurs, par exemple, dans le dossier Tournoi 1, alors toutes les sauvegardes et autres fichiers seront enregistrés dans ce dossier. Pour reprendre le Tournoi 1, il suffira à l'utilisateur d'importer le fichier de sauvegarde des joueurs se situant dans le dossier Tournoi 1. C'est ainsi qu'aucune sauvegarde ne sera perdue.

Emplacement fichiers sous Linux

Sous Linux, il n'est pas possible de faire plusieurs tournois en même temps. Il faudrait sinon créer plusieurs dossiers avec tous nos fichiers code ainsi que le fichier des joueurs.

Lorsqu'un tournoi débute, si le nom du fichier écrit, lors de l'importation, ne se situe pas dans le même dossier que les codes, ce dernier ne fonctionnera pas. De plus, une fois le fichier importé, les sauvegardes ainsi que les fichiers de classements ou à imprimer seront enregistrés dans ce même dossier.

De ce fait, pour pouvoir réaliser plusieurs tournois, et perdre aucune sauvegarde, il faut créer un dossier par tournoi. Ce dernier devra contenir tous les codes, le Makefile, le fichier de joueurs et les sauvegardes correspondants (si le tournoi a débuté).

Sous Linux

Pour notre première version nous avons réalisé le logiciel d'arbitrage sous Linux. En effet, nous voulions commencer à programmer sur une interface que nous connaissions et qui nous permettait de travailler en même temps sur le code à l'aide de git. Pour cela nous avons suivi notre schéma conceptuel et créer les classes Joueur, Match et Tournoi. Pour compiler notre programme dans le terminal nous avons utilisé un Makefile. Celui-ci permet d'exécuter toutes les commandes de compilation en une fois et ainsi éviter un travail fastidieux.

Nous voulions créer une interface simple et efficace. L'utilisateur n'as qu'à suivre les instructions que le terminal renvoie pour faire avancer son tournoi. Une des fonctions qui pour nous était essentiel fut de pouvoir reprendre un tournoi à n'importe quel moment, il peut y avoir une pause lors du tournoi ou par m'égard l'utilisateur ferme le logiciel. C'est pourquoi, dès le début nous demandons à l'utilisateur s'il souhaite commencer un nouveau tournoi ou s' il souhaite en continuer un. Le seul document qui est demandé à l'utilisateur est le fichier joueur.txt. Si ce fichier n'est pas dans le bon format, le tournoi ne pourra commencer.
Dans notre version Linux, l'interface n'est pas visuelle et il faut rentrer à la main le nom du fichier joueur.txt pour commencer le tournoi. Ce seront deux points que nous allons améliorer par la suite.
Pour que notre logiciel fonctionne, nous avons réalisé un programme principal qui permet d'utiliser toutes les classes vues précédemment et ainsi exécuter un tournoi. Pour ce faire nous avons réalisé plusieurs boucles avec des conditions selon l'entrée de l'utilisateur. Pour faciliter la saisie, nous demandons à l'utilisateur d'entrer un numéro correspondant à la commande souhaitée. Selon cette entrée plusieurs scénarios peuvent se dérouler. Pour le menu avec toutes les options qui peuvent se répéter nous avons utilisé une boucle qui se répète tant que l'utilisateur ne quitte pas le tournoi en entrant '0' ou que le tournoi n'est pas terminé. De plus, pour différencier les cas entre les fonctionnalités que l'utilisateur souhaite utiliser, nous avons utilisé la déclaration switch(choix3). Ainsi selon le choix de l'utilisateur la fonction souhaitée s'exécute.

```
switch(choix3){
    case 1:{
        if (poule){
            nomfichier="Rencontre_Poule.txt";
        }
        else{
            nomfichier="Rencontre_Matchs.txt";
        }
        cout<<"Entrez le numéro de la rencontre : "<<endl;
        cin>>num;
        if ( 0<num &&  num<=T.getNombreMatch() ){ //Si le match existe
            T.entrerScores(num);
            T.joueur();
            T.match(nomfichier);
        }
        else{
            cout<< "Erreur ce numéro de match n'existe pas"<<endl;
        }

        break;
    }
    case 2 :{ //Pour imprimer une rencontre il nous faut son numéro
        cout <<"Entrer le numéro rencontre que vous voulez imprimer" <<endl;
        cin >> num;
        if ( 0<num &&  num<=T.getNombreMatch() ){ //Si le match existe
            T.imprimerRencontre(num);
        }
```

Figure III - Programme Principal

Nous avons aussi codé toutes les exceptions dans le Main.cpp. Par exemple, si le numéro de la rencontre n'existe pas, nous envoyons un message d'erreur et la fonction n'est pas exécutée puisque la condition l'en empêche.

Sous Qt les différences avec Linux

Pour réaliser cette version nous avons repris notre code sous Linux mais avons dû le modifier pour l'adapter à cette interface. En effet, la gestion des fichiers sous Linux et sous Qt ne fonctionne pas de la même façon. Pour lire un fichier ou en créer un nous avons dû tout adapter en utilisant la classe QFile inclut dans Qt.

Sous Qt nous avons pu améliorer notre logiciel d'arbitrage sur plusieurs points. Tout d'abord, l'utilisateur doit choisir son fichier joueur.txt directement dans son dossier. Cette fonctionnalité lui permet d'éviter de taper le nom du fichier, de commettre une faute et de prendre n'importe quel fichier de son ordinateur.

Des erreurs sous Linux sur les conditions pour entrer le score d'un match ont été corrigées. Grâce à des QspinBox nous avons pu directement bloquer la saisie pour des score ou numéro de match négatif ce qui enlève des conditions sur la saisie de l'utilisateur. De plus, l'utilisation des QspinBox empêche la saisie de caractères, seuls les entiers sont acceptés, donc nous n'avons plus d'erreur sur la saisie. Enfin, à la différence du Linux, le logiciel affiche la rencontre avant la saisie des scores, afin d'être sûr qu'il est sur la bonne rencontre.

Notre fonction imprimer une rencontre a aussi été modifiée. Nous avons préféré pouvoir imprimer toutes les rencontres du tournoi. Pour faciliter l'impression, ce fichier est créé pour faire en sorte que quatre matchs se tiennent sur une seule page.

Enfin, pour reprendre un tournoi nous demandons à l'utilisateur de sélectionner dans ses fichiers le fichier que le logiciel à créer : joueur_maj.txt.

Utilisation du logiciel

Sous Linux

Dans cette version, l'utilisateur doit d'abord choisir s'il reprend ou démarre un tournoi puis il doit écrire le nom de son fichier qui se situe dans le même dossier que ses codes.

C'est en saisissant des chiffres qu'il pourra réaliser un tournoi. Chaque chiffre correspond à une fonctionnalité. Si '0' est saisie, le terminal se ferme. Il est possible de reprendre le tournoi si celui-ci n'est pas terminé.

Lorsque l'utilisateur décide de démarrer un tournoi, il se trouve face à cinq cas *[Figure I]*.

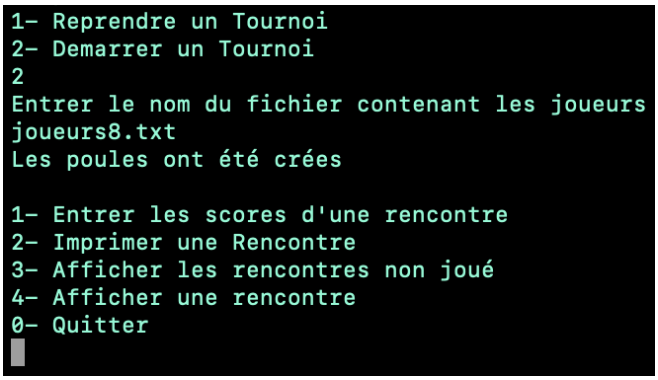


Figure I - Début d'un Tournoi

Sinon, il reprend un tournoi et doit alors préciser s'il est en phase de poules ou matchs (afin d'importer le bon fichier de sauvegarde) *[Figure II]* et se retrouve encore face aux cinq cas :

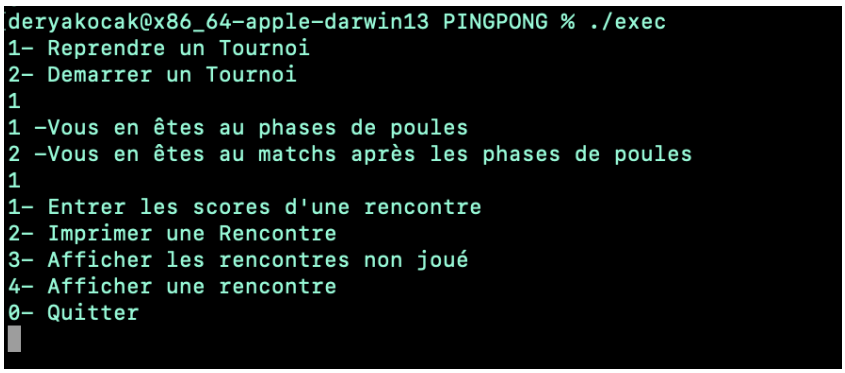


Figure II - Reprendre un Tournoi

Cas 1 : Entrer les scores d'une rencontre *[Figure III]*

Dans ce cas-là de nouvelles questions sont affichées. D'abord, le numéro de rencontre doit être donné puis le score du joueur A et celui du joueur B. Une confirmation est demandée pour valider le score.

