27/04/2015

Marc-Antoine Fernandes

1G3 Informatique | IUT Lyon 1 – Site de Bourg-En-Bresse

ChessQuito

Compte-rendu

Table des matières

[Etape 0 – Diagramme de Classe 3](#_Toc417932896)

[Etape 1 – Partie & Joueur 4](#_Toc417932897)

[Etape 2 – L’Héritage 4](#_Toc417932898)

[Etape 3 – Affichage 4](#_Toc417932899)

[Etape 4 – Interface + déplacement 4](#_Toc417932900)

[Etape 5 – Surcharge Operateurs 6](#_Toc417932901)

[Etape 6 – Sauvegarde 6](#_Toc417932902)

[Etape 7 – SDL 7](#_Toc417932903)

[Code : 8](#_Toc417932904)

[Bouton.h 8](#_Toc417932905)

[Bouton.cpp 8](#_Toc417932906)

[Cavalier.h 10](#_Toc417932907)

[Cavalier.cpp 10](#_Toc417932908)

[EditBox.h 11](#_Toc417932909)

[EditBox.cpp 12](#_Toc417932910)

[Fou.h 18](#_Toc417932911)

[Fou.cpp 19](#_Toc417932912)

[Joueur.h 20](#_Toc417932913)

[Joueur.cpp 21](#_Toc417932914)

[Partie.h 23](#_Toc417932915)

[Partie.cpp 25](#_Toc417932916)

[Piece.h 25](#_Toc417932917)

[Piece.cpp 26](#_Toc417932918)

[Pion.h 27](#_Toc417932919)

[Pion.cpp 27](#_Toc417932920)

[Reine.h 28](#_Toc417932921)

[Reine.cpp 29](#_Toc417932922)

[Roi.h 30](#_Toc417932923)

[Roi.cpp 30](#_Toc417932924)

[SDL\_INIT.h 31](#_Toc417932925)

[Tour.h 31](#_Toc417932926)

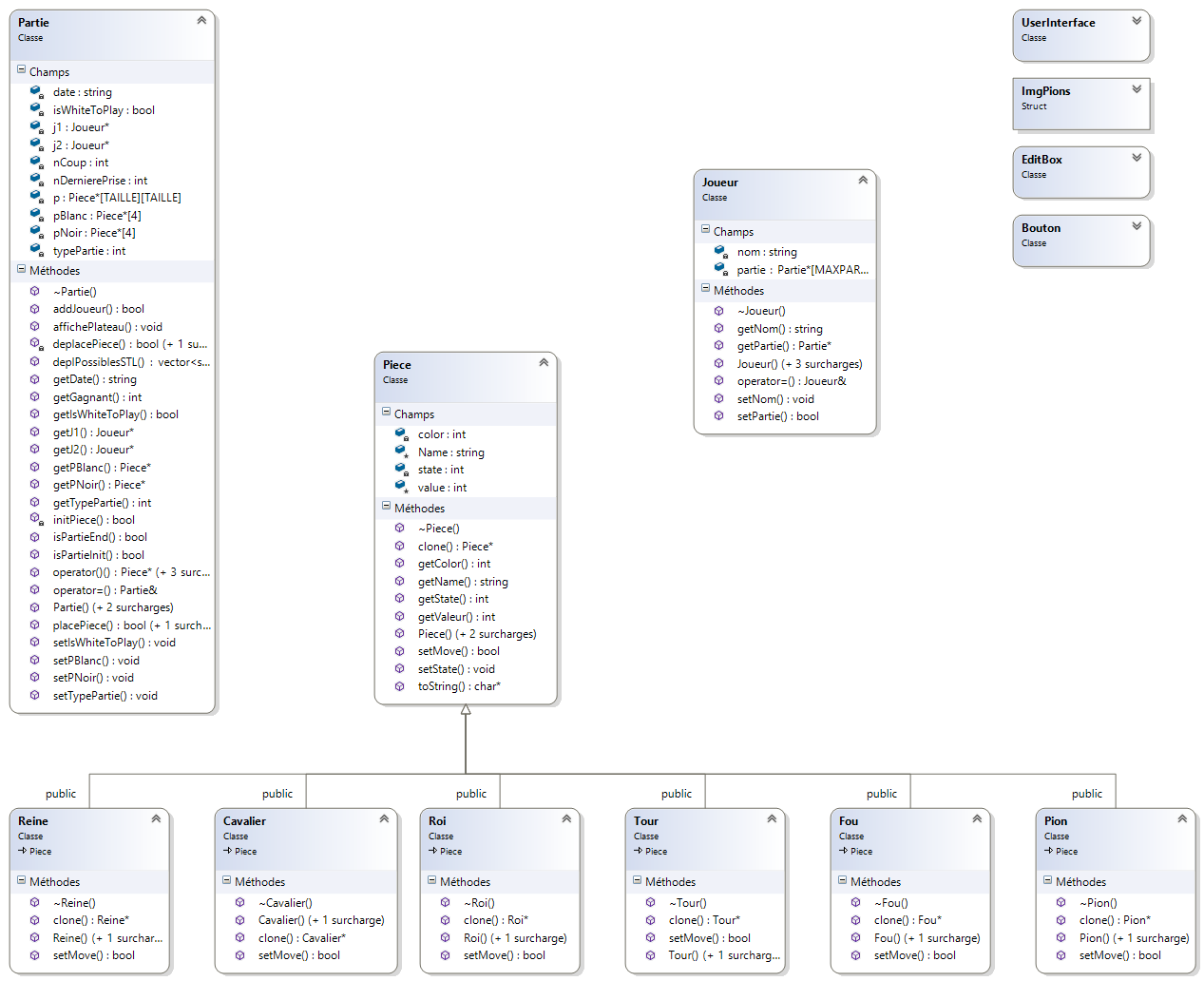
[Tour.cpp 32](#_Toc417932927)

[UserInterface.h 33](#_Toc417932928)

[UserInterface.cpp 35](#_Toc417932929)

[Main.cpp 57](#_Toc417932930)

## Etape 0 – Diagramme de Classe



La modélisation du jeu n’a pas été très compliquée, j’ai pu vérifier la validité du modèle en testant les possibilités et également durant la suite du projet. Les différents objets / structure de droite sont utilisés pour l’interface SDL.

## Etape 1 – Partie & Joueur

Dans cette étape, j’ai commencé à écrire les premières classes Joueur et Partie, pas de difficulté particulière, le plus dur dans cette étape fut la destruction d’une partie qui devait mettre à jour les parties de chaque joueur sans créer de boucle infini en resupprimant la même partie. De plus, il fallait en même temps s’habituer à la POO.

## Etape 2 – L’Héritage

L’étape 2 était relativement longue. Tout d’abord, l’héritage de deux pièces Tour et Fou, pas de difficulté. Ensuite pour l’initialisation de l’échiquier : pareil, rien de compliqué, j’ai d’abord utilisé un typedef Plateau mais je l’ai enlevé par la suite car il n’est utilisé qu’une seule fois dans tout le programme. Ensuite j’ai créé la fonction verifierDeplacement(), j’ai dû passer la fonction en virtual et donc passer la classe Piece en abstraite afin de pouvoir vérifier le déplacement en fonction de chaque type de pièce. Pour cette vérification, j’ai juste vérifié l’égalité des coordonnées x ou y en fonction des mouvements possibles. Ensuite, j’ai écrit la fonction placePiece() dans Partie, sans difficulté. Ne sachant pas comment écrire une liste d’initialisation, j’ai cherché comment faire. Puis j’ai essayé de faire la surcharge de () et y suis arrivé assez rapidement en reprenant un tutoriel sur un stackoverflow.

## Etape 3 – Affichage

Vu que j’avais déjà transformé ma classe Piece en classe abstraite, l’étape 3 s’est résumé à faire l’affichage d’une partie. J’ai voulu modifier toutes mes pièces afin de leur rajouter la fonction toString(), mais j’ai préféré utiliser la première lettre de leur nom ( lettres uniques pour chaque règle). Puis j’ai surchargé l’opérateur << pour pouvoir faire cout << mPartie ; pas de problème sur cette partie.

## Etape 4 – Interface + déplacement

J’ai divisé cette étape en 3 parties.

**Première partie :**

J’ai commencé par finir de créer mes pièces restantes, pas de difficulté, sauf pour le Pion où je ne savais pas trop quand renvoyer TRUE dans la fonction verifierDeplacement() (setMove() dans mon programme), et j’ai finalement choisi de renvoyer TRUE du moment que le déplacement est bon pièce à coté ou pas. J’en ai profité pour leur attribuer une valeur et ai rajouté une fonction pour avoir le gagnant d’une partie.

**Deuxième partie :**

Dans la deuxième partie, j’ai fait la fonction qui vérifie un déplacement. Cette fonction fût pour moi la plus dur du projet, j’ai pas mal réfléchi à comment faire, et un camarade m’a fait penser à utiliser des vecteurs. J’ai trouvé l’idée bonne donc je me suis penché sur cette solution-là. Je suis resté bloqué plusieurs heures en essayant plusieurs systèmes. Et finalement j’ai trouvé un algorithme fonctionnel : Le but de l’algorithme est de parcourir chaque case du plateau et de vérifié si cette case est comprise entre la position de départ de la pièce et sa position finale. Si la case testée est vide, on continue sinon on sort de la fonction en renvoyant FALSE. J’ai dû mettre des exceptions pour les trois pièces qui ont un déplacement « fixe » (ROI | CAVALIER | PION) qui ne m’ont pas posé de problème. Voici la partie de l’algorithme où on teste d’abord si un point appartient à la droite (pos1, pos2) (On parcourt le plateau uniquement entre les deux points de la droite).

for (int x = xMin; x <= xMax; x++) {

for (int y = yMin; y <= yMax; y++) {

if (p[x][y] != NULL && (x != x1 || y != y1) && (x != x2 || y != y2)) {

if (((x2 - x1) \* (y - y1)) - ((x - x1) \* (y2 - y1)) == 0) {

return false;

}

}

}

}

J’ai pu réduire la taille de la boucle en limitant les tests entre les des points, j’ai eu des problèmes au début car si le déplacement était horizontale ou verticale, on ne parcourait pas la boucle. Dans le premier « if » on teste si la case est NULL ou pas (si NULL on passe) et si la case testé n’est pas celle de départ ou d’arrivé. Dans le deuxième « if », on teste si le point (x, y) appartient à la droite (x1, y1) (y1, y2) grâce à un produit scalaire. Enfin après la fin de cet algorithme on bouge la pièce et met l’état de celle d’arrivée (si elle existe) à 2 (prise) (On ne la supprime pas car elle reste dans les deux tableaux d’initialisations).

**Troisième partie :**

La partie sur l’interface Homme/Machine n’a pas été compliqué, mais longue, elle n’est plus dans le code car j’ai implémenté une interface SDL, mais elle se divisait en 3 parties : Un gestionnaire de joueur, un gestionnaire de partie, et une interface pour ajouter un joueur à une partie. J’ai eu deux trois problèmes sur les fonctions pour ajouter et supprimer des parties/joueurs d’où mes listes (listeJoueur et listePartie) qui sont de type Joueur\*\*\* et Partie\*\*\*.

## Etape 5 – Surcharge Operateurs

Pour cette étape, j’ai dû modifier quelques fonctions dans ma classe Joueur afin de pouvoir avoir 5 parties. J’ai également écrit les fonctions de copie et d’assignation pour Partie et Joueur. Pas beaucoup de difficulté la dessus, juste ne pas oublier de supprimer l’ancien plateau avant de cloner le nouveau (pour sa j’ai dû rajouter une fonction virtuel clone() dans mes pièces afin de créer une pièce du bon type). Lorsque je clone une partie, les joueurs ne sont pas copiés (je n’ai pas voulu perdre de temps la dessus sachant que copier une partie pour jouer avec les mêmes joueurs n’a pas beaucoup d’intérêt.). Pour finir l’étape, j’ai utilisé la fonction friend afin de pouvoir surcharger le cout avec un Joueur.

## Etape 6 – Sauvegarde

Dans cette étape, je n’ai fait que les 2 premières questions car j’ai préféré implémenter la SDL en priorité.

Pour la première question, j’ai essayé au brouillon de structurer mes fichiers de sauvegarde. J’en ai conclu par créer un fichier Joueurs.txt qui contient la liste des joueurs, un fichier Parties.txt qui liste les parties en cours (enregistrées par leur date), et enfin un dossier parties/ qui contient un fichier par partie qui ont lieu (supprimer les anciennes parties du dossier était trop complexe par rapport à son réel utilité). Chaque fichier de partie liste chaque attribut nécessaire, les pions sont enregistré sous la forme NOMDELAPIECE\_COULEUR\_ETAT (si la case est vide : NULL\_0\_0). Les joueurs sont enregistrés par leur pseudo (unique).

Pour la lecture dans ces fichiers, c’était légèrement plus complexe, il ne fallait pas se louper sur la position du curseur dans le fichier. Les joueurs sont chargés avant les parties afin de pouvoir relier les joueurs à une partie lors de son chargement.

## Etape 7 – SDL

J’ai ensuite voulu implémenter une interface graphique au jeu, je n’ai pas voulu implémenter la troisième règle avant car après avoir réfléchi à comment faire, j’ai trouvé ça complexe et sans vraiment de résultat satisfaisant.

Pour la SDL, j’ai pas mal réfléchi à si je devais ou non faire une classe spécifique, et qu’est-ce que je devais mettre dedans si oui. J’ai comencé à faire l’interface dans le main puis j’ai vite vu que cela serait plus facile dans une classe spécifique (UserInterface). Pour l’interface j’ai aussi eu besoin de « boutons » donc j’ai créé une autre classe Bouton qui permet de savoir si des coordonnées sont sur ce bouton (je l’utilise également pour les listes). J’ai également décidé de créer une classe EditBox pour pouvoir écrire dans une interface SDL. Pour ajouter un joueur à une partie, un bouton apparait lorsqu’on joue à une partie ou il n’y a pas assez de joueur, il faut écrire dans la console le nom du joueur (manque de temps pour ajouter la liste des joueurs). L’interface de jeu se joue avec les flèches directionnelles et Entrer, le plateau de jeu se découpe en trois tableaux : Le plateau de jeu, et les deux tableaux « d’initialisation » qui permettent d’afficher l’état des pièces de chaque joueur (Normal quand non placé, Semi-transparente quand placé et Barré quand prise). L’interface globale ce découpe en deux parties, le Plateau et la barre de navigation.

## Code :

### Bouton.h

#ifndef \_BOUTON\_H

#define \_BOUTON\_H

/\*

Bouton:

Permet de creer un bouton avec un texte,

une couleur de fond et une couleur de police.

A une fonction de colision (isClicked),

Positionné aux coordonnées passées.

\*/

#include "SDL\_INIT.h"

#include <string>

#include <iostream>

using namespace std;

class Bouton

{

private:

SDL\_Surface\* ecran;

TTF\_Font\* police; // Police d'écriture pour tout l'affichage

SDL\_Surface\* btn; // Image du bouton ( créé au moment de la création de l'instance)

SDL\_Rect btnRect; // Contient les positions et la taille du bouton

public:

Bouton(SDL\_Surface\* ecran,TTF\_Font\* police, string txtBtn, int x, int y, int width, int height, Uint32 bgColor, SDL\_Color ftColor);

void afficherBtn(); // Affiche le bouton sur l'écran

bool isClicked(int x, int y); // Gestion des colisions

~Bouton();

};

#endif

### Bouton.cpp

#include "Bouton.h"

Bouton::Bouton(SDL\_Surface \* ecran, TTF\_Font \* police, string txtBtn, int x, int y, int width, int height, Uint32 bgColor, SDL\_Color ftColor)

{

this->ecran = ecran;

this->police = police;

btnRect.h = height;

btnRect.w = width;

btnRect.x = x;

btnRect.y = y;

btn = SDL\_CreateRGBSurface(SDL\_HWSURFACE, btnRect.w, btnRect.h, 32, 0, 0, 0, 0);

SDL\_FillRect(btn, NULL, bgColor); // On rempli le fond du bouton

SDL\_Surface\* tmp;

tmp = TTF\_RenderText\_Blended(police, txtBtn.c\_str() ,ftColor);

if (tmp->w > width - 20)

resizeImage(tmp, width - 10, width - 10, true);

if (tmp->h > height - 8)

resizeImage(tmp, height - 4, height - 4, false);

SDL\_Rect tmpRect;

tmpRect.x = (width - tmp->w) / 2;

tmpRect.y = (height - tmp->h) / 2;

SDL\_BlitSurface(tmp, NULL, btn, &tmpRect);

SDL\_FreeSurface(tmp);

}

void Bouton::afficherBtn()

{

SDL\_BlitSurface(btn, NULL, ecran, &btnRect);

SDL\_Flip(ecran);

}

bool Bouton::isClicked(int x, int y)

{

if (x > btnRect.x && x < btnRect.x + btnRect.w && y > btnRect.y && y < btnRect.y + btnRect.h)

return true;

return false;

}

Bouton::~Bouton()

{

SDL\_FreeSurface(btn);

}

### Cavalier.h

#ifndef \_CAVALIER\_H

#define \_CAVALIER\_H

/\*

Cavalier:

Hérite de Pièce,

Mouvement en L

\*/

#include "Piece.h"

class Cavalier :

public Piece

{

public:

Cavalier(int);

Cavalier(int, int);

Cavalier\* clone();

bool setMove(char[], char[]);

~Cavalier(void);

};

#endif

### Cavalier.cpp

#include "Cavalier.h"

Cavalier::Cavalier(int color) : Piece(color)

{

Name = "Cavalier";

value = 3;

}

Cavalier::Cavalier(int color, int state) : Piece(color, state)

{

Name = "Cavalier";

value = 3;

}

Cavalier\* Cavalier::clone(){

Cavalier\* tmp = new Cavalier(this->getColor());

tmp->setState(this->getState());

return tmp;

}

bool Cavalier::setMove(char pos1[2], char pos2[2]) {

int x1 = pos1[0] - 'a';

int y1 = pos1[1] - '0';

int x2 = pos2[0] - 'a';

int y2 = pos2[1] - '0';

if (fabs((float)x2 - x1) == 2 && fabs((float)y2 -y1) == 1) { // En forme de L v1

return true;

}

if (fabs((float)x2 - x1) == 1 && fabs((float)y2 - y1) == 2) { // En forme de L v2

return true;

}

return false;

}

Cavalier::~Cavalier(void)

{

}

### EditBox.h

#ifndef \_EDITBOX\_H

#define \_EDITBOX\_H

/\*

EditBox :

Permet d'avoir une zone d'édition en SDL, ne prend en charge que quelques caractères spéciaux

Pour une majuscule, il faut déjà presser LSHIFT ou RSHIFT puis écrire la lettre demandé

\*/

#include "SDL\_INIT.h"

#include <string>

#include <iostream>

using namespace std;

class EditBox

{

private:

SDL\_Surface\* ecran;

TTF\_Font\* police; // Police d'écriture pour tout l'affichage

SDL\_Rect boxRect; // Position de la boite d'édition

SDL\_Color fgColor; // Couleur de l'écriture

Uint32 bgColor; // Couleur de fond

string txt; // Contient le texte tapé

bool isSelect; // Permet de régler la couleur de l'encadré ( Vert si on peut écrire, Rouge sinon)

public:

EditBox(SDL\_Surface\* ecran, TTF\_Font\* police, int x, int y, int width, int height, Uint32 bgColor, SDL\_Color fgColor);

int start(); // Gère l'écriture, renvoie -2 si QUIT

void drawBox(); // Dessine la boite en fonction du texte ( rognage auto)

string getText() { return txt; }

void setText(string newTxt) { txt = newTxt; }

bool isClicked(int x, int y); // Colision

~EditBox();

};

#endif

### EditBox.cpp

#include "EditBox.h"

EditBox::EditBox(SDL\_Surface \* ecran, TTF\_Font \* police, int x, int y, int width, int height, Uint32 bgColor, SDL\_Color fgColor)

{

/\* On enregsitre les données nécessaires \*/

this->ecran = ecran;

this->police = police;

this->bgColor = bgColor;

this->fgColor = fgColor;

boxRect.h = height;

boxRect.w = width;

boxRect.x = x;

boxRect.y = y;

isSelect = false; // False

}

int EditBox::start()

{

isSelect = true;

drawBox();

bool isShiftPressed = false;

SDL\_Event event;

SDL\_EnableUNICODE(true);

while (true) {

SDL\_WaitEvent(&event);

switch (event.type) {

case SDL\_QUIT:

isSelect = false;

return -2;

break;

case SDL\_KEYDOWN:

if (event.key.keysym.sym == SDLK\_ESCAPE) {

isSelect = false;

drawBox();

return -2;

}

if (event.key.keysym.sym == SDLK\_RETURN) {

isSelect = false;

drawBox();

return 0;

}

if (event.key.keysym.sym == SDLK\_BACKSPACE && txt.length() > 0)

txt.pop\_back();

if (event.key.keysym.sym == SDLK\_LSHIFT || event.key.keysym.sym == SDLK\_RSHIFT)

isShiftPressed = true;

else {

switch (event.key.keysym.sym) {

case SDLK\_MINUS:

txt += '-';

break;

case SDLK\_KP0:

txt += '0';

break;

case SDLK\_KP1:

txt += '1';

break;

case SDLK\_KP2:

txt += '2';

break;

case SDLK\_KP3:

txt += '3';

break;

case SDLK\_KP4:

txt += '4';

break;

case SDLK\_KP5:

txt += '5';

break;

case SDLK\_KP6:

txt += '6';

break;

case SDLK\_KP7:

txt += '7';

break;

case SDLK\_KP8:

txt += '8';

break;

case SDLK\_KP9:

txt += '9';

break;

}

if (isShiftPressed) {

switch (event.key.keysym.unicode) {

case SDLK\_a:

txt += 'A';

break;

case SDLK\_b:

txt += 'B';

break;

case SDLK\_c:

txt += 'C';

break;

case SDLK\_d:

txt += 'D';

break;

case SDLK\_e:

txt += 'E';

break;

case SDLK\_f:

txt += 'F';

break;

case SDLK\_g:

txt += 'G';

break;

case SDLK\_h:

txt += 'H';

break;

case SDLK\_i:

txt += 'I';

break;

case SDLK\_j:

txt += 'J';

break;

case SDLK\_k:

txt += 'K';

break;

case SDLK\_l:

txt += 'L';

break;

case SDLK\_m:

txt += 'M';

break;

case SDLK\_n:

txt += 'N';

break;

case SDLK\_o:

txt += 'O';

break;

case SDLK\_p:

txt += 'P';

break;

case SDLK\_q:

txt += 'Q';

break;

case SDLK\_r:

txt += 'R';

break;

case SDLK\_s:

txt += 'S';

break;

case SDLK\_t:

txt += 'T';

break;

case SDLK\_u:

txt += 'U';

break;

case SDLK\_v:

txt += 'V';

break;

case SDLK\_w:

txt += 'W';

break;

case SDLK\_x:

txt += 'X';

break;

case SDLK\_y:

txt += 'Y';

break;

case SDLK\_z:

txt += 'Z';

break;

case SDLK\_UNDERSCORE:

txt += '\_';

break;

}

isShiftPressed = false;

}

else {

switch (event.key.keysym.unicode) {

case SDLK\_a:

txt += 'a';

break;

case SDLK\_b:

txt += 'b';

break;

case SDLK\_c:

txt += 'c';

break;

case SDLK\_d:

txt += 'd';

break;

case SDLK\_e:

txt += 'e';

break;

case SDLK\_f:

txt += 'f';

break;

case SDLK\_g:

txt += 'g';

break;

case SDLK\_h:

txt += 'h';

break;

case SDLK\_i:

txt += 'i';

break;

case SDLK\_j:

txt += 'j';

break;

case SDLK\_k:

txt += 'k';

break;

case SDLK\_l:

txt += 'l';

break;

case SDLK\_m:

txt += 'm';

break;

case SDLK\_n:

txt += 'n';

break;

case SDLK\_o:

txt += 'o';

break;

case SDLK\_p:

txt += 'p';

break;

case SDLK\_q:

txt += 'q';

break;

case SDLK\_r:

txt += 'r';

break;

case SDLK\_s:

txt += 's';

break;

case SDLK\_t:

txt += 't';

break;

case SDLK\_u:

txt += 'u';

break;

case SDLK\_v:

txt += 'v';

break;

case SDLK\_w:

txt += 'w';

break;

case SDLK\_x:

txt += 'x';

break;

case SDLK\_y:

txt += 'y';

break;

case SDLK\_z:

txt += 'z';

break;

case SDLK\_UNDERSCORE:

txt += '\_';

break;

case SDLK\_MINUS:

txt += '-';

break;

}

}

}

drawBox();

break;

case SDL\_MOUSEBUTTONDOWN:

int x = event.button.x;

int y = event.button.y;

if (!isClicked(x,y)) {

isSelect = false;

drawBox();

return -1;

}

break;

}

}

isSelect = false;

drawBox();

return -1;

}

void EditBox::drawBox()

{

SDL\_Surface\* box;

box = SDL\_CreateRGBSurface(SDL\_HWSURFACE, boxRect.w, boxRect.h, 32, 0, 0, 0, 0);

/\* On met le contour ( selection ou pas) \*/

SDL\_Rect tmpRect\_select;

tmpRect\_select.h = boxRect.h + 4;

tmpRect\_select.w = boxRect.w + 4;

tmpRect\_select.x = boxRect.x - 2;

tmpRect\_select.y = boxRect.y - 2;

Uint32 tmpColor;

if (isSelect && txt.length() > 0)

tmpColor = SDL\_MapRGB(ecran->format, 0, 255, 0);

else

tmpColor = SDL\_MapRGB(ecran->format, 255, 0, 0);

SDL\_FillRect(ecran, &tmpRect\_select, tmpColor); // On met le contour de la box

SDL\_FillRect(box, NULL, bgColor); // On rempli le fond du bouton

if(txt.length() > 0){

SDL\_Surface\* tmp;

tmp = TTF\_RenderText\_Blended(police, txt.c\_str(), fgColor);

if (tmp->h > boxRect.h - 8)

resizeImage(tmp, boxRect.h - 4, boxRect.h - 4, false);

if (tmp->w > boxRect.w - 20) { // Si le texte est trop large

SDL\_Rect tmpRect;

tmpRect.x = 10;

tmpRect.y = (boxRect.h - tmp->h) / 2;

SDL\_Rect tmp2Rect;

tmp2Rect.w = boxRect.w - 20;

tmp2Rect.x = tmp->w - tmp2Rect.w;

tmp2Rect.y = 0;

tmp2Rect.h = tmp->h;

SDL\_BlitSurface(tmp, &tmp2Rect, box, &tmpRect);

}

else {

SDL\_Rect tmpRect;

tmpRect.x = 10;

tmpRect.y = (boxRect.h - tmp->h) / 2;

SDL\_BlitSurface(tmp, NULL, box, &tmpRect);

}

SDL\_FreeSurface(tmp);

}

SDL\_BlitSurface(box, NULL, ecran, &boxRect);

SDL\_Flip(ecran);

SDL\_FreeSurface(box);

}

bool EditBox::isClicked(int x, int y)

{

if (x > boxRect.x && x < boxRect.x + boxRect.w && y > boxRect.y && y < boxRect.y + boxRect.h)

return true;

return false;

}

EditBox::~EditBox()

{

}

### Fou.h

#ifndef \_FOU\_H

#define \_FOU\_H

/\*

Fou:

Hérite de Pièce,

Mouvement diagonaux

\*/

#include "Piece.h"

class Fou :

public Piece

{

public:

Fou(int);

Fou(int, int);

Fou\* clone();

bool setMove(char[], char[]);

~Fou(void);

};

#endif

### Fou.cpp

#include "Fou.h"

Fou::Fou(int color) : Piece(color)

{

Name = "Fou";

value = 2;

}

Fou::Fou(int color, int state) : Piece(color, state)

{

Name = "Fou";

value = 2;

}

Fou\* Fou::clone(){

Fou\* tmp = new Fou(this->getColor());

tmp->setState(this->getState());

return tmp;

}

bool Fou::setMove(char pos1[2], char pos2[2]){

int x1 = pos1[0] - 'a';

int y1 = pos1[1] - '0';

int x2 = pos2[0] - 'a';

int y2 = pos2[1] - '0';

if ( fabs((float)x2 - x1) == fabs((float)y2 - y1)){ // Diagonal

return true;

}

return false;

}

Fou::~Fou(void)

{

}

### Joueur.h

#ifndef \_JOUEUR\_H

#define \_JOUEUR\_H

/\*

Joueur:

Contient les informations du joueur,

ainsi que les parties aux quelles il joue.

\*/

#include <string>

#include "Partie.h"

using namespace std;

const int MAXPARTIES = 5; // Nombre de partie max par joueur

class Partie;

class Joueur

{

private:

string nom; // Nom du joueur

Partie\* partie[MAXPARTIES]; // Partie à laquelle est relié le joueur ( NULL si aucune partie en cours);

public:

Joueur(void);

Joueur(string); // Pseudo du joueur

Joueur(string, Partie\*); // Pseudo du joueur, Partie à rejoindre

Joueur(const Joueur& cpyJoueur); // Constructeur par copie

Joueur& operator=(Joueur& cpyJoueur); // Operateur d'affectation

string getNom() const;

void setNom(string); // Changer le pseudo du joueur

Partie\* getPartie(int) const;

bool setPartie(Partie\*); // Permet d'ajouter le joueur à une partie ( si il lui reste une place)

friend ostream& operator<<(ostream &flux, Joueur const& mJoueur); // Surcharge operateur <<

~Joueur(void);

};

ostream& operator<<(ostream &flux, Joueur const& mJoueur);

#endif

### Joueur.cpp

#include "Joueur.h"

ostream& operator<<(ostream &flux, Joueur const& mJoueur){

int nbPartie = 0;

for(int i = 0;i < MAXPARTIES;i++){

if(mJoueur.partie[i] != NULL)

nbPartie++;

}

flux << endl << mJoueur.nom << " a " << nbPartie << " partie(s) en cours :" << endl;

for(int i = 0;i < MAXPARTIES;i++){

if(mJoueur.partie[i] != NULL)

flux << "\t" << i << ". Du " << mJoueur.partie[i]->getDate();

}

flux << endl;

return flux;

}

Joueur::Joueur(void)

{

nom = "Default";

for(int i = 0; i < MAXPARTIES; i++)

partie[i] = NULL;

}

Joueur::Joueur(string nom)

{

this->nom = nom;

for(int i = 0; i < MAXPARTIES; i++)

this->partie[i] = NULL;

}

Joueur::Joueur(string nom, Partie\* partie)

{

this->nom = nom;

for(int i = 0; i < MAXPARTIES; i++)

this->partie[i] = NULL;

setPartie(partie);

}

Joueur::Joueur(const Joueur& cpyJoueur){

nom = cpyJoueur.nom;

nom.append("1");

}

Joueur& Joueur::operator=(Joueur& cpyJoueur){

nom = cpyJoueur.nom;

nom.append("1");

return cpyJoueur;

}

void Joueur::setNom(string nom)

{

this->nom = nom;

}

string Joueur::getNom() const

{

return nom;

}

bool Joueur::setPartie(Partie\* partie)

{

/\*if(partie == NULL){ // Si on recoit une partie à NULL, on supprime la partie du joueur

this->partie = NULL;

return true;

}\*/

/\* Si la partie envoyée est déjà dans les parties du joueur, on l'enleve des parties du joueur\*/

for(int i = 0; i < MAXPARTIES; i++){

if(this->partie[i] == partie){

this->partie[i] = NULL;

return false;

}

}

/\* Ensuite, on regarde si le joueur peut encore rejoindre une autre partie \*/

int nbParties = 0;

for(int i = 0; i < MAXPARTIES; i++)

if(this->partie[i] != NULL)

nbParties++;

if(nbParties == MAXPARTIES){return false;} // Si le joueur a atteinds la limite de parties

/\* Sinon, on trouve la premiere case vide et on met la partie dedans si il reste de la place \*/

int i = 0;

for(; this->partie[i] != NULL; i++);

if(partie->addJoueur(this)){ // Si il reste de la place dans la partie

this->partie[i] = partie;

return true;

}

return false;

}

Partie\* Joueur::getPartie(int i) const

{

return partie[i];

}

Joueur::~Joueur(void)

{

for(int i = 0; i < MAXPARTIES; i ++)

if( partie[i] != NULL)

delete partie[i];

}

### Partie.h

#ifndef \_PARTIE\_H

#define \_PARTIE\_H

/\*

Partie:

Gère tout ce qui est nécessaire pour une partie

\*/

#include <string>

#include <iostream>

#include <ctime>

#include <vector>

#include "Joueur.h"

#include "Piece.h"

#include "Fou.h"

#include "Tour.h"

#include "Cavalier.h"

#include "Reine.h"

#include "Roi.h"

#include "Pion.h"

using namespace std;

const int TAILLE=4;

class Joueur; // Pour que partie connaisse la class Joueur

class Partie

{

private:

string date; // Date sous le format : "2015-11-30 23-42-55"

Joueur\* j1; // Pointeur sur le joueur 1

Joueur\* j2; // Pointeur sur le joueur 2

Piece\* p[TAILLE][TAILLE]; // Un échiquier par partie

Piece\* pBlanc[4]; // Contient les pièces

Piece\* pNoir[4];

int typePartie; // Contient le numéro de la règle utilisé

bool isWhiteToPlay; // Permet de s'arreter dans une partie

int nCoup; // Comptabilise le nombre de coup fait durant la partie

int nDernierePrise; // Contient le numero du coup de la dernière prise

bool initPiece(Piece\*, char[3]); // Gère le placement initial des pièces privée car appelé par la fonction placePiece uniquement

bool deplacePiece(Piece\*, char[]);

bool deplacePiece(Piece\*, int, int);

public:

Partie(); // Date généré automatiquement

Partie(string); // Date

Partie(const Partie& cpyPartie); // Constructeur par copie

Partie& operator=(Partie& cpyPartie); // Operateur d'affectation

string getDate() const{ return date;}

Joueur\* getJ1() const {return j1;}

Joueur\* getJ2() const {return j2;}

int getTypePartie() const {return typePartie;}

bool getIsWhiteToPlay() const { return isWhiteToPlay; }

Piece\* getPNoir(int) const; // Renvoi la pièce i du tableau d'init

Piece\* getPBlanc(int) const;

void setPNoir(int, Piece\*);

void setPBlanc(int, Piece\*);

void setTypePartie(int, bool = false); // Met le type de partie ( num de la regle - possible une seule fois)

void setIsWhiteToPlay(bool); // Permet de choisir le prochain joueur

void affichePlateau(ostream& flux) const; // Permet cout << mPartie;

Piece\*& operator()(int ,int);

Piece\* operator()(int, int) const; // Permet de faire mPartie(i,j)

Piece\*& operator()(char[3]);

Piece\* operator() (char[3]) const; // Permet de faire mPartie('a0');

bool placePiece(Piece\*, char[3]); // Gère les déplacements

bool placePiece(Piece\*, int, int);

bool addJoueur(Joueur\*); // Permet d'ajouter les joueurs à la partie

vector<string> deplPossiblesSTL(string pos);

bool isPartieInit(); // Retourne true si la partie a fini l'initialisation

bool isPartieEnd(); // Renvoie true si la partie est terminé

int getGagnant(); // Renvoie la couleur gagnante (Blanc : 0 | Noir : 1 | Egalité : 2)

~Partie(void);

};

#endif

### Partie.cpp

### Piece.h

#ifndef \_PIECE\_H

#define \_PIECE\_H

/\*

Pièce:

Class mère, abstraite,

\*/

#include <string>

#include <iostream>

using namespace std;

class Piece

{

private:

int color; // Blanc : 0 Noir : 1

int state; // Disponible : 0 Placé : 1 Pris : 2

protected :

string Name; // Nom de la piece

int value; // Valeur de la pièce

public:

Piece(void);

Piece(int); // Couleur

Piece(int, int); // Couleur, Etat

virtual Piece\* clone() = 0; // Permet de dupliquer une pièce ( virtual car pièces typées)

string getName(){return Name;}

int getColor(){return color;}

int getState() { return state;}

int getValeur() {return value;}

void setState(int);

virtual bool setMove(char[], char[]) = 0; // Renvoi true si le mouvement est possible false sinon

char\* toString(); // Convertie une pièce en string de type '{Première lettre}{Couleur(N|B)}'

virtual ~Piece(void);

};

#endif

### Piece.cpp

#include "Piece.h"

Piece::Piece(void){}

Piece::Piece(int color)

{

Name = "Unkown";

this->color = color;

this->state = 0; // Disponible par defaut

}

Piece::Piece(int color, int state)

{

Name = "Unkown";

this->color = color;

this->state = state;

}

void Piece::setState(int state)

{

if(state >= 0 && state < 3)

this->state = state;

}

char\* Piece::toString() {

char mString[2];

if (color)

mString[1] = 'N';

else

mString[1] = 'B';

mString[0] = Name[0]; // Premiere lettre du Nom

return mString;

}

Piece::~Piece(void){

}

### Pion.h

#ifndef \_PION\_H

#define \_PION\_H

/\*

Pion:

Hérite de Pièce,

Mouvement verticaux ou diagonaux en fonction des pièces aux alentours

\*/

#include "Piece.h"

class Pion :

public Piece

{

public:

Pion(int);

Pion(int, int);

Pion\* clone();

bool setMove(char[], char[]);

~Pion(void);

};

#endif

### Pion.cpp

#include "Pion.h"

Pion::Pion(int color) : Piece(color)

{

Name = "Pion";

value = 1;

}

Pion::Pion(int color, int state) : Piece(color, state)

{

Name = "Pion";

value = 1;

}

Pion\* Pion::clone(){

Pion\* tmp = new Pion(this->getColor());

tmp->setState(this->getState());

return tmp;

}

bool Pion::setMove(char pos1[2], char pos2[2]) {

int x1 = pos1[0] - 'a';

int y1 = pos1[1] - '0';

int x2 = pos2[0] - 'a';

int y2 = pos2[1] - '0';

if (fabs((float)x2 - x1) == fabs((float)y2 - y1) && fabs((float)x2 - x1) == 1 && fabs((float)y2 - y1) == 1) { // Diagonale

return true;

}

if (x1 == x2 && fabs((float)y2 - y1) == 1) { // Vertical

return true;

}

return false;

}

Pion::~Pion(void)

{

}

### Reine.h

#ifndef \_REINE\_H

#define \_REINE\_H

/\*

Reine:

Hérite de Pièce,

Mouvement verticaux ou horizontaux ou diagonaux

\*/

#include "Piece.h"

class Reine :

public Piece

{

public:

Reine(int);

Reine(int, int);

Reine\* clone();

bool setMove(char[], char[]);

~Reine(void);

};

#endif

### Reine.cpp

#include "Reine.h"

Reine::Reine(int color) : Piece(color)

{

Name = "Reine";

value = 5;

}

Reine::Reine(int color, int state) : Piece(color, state)

{

Name = "Reine";

value = 5;

}

Reine\* Reine::clone(){

Reine\* tmp = new Reine(this->getColor());

tmp->setState(this->getState());

return tmp;

}

bool Reine::setMove(char pos1[2], char pos2[2]) {

int x1 = pos1[0] - 'a';

int y1 = pos1[1] - '0';

int x2 = pos2[0] - 'a';

int y2 = pos2[1] - '0';

if (fabs((float)x2 - x1) == fabs((float)y2 - y1)) { // Diagonale

return true;

}

if (x1 == x2 && y1 != y2) { // Vertical

return true;

}

if (y1 == y2 && x1 != x2) { // Horizontal

return true;

}

return false;

}

Reine::~Reine(void)

{

}

### Roi.h

#ifndef \_ROI\_H

#define \_ROI\_H

/\*

Roi:

Hérite de Pièce,

Mouvement limité à 1 case

\*/

#include "Piece.h"

class Roi :

public Piece

{

public:

Roi(int);

Roi(int, int);

Roi\* clone();

bool setMove(char[], char[]);

~Roi(void);

};

#endif

### Roi.cpp

#include "Roi.h"

Roi::Roi(int color) : Piece(color)

{

Name = "Roi";

value = 0;

}

Roi::Roi(int color, int state) : Piece(color, state)

{

Name = "Roi";

value = 0;

}

Roi\* Roi::clone(){

Roi\* tmp = new Roi(this->getColor());

tmp->setState(this->getState());

return tmp;

}

bool Roi::setMove(char pos1[2], char pos2[2]) {

int x1 = pos1[0] - 'a';

int y1 = pos1[1] - '0';

int x2 = pos2[0] - 'a';

int y2 = pos2[1] - '0';

if (fabs((float)x2 - x1) <= 1 && fabs((float)y2 - y1) <= 1 && (x1 != x2 || y1 != y2)) { // Deplacement de 1

return true;

}

return false;

}

Roi::~Roi(void)

{

}

### SDL\_INIT.h

/\*

Includes SDL et

définition de la fonction resize ( ==> main)

\*/

#ifndef \_SDL\_INIT\_H

#define \_SDL\_INIT\_H

#include <SDL/SDL.h>

#include <SDL/SDL\_ttf.h>

#include <SDL/SDL.h>

#include <SDL/SDL\_image.h>

#include <SDL/SDL\_rotozoom.h>

#include <SDL/SDL\_gfxPrimitives.h>

void resizeImage(SDL\_Surface\*& img, const double newwidth, const double newheight, bool x);

#endif

### Tour.h

#ifndef \_TOUR\_H

#define \_TOUR\_H

/\*

Tour:

Hérite de Pièce,

Mouvement verticaux ou horizontaux

\*/

#include "Piece.h"

class Tour :

public Piece

{

public:

Tour(int);

Tour(int, int);

Tour\* clone();

bool setMove(char[], char[]);

~Tour(void);

};

#endif

### Tour.cpp

#include "Tour.h"

Tour::Tour(int color) : Piece(color)

{

Name = "Tour";

value = 4;

}

Tour::Tour(int color, int state) : Piece(color, state)

{

Name = "Tour";

value = 4;

}

Tour\* Tour::clone(){

Tour\* tmp = new Tour(this->getColor());

tmp->setState(this->getState());

return tmp;

}

bool Tour::setMove(char pos1[2], char pos2[2]){

int x1 = pos1[0] - 'a';

int y1 = pos1[1] - '0';

int x2 = pos2[0] - 'a';

int y2 = pos2[1] - '0';

if ( x1 == x2 && y1 != y2){ // Vertical

return true;

}

if ( y1 == y2 && x1 != x2){ // Horizontal

return true;

}

return false;

}

Tour::~Tour(void)

{

}

### UserInterface.h

#ifndef \_USERINTERFACE\_H

#define \_USERINTERFACE\_H

/\*

UserInterface:

Class qui gère tout la partie Interface,

Affichage / Events / etc...

\*/

#include "SDL\_INIT.h"

#include "Joueur.h"

#include "Partie.h"

#include "Bouton.h"

#include "EditBox.h"

/\* On importe les fonctions qu'on devra utiliser (==> main) \*/

bool ajouterJoueur(string, Joueur\*\*\*&);

bool updateJoueur(string, Joueur\*\*\*&, int i);

void deleteJoueur(Joueur\*\*\*&, Joueur\*);

void newPartie(Partie\*\*\*&);

void deletePartie(Partie\*\*\*&, Partie\* mPartie);

/\* Constantes \*/

const int TX = 1200; // Largeur de la fenetre

const int TY = 900; // Hauteur de la fenetre

const int WIDTH = 900; // Largeur de la zone de "JEU"

const int CASE\_X = 100; // Largeur d'une case

const int CASE\_Y = 100; // Hauteur d'une case

typedef struct ImgPions { // Structure qui contient les images de pions

SDL\_Surface\* cavalier;

SDL\_Surface\* fou;

SDL\_Surface\* reine;

SDL\_Surface\* roi;

SDL\_Surface\* pion;

SDL\_Surface\* tour;

} ImgPions;

class UserInterface

{

private:

SDL\_Surface\* ecran; // Contient l'ecran d'affichage

TTF\_Font\* police; // Police d'écriture pour tout l'affichage

SDL\_Rect navBar; // Rectangle de la barre de navigation

SDL\_Rect plateau; // Rectangle du plateau

Joueur\*\*\* listeJoueur; // Utilisé seulement pour la lecture

Partie\*\*\* listePartie; // ( affichage des listes des parties et joueurs)

/\* Liste des boutons (leur nom permet de les identifier) \*/

SDL\_Color btnFontColor; // Couleur de la police

Uint32 btnColor; // Couleur du fond des boutons

Bouton\* btnSortir;

Bouton\* btnGestJoueurs;

Bouton\* btnGestParties;

Bouton\* btnAddJoueur;

Bouton\* btnUpdateJoueur;

Bouton\* btnDeleteJoueur;

Bouton\* btnValider;

Bouton\* btnPlayPartie;

Bouton\* btnSetJoueurPartie;

Bouton\* btnNewPartie;

Bouton\* btnDeletePartie;

Bouton\*\* btnListe;

EditBox\* eb; // EditBox pour le pseudo du joueur

int mode; // Modes d'affichage 0. Jouer Partie 1. Accueil 2. Gestionnaire Joueurs 3. Gestionnaire Parties 4. AjouterJoueur 5. UpdateJoueur

int selection; // Int qui contient l'indice de la partie ou du joueur selectionné -1 si rien de selectionné

string pseudo; // Utilisé pour ajouter un joueur ou modifier un joueur

int xPartie; // Utilisé pendant une partie

int yPartie; // afin de bouger un pion

int xSelectPartie; // Utilisé pour savoir

int ySelectPartie; // le pion qui se joue

ImgPions imgNoir; // Liste des images des pièces Noirs

ImgPions imgBlanc; // Liste des images des pièces Blanches

SDL\_Surface\* croix;

int checkEventMenu(int x, int y); // Permet de naviguer dans les différents menus, si mode == 0 alors on verifie que la barre laterale

int checkEventListe(int x, int y); // Change l'état de selection en l'id de la partie/ du joueur ou -1 si ailleurs

int checkEventEditBox(); // Gère ( a peu près) l'edition

void playPartie(Partie\*& mPartie); // Gère une partie de A à Z

void dPartie(Partie \*& mPartie); // Affiche une partie

void dPlateau(); // Affiche la partie tableau

void dNavBar(); // Affiche la barre de navigation

public:

UserInterface(Joueur\*\*\*, Partie\*\*\*); // Liste des joueurs, liste des parties

void start(); // Lance la gestion des events

int getMode() { return mode; }

int getSelect() { return selection; }

string getPseudo() { return pseudo; }

~UserInterface();

};

#endif

### UserInterface.cpp

#include "UserInterface.h"

UserInterface::UserInterface(Joueur\*\*\* listeJoueur, Partie\*\*\* listePartie)

{

this->listeJoueur = listeJoueur; // On enregistre les listes des joueurs et des parties

this->listePartie = listePartie;

mode = 1; // Page d'accueil

selection = -1; // Pas de selection

/\* Initialisation de la SDL \*/

SDL\_Init(SDL\_INIT\_VIDEO);

if (TTF\_Init() == -1)

{

fprintf(stderr, "Erreur d'initialisation de TTF\_Init : %s\n", TTF\_GetError());

exit(EXIT\_FAILURE);

}

police = TTF\_OpenFont("fonts/Roboto-Regular.ttf", 60); // Police/Fonts du texte

ecran = SDL\_SetVideoMode(TX, TY, 32, SDL\_HWSURFACE);

SDL\_WM\_SetCaption("ChessQuito | Projet C++ | Fernandes Marc-Antoine", NULL);

/\* On charge les images et on redimenssionne à la taille de la case\*/

imgNoir.tour = IMG\_Load("res/n\_tour.png");

imgNoir.roi = IMG\_Load("res/n\_roi.png");

imgNoir.fou = IMG\_Load("res/n\_fou.png");

imgNoir.reine = IMG\_Load("res/n\_reine.png");

imgNoir.cavalier = IMG\_Load("res/n\_cavalier.png");

imgNoir.pion = IMG\_Load("res/n\_pion.png");

imgBlanc.tour = IMG\_Load("res/b\_tour.png");

imgBlanc.roi = IMG\_Load("res/b\_roi.png");

imgBlanc.fou = IMG\_Load("res/b\_fou.png");

imgBlanc.reine = IMG\_Load("res/b\_reine.png");

imgBlanc.cavalier = IMG\_Load("res/b\_cavalier.png");

imgBlanc.pion = IMG\_Load("res/b\_pion.png");

croix = IMG\_Load("res/croix.bmp");

SDL\_SetColorKey(croix, SDL\_SRCCOLORKEY, SDL\_MapRGB(croix->format, 255, 255, 255));

SDL\_SetColorKey(imgNoir.tour, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgNoir.tour->format, 255, 0, 0));

SDL\_SetColorKey(imgNoir.roi, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgNoir.roi->format, 255, 0, 0));

SDL\_SetColorKey(imgNoir.fou, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgNoir.fou->format, 255, 0, 0));

SDL\_SetColorKey(imgNoir.reine, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgNoir.reine->format, 255, 0, 0));

SDL\_SetColorKey(imgNoir.cavalier, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgNoir.cavalier->format, 255, 0, 0));

SDL\_SetColorKey(imgNoir.pion, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgNoir.pion->format, 255, 0, 0));

SDL\_SetColorKey(imgBlanc.tour, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgBlanc.tour->format, 255, 0, 0));

SDL\_SetColorKey(imgBlanc.roi, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgBlanc.roi->format, 255, 0, 0));

SDL\_SetColorKey(imgBlanc.fou, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgBlanc.fou->format, 255, 0, 0));

SDL\_SetColorKey(imgBlanc.reine, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgBlanc.reine->format, 255, 0, 0));

SDL\_SetColorKey(imgBlanc.cavalier, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgBlanc.cavalier->format, 255, 0, 0));

SDL\_SetColorKey(imgBlanc.pion, SDL\_SRCCOLORKEY, SDL\_MapRGB(imgBlanc.pion->format, 255, 0, 0));

resizeImage(imgNoir.tour, CASE\_Y, CASE\_Y, false);

resizeImage(imgNoir.roi, CASE\_Y, CASE\_Y, false);

resizeImage(imgNoir.fou, CASE\_Y, CASE\_Y, false);

resizeImage(imgNoir.reine, CASE\_Y, CASE\_Y, false);

resizeImage(imgNoir.cavalier, CASE\_Y, CASE\_Y, false);

resizeImage(imgNoir.pion, CASE\_Y, CASE\_Y, false);

resizeImage(imgBlanc.tour, CASE\_Y, CASE\_Y, false);

resizeImage(imgBlanc.roi, CASE\_Y, CASE\_Y, false);

resizeImage(imgBlanc.fou, CASE\_Y, CASE\_Y, false);

resizeImage(imgBlanc.reine, CASE\_Y, CASE\_Y, false);

resizeImage(imgBlanc.cavalier, CASE\_Y, CASE\_Y, false);

resizeImage(imgBlanc.pion, CASE\_Y, CASE\_Y, false);

/\* On crée les boutons necessaires et on met les autres à NULL \*/

btnColor = SDL\_MapRGB(ecran->format, 129, 199, 132); // Couleur du fond d'un bouton (vert)

//btnFontColor = { 0,0,0 }; // Couleur d'écriture d'un bouton (noir)

btnFontColor.r = 0;

btnFontColor.g = 0;

btnFontColor.b = 0;

btnSortir = new Bouton(ecran, police, "SORTIR", WIDTH + 10, TY - 75, TX - WIDTH - 20, 70, btnColor, btnFontColor);

btnGestJoueurs = new Bouton(ecran, police, "Gestionnaire Joueurs", WIDTH + 10, 100, TX - WIDTH - 20, 70, btnColor, btnFontColor);

btnGestParties = new Bouton(ecran, police, "Gestionnaire Parties", WIDTH + 10, 200, TX - WIDTH - 20, 70, btnColor, btnFontColor);

// On initialise les boutons à leur première utilisation

btnAddJoueur = NULL;

btnUpdateJoueur = NULL;

btnDeleteJoueur = NULL;

btnValider = NULL;

btnPlayPartie = NULL;

btnSetJoueurPartie = NULL;

btnNewPartie = NULL;

btnDeletePartie = NULL;

btnListe = NULL;

/\* On initialise l'EditBox (Noir sur fond Blanc) \*/

Uint32 bgColor = SDL\_MapRGB(ecran->format, 255, 255, 255);

SDL\_Color fontColor = { 0, 0, 0 };

eb = new EditBox(ecran, police, 300, 300, 250, 45, bgColor, fontColor);

/\* On positionne la barre de navigation et le plateau \*/

navBar.h = TY;

navBar.w = TX - WIDTH;

navBar.x = WIDTH;

navBar.y = 0;

plateau.h = TY;

plateau.w = WIDTH;

plateau.x = 0;

plateau.y = 0;

dNavBar();

dPlateau();

SDL\_Flip(ecran);

}

void UserInterface::start()

{

/\* Démarre l'interface \*/

bool continuer = true;

SDL\_Event event;

while (continuer) {

SDL\_WaitEvent(&event);

switch (event.type) {

/\* Event de sorti \*/

case SDL\_QUIT:

continuer = 0;

break;

case SDL\_KEYDOWN:

if (event.key.keysym.sym == SDLK\_ESCAPE)

continuer = false;

break;

/\* Event du menu \*/

case SDL\_MOUSEBUTTONDOWN:

int action = checkEventMenu(event.button.x, event.button.y); // On recupère si possible l'action

if (action == 0) // Si SORTIR

continuer = false;

else if (action == -1) { // Pas dans le menu

if (mode == 2 || mode == 3) { // Liste joueurs + parties

checkEventListe(event.button.x, event.button.y);

}

else if (mode == 4 || mode == 5) { // Edit Box

int jsp = checkEventEditBox();

if (jsp == -1)

continuer = false;

else {

mode = 2;

dPlateau();

dNavBar();

}

}

}

break;

}

}

}

/\* Affichage menus \*/

void UserInterface::dPlateau()

{

SDL\_FillRect(ecran, &plateau, SDL\_MapRGB(ecran->format, 207, 216, 220)); // On met l'arriere plan

if (mode == 1) { // Accueil

SDL\_Surface\* tmp1;

SDL\_Surface\* tmp2;

SDL\_Surface\* tmp3;

SDL\_Color fontColor = { 0, 0, 0 };

tmp1 = TTF\_RenderText\_Blended(police, "Bienvenue", fontColor);

tmp2 = TTF\_RenderText\_Blended(police, "Sur le jeu", fontColor);

tmp3 = TTF\_RenderText\_Blended(police, "ChessQuito", fontColor);

SDL\_Rect txtTmp;

txtTmp.x = 100;

txtTmp.y = 100;

SDL\_BlitSurface(tmp1, NULL, ecran, &txtTmp);

txtTmp.x += 100;

txtTmp.y += 100;

SDL\_BlitSurface(tmp2, NULL, ecran, &txtTmp);

txtTmp.x += 100;

txtTmp.y += 100;

SDL\_BlitSurface(tmp3, NULL, ecran, &txtTmp);

SDL\_FreeSurface(tmp1);

SDL\_FreeSurface(tmp2);

SDL\_FreeSurface(tmp3);

}

else if (mode == 2) {

SDL\_Surface\* tmp;

tmp = TTF\_RenderText\_Blended(police, "Accueil > Gestionnaire des Joueurs", btnFontColor);

resizeImage(tmp, WIDTH - 100, WIDTH - 100, true);

SDL\_Rect tmpRect;

tmpRect.x = (WIDTH - tmp->w) / 2;

tmpRect.y = 20;

SDL\_BlitSurface(tmp, NULL, ecran, &tmpRect);

SDL\_FreeSurface(tmp);

Uint32 bgColor = SDL\_MapRGB(ecran->format, 255, 255, 255);

SDL\_Rect listeRect;

listeRect.x = 50;

listeRect.y = 100;

if (btnListe != NULL) {

for (int i = 0; btnListe[i] != NULL; i++)

delete btnListe[i];

delete btnListe;

}

int nb;

for (nb = 0; (\*listeJoueur)[nb] != NULL; nb++);

btnListe = new Bouton\*[nb + 1];

btnListe[nb] = NULL;

for (int i = 0; btnListe[i] != NULL; i++) {

if (i == selection) {

Uint32 bgColorbis = SDL\_MapRGB(ecran->format, 200, 200, 200);

btnListe[i] = new Bouton(ecran, police, (\*listeJoueur)[i]->getNom(), listeRect.x, listeRect.y, WIDTH - 100, 26, bgColorbis, btnFontColor);

}

else {

btnListe[i] = new Bouton(ecran, police, (\*listeJoueur)[i]->getNom(), listeRect.x, listeRect.y, WIDTH - 100, 26, bgColor, btnFontColor);

}

btnListe[i]->afficherBtn();

listeRect.y += 28;

}

}

else if (mode == 3) {

SDL\_Surface\* tmp;

tmp = TTF\_RenderText\_Blended(police, "Accueil > Gestionnaire des Parties", btnFontColor);

resizeImage(tmp