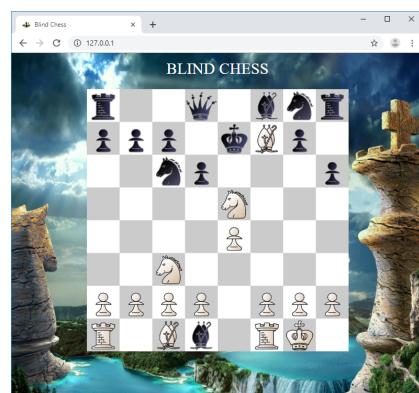




## Module Système 6

Partie d'échecs en réseau retransmise  
sur Internet et sur écran géant à LED



### Table des matières

Objectifs, matériaux et ressources.....	2
TD1 – Cahier des charges et analyse UML.....	4
TD2 – Analyse des diagrammes de classes.....	6
TP – Défi 1 – Déplacement des pièces.....	8
TP – Défi 2 – Gestion du trait -le tour de jeu-.....	12
TP – Défi 3 – Gestion des déplacements particuliers.....	14
TP – Défi 4 – Jeu en réseau : connexion et envoi de la couleur.....	17
TP – Défi 5 – Jeu en réseau : envoi et réception des déplacements.....	19
TP – Défi 6 – Génération de l'image de la position.....	22
TP – Défi 7 – Serveur HTTP.....	24
TP – Défi 8 – Interface graphique et gestion du temps.....	25
Annexe 1 : les règles du jeu.....	28
Annexe 2: diagrammes de classes.....	31

## Objectifs, matériels et ressources

### Objectifs

A partir d'un diagramme de classe détaillé, concevoir une IHM (interface homme-machine) client-serveur, permettant de jouer aux échecs à l'aveugle (ou non), dans une cadence choisie. Les règles du jeu doivent être vérifiées. Le logiciel intégrera un serveur Web en C++ afin que les parties soient visibles depuis un navigateur Web.

- Client TCP en C++
- Serveur TCP en C++
- Serveur HTTP en C++
- Interface graphique
- Thread C++ Builder
- Algorithmie avancée : tableaux à deux dimensions
- UML (cas d'utilisation, déploiement, classe, séquence, activité)

### Matériaux

- Afficheur géant lumineux à LED



- Journal lumineux à LED

### Codes sources

- SNIImage.h
- SNIImage.obj
- SNClientTCP.h
- SNClientTCP.cpp
- SNServeurHTTP.h
- SNServeurHTTP.cpp
- SNServeurTcpMonoclient.h
- SNServeurTcpMonoclient.cpp
- ReglesEtNotation.h
- ReglesEtNotation.cpp
- Deplacement.h

### Ressources et documentation

- Pièces.zip
- Web.zip
- Diagrammes UML de classes : impression recto-verso séparée.
- Documentation constructeur de l'afficheur.

- Deplacement.cpp

### Logiciels

- C++ Builder
- Logiciel constructeur de l'afficheur.
- Logiciel constructeur du journal lumineux
- PositionBMP2JPG.exe

### Annexes

- extrait du site : <https://www.pousseurdebois.fr/cours/regles-du-jeu-d-echecs/>
- Diagrammes de classes

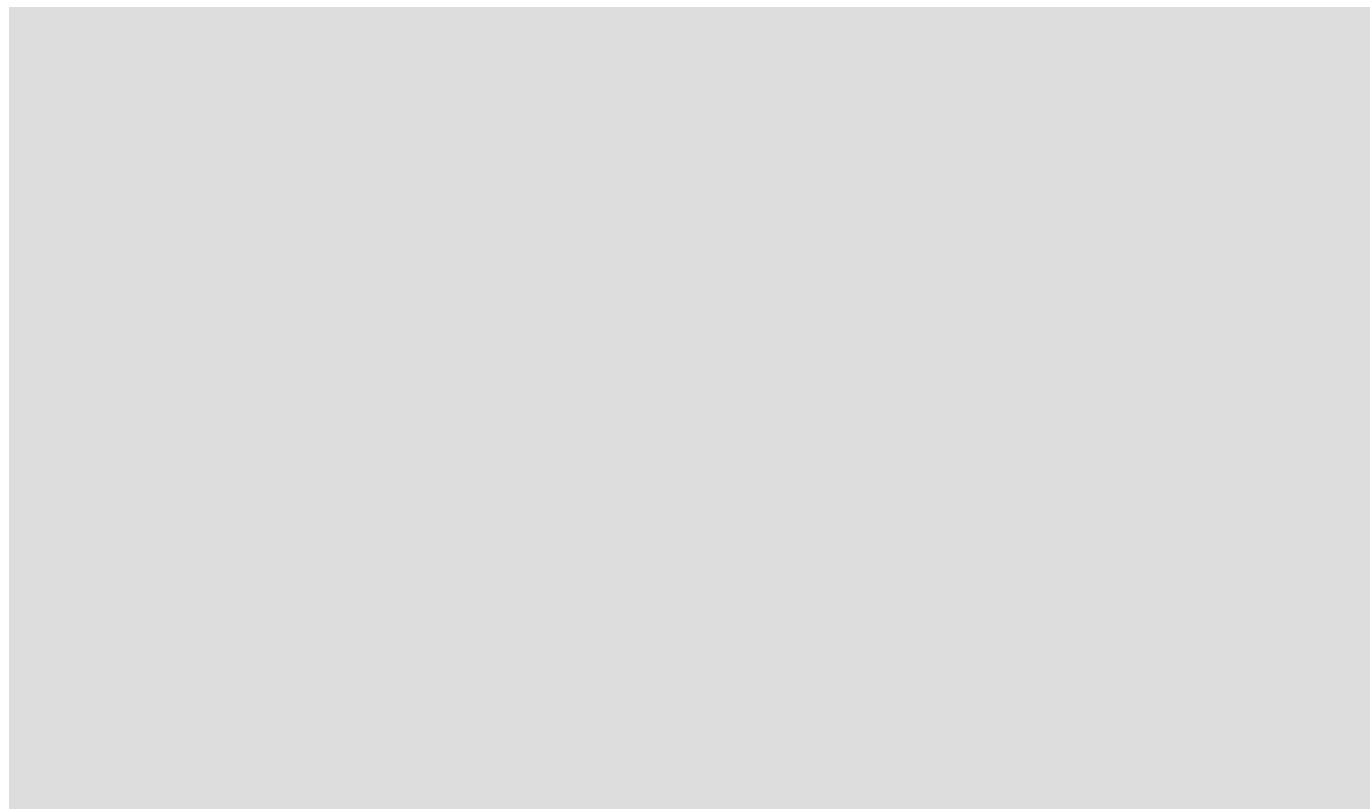
## TD1 – Cahier des charges et analyse UML

### Le cahier des charges

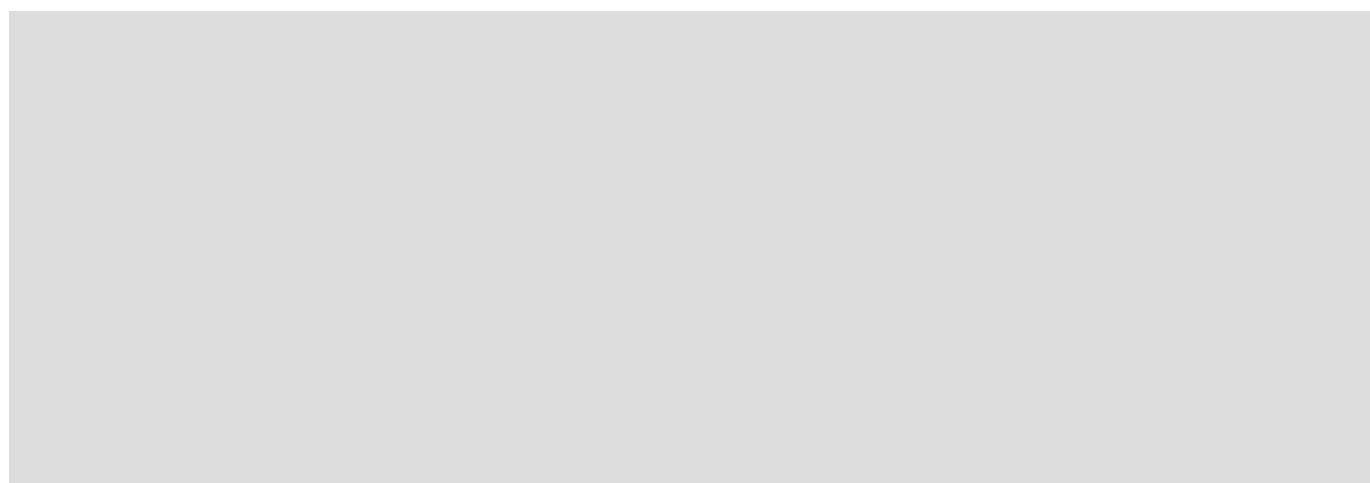
Lors des tournois d'échecs à l'aveugle, les compétiteurs jouent en réseau sur des ordinateurs, en déplaçant des pièces invisibles sur un échiquier vide. Le public peut suivre les matchs grâce à une retransmission sur écran géant à LED avec les pièces visibles, le temps restant à chaque joueur est indiqué (en l'absence d'écran géant, les coups sont indiqués sur un journal lumineux). Les parties sont également suivies sur Internet en temps réel, pour les abonnés disposant d'un login et d'un mot de passe. Les parties seront stockées au format PGN dans une base de données, elles pourront être rejouées.

### Les diagrammes de cas d'utilisation, de déploiement et de séquence

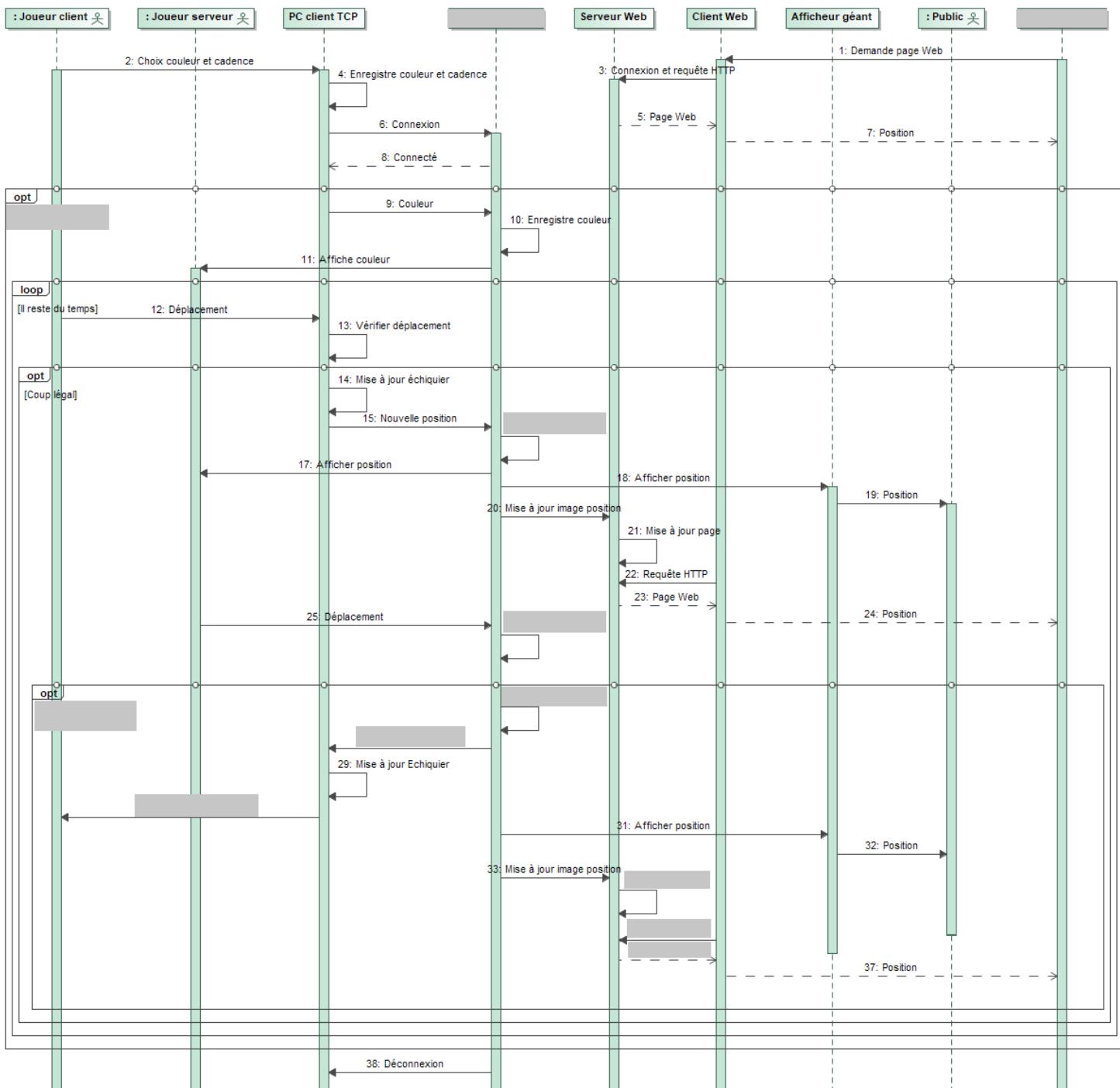
Dessiner le diagramme de cas d'utilisation :



Dessiner le diagramme de déploiement :



## Compléter le diagramme UML de séquence du système complet en remplaçant les zones grisées :



## TD2 – Analyse des diagrammes de classes

**Les diagrammes de classes sont donnés en annexe.**

Convention : les noms des variables et des attributs commencent par des minuscules. Une majuscule sépare les mots qui composent une variable. Une méthode ou une fonction commence par une majuscule.

Un coup est composé d'un déplacement blanc suivi d'un déplacement noir, qu'est ce qu'un demi-coup ?

Le trait est le tour de jeu : on dit que les blancs ont le trait signifie que c'est à eux de jouer. Donner et justifier le type de l'attribut trait, à quelle classe appartient cet attribut ?

Quels attributs permettent de gérer le temps restant ?

tabEchiquier est un tableau à 2 dimensions contenant autant de caractères que de cases sur l'échiquier : donner l'instruction permettant de déclarer ce tableau :

A quoi sert la classe partie ?

Que signifient les 2 liaisons entre les classes Echiquier et Joueur, quel nom porte ce type de lien ?

Quelles classes permettent de vérifier les règles ?



Quelle méthode indique si un déplacement est possible ? justifier son type de retour.



Nommer et justifier le lien entre Deplacement et DeplacementDame :



Quelles classes permettent de jouer en réseau ?



Quelle classe permet le suivi de la partie sur Internet ?



Un Thread est un processus léger : quel est son utilité ?



Donner le nom des 3 objets de type thread dans le diagramme de classes.



Dans la classe thread : ThClientServeur, quelle instruction permet d'initialiser le serveur TCP ?



Dans la classe thread : Quelle instruction permet au client de se connecter ?



Quelle classe permet de gérer l'interface graphique ?

## TP – Défi 1 – Déplacement des pièces

La première classe à coder est la classe Echiquier.



Donner la déclaration de la classe Echiquier partielle donnée ici, sachant que tabEchiquier est tableau à 2 dimensions : [8][8] à la place de [0..\*], les types char"\*\$\*" doivent être remplacés par char\*.

Echiquier
-tabEchiquier : char [0..*]
-demiCoup : int
+Echiquier()
+LireCaseEchiquier( ligne : int, colonne : int ) : char
+InitialiserEchiquier() : void
+VisualiserEchiquier() : string
+Les64Caracteres() : string
+ChargerEchiquierComplet( les64Caracteres : char \$" \$" ) : void
+ModifierCaseEchiquier( cPiece : char, ligne : int, colonne : int ) : void
+EstUnePieceNoire( ligne : int, colonne : int ) : bool
+EstUnePieceBlanche( ligne : int, colonne : int ) : bool
+EstVide( ligne : int, colonne : int ) : bool
+ChangerLeTrait() : void
+Deplacer( idep : int, jdep : int, iarr : int, jarr : int, piecePromotion : char ) : bool
+Trait() : string
+DemiCoup() : int
+SauvegarderEchiquierBMP( fichier : char \$" \$" ) : void

Pour analyser la position (vérifier les règles), et pour transmettre la position en TCP/IP, il est nécessaire de la stocker dans un tableau de lettres à 2 dimensions : tabEchiquier est un tableau contenant des lettres minuscules pour les noirs, et majuscules pour les blancs. Les cases vides sont représentées par un espace ''.

Le premier indice est la ligne, le second la colonne : ainsi tabEchiquier[7][3] contient une dame blanche 'D'.

tcfdrfct  
pppppppp

PPPPPPPP  
TCFDRFCT



Compléter la définition de la méthode InitialiserEchiquier() :

```
void Echiquier::InitialiserEchiquier()
{
    tabEchiquier[0][0] = 't'; tabEchiquier[0][1] = 'c';
    tabEchiquier[0][2] = 'f'; tabEchiquier[0][3] = 'd';
    tabEchiquier[0][4] = 'r'; tabEchiquier[0][5] = 'f';
    tabEchiquier[0][6] = 'c'; tabEchiquier[0][7] = 't';
    tabEchiquier[ ] [ ] = ; tabEchiquier[ ] [ ] = ;
    tabEchiquier[ ] [ ] = ; tabEchiquier[ ] [ ] = ;
    tabEchiquier[ ] [ ] = ; tabEchiquier[ ] [ ] = ;
    tabEchiquier[ ] [ ] = ; tabEchiquier[ ] [ ] = ;
    for(int j=0; j<8 ; j++)
    {   tabEchiquier[1][j] = 'p';
        [REDACTED]
    }
    for(int i=2; i<=5 ; i++)
    {
        [REDACTED]
        {   tabEchiquier[i][j] = [REDACTED] ;
        }
    }
}
```



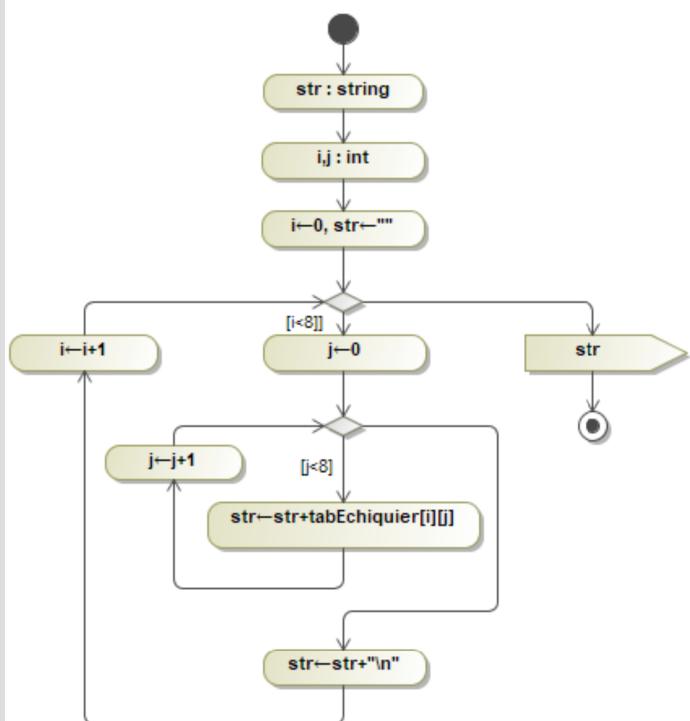
Donner la définition du constructeur chargé d'appeler la méthode d'initialisation :

[REDACTED]



En suivant le diagramme d'activité fourni, donner la définition de la méthode VisualiserEchiquier(), méthode qui construit et retourne une chaîne de caractères de type string contenant la position sur l'échiquier :

[REDACTED]





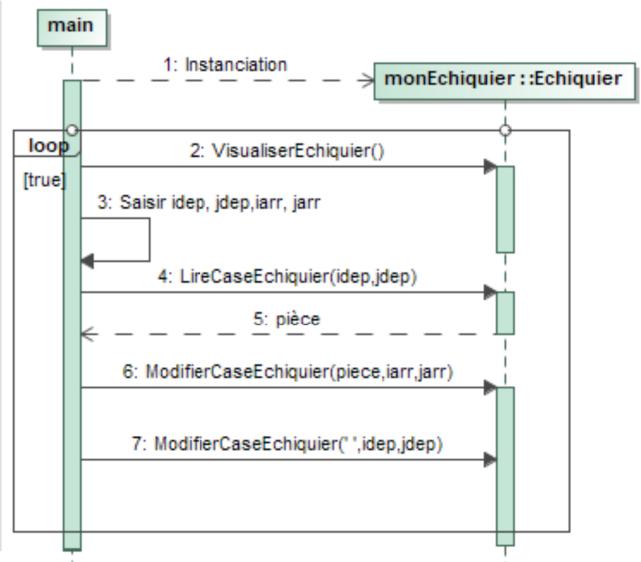
Donner la définition de la méthode LireCaseEchiquier(...) chargé de retourner le caractère présent dans la case [ligne] [colonne] du tableau tabEchiquier :



Donner la définition de la méthode ModifierCaseEchiquier(...) chargé de placer le caractère cPiece dans la case [ligne] [colonne] du tableau tabEchiquier :



Donner le programme principal correspondant au diagramme de séquence suivant :



Coder et tester votre programme en mode console.



Modifier le programme principal : la saisie de -1 pour la ligne de départ entraîne la sortie de la boucle (instruction break).

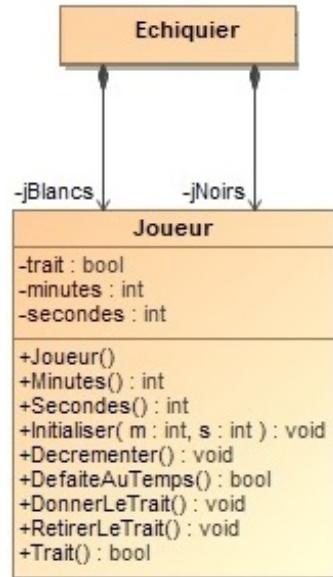


Afin d'alléger le main, placer les méthodes permettant le déplacement dans la méthode Deplacer. Tester votre programme en appelant dans le main cette méthode (mettre un espace '' dans l'argument piecePromotion).

## TP – Défi 2 – Gestion du trait -le tour de jeu-

Chaque joueur doit jouer à son tour. Pour gérer le trait, une classe Joueur doit être codée, deux objets jBlancs et jNoirs seront ajoutés à la classe Echiquier.

**Donner la déclaration de la classe Joueur :**



**Donner la définition des méthodes DonnerLeTrait() et RetirerLeTrait(), ces méthodes permettent de mettre l'attribut trait à true ou bien à false. Donner la définition de la méthode Joueur::Trait() permettant de donner l'accès en lecture à l'attribut du même nom :**

**Quelle ligne faut-il ajouter à InitialiserEchiquier() afin de donner le trait au blancs en début de partie :**

**Donner la définition de la méthode ChangerLeTrait() : si le trait est aux blancs, il faut le donner aux noirs et réciproquement :**



Donner la définition de la méthode Echiquier::Trait() qui reverra "blancs" si les blancs ont le trait, et "noirs" si c'est aux noirs de jouer :

Pour détecter la couleur d'une pièce, où bien l'absence de pièce sur une case, il faut tester si la case contient un espace, une majuscule ou bien une minuscule.



Donner la définition des méthodes EstVide, EstUnePieceNoire et EstUnePieceBlanche. :

Il ne reste plus qu'à vérifier qu'au tour des blancs on déplace une pièce blanche, et une pièce noire au tour des noirs.  
Dans la méthode Deplacer il faut soumettre le déplacement à une longue condition :

SI les cases de départ et d'arrivée sont bien dans l'échiquier (lignes et colonnes entre 0 et 7)

ET

( les blancs ont le trait

ET

une pièce blanche est bien sur la case de départ

ET

une pièce blanche ne se trouve pas sur la case d'arrivée)

OU

( les noirs ont le trait

ET

une pièce noire est bien sur la case de départ

ET

une pièce noire ne se trouve pas sur la case d'arrivée)

)

ALORS effectuer le déplacement puis changer le trait.



Donner en C++ la condition nécessaire au déplacement :

La méthode Deplacer renverra true si le déplacement est possible, false dans le cas contraire.



Coder et tester votre programme : un message "coup illégal" indiquera si le déplacement ne respecte pas le tour de jeu.

## TP – Défi 3 – Gestion des déplacements particuliers

Les règles du jeu sont données en annexe.

### Promotion :

Dans le programme principal, lorsqu'un pion atteint la case de promotion, il faut demander au joueur de choisir la pièce de promotion : D T F C ou bien d t f c.

Donner la condition de promotion en C++ :

Lorsqu'il y a promotion, la pièce choisie est transmise à la méthode Deplacer (piecePromotion). Dans la méthode Deplacer, après avoir effectuer le déplacement, si un pion se retrouve sur la case d'arrivée et que celle ci est située sur la ligne 0 pour les blancs ou sur la ligne 7 pour les noirs, alors ce pion doit être remplacé par piecePromotion.

 Coder et tester votre programme.

### Roque :

Dans déplacer, il faut gérer 4 cas différents : le grand roque blancs, le petit roque blancs, le grand roque noirs, le petit roque noirs.

Cas du grand roque blancs :

7 T R  
0 1 2 3 4

Les cases entre R et T doivent être vides, R doit être en (7,4) et T en (7,0) : alors si R se déplace en (7,2) la tour se déplace automatiquement en (7,3).

Donner le code permettant de tester les conditions de ce grand roque et de déplacer la T de (7,0) à (7,3) :

 Coder et tester votre programme afin de prendre en compte les 4 différents roques.

### Prise en passant :

Un pion capture en diagonale, dans un seul cas il se rend en diagonale sur une case vide : il prend en passant un autre pion qui vient de sauter une case.

Cas du trait aux blancs :

lorsqu'un pion blanc est sur la ligne 3, s'il se déplace en diagonale sur une case vide (2,jdep-1) ou (2,jdep+1) et qu'un pion adverse se trouve respectivement en (3,jdep-1) ou (3,jdep+1), alors ce dernier disparaît.

<b>0</b>	<b>1</b>	<b>0</b>
<b>1</b>	<b>1</b>	<b>1</b>
<b>2</b>	=>	<b>2 P</b>
<b>3 P P</b>		<b>3</b>
<b>0 1 2 3</b>		<b>0 1 2 3</b>

 Donner en C++ la condition de prise en passant d'un pion noir par un pion blanc. :

Dans le cas du trait au noir, les pions vont vers le bas, la ligne de départ doit être la ligne 4, celle d'arrivée la ligne 5.

 Donner en C++ la condition de prise en passant d'un pion blanc par un pion noir :

 Modifier la méthode Deplacer afin de permettre d'effectuer la prise en passant. Tester votre programme.

## TP – Défi 4 – Jeu en réseau : connexion et envoi de la couleur

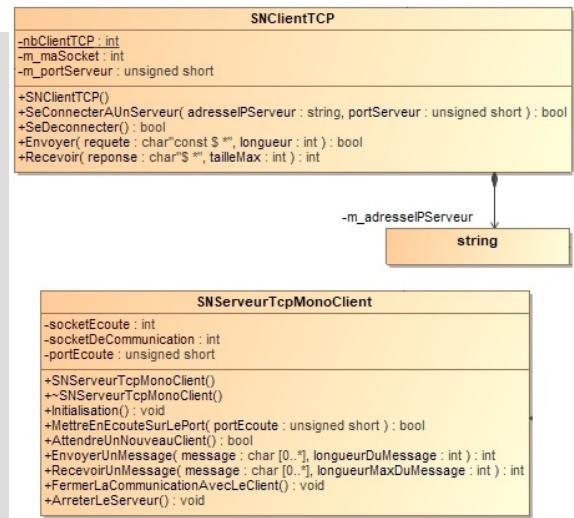
Les classes C++ permettant de coder un serveur TCP et un client TCP sont données. Seul le programme principal sera modifié : une variable entière mode permettra de gérer 3 modes (local, serveur et client). Une variable jaiLesBlancs indiquera la couleur du joueur lors du jeu en réseau. 2 tableaux de caractères seront nécessaires : IP pour stocker l'adresse IP du serveur, et message qui stockera le message reçu ou bien le message à envoyer. Le message contiendra l'ensemble des caractères représentant une position.



Donner la déclaration des 4 variables à ajouter, en précisant la taille des 2 tableaux :



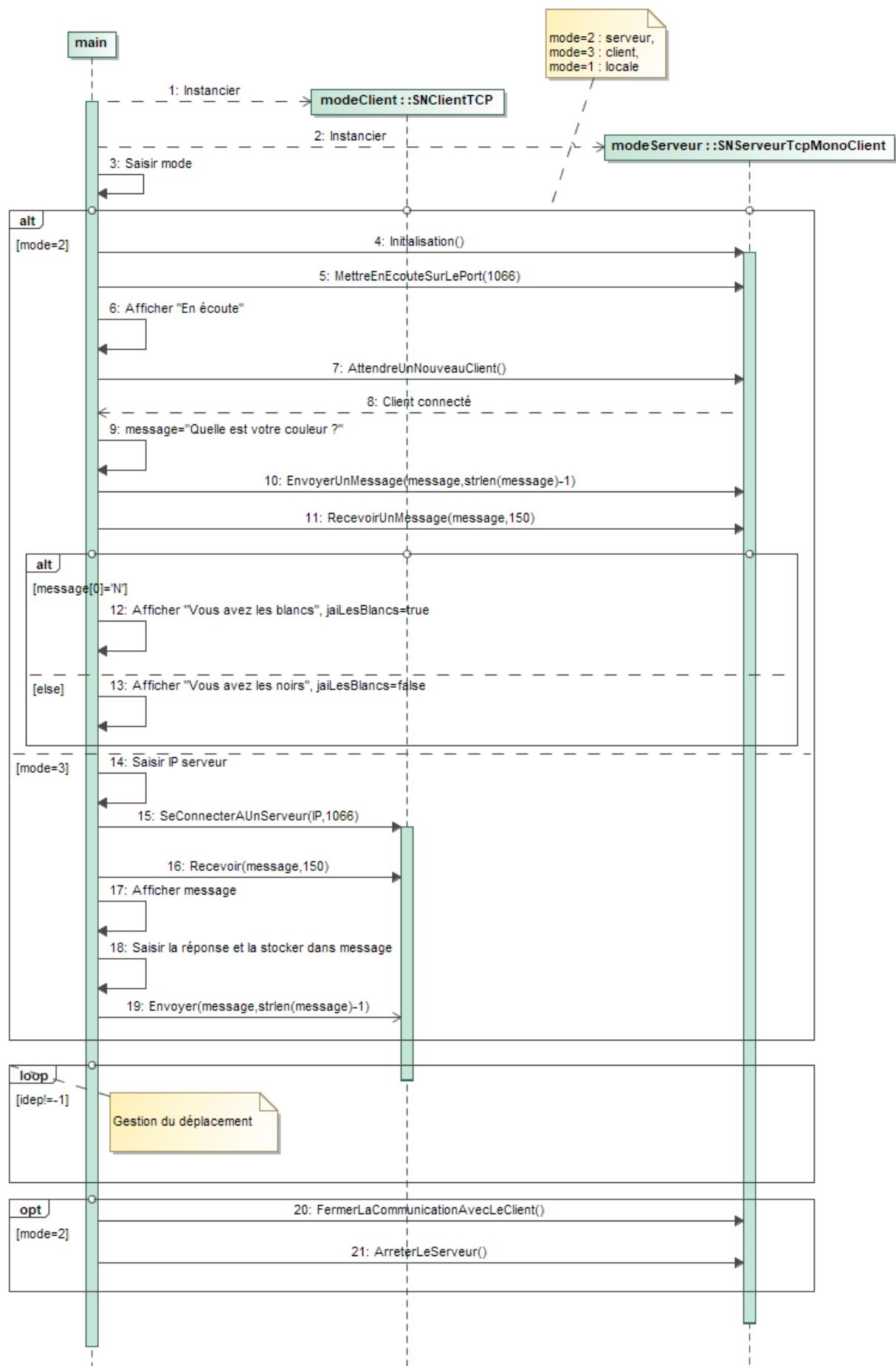
Donner le code partiel du main (sans la gestion du déplacement), correspondant au diagramme de séquence donné page suivante :



Qui choisit sa couleur, le serveur ou bien le client ?



Compléter et tester votre programme principal.



## TP – Défi 5 – Jeu en réseau : envoi et réception des déplacements

Dans la classe Echiquier, deux méthodes doivent être codées : Les64Caracteres et ChargerEchiquierComplet.

Les64Caracteres() vise à mettre sur une même ligne les 64 caractères de l'échiquier. Cette méthode ressemble fortement à VisualiserEchiquier() : il faut y ôter les "\n".

Donner la définition de la méthode Les64Caracteres() :

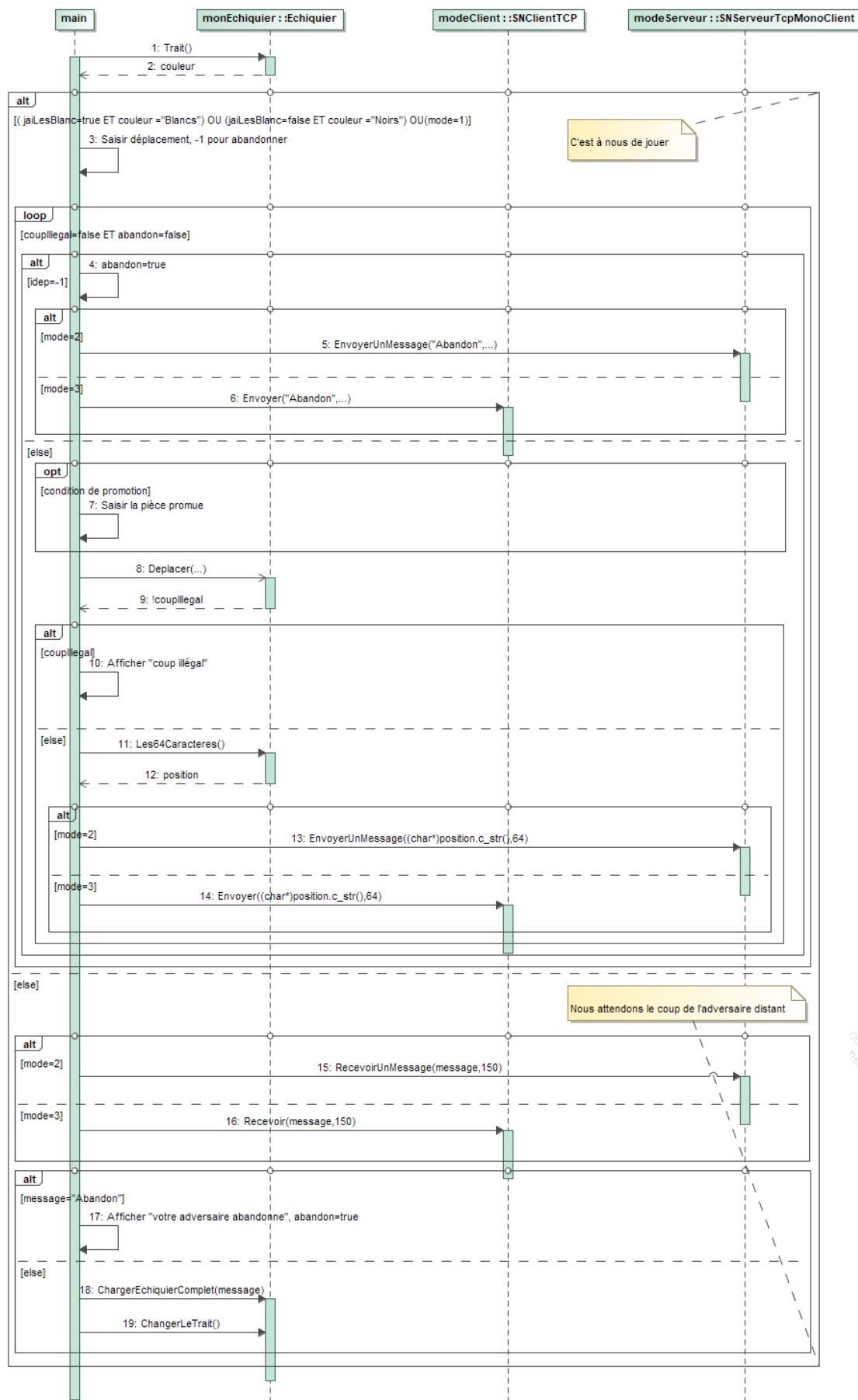
La méthode ChargerEchiquierComplet permet de modifier l'échiquier en y chargeant la nouvelle position envoyée par un adversaire distant : les64Caractères est le tableau de caractères reçu.

Compléter l'algorithme suivant correspondant à la méthode ChargerEchiquierComplet :

```
k←0  
Pour i allant de 0 à 7 par incrément de 1  
    Pour j allant de 0 à 7 par incrément de 1  
        tabEchiquier[i] [j]←  
        k←k+1  
    FinPour  
FinPour
```

Donner la définition de la méthode ChargerEchiquierComplet :

Compléter le programme principal en suivant le diagramme de séquence suivant : ce diagramme représente le contenu de la boucle de déplacement do{ ... }while (!abandon); : tester votre programme.



## Facultatif : Vérification des règles et notation des coups

Les classes ReglesEtNotation et Deplacement sont données.

### Dans la classe Echiquier :

Il faut ajouter les attributs et les méthodes concernant le roque, la mémorisation de l'échiquier précédent (pour la prise en passant notamment), la notation et le numéro du demi-coup :

int demiCoup; //0 au départ, 1 après le premier coup des blancs, 2 après celui des noirs ...  
(à modifier avant notation)

char tabEchiquierPrecedent[8][8]; //contient la position précédente

bool grandRoqueNoirPossible, petitRoqueNoirPossible, grandRoqueBlancPossible,  
petitRoqueBlancPossible;

//sont positionnés à false si le roi ou la tour concernée ont bougé

void SauvegarderEchiquierPrecedent(char tEchiquier[8][8]);

//pour la prise en passant ET après notation

//permet de copier les 64 caractères de tEchiquier dans tabEchiquierPrecedent

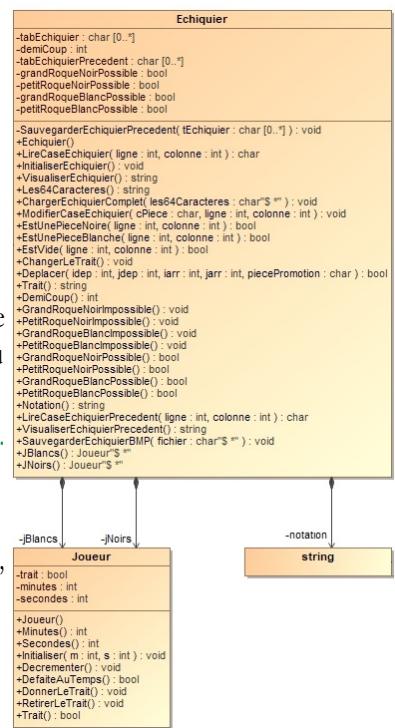
string notation; //contiendra la notation du coup, par exemple 1:e2-e4 signifie que le pion e2 se déplace en e4

int DemiCoup(); //permet l'accès à demiCoup

void GrandRoqueNoirImpossible(); //positionne grandRoqueNoirPossible à false, même principe pour les 3 autres cas

bool PetitRoqueBlancPossible(); //permet l'accès à petitRoqueBlancPossible, même principe pour les 3 autres cas

string Notation(); //permet l'accès à notation



Modifier la déclaration de la classe Echiquier et ajouter la définition des nouvelles méthodes.

### Dans la méthode Déplacer :

Créer un objet de la classe ReglesEtNotation.

Un déplacement sera possible si DeplacementPossible de la classe ReglesEtNotation renvoie true. Le pointeur sur Echiquier à passer en argument est l'objet courant : indiquer le mot clé **this**.

Après un déplacement, incrémenter demicoup.

Stocker dans l'attribut notation le résultat de la méthode EnregistrerNotation.

Appeler la méthode SauvegarderEchiquierPrecedent en passant en argument tabEchiquier.

Modifier la méthode Deplacer et tester le fonctionnement du programme : le programme principal affichera la notation correspondante à chaque coups. Les déplacements illégaux ne sont pas permis.

## TP – Défi 6 – Génération de l'image de la position

La méthode SauvegarderEchiquierBMP utilise la classe SNImage donnée : .h et .obj (le .obj est le .cpp déjà compilé). Cette méthode va générer une image de la position en utilisant les images des pièces (100x100), ainsi que Echiquier.bmp : une image blanche de 800x800. Elle contient quatre boucles for imbriquées.

En étudiant SNimage.h, que représente le type de variable Pixel, s'agit-il d'un tableau, d'une classe, d'une structure... ?

Que représente l'attribut image de la classe SNimage, que signifie \*\* lors de sa déclaration ?

Quelle méthode permet de charger une image ?

Qu'est ce qu'un modulo, et quel symbole représente cet opérateur en langage C ?

A quelle classe appartient la méthode SauvegarderEchiquierBMP ?

Traduire en C++ l'algorithme suivant de la méthode SauvegarderEchiquierBMP :

Créer un objet SNImage pour chaque type de pièces de l'échiquier : **cavB**, **cavN**, **fouB**...

Charger les images avec les objets SNImage créés (cavB.bmp, cavN.bmp, fouB.bmp...) .

Créer un objet **position** de la classe SNImage et charger Echiquier.bmp.

Créer 2 variables de type Pixel : **blanc**={255,255,255} et **gris**={200,200,200}

Pour **ligne** allant de 0 à 8 par incrément de 1

    Pour **colonne** allant de 0 à 8 par incrément de 1

        Pour i **allant** de 0 à 100 par incrément de 1

            Pour j **allant** de 0 à 100 par incrément de 1

                Créer une variable **couleur** de type Pixel

                Initialiser **couleur** à la valeur de **cavB.image[0][0]**

                Si **tabEchiquier[ligne][colonne] = 'C'**

**couleur = cavB.image[i][j]**

                FinSi

                ... faire ainsi pour les 12 types de pièces ...

                Si **couleur=cavB.image[0][0]** (la couleur de transparence)

                    Si (**ligne+colonne**) modulo 2 est différent de 0

**position.image[ligne\*100+i][colonne\*100+i] = gris**

                    Sinon

**position.image[ligne\*100+i][colonne\*100+i] = blanc**

                    FinSi

                Sinon

**position.image[ligne\*100+i][colonne\*100+i] = couleur**

                    FinSi

                FinPour

    FinPour

FinPour

FinPour

Sauvegarder à l'aide de l'objet **position** l'image dans "Position.bmp".

Votre code de la méthode SauvegarderEchiquierBMP :



Coder cette méthode SauvegarderEchiquierBMP, puis appeler cette méthode après VisualiserEchiquier dans le programme principal. Tester votre programme.

**Il reste à convertir le BMP et JPG :**



Quelles sont les différences entre une image bitmap (.bmp) et une image jpeg (.jpg) ?



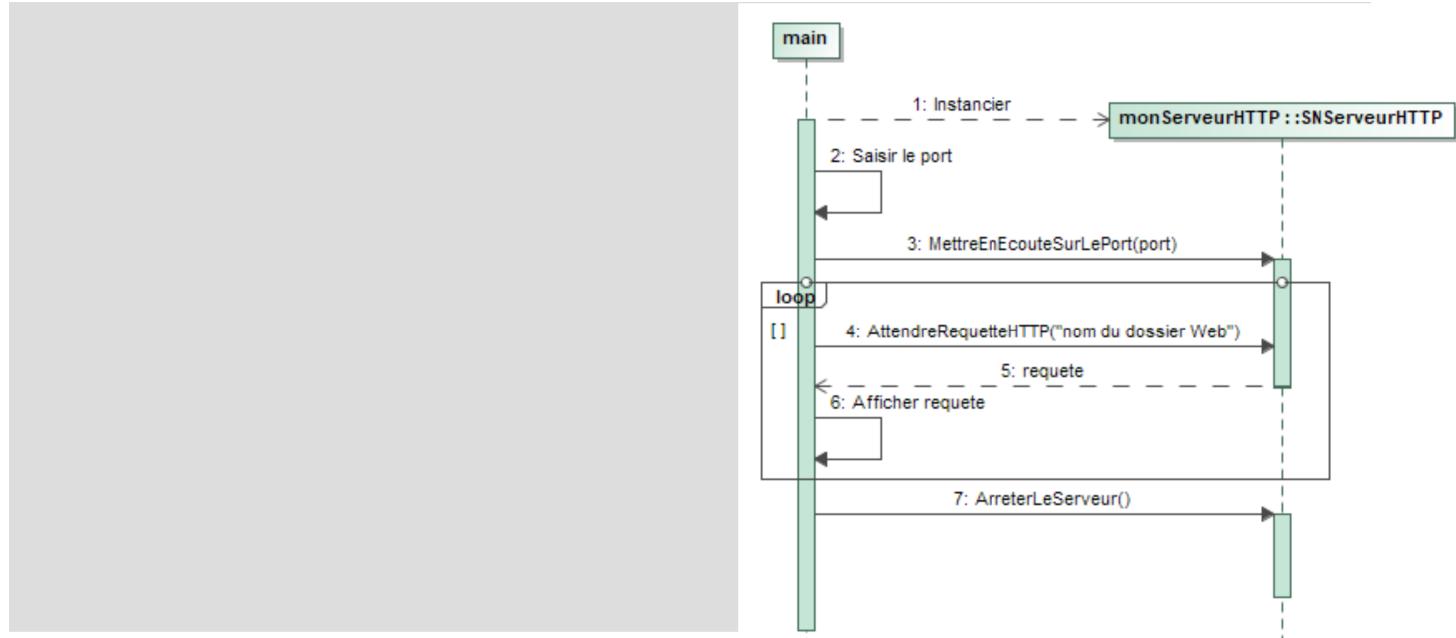
Dans le programme principal, après avoir sauvegarder l'échiquier en BMP, appeler le programme PositionBMP2JPG.exe à l'aide de la fonction WinExec (nécessite Windows.h). Utiliser l'aide de C++ builder ou bien Internet pour connaître le prototype de WinExec. Tester votre programme : une image Position.jpg doit apparaître dans le répertoire de compilation.

## TP – Défi 7 – Serveur HTTP

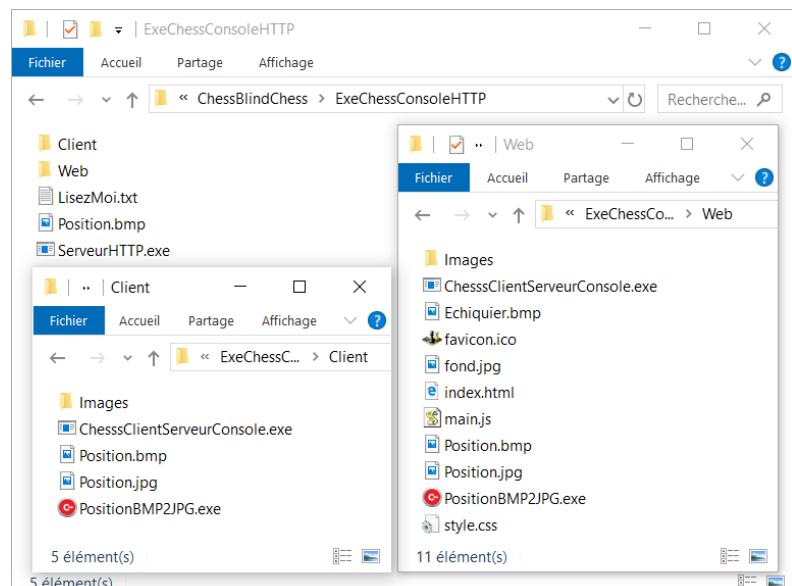
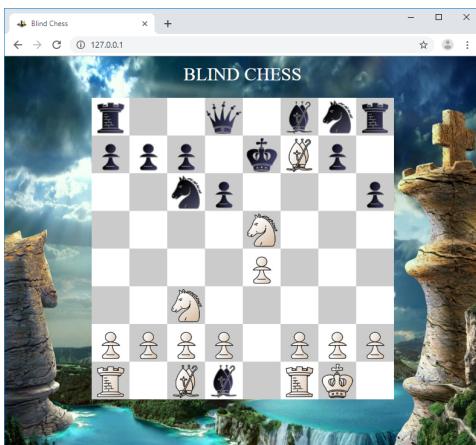
Pour coder et tester le serveur Web en C++, il est nécessaire de créer un nouveau projet en mode console. La classe SNServeurHTTP est donnée. Un exemple de page web minimaliste est proposé dans l'archive Web.zip. Cet exemple est à personnaliser.



Traduire en C++ le diagramme de séquence suivant (le « dossier Web » est un dossier à créer dans le répertoire de compilation) :



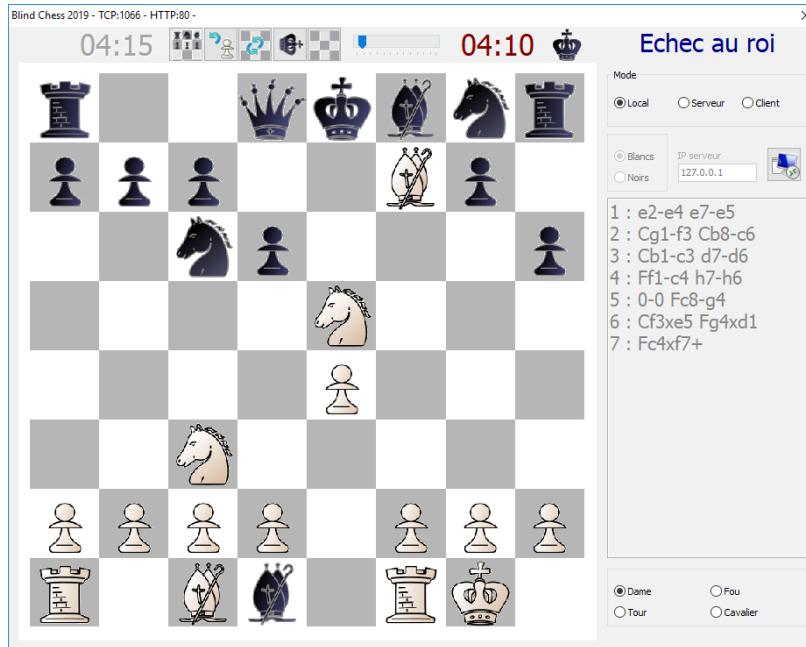
Coder et tester votre programme : il faut utiliser un navigateur Web et se connecter à 127.0.0.1 (serveur Web local).



Tester le jeu complet en plaçant l'exécutable du jeu en mode console ainsi que PositionBMP2JPG.exe dans le répertoire contenant la page Web. Lancer le serveur Web seul, lancer l'exécutable du jeu, connecter un navigateur au serveur Web : la partie peut commencer ! Remarque : pour tester le jeu en client-serveur sur une même machine, il est nécessaire de dupliquer le répertoire contenant l'exécutable du jeu.

## TP – Défi 8 – Interface graphique et gestion du temps

Nous allons maintenant créer une nouvelle application graphique sous C++ Builder. Puis y intégrer nos classes créées en mode console.



### Création de l'IHM et déplacement des pièces (Drag & Drop) :



Programmer sous C++ Builder l'IHM permettant de déplacer une pièce d'une case vers une autre. La méthode LacherPiece de la classe TForm1 est la méthode centrale du déplacement (DRAG and DROP) : elle doit donc être codée en partie. Placer les 64 images (Timage) dans le bon ordre dans la forme (en commençant en haut à gauche par Image1), et ajouter l'attribut TImage \*laCase[8][8] à TForm1. Dans le constructeur, placer les adresses des 64 images dans le tableau laCase :

```
laCase[0][0]=Image1; laCase[0][1]=Image2; laCase[0][2]=Image3; laCase[0][3]=Image4;
laCase[0][4]=Image5; laCase[0][5]=Image6; laCase[0][6]=Image7; laCase[0][7]=Image8;
laCase[1][0]=Image9; laCase[1][1]=Image10; laCase[1][2]=Image11; laCase[1][3]=Image12;
laCase[1][4]=Image13; laCase[1][5]=Image14; laCase[1][6]=Image15; laCase[1][7]=Image16;
laCase[2][0]=Image17; laCase[2][1]=Image18; laCase[2][2]=Image19; laCase[2][3]=Image20;
laCase[2][4]=Image21; laCase[2][5]=Image22; laCase[2][6]=Image23; laCase[2][7]=Image24;
laCase[3][0]=Image25; laCase[3][1]=Image26; laCase[3][2]=Image27; laCase[3][3]=Image28;
laCase[3][4]=Image29; laCase[3][5]=Image30; laCase[3][6]=Image31; laCase[3][7]=Image32;
laCase[4][0]=Image33; laCase[4][1]=Image34; laCase[4][2]=Image35; laCase[4][3]=Image36;
laCase[4][4]=Image37; laCase[4][5]=Image38; laCase[4][6]=Image39; laCase[4][7]=Image40;
laCase[5][0]=Image41; laCase[5][1]=Image42; laCase[5][2]=Image43; laCase[5][3]=Image44;
laCase[5][4]=Image45; laCase[5][5]=Image46; laCase[5][6]=Image47; laCase[5][7]=Image48;
laCase[6][0]=Image49; laCase[6][1]=Image50; laCase[6][2]=Image51; laCase[6][3]=Image52;
laCase[6][4]=Image53; laCase[6][5]=Image54; laCase[6][6]=Image55; laCase[6][7]=Image56;
laCase[7][0]=Image57; laCase[7][1]=Image58; laCase[7][2]=Image59; laCase[7][3]=Image60;
laCase[7][4]=Image61; laCase[7][5]=Image62; laCase[7][6]=Image63; laCase[7][7]=Image64;
```

Deux événements (générés par les 64 TImages) gèrent le DRAG and DROP : ils nous permettront de déplacer une image sur une autre : OnDragOver et OnDragDrop.

Sélectionner les 64 TImages, dans les propriétés préciser DragMode=dmAutomatic.

Dans l'onglet événements : double cliquer dans OnDragOver, puis renommer l'événement en AccepterPiece.

`void __fastcall AccepterPiece(TObject *Sender, TObject *Source, int X, int Y, TDragState State, bool &Accept);`  
est alors l'événement (OnDragOver) généré lorsqu'un objet image arrive sur un autre objet image, il faut l'accepter.

Ajouter `Accept=1;` dans le corps de la méthode.

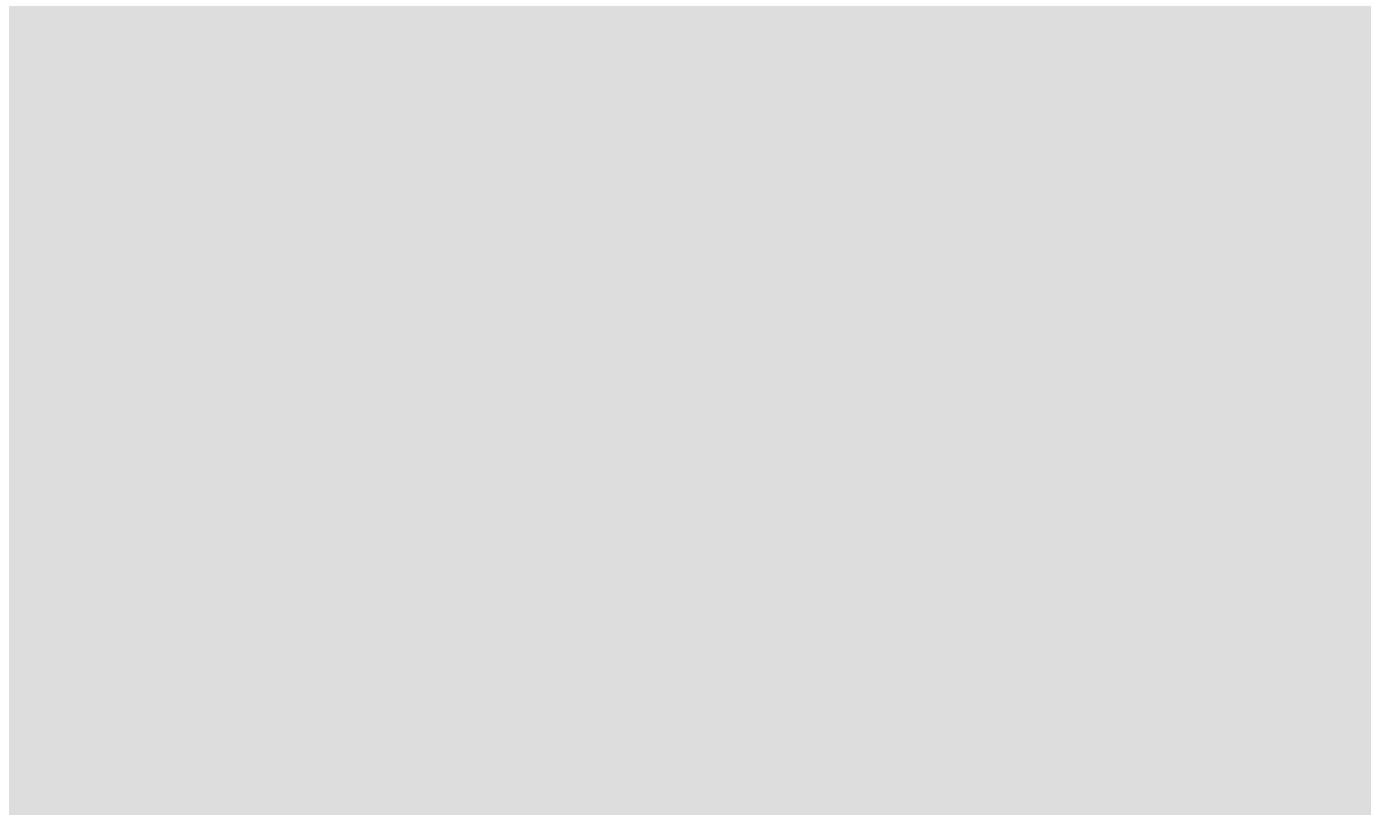
Dans l'onglet événements : double cliquer dans OnDragDrop, puis renommer l'événement en LacherPiece.

`void __fastcall LacherPiece(TObject *Sender, TObject *Source, int X, int Y);`

est alors l'événement (OnDragDrop) généré lorsqu'un objet image quitte son emplacement et arrive sur un autre objet image. En comparant les adresses source (Source) et destination (Sender) avec celles contenues dans le tableau 2D laCase on retrouve les coordonnées de départ et d'arrivée (idep jdep et iarr jarr) : i pour les lignes et j pour les colonnes. Source et Sender doivent être différents. L'algorithme est le suivant :

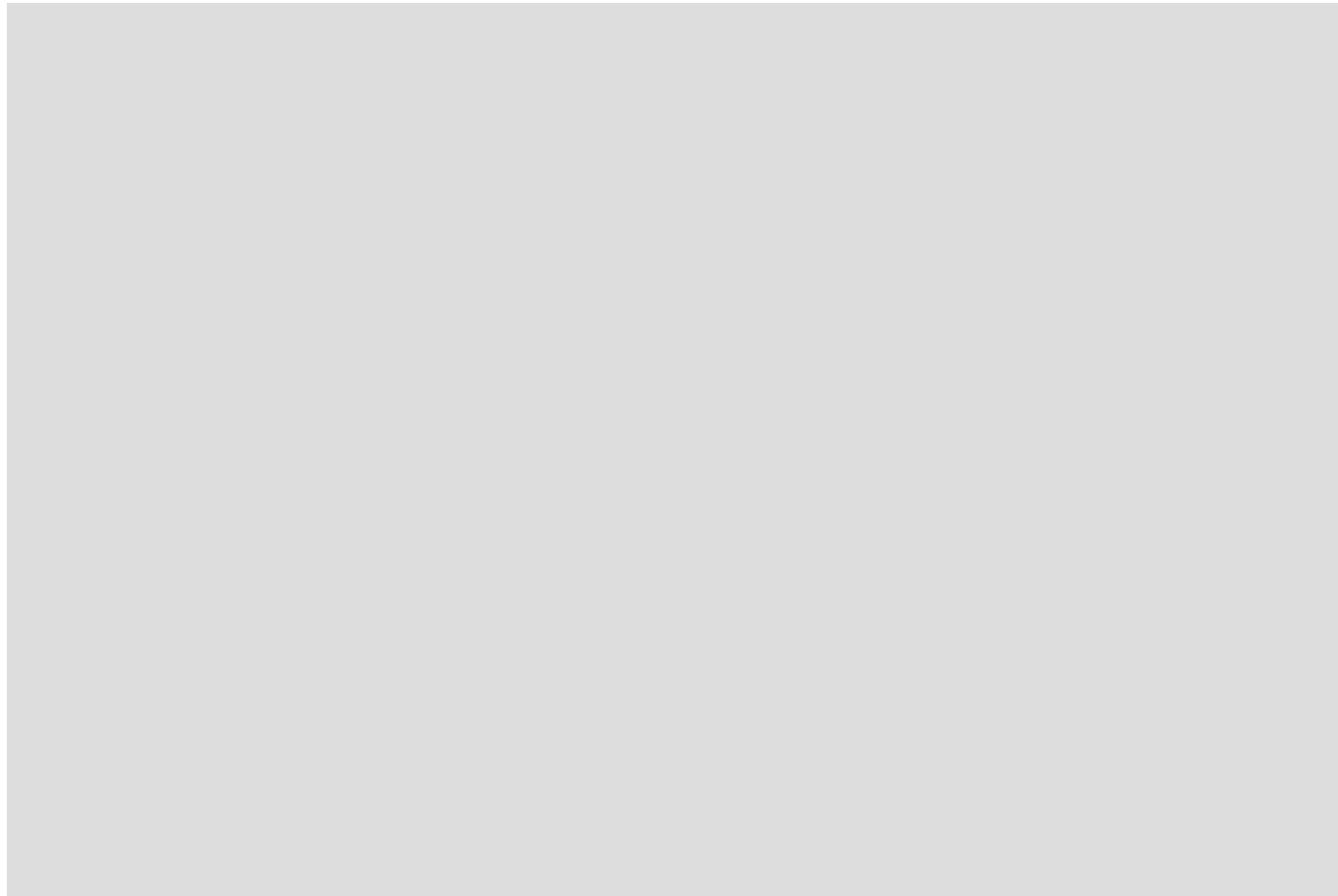
```
Si Source et Sender sont différents Alors
    Pour i allant de 0 à 7 par incrément de 1
        Pour j allant de 0 à 7 par incrément de 1
            Si Source = laCase[i][j] Alors
                idep ← i
                jdep ← j
            FinSi
            ... même principe avec Sender, iarr et jarr ...
        FinPour
    FinPour
FinSi
```

Donner le code de la méthode LacherPiece :



 **Ajouter un objet monEchiquier de la classe Echiquier à la classe TForm1. Tester la compilation du code.**

Chaque mouvement sera d'abord effectué dans l'objet monEchiquier, puis chargé dans l'IHM à l'aide de la méthode MiseAJourEchiquierIHM de la classe TForm1. La méthode MiseAJourEchiquierIHM : Pour chaque case de l'échiquier, en fonction de la pièce lue dans monEchiquier (LireCaseEchiquier(i,j)), la bonne pièce sera chargée dans l'image laCase[i][j] (laCase[i][j] → Picture → LoadFromFile(...)), parmi les images cavB.bmp, cavN.bmp....

 **Donner le code de la méthode MiseAJourEchiquierIHM :** **Dans LacherPiece, appeler et tester la méthode Deplacer avec l'objet monEchiquier, en cas de promotion un groupe de RadioButton permettra de choisir la pièce de promotion et de passer la caractère correspondant à la méthode Deplacer. Si le déplacement est correct appeler la méthode MiseAJourEchiquierIHM. Tester le programme : l'image se déplace...** **Ajouter un indicateur (Shape ou Label) dans l'IHM permettant de savoir qui a le trait.** **Ajouter et coder un bouton "Nouvelle partie". Ajouter le bouton « Aveugle », les pièces ne sont plus visibles, puis un nouvel appui permet de revoir les pièces.**

### Gestion du temps :

- Keyboard Coder les méthodes et attributs nécessaires dans la classe Joueur. Un objet TTimer ajouté à la Form1 sera chargé de décrémenter le temps des joueurs lorsqu'ils auront le trait, des labels permettront d'afficher le temps restant de chaque joueur.

### Jeu en réseau :

- Keyboard Créer un Thread : Fichier > Nouveau > Autre > Fichiers C++ Builder > Objet thread. Ce Thread (voir le diagramme de classes) contiendra une partie du programme principal du jeu en mode console : ce thread sera chargé d'attendre le coup de l'adversaire. Dans la fenêtre principale, ajouter la possibilité de saisir l'IP du serveur ainsi que le choix entre : partie locale, serveur ou client.

### Création de l'image de la position :

- Keyboard Dans le diagramme de classe, threadPosition permet de générer l'image de la position. Créer et coder ce thread en y plaçant la création de l'image BMP. Le code suivant permet de générer un JPG sans utiliser le programme PositionBMP2JPG.exe. Il nécessite d'inclure jpeg.hpp. Ici Image71 contiendra l'image de la position.

```
Form1->LEchiquier()->SauvegarderEchiquierBMP("Web\\Position.bmp");
TJPEGImage *imagejpg = new TJPEGImage;
Form1->Image71->Picture->LoadFromFile("Web\\Position.bmp");
imagejpg->Assign(Form1->Image71->Picture->Bitmap);
imagejpg->SaveToFile("Web\\Position.jpg");
```

### Serveur Web :

- Keyboard Créer un Thread (voir le diagramme de classes) chargé de gérer la communication entre le navigateur et le serveur Web. Il contiendra la main du serveur Web en mode console.

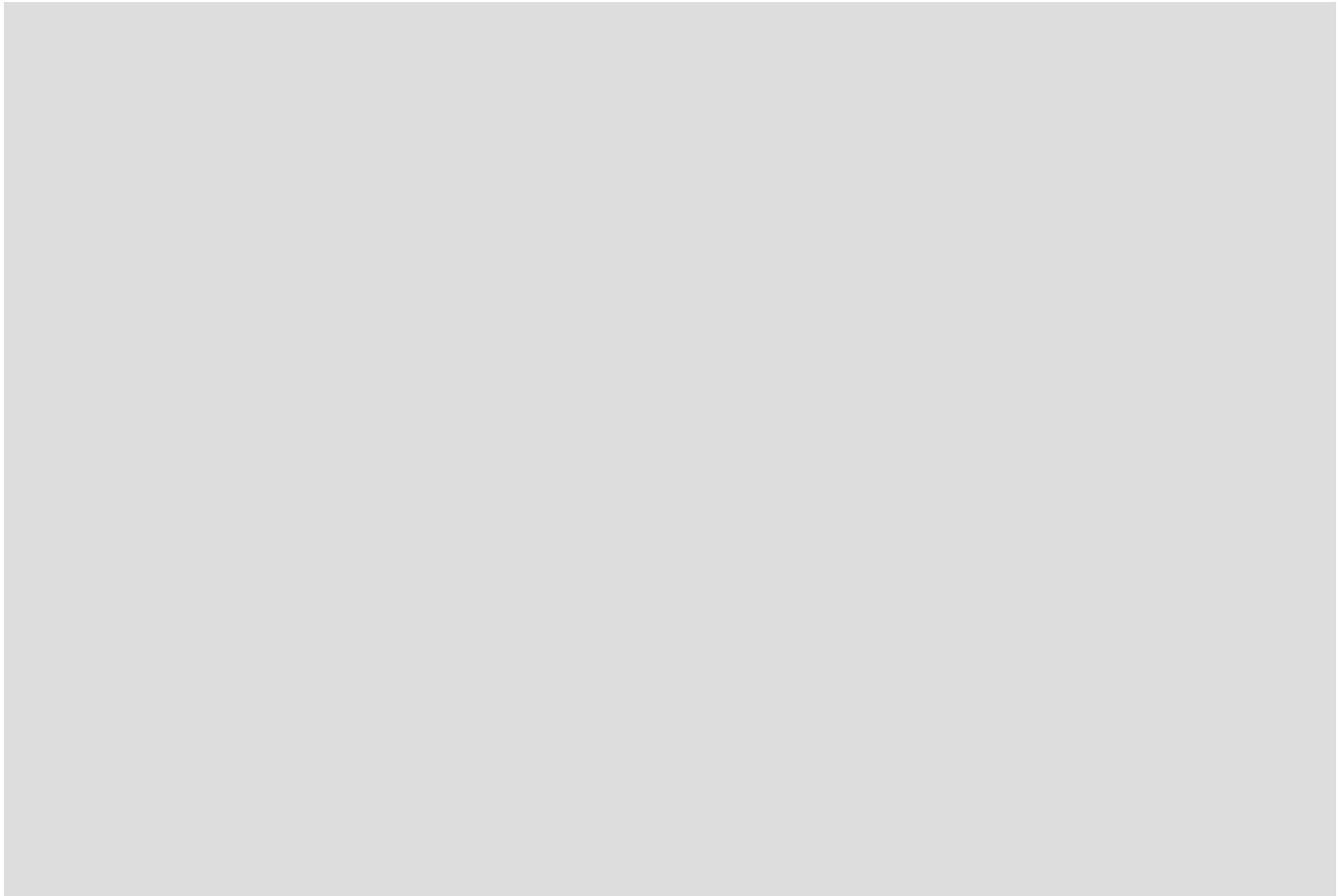
### Mémorisation de la partie : la classe Partie

- Keyboard Ajouter la possibilité de reprendre son coup en créant une classe Partie (voir le diagramme de classes). Le principe est de stocker à chaque demi-coup l'objet monEchiquier dans un tableau d'Echiquiers : suiteEchiquier. Tester sur PC vos méthodes en ajoutant un objet maPartie de la classe Partie à la classe TForm1. Ajouter un bouton "retour" permettant de revenir au coup précédent.

## Rotation de l'échiquier



Donner le diagramme d'activité permettant de tourner de  $180^\circ$  le tableau à 2 dimensions laCase (il faudra ajouter une image temporaire : `TImage *temp=new TImage(this);`).



Coder cette rotation et ajouter un bouton permettant de tourner l'échiquier.

## Pour aller plus loin...

Affichage des coups dans l'interface...

Gestion des parties nulles...

Gestion des cadences Fisher...

Entrer une position libre...

Envoyer des commentaires à l'adversaire (Chat)...

Affichage des coups sur un journal lumineux...

Affichage de la position sur écran géant...

## Annexe 1 : les règles du jeu

### I – L’ÉCHIQUIER



#### ■ But du jeu :

Faire **échec et mat**, (<— définition et tableaux de mats) évidemment, l’adversaire peut abandonner et vous avez la possibilité de gagner au temps.

On dit que le **Roi est en échec**, lorsque la case qu'il occupe est contrôlée par une pièce adverse. (en d'autres termes, lorsqu'une pièce peut le manger). → le Roi doit donc OBLIGATOIREMENT parer cet échec.

Si le Roi ne peut parer l'échec, il perd la partie, puisqu'il est **échec et mat**.

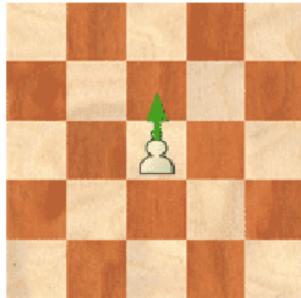
→ Caractéristiques de l'échec et mat :

- Le Roi ne peut plus se déplacer
- Aucune pièce alliée ne peut s'interposer pour parer l'échec
- La pièce qui fait l'échec ne peut être éliminée

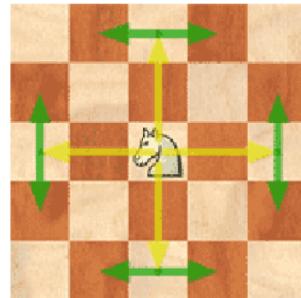
#### ■ Voici la liste des cas qui déterminent le match nul :

- Il y a **Pat** (le Roi n'est pas en échecs, mais aucune pièce ne peut être déplacée)
- Il n'y a **pas assez de matériel** pour mater les Rois
- Un joueur fait des **échecs perpétuels** (échec à l'infini)
- La **même position est répétée trois fois**
- **50 coups sont joués sans aucune prise, ni aucun déplacement de pion.**

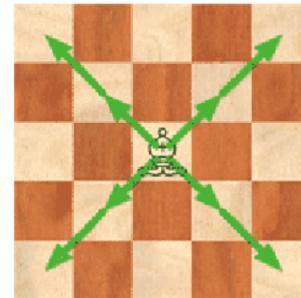
## II- LA MARCHE DES PIÈCES



— Le pion (1)



— Le Cavalier (3)



— Le Fou (3)

- Avance droit devant lui
- Ne recule jamais
- Avance d'1 case à la fois SAUF lors de son 1er coup où il peut avancer de 2 cases
- **Capture en diagonale**

- Déplacement en "L"
- Change de couleur de case à chaque déplacement
- Saute par dessus les autres

- Déplacement en diagonale
- d'autant de cases qu'il veut



— La Tour (5)



— Le Roi



— La Dame (9)

- Déplacement horizontal/vertical
- d'autant de cases qu'elle veut
- Peut avancer/reculer

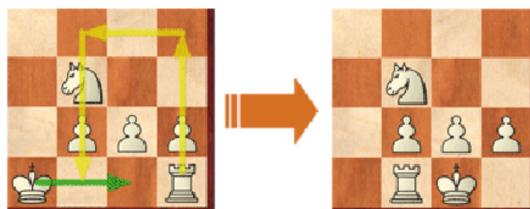
- Déplacement toutes directions
- d'1 seule case à la fois
- Il doit y avoir au moins une case entre 2 Rois adverses

- Pièce la plus puissante
- Cumule déplacement de la Tour (horizontal/vertical) et du Fou (diagonal)
- Peut contrôler 27 cases !

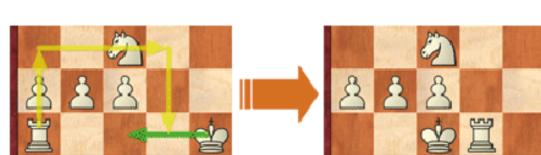
### III – RÈGLES PARTICULIÈRES

#### LE ROQUE

- Le **Roi** se décale de deux cases, en direction d'une de ses Tours. – La **Tour** vient sauter par dessus le Roi pour se placer juste à côté (sur une case adjacente)



Petit Roque O-O



Grand Roque O-O-O

#### Les conditions du Roque :

- Ni le **Roi**, ni la **Tour** concernés, ne doivent avoir bougé pendant le jeu et aucune pièce ne doit les séparer.
- Le **Roi** ne peut être en échec.
- Aucune pièce ennemie ne doit contrôler les deux cases que le Roi parcourt pour roquer.

#### LE PION

##### ■ La Promotion :

Quand le pion atteint la dernière rangée de l'échiquier, il se **transforme** en une autre pièce de sa couleur (en général une Dame). Le pion ne peut se transformer en Roi.

##### ■ La Prise en passant :



Un pion peut capturer un pion adverse (de colonne adjacente), si celui-ci saute deux cases, **comme s'il n'avait avancé que d'une case**. On dit que ce pion prend le pion ennemi "en passant".

Pour vous aider à visualiser la scène, imaginez deux chevaliers au galop, face à face, pratiquant la **joute équestre**. Les deux chevaliers se croisent, mais l'un des deux tombe (celui qui a sauté deux cases). Une prise en passant s'effectue **immédiatement** (le tour suivant, il est trop tard).

## Annexe 2: diagrammes de classes