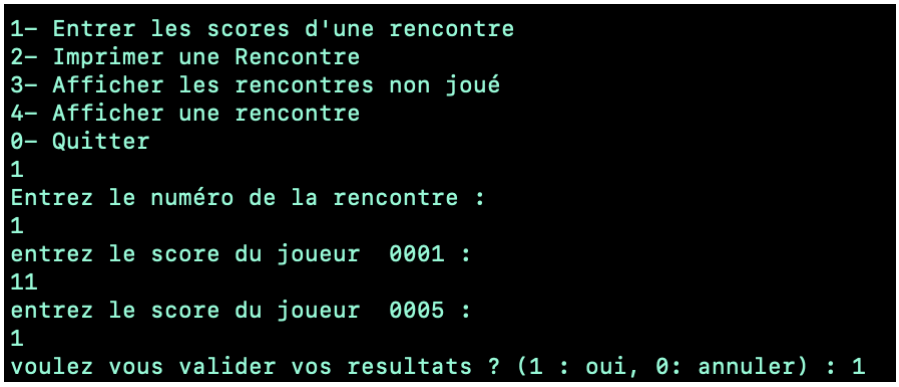


Figure III - Entrer un score

Cas 2 : Imprimer une Rencontre *[Figure IV]*

Un message sur le terminal est affiché. Il faut entrer le numéro de la rencontre puis, si elle existe, elle est enregistrée dans un fichier .txt qui se situe dans le dossier avec le fichier joueur.txt.

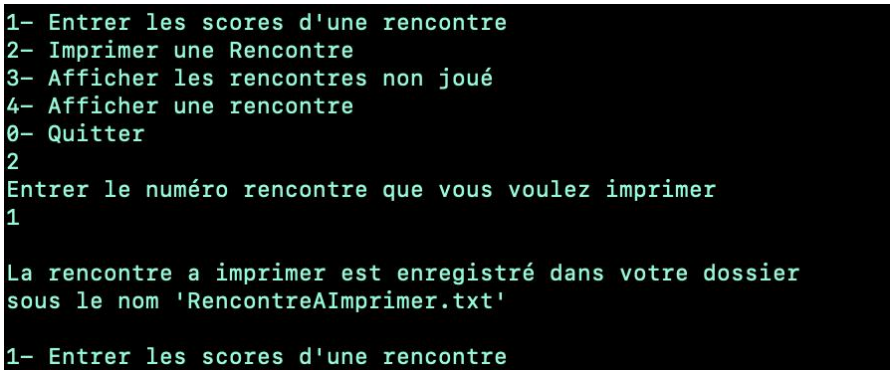


Figure IV - Imprimer une rencontre

Cas 3 : Afficher les rencontres non jouées

Toutes les rencontres non jouées sont affichées dans le terminal. On retrouve le numéro de la rencontre, le numéro de licence, le nom et le prénom des joueurs concernés : le même affichage que la *Figure V* répété pour chaque rencontre

Cas 4 : Afficher une rencontre *[Figure V]*

Après que l'utilisateur ai saisi le numéro de la rencontre, les informations de cette dernière sont affichées.

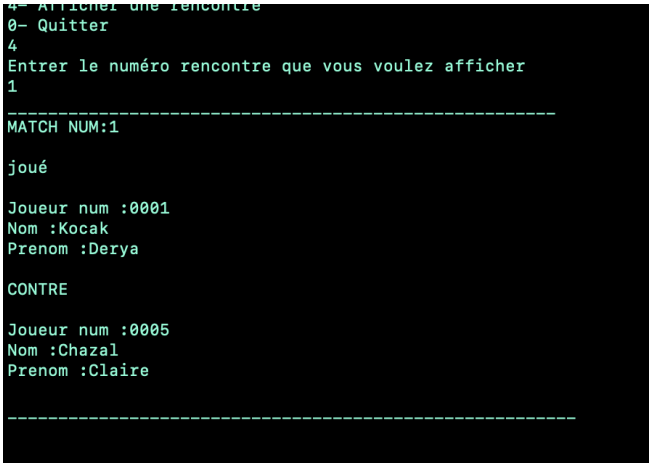


Figure V - Afficher une rencontre

Une fois les derniers scores entrés, le logiciel enregistre alors un classement des poules. De ce fait, il affiche un message pour en informer l'utilisateur. *[Figure VI]*

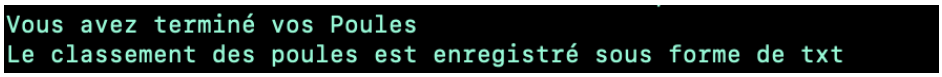


Figure VI - Message fin poules

Sont alors proposés à l'utilisateur les cinq mêmes cas pour la phase éliminatoire jusqu'à la fin des matchs de la phase après poules. Puis de nouveaux matchs sont créés avec ceux ayant passé la première phase éliminatoire. Le même message est affiché à chaque fin de phase. *[Figure VII]*

Figure VII - Message fin d'une phase

Enfin, après la finale, le tournoi se termine. Le logiciel enregistre le classement final composé uniquement des joueurs ayant passé les poules. Le logiciel affiche donc à l'écran un message d'information et se ferme *[Figure VIII]*.

Figure VIII - Message fin tournoi

Sous Qt

Dans cette partie, nous allons expliquer l'utilisation du logiciel sous l'interface Qt qui est notre version la plus élaborée. Nous avons repris notre code sous Linux et l'avons adapté à l'interface Qt.

Le logiciel, une fois lancé, propose à l'utilisateur de démarrer un tournoi ou un reprendre un *[Figure 1]*.

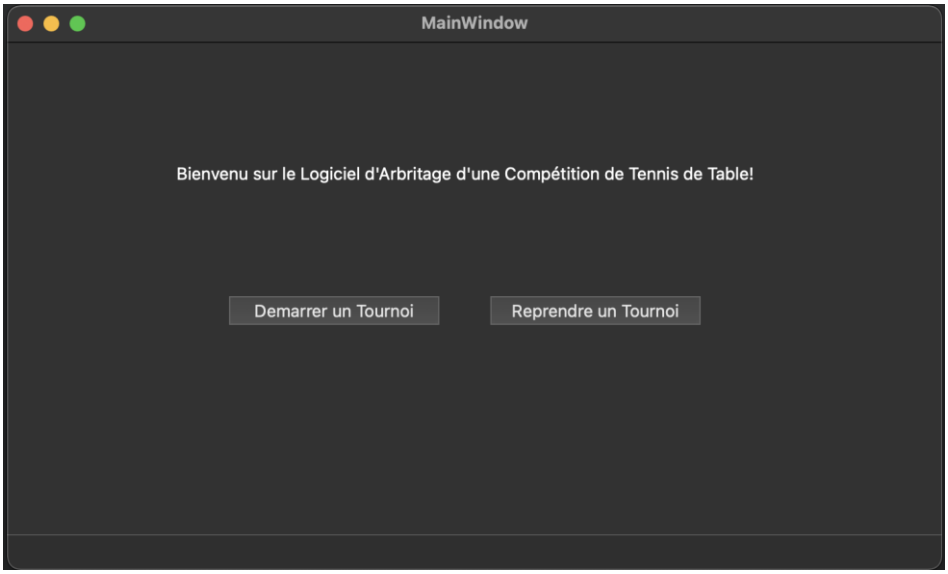


Figure 1 - Accueil

Dans les 2 cas, l'utilisateur devra choisir un fichier sous forme txt :

- Cas 1 : démarrer un tournoi *[Figure 2]*

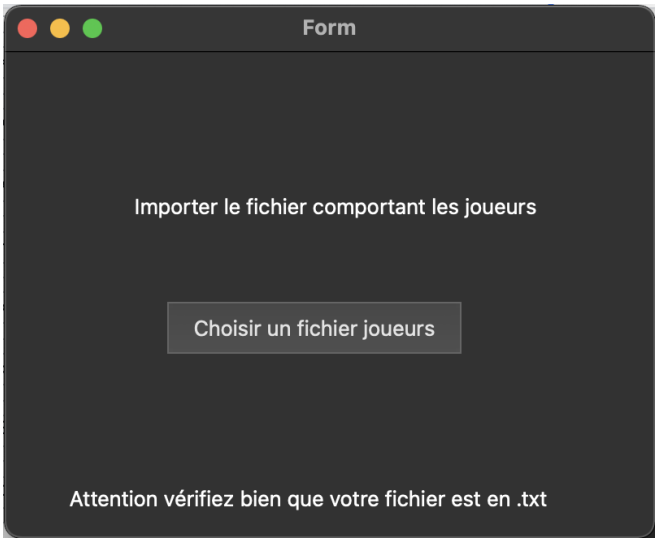


Figure 2 - Importer fichier de joueurs

C'est alors que logiciel redirige l'utilisateur vers ses fichiers, afin d'en importer un :

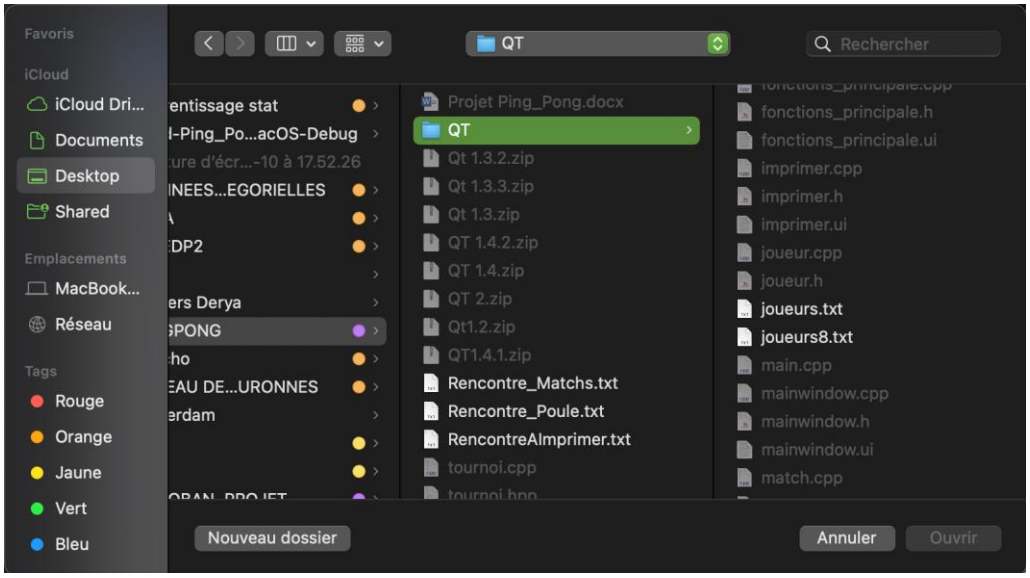


Figure 3 - Importer fichier de joueurs

- Cas 2 : Reprendre un tournoi : [Figure 4]

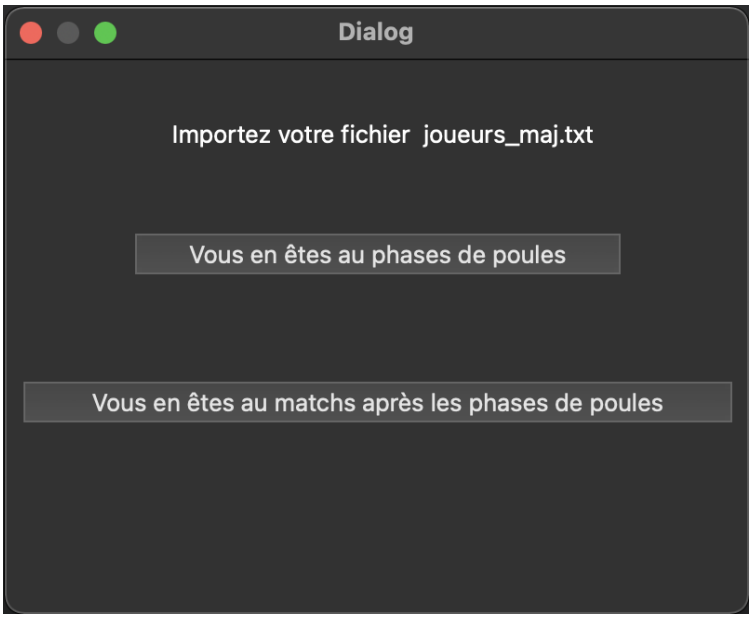


Figure 4 - Reprendre un tournoi

Ainsi l'utilisateur est renvoyé à la Figure 3 pour importer le fichier joueurs_maj.txt.

Dans les 2 cas, les fonctions principales s'affichent : [Figure 5]

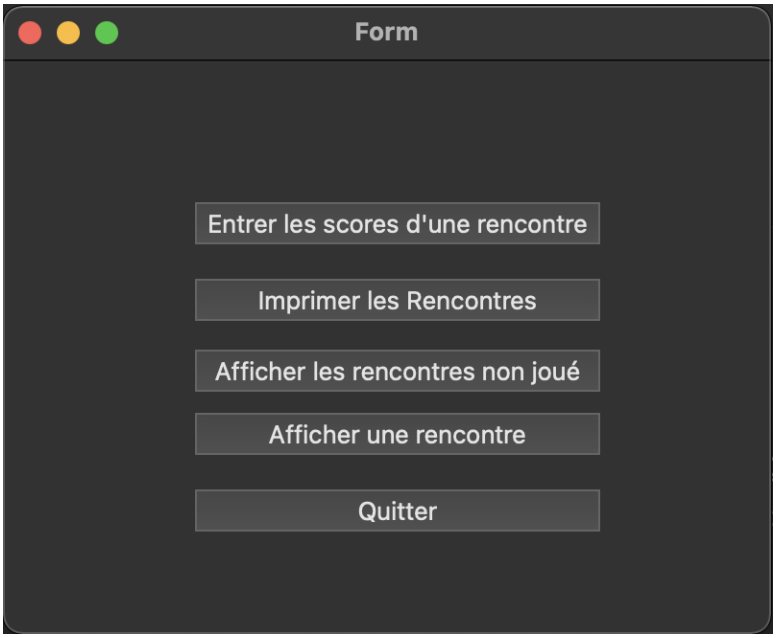


Figure 5 - Fonctions principales

L'utilisateur est face à plusieurs possibilités :

- Entrer les scores d'une rencontre :

D'abord le logiciel demande le numéro de match *[Figure 6]* puis affiche le match afin de permettre à l'utilisateur de saisir les scores *[Figure 7].* Si ce dernier saisit de mauvaises valeurs, alors un message d'erreur s'affiche, et renvoi l'utilisateur à la page des fonctions principales.

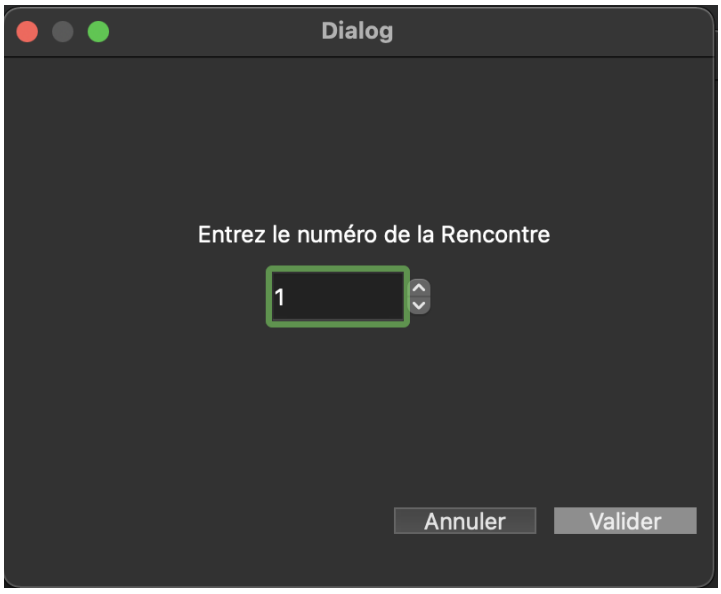


Figure 6 - Saisie du numéro de la rencontre

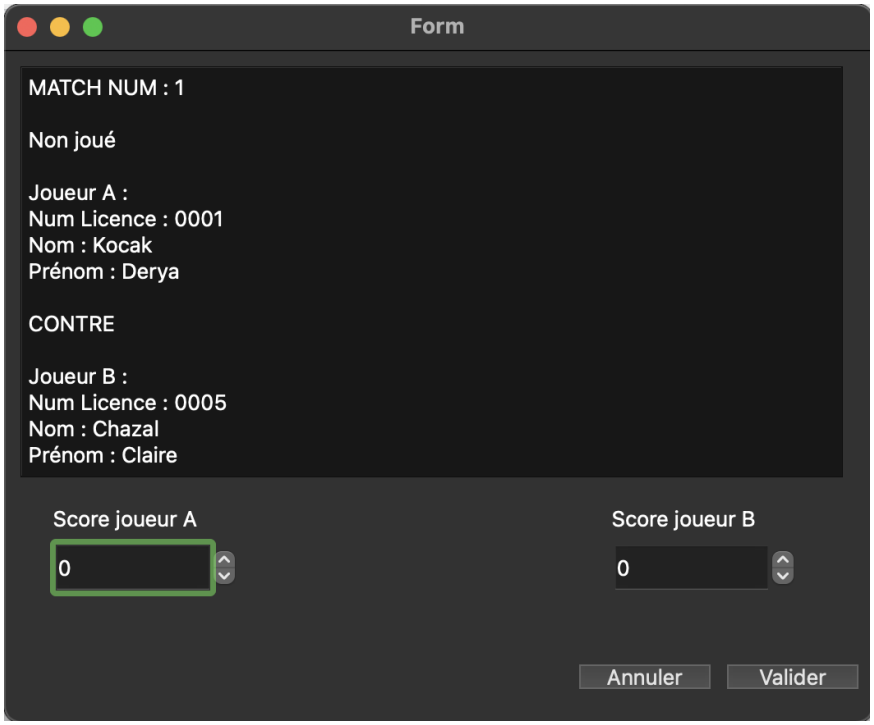


Figure 7 - Saisie des scores

- Imprimer les rencontres :
  S'affiche alors une boîte de message *[Figure 8],* le fichier des rencontres nommé "RencontreAImprimer.txt" est ainsi enregistré dans le dossier de l'utilisateur.

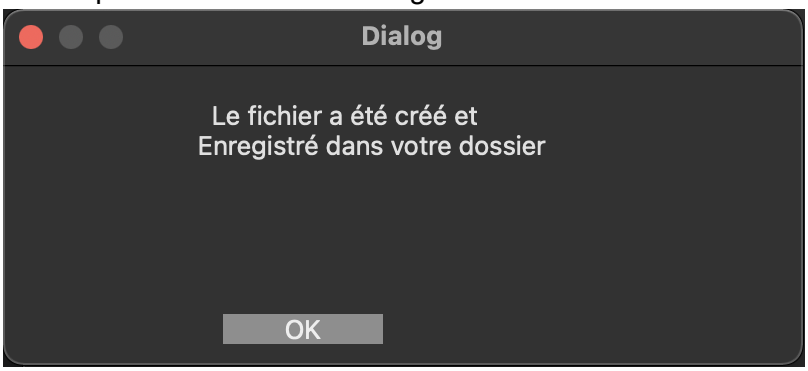


Figure 8 - Boîte Message

- Afficher les rencontres non joué : *[Figure 9]*
  Permet de visualiser toutes les rencontres qu'il reste à jouer ou ceux dont les scores ne sont toujours pas entrés.

Figure 9 - Rencontres non joués

- Afficher une rencontre :
  L'utilisateur doit donc saisir un numéro de rencontre [Figure 10]. Si la saisie est mauvaise, et que la rencontre n'existe pas, un message d'erreur est affiché à l'écran, sinon s'affiche à l'écran la rencontre demandée [Figure 11].

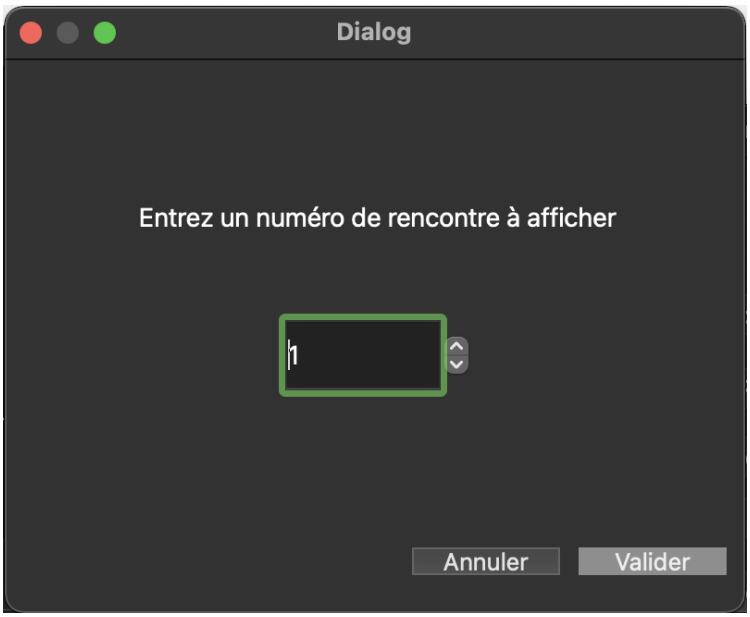


Figure 10- Saisie numéro de rencontre

Figure 11- Affichage de la rencontre

- Quitter :
  Ce bouton permet de quitter le logiciel.

Après la fin des poules, la fin d'une phase éliminatoire ou encore à la fin du tournoi s'affiche une boîte de message tel que la *Figure 8* contenant les mêmes messages que pour la version Linux selon la situation.

Dans le dossier où se situait le premier fichier joueurs.txt qui a été importé lors du démarrage ou lors de la reprise se trouvent donc :

- joueurs_maj.txt : la liste des joueurs mise à jour au fur et à mesure des rencontres enregistrées. Elle est de la même forme que le fichier joueur importé.

- Rencontre_Poule.txt : liste des rencontres à jouer/ jouées mise à jour avec les scores tout au long de la phase Poule. (Annexe)

- Classement_Poules.txt : Le classement des poules comportant uniquement les 2 premiers de chaque poule. (Annexe)

- Rencontre_Matchs.txt : liste des matchs qui se met à jour à chaque étape de la phase éliminatoire

- Classement_eliminatoire.txt : liste des joueurs de la phase éliminatoire, classés tout au long de cette phase. Ce dernier sert pour la récupération d'un tournoi, et n'est significatif en termes de classement qu'à la fin du tournoi. (Annexe)

Bilan

Travail en amont

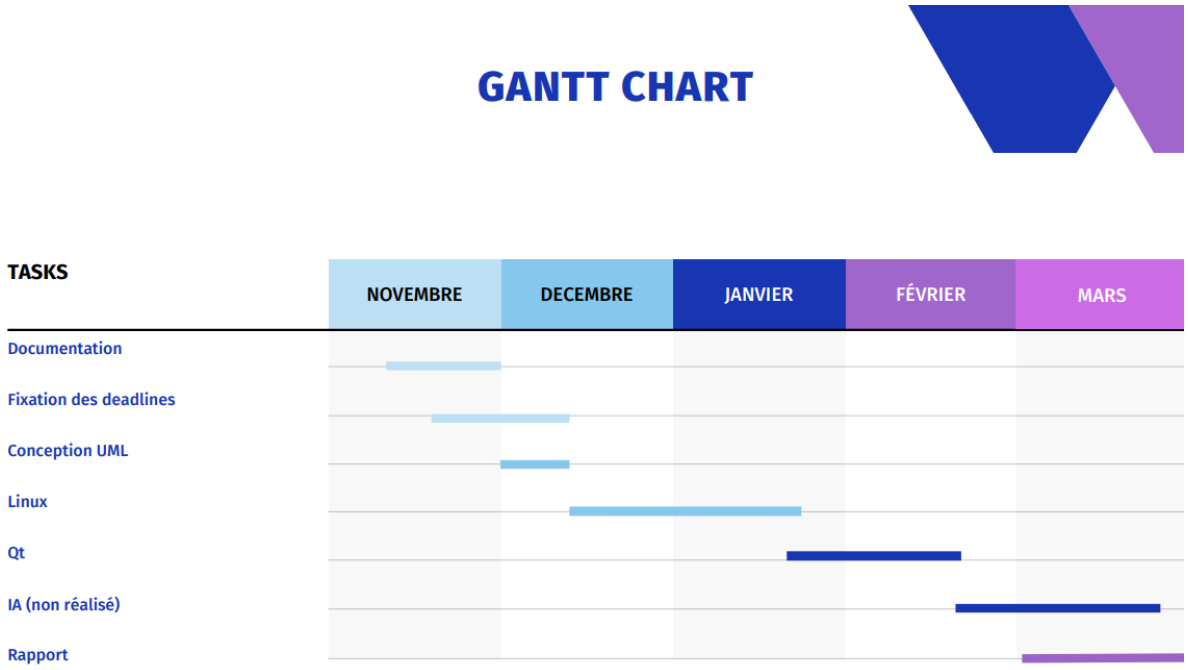


Figure Bilan - GANTT

Dans un premier temps, nous avons pris connaissance des règles et du fonctionnement d'un tournoi de tennis de table. Nous en avons fait une fiche méthode afin de ne pas oublier les points essentiels lors du développement du logiciel. Dans nos contraintes de projet se trouvaient : programmation sous Linux, utilisation de MakeFile et version Qt.
Etant donné que nous utilisons les ordinateurs de Polytech en temps normal, nous maîtrisons bien Linux. Cependant, MakeFile et Qt étaient pour nous deux notions complètement inconnues. De ce fait, nous avons dû visionner des tutoriels mis à notre disposition par notre tuteur de stage et en faire des résumés.

Ensuite, nous avons fixé les étapes de la conception du logiciel avec la conception d'un diagramme UML de classe d'analyse. Ce diagramme à été notre fil conducteur pour toute la conception du programme. Ces étapes, nous les avons mises dans un Gantt *[Figure Bilan].*

Avancement du projet

Nous avons dans un premier temps, respecté les délais du Gantt jusqu'au Linux *[Figure Bilan].* Au lieu de nous prendre 30-60 jours, ce dernier nous en a pris d'avantage. Tout d'abord nous avons eu des problèmes de machine virtuelle pour utiliser Linux à partir de nos ordinateurs. De ce fait, nous avons pris contact avec notre tuteur pour régler le souci de Linux. Nous avons donc commencé à travailler sur les ordinateurs de Polytech, mais ces derniers n'étaient pas souvent disponibles, puis nous sommes entrés en période d'examens.
Pendant les vacances, nous avons donc programmé sur des logiciels qui permettaient de coder en C++ : Visual Studio Code. Pour vérifier le fonctionnement sur Linux, c'est à la rentrée que nous avons testé nos codes sur les ordinateurs de Polytech. C'est ainsi que nous avons été retardés dans le travail.

Durant le cursus 4A, nous avons eu l'occasion de découvrir Qt creator de près en IHM et MakeFile pour le projet en programmation objet orienté. Nous sommes arrivés en phase de programmation en Qt du projet après les enseignements d'IHM. De ce fait, cette partie nous a pris le temps qu'il fallait, sans débordement malgré des complications pour le passage de Linux à Qt. Avec le retard de la partie Linux, nous avons terminé la phase de programmation début mars.

Etant donné que nous n'avons pas de créneau dédié à ce projet, pour travailler en binôme de chez nous, nous avons utilisé GIT avec Visual Studio Code. Cependant, il fallait que l'on soit connecté en même temps pour travailler. Lorsque nous ne pouvions pas travailler en même temps sur GIT, nous avons enregistré notre travail après chaque modification et envoyé à notre binôme. De ce fait, nous avons eu des enregistrements de chaque version de notre logiciel, afin de pouvoir accéder au précédent lors d'un problème.

Difficultés rencontrées

Au cours de ce projet nous avons rencontré plusieurs difficultés qu'elles soient liées au code ou l'utilisation des outils mis à disposition de notre tuteur.

Forge

Dans un premier temps, notre tuteur de stage nous a inscrit dans un projet sur Forge. Dans ce dernier, il est possible de faire un Gantt, de mettre des dossiers, d'écrire le rapport, d'utiliser Git. Cependant, malgré les tutoriels et les fiches méthodes nous n'avons pas réussi à utiliser Git avec Forge.

En guise de solution, nous avons appris à utiliser Git sous Visual Studio Code et avons utilisé un google drive pour le rapport.

Modification du score d'un match joué

Dans notre logiciel, lorsque l'utilisateur souhaite entrer les scores d'une rencontre, d'abord la saisie est vérifiée, puis une confirmation de la saisie est demandée à l'utilisateur. Enfin, les scores sont récupérés pour mettre à jour les points des joueurs.
Cependant, pour une quelconque raison, si l'utilisateur demande à modifier les scores d'une rencontre, il ne pourra pas. Car cela impliquerait d'enregistrer chaque mise à jour pour retourner à une version précédente afin de pouvoir récupérer les anciens points des joueurs. Mais aussi de modifier, si nécessaire, le classement et donc les rencontres générées suite à l'ancien classement. Il n'est donc pas possible de modifier les scores d'une rencontre.

Passage en Qt

Notre première version du logiciel était sous Linux. Un simple affichage dans le terminal avec un menu etc. Cependant, ce dernier ne filtrait pas les chiffres, les lettres. Etant donné que nous allions passer sous Qt, nous avons directement mis en place des filtres en Qt.

La difficulté était d'adapter le code du Linux en Qt : La lecture des fichiers ainsi que leurs enregistrements était impossible. A chaque compilation, Qt s'arrêtait. Nous avons donc poussé nos recherches dans le domaine. Il fallait, en effet, changer les fonctions utilisées. C'est-à-dire que pour Qt, l'utilisation des fichiers passe par QFile.

De plus, sous Linux nous avions un programme principal qui permettait de faire le lien entre l'utilisateur et le code ce qui n'est pas le cas sous Qt. De ce fait, nous avons dû reprendre toutes les fonctions pour les adapter à l'interface Qt. Et ainsi, créer plusieurs classes en Qt pour permettre de faire les liens entre les codes :

- Une classe pour demander à l'utilisateur s'il démarre ou reprend un tournoi
- Une classe pour savoir (s'il reprend un tournoi) la phase dans laquelle il était
- Une classe pour les fonctions principales
- Une classe par fonctions principales
- Une classe pour entrer les numéros de rencontres afin d'accéder à une fonction
- Une classe message qui affiche un message différent selon la situation

C'est ainsi que le code est passé de trois classes à une dizaine de classes.

Intelligence Artificielle

Pour aller plus loin dans le projet, l'utilisation d'une intelligence artificielle était proposée. En effet, nous avions une reconnaissance visuelle à programmer. C'est-à-dire que l'utilisateur aurait juste eu à montrer à sa webcam la fiche de rencontre remplie et le logiciel récupérerait les scores.
Au début du projet, nous avons pensé laisser un mois à la programmation de cette partie. Cependant, nous avons rencontré des problèmes de compilations, des problèmes de fichiers qui nous ont retardés. De ce fait, il ne nous restait plus que 10 jours pour cette partie. Nous avions tout de même effectué quelques recherches sur ce domaine mais an vue du temps qu'il nous restait nous avons préféré ne pas développer cette partie. Nous avions mal prévu

le temps consacré à cette partie car nous manquons de compétences. De ce fait, un mois ou deux pour réaliser la reconnaissance visuelle auraient été plus juste.

Nous nous sommes donc concentrées sur la rédaction du rapport et avons poussé Qt plus loin en permettant par exemple de saisir un fichier à la souris et non en écrivant tout le chemin de ce dernier.

Amélioration éventuelle

Avec plus de temps nous aurions pu ajouter plusieurs fonctionnalités à notre logiciel. Par exemple, la modification du score d'un match dont la saisie a été validée ou encore la possibilité d'ajouter un ou plusieurs joueurs au début du tournoi. Il aurait été intéressant que l'utilisateur puisse choisir le type de tournoi qu'il souhaite. Il pourrait ainsi paramétrer le tournoi en fonction du nombre de poules qu'il souhaite, s'il y a des matchs retour, repêchage ou non etc. De plus, la déclaration de forfait lors d'une rencontre pourrait être une possibilité à ajouter au logiciel. La reconnaissance visuelle serait également une grande amélioration du projet.

Ressenti

Ce projet a été un bon approfondissement de nos compétences apprises au cours de l'année. En effet, en début de 4A nous avons appris à coder en C++ puis en même temps que le commencement de notre projet nous avons pu découvrir l'utilisation de Qt en cours. Ainsi nous avions déjà des connaissances et cela nous a permis de les renforcer mais aussi d'avancer plus facilement sur le projet. Qt et C++ n'ont plus de secret pour nous !

Ce travail a été très intéressant, nous sommes déçus de ne pas avoir eu plus de temps pour pouvoir réaliser la partie sur l'intelligence artificielle qui pour nous était la plus enrichissante techniquement.

On a dû passer deux heures par semaine en début de projet puis après les vacances une intensification du travail pouvant aller jusqu'à 6h par semaine. Ce qu'il nous manquait, nous pensons, ce sont des créneaux dédiés à notre projet. Car nous ne sommes pas disponibles en même temps durant les weekends ou en semaine ce qui complique l'avancement du travail. Malgré cela nous avons pu travailler en binôme sur Linux puis nous avons avancé chacune de notre côté sur Qt avec un envoi par mail des différentes améliorations de notre projet.

En suggestion d'amélioration, nous proposons une formation sur Git, GitHub ou GitLab. Ces derniers nous auraient facilité le travail en binôme.

Conclusion

Avec cinq mois de travail sur ce projet nous avons pu finaliser une version fonctionnelle de notre logiciel d'arbitrage. Il est possible de démarrer un tournoi ou d'en reprendre un en cours. Le logiciel réalise automatiquement les phases de poules et génère toutes les rencontres du tournoi. L'utilisateur peut entrer les scores des différentes rencontres et d'en afficher une ou toutes celles non jouées. Il est aussi possible d'imprimer des feuilles de rencontres.

Ainsi nous sommes fières de notre rendu de logiciel et pensons qu'il serait intéressant qu'un nouveau groupe de 4A IMDS reprennent notre projet l'année prochaine. Cela en vue de réaliser les améliorations que nous avons proposées. Ainsi le logiciel serait totalement fonctionnel et pourrait, qui sait, être commercialisé.

Remerciements

Nous souhaitons tout d'abord remercier M. LENGAGNE, notre tuteur de Projet, pour ses explications et son suivi tout au long de ce projet. Il a su nous expliquer ces différentes attentes pour projet. Il nous mit à disposition ses plusieurs tutoriels pour l'utilisation de GIT ainsi que pour l'utilisation de MakeFile.

De plus, nous voulons remercier M. de Vaulx pour son aide en Qt. Il a pris le temps de nous expliquer l'affichage des boîtes message. Grâce à lui nous avons pu approfondir nos connaissances en Qt et ainsi développer une meilleure interface.

Enfin, nous aimerions remercier M. Bouchon pour avoir répondu rapidement à nos questions sur notre rapport et le déroulement de notre soutenance.

Annexes

Méthode importante classe Match en Qt

La seule méthode “compliquée” de la classe match est la méthode setEcart(). Les autres ne contribuent point à la compréhension du code.

```
//Méthode pour modifier les points et les écarts des joueurs

void Match::setEcart(int a, int b){

  if (a<b){ //si joueur A gagne
     joueurB->setEcart(b-a);
     joueurA->setEcart(a-b);
     joueurB->setPoint(2); //2 points pour le gagnant

  }
  if (a>b){ //si joueur B gagne
     joueurA->setEcart(a-b);
     joueurB->setEcart(b-a);
     joueurA->setPoint(2);
  }
  if(a==b){ // si egalite , tout le monde a 1 point
     joueurA->setPoint(1);
     joueurB->setPoint(1);
  }

}
```

Méthodes importantes classe Tournoi en Qt

Dans cette partie, les méthodes get et set n’ont pas été mises ainsi que les constructeurs. Elles n’ont aucune particularité et ne contribuent pas à la compréhension du code.

```
// Méthode pour créer un fichier au nom donné

void Tournoi::creerFichier(string nomFichier){

// Création d'un objet QFile
  QString nom=(chemin+nomFichier).c_str();
  QFile fichier(nom);
```

```
  // On ouvre notre fichier en lecture seule et on vérifie l'ouverture
  if (!fichier.open(QIODevice::WriteOnly | QIODevice::Text))
      cerr << "Impossible d'ouvrir le fichier !" <<nomFichier<< endl;

  else
      fichier.close();
}
```

//Méthode pour créer liste joueur à partir d'un fichier

```
void Tournoi:: fichierJoueurs(string nomFichier){

//initialisation des données
   string Licence;
   string nom;
   string prenom;
   int point;
   int ecart;
   int poule;
   int nb;
   QFile fichier((nomFichier).c_str());

//ouverture fichier en lecture seule
   if(fichier.open(QIODevice::ReadOnly | QIODevice::Text))
   {
      QTextStream flux(&fichier);
      nb=flux.readLine().toInt();
      nbjoueurs=nb; // récupère le nb de joueurs

      for(int i=0;i<nbjoueurs;i++){
         Licence=flux.readLine().toStdString();
         nom=flux.readLine().toStdString();
         prenom=flux.readLine().toStdString();
         point=flux.readLine().toInt();
         ecart=flux.readLine().toInt();
         poule=flux.readLine().toInt(); // récupère les attributs du joueur

          // création joueur et initialisation des point, écart et poule a zéro
          Joueur* j=new Joueur(Licence,nom,prenom);

         //attribution valeurs
         j->setPoint(point);
         j->setEcart(ecart);
         j->setPoule(poule);
```

```
            listejoueurs.push_back(j); // ajout dans la liste
        }
        fichier.close();

    }
        else
          erreur=true; // déclare erreur pour ne pas continuer le tournoi malgré fichier mal
importé

}
```

//Méthode pour la création des poules

```
void Tournoi :: Poules(){
    int nbpoules;
//  determination nombre de poules
    if (nbjoueurs<24){
        nbpoules= 4; //  car max 6 joueurs par poules
    }
    else { nbpoules=8;}

    // attribuer les poules aux joueurs qui complètent les poules
    int i=0;
    int mod= int(nbjoueurs/nbpoules);
    while(i<( mod *4)){
        for(int j=1;j<=nbpoules;j++){
            listejoueurs[i]->setPoule(j);
            i++;
        }
    }

    // ceux qui sont en plus des 4 par poules
    int j = 1;
    for (i=nbpoules*4; i<nbjoueurs; i++){
        listejoueurs[i]->setPoule(j);
        j++;
    }

//  mise a jour des joueurs
    joueur();

    // Créer les matchs dans chaque poule
    i = 0;
    int num_match = 1;
```

```
   while(i<nbjoueurs){
      for (j=i+1; j<nbjoueurs; j++){ //parcours les joueurs qui sont après le joueur i
          //si joueur i et joueur j sont dans la même poule
          if (listejoueurs[i]->getPoule() == listejoueurs[j]->getPoule()){
                    Match *M = new Match(num_match,listejoueurs[i],listejoueurs[jj],0,0,false); //
          creation match de i contre j
             matchs.push_back(M);
             num_match ++;
          }
      }
      i++;
   }
   nbmatch = matchs.size();

   // Création de notre fichier rencontre de poules avec toutes les rencontres d'une poule
   match("Rencontre_Poule.txt");
}
```

//Méthode mise à jour des joueurs

```
void Tournoi::joueur(){

   string nomFichier="Joueurs_maj.txt";
   creerFichier(nomFichier);
   QFile fichier((chemin+nomFichier).c_str());

   if (fichier.open(QIODevice::ReadWrite | QIODevice::Text)) {

   // Si l'ouverture du fichier en écriture à réussie
      QTextStream out(&fichier);
      out<<nbjoueurs<<"\n";
      for (int i=0;i<nbjoueurs;i++){
          out<<listejoueurs[i]->getnumLicence().c_str()<<"\n"
          <<listejoueurs[i]->getnom().c_str()<<"\n"
         <<listejoueurs[i]->getprenom().c_str()<<"\n"
          <<listejoueurs[i]->getPoint()<<"\n"
          <<listejoueurs[i]->getEcart()<<"\n"
          <<listejoueurs[i]->getPoule()<<"\n";}

      // Fermer le fichier
      fichier.close();
   }}
```

//Méthode saisie des scores

```
void Tournoi::entrerScores(int num,int scoreA,int scoreB){

   num--;

// les saisies sont vérifiés avant d'appeler la fonction

   matchs[num]->setScoreA(scoreA);
   matchs[num]->setScoreB(scoreB);
   matchs[num]->setEcart(scoreA,scoreB);
   matchs[num]->setJoue(1);

// mise à jour des joueurs et des rencontres selon la phase
   joueur();
   if ((this->phase)==0){
      match("Rencontre_Matchs.txt");
   }
   else{
      match("Rencontre_Poule.txt");
   }}

//Méthode création des matchs pour phase juste après poules

void Tournoi::creerMatchsFinPoule(){
   nbjoueurs=classement.size();
   int j=1;
   matchs.clear();
   listejoueurs.clear();

   for(int i=0;i<nbjoueurs;i+=2){
// dans le classement il y a : 1er de P1, 2e de P1, 1er de P2, 2e de P3 etc..

      Joueur *A= new Joueur;
      Joueur *B= new Joueur;

      if(i==(nbjoueurs-2)){ // pour éviter les "out of range"
         A=classement[1]; // le 2e de la poule 1 contre
         B=classement[i]; // le 1er de la dernière poule

      }
      else{
         A=classement[i]; // 1er P1 contre
         B=classement[i+3]; //2e P2 etc..

      }

      Match *M = new Match(j,A,B,0,0,false);

 //mise à jour de la liste des joueurs après les poules
```

```
        listejoueurs.push_back(A);
        listejoueurs.push_back(B);
        matchs.push_back(M);
        j++;
    }
    nbmatch = matchs.size();

//mise à jour
    joueur();
    match("Rencontre_Matchs.txt"); // création du fichier

}
```

//Méthode création des matchs pour phase après poules : éliminatoires

```
void Tournoi::creerMatchsEliminatoire(){

// création du nouveau vecteur matchs qui va contenir les nouveaux matchs
    vector<Match*> matchs2;

//s'il ne reste plus qu'un seul match : finale
    if (nbmatch==1){
        if (matchs[0]-> getScoreA() > matchs[0]->getScoreB()){ //si A a gagné
            classement_eliminatoire.push_back(&matchs[0]->getJoueurB() );
            classement_eliminatoire.push_back(&matchs[0]->getJoueurA() );
        }
        else{
            classement_eliminatoire.push_back(&matchs[0]->getJoueurA() );
            classement_eliminatoire.push_back(&matchs[0]->getJoueurB() );
        }
        matchs.clear();
        nbmatch=0;
    }

    else{ //si plus d'un match

//on parcours tous les matchs deux par deux pour faire les matchs suivants
    for (int i=0; i<nbmatch; i=i+2){

//on regarde le 1er match pour savoir qui as gagne
        if ( matchs[i]-> getScoreA() > matchs[i]->getScoreB() ){  //si A1 a gagne
            //on regarde le matchs suivant pour savoir qui as gagne
            if ( matchs[i+1]-> getScoreA() > matchs[i+1]->getScoreB()){ //si A2 a  gagne
                Match *M = new Match((i/2)+1,&matchs[i]->getJoueurA(),&matchs[i+1]-
>getJoueurA(),0,0,false);
                matchs2.push_back(M);
```

```
                classement_eliminatoire.push_back( &matchs[i+1]->getJoueurB() );//joueur B2 a
perdu
            }
            else{ //si B2 gagne

                Match *M = new Match((i/2)+1, &matchs[i]->getJoueurA(), &matchs[i+1]-
>getJoueurB(),0,0,false);
                matchs2.push_back(M);
                classement_eliminatoire.push_back( &matchs[i+1]->getJoueurA() ); //joueur A2
a  perdu
            }
            classement_eliminatoire.push_back( &matchs[i]->getJoueurB()); //joueur B1 a
perdu
        }
        else{//si B1 a  gagne
            if ( matchs[i+1]-> getScoreA() > matchs[i+1]->getScoreB()){ //si A2 a gagne
                Match *M = new Match((i/2)+1, &matchs[i]->getJoueurB(), &matchs[i+1]-
>getJoueurA(),0,0,false);
                matchs2.push_back(M);
                classement_eliminatoire.push_back( &matchs[i+1]->getJoueurB() ); //joueur B2
Ã  perdu
            }
            else{ //si B2 gagne
                Match *M = new Match((i/2)+1, &matchs[i]->getJoueurB(), &matchs[i+1]-
>getJoueurB(),0,0,false);
                matchs2.push_back(M);
                classement_eliminatoire.push_back( &matchs[i+1]->getJoueurA() );//joueur A2 a
perdu
            }
            classement_eliminatoire.push_back(&matchs[i]->getJoueurA() );// joueur A1 a
perdu
        }

      }
      matchs.clear(); //on efface les anciens matchs qui sont passés
      matchs = matchs2; //on affecte les nouveau matchs à faire
      nbmatch = matchs.size();}
   match("Rencontre_Matchs.txt");
}
```

//Méthode création fichier à imprimer

```
void Tournoi::imprimerRencontre (){
   string nomFichier="RencontreAImprimer.txt";
   creerFichier(nomFichier);
   QFile fichier((chemin+nomFichier).c_str());
```

```
    if (fichier.open(QIODevice::ReadWrite | QIODevice::Text)) {
    // Si l'ouverture du fichier en écriture à réussie

        QTextStream out(&fichier);
        int j=0;
        for (int i=0;i<nbmatch;i++){
//après 4 rencontres on saute des lignes pour avoir 4 rencontres par feuille
            if(j==4){
                out<<"\n\n";
                j=0;
            }
        out<<"_____"<<"\n"
          <<"Match num:"<<matchs[i]->getNumMatch()<<"\n"
          <<"\n"
          <<"Joueur num :"<<matchs[i]->getJoueurA().getnumLicence().c_str()<<"\n"
          <<"Nom :"<<matchs[i]->getJoueurA().getnom().c_str()<<"\n"
          <<"Prenom :"<<matchs[i]->getJoueurA().getprenom().c_str()<<"\n"
          <<"SCORE : "
          <<"\n"
          <<"\n"
          <<"Joueur num :"<<matchs[i]->getJoueurB().getnumLicence().c_str()<<"\n"
          <<"Nom :"<<matchs[i]->getJoueurB().getnom().c_str()<<"\n"
          <<"Prenom :"<<matchs[i]->getJoueurB().getprenom().c_str()<<"\n"
          <<"SCORE : "
          <<"\n"
          <<"_____"<<"\n";
        j++;
        }// Fermer le fichier
        fichier.close();
    }
    else{
        cout<<"erreur"<<endl;
    }}
```

//Méthode pour afficher une rencontre

```
string Tournoi::afficherRencontre(int num){
    string ss;
    num--;

    ss="MATCH NUM : ";
    string str;
    stringstream s;
    s<<(matchs[num]->getNumMatch());
    s>>str;
    ss=ss+str;
```

```
   ss=ss+"\n\n";
   if (matchs[num]->getJoue()){ //selon si rencontre joué
      ss=ss+"Joué \n";
   }
   else{
      ss =ss+"Non joué";
   }
   ss = ss + "\n\n";
   ss = ss + "Joueur A : \n";
   ss= ss+ "Num Licence : " + matchs[num]->getJoueurA().getnumLicence()+"\n";
   ss = ss + "Nom : "+ matchs[num]->getJoueurA().getnom()+ "\n";
   ss = ss + "Prénom : "+ matchs[num]->getJoueurA().getprenom() +"\n";
   ss=ss+"\n" + "CONTRE" + "\n\n";
   ss = ss + "Joueur B : \n";
   ss= ss+ "Num Licence : " + matchs[num]->getJoueurB().getnumLicence() + "\n";
   ss = ss + "Nom : "+ matchs[num]->getJoueurB().getnom()+ "\n";
   ss = ss + "Prénom : "+ matchs[num]->getJoueurB().getprenom()+"\n";
   ss=ss+"\n"+ "_____________________________________________________\n\n";

 //renvoi l'affichage sous forme de chaine de caractère, pour insérer dans le textEdit
return ss; }
```

//Méthode pour afficher toutes les rencontres non jouées

```
string Tournoi::afficherRencontresNonJoue(){
   string ss="";

   for(int i=0; i<nbmatch;i++){
      if(!(matchs[i]->getJoue())){ //vérifie que non joué
         ss=ss+afficherRencontre(i+1);  // fait appel à la fonction pour afficher une rencontre
      }
   }
   return ss;
}
```

//Méthode pour créer/ mettre à jour fichier de sauvegarde des rencontres

```
void Tournoi::match(string nomFichier){
   creerFichier(nomFichier);
   QFile fichier((chemin+nomFichier).c_str());
   if (fichier.open(QIODevice::ReadWrite | QIODevice::Text)) {
   // Si l'ouverture du fichier en écriture à réussie
      QTextStream out(&fichier);
      out<<nbmatch<<"\n";
```

```
    for  (int i=0;i<nbmatch;i++){
        out<<matchs[i]->getNumMatch()<<"\n"
          <<matchs[i]->getJoue()<<"\n"
          <<matchs[i]->getJoueurA().getnumLicence().c_str()<<"\n"
          <<matchs[i]->getScoreA()<<"\n"
          <<matchs[i]->getJoueurB().getnumLicence().c_str()<<"\n"
          <<matchs[i]->getScoreB()<<"\n";}
      // Fermer le fichier
      fichier.close();
  }}
```

//Méthode pour créer fichier de classement des matchs éliminatoire

```
void Tournoi::classerEliminatoire(){
  string nomFichier="Classement_Eliminatoire.txt";

  creerFichier(nomFichier);
  QFile fichier((chemin+nomFichier).c_str());
  if (fichier.open(QIODevice::ReadWrite | QIODevice::Text)) {
  // Si l'ouverture du fichier en écriture à réussie
      QTextStream out(&fichier);

//parcours vecteur de classement_eliminatoire
      for (int i=0;i<int(classement_eliminatoire.size());i++){
          out<< "Joueur arrivee : "<< int(classement_eliminatoire.size()) - i<<"e"<<"\n"
          <<classement_eliminatoire[i]->getnumLicence().c_str()<<"\n"
          <<classement_eliminatoire[i]->getnom().c_str()<<"\n"
          <<classement_eliminatoire[i]->getprenom().c_str()<<"\n";
      }
      fichier.close();
  }
}
```

//Méthode pour vérifier si nous avons fini la phase

```
bool Tournoi::finMatchs(){
  for(int i=0;i<nbmatch;i++){
      if (!matchs[i]->getJoue()){ //vérifie qu'ils sont tous joués
          return false;
     } }
  return true;}
```

//Méthode pour classer les poules et créer le fichier classement

```
void Tournoi::classerPoule(){ //creer fichier de classement et vecteur classement
  if (finMatchs()){
     int nbpoule;
     if (nbjoueurs<24){
        nbpoule= 4; //  car max 6 joueurs par poules
     }
     else { nbpoule=8;}
     int p;
     int p2;
     int max1;
     int max2;


     for (int poule=1;poule<=nbpoule;poule++){
        p=0;
        p2=0;
        max1=-1;
        max2=-2;
        for(int i=0;i<nbjoueurs;i++){ // on trie en fonction des points et écart si égalité
           if (listejoueurs[i]->getPoule()==poule){ // on va dans la bonne poule

              if(max1<=listejoueurs[i]->getPoint()|| max2<=listejoueurs[i]->getPoint()){
                 if(max1<listejoueurs[i]->getPoint()){// cas ou c'est le plus grand point
                    if(max1>max2){ //l'ancien premier devient 2e
                       max2=max1;
                       max1=listejoueurs[i]->getPoint();
                       p2=p;
                       p=i;
                    }
                    else if (max1==max2){ // si 2e même points que l'ancien premier, on
regarde les écarts
                       if (listejoueurs[p2]->getEcart()<listejoueurs[p]->getEcart()){
                          max1=listejoueurs[i]->getPoint();
                          p2=p;
                          p=i;
                       }
                       else if(listejoueurs[p2]->getEcart()>listejoueurs[p]->getEcart()){
                          max1=listejoueurs[i]->getPoint();
                          p=i;
                       }
                    }

                 }
                 else if(max1==listejoueurs[i]->getPoint()){ //le cas où le joueur a le même
point que le premier
                    if (listejoueurs[p]->getEcart()<listejoueurs[i]->getEcart()){ //on regarde les
écarts
```

```
                    if (max1>max2){ // il devient 2e l'ancien premier
                        max2=max1;
                        p2=p;
                        p=i;
                    }
                    else if(max1==max2){
                        if (listejoueurs[p2]->getEcart()<listejoueurs[p]->getEcart()){
                            p2=p;
                            p=i;
                            }
                    }
                    else {
                        p=i;
                    }

                }
                else if (listejoueurs[p]->getEcart()>listejoueurs[i]->getEcart()){ //on regarde
les écarts
                    if (max1>max2){ // joueur devient 2e
                        max2=max1;
                        p2=p;
                        p=i;
                    }
                    else if(max1==max2){ // on regarde les écarts entre joueur et 2e
                        if (listejoueurs[p2]->getEcart()<listejoueurs[i]->getEcart()){
                            p2=i;
                        }            } }
            }
            else if (max2<listejoueurs[i]->getPoint()){
                p2=i;
                max2=listejoueurs[i]->getPoint();
            }
            else if(max2==listejoueurs[i]->getPoint() && i!=p2){
                if (listejoueurs[p2]->getEcart()<listejoueurs[i]->getEcart()){
                    p2=i;
                } }} }}
//ajout dans vecteur classement
        classement.push_back(listejoueurs[p]);
        classement.push_back(listejoueurs[p2]);
  } }
//création fichier
  string nomFichier="Classement_Poules.txt";
  creerFichier(nomFichier);
  QFile fichier((chemin+nomFichier).c_str());
  if (fichier.open(QIODevice::ReadWrite | QIODevice::Text)) {
     QTextStream out(&fichier);
     int j=1;
     int w=1;
```

```
        for (int i=0;i<nbjoueurs;i++){
            out<<"Joueur arrivee : "<<j<<"e de la Poule "<<w<<"\n"
            <<classement[i]->getnumLicence().c_str()<<"\n"
            <<classement[i]->getnom().c_str()<<"\n"
            <<classement[i]->getprenom().c_str()<<"\n"
            <<classement[i]->getPoint()<<"\n";
            j++;
            if (j==3){
                w++;
                j=1;  }
        fichier.close();
    }
}
```

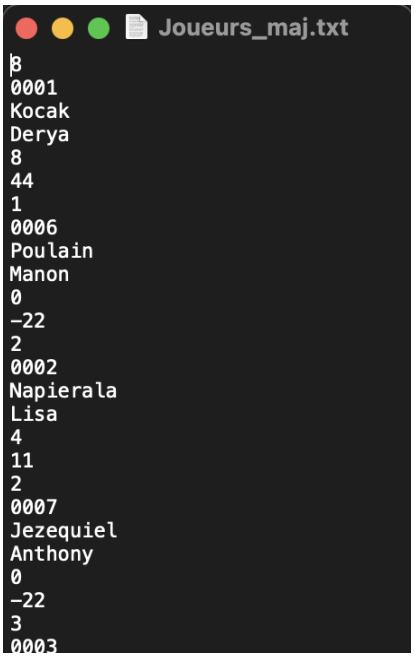
//Méthode pour reprendre un tournoi

```
void Tournoi::recupRencontres(string nomFichier){
    matchs.clear();

//il va récupérer fichier Rencontre_Poule ou Rencontre_match et va le parcourir
    QFile fichier((chemin+nomFichier).c_str());
    if(fichier.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        QTextStream flux(&fichier);
        nbmatch=flux.readLine().toInt();
        Joueur *jB;
        Joueur *jA;
        string numLicenceA,numLicenceB;
        int num,joue,pointA,pointB;

        for(int i=0;i<nbmatch;i++){
            num=flux.readLine().toInt();
            joue=flux.readLine().toInt();
            numLicenceA=flux.readLine().toStdString();
            pointA=flux.readLine().toInt();
            numLicenceB=flux.readLine().toStdString();
            pointB=flux.readLine().toInt();
            for(int j=0;j<nbjoueurs;j++){
                if((listejoueurs[j]->getnumLicence())==numLicenceA){
                    jA=listejoueurs[j];
                }
                if(listejoueurs[j]->getnumLicence()==numLicenceB){
                    jB=listejoueurs[j];

                }
            }
//on recréer les matchs
            Match *M= new Match(num,jA,jB,pointA,pointB,joue);
```

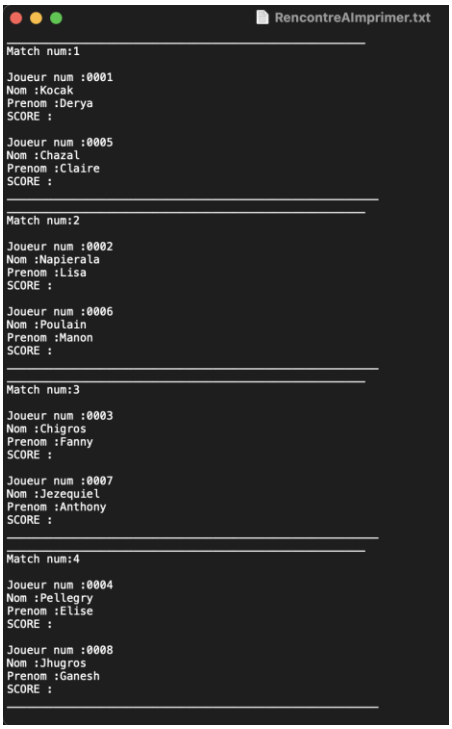
```
            matchs.push_back(M);

        }
        fichier.close();


    }
        else
          cerr << "Impossible d'ouvrir le fichier !" <<nomFichier<< endl;

    //si phase éliminatoire, on récupère les joueurs qui ont été éliminés mais classé
    if(phase==0){
        classement_eliminatoire.clear();
        string blabla;
        QFile file((chemin+"Classement_Eliminatoire.txt").c_str());
        if(file.open(QIODevice::ReadOnly | QIODevice::Text))
        {
            QTextStream flux(&file);
            while (!flux.atEnd()) {
                blabla=flux.readLine().toStdString();
                Joueur *J;
                string numLicence;
                numLicence=flux.readLine().toStdString();
                for (int i=0;i<nbjoueurs;i++){
                    if(listejoueurs[i]->getnumLicence()==numLicence){
                        J=listejoueurs[i];
                        classement_eliminatoire.push_back(J);
                    }}} }
        else
            cout<<"erreur pour fichier classement eliminatoire"<<endl;
}}
```

Exemples de fichiers enregistrés

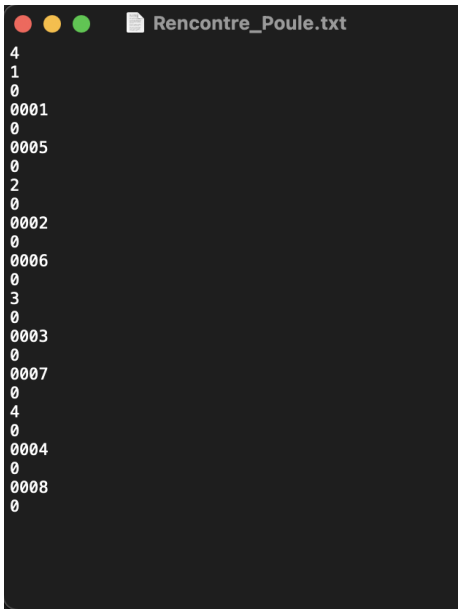
Joueurs_maj.txt



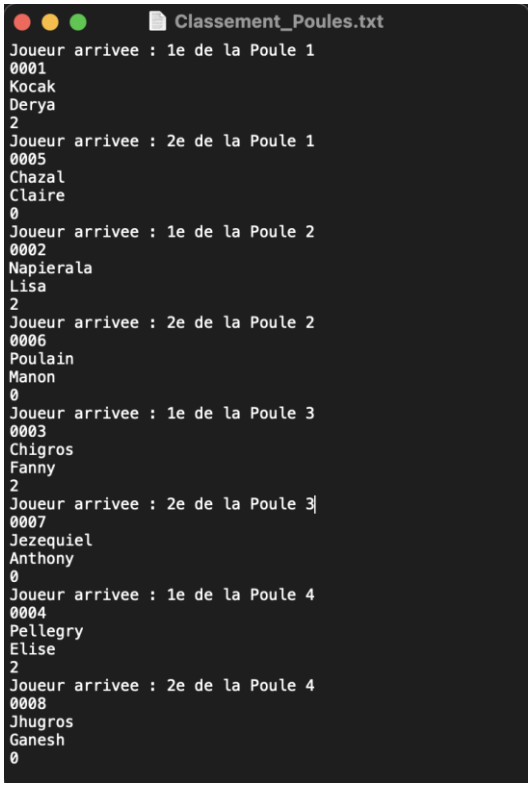
RencontreAImprimer.txt



Rencontre_Poule.txt



Classement_Poule.txt



Classement_Eliminatoire.txt


Classement_Eliminatoire.txt
Joueur arrivee : 8e
0007
Jezequiel
Anthony
Joueur arrivee : 7e
0006
Poulain
Manon
Joueur arrivee : 6e
0004
Pellegry
Elise
Joueur arrivee : 5e
0008
Jhugros
Ganesh
Joueur arrivee : 4e
0005
Chazal
Claire
Joueur arrivee : 3e
0002
Napierala
Lisa
Joueur arrivee : 2e
0003
Chigros
Fanny
Joueur arrivee : 1e
0001
Kocak
Derya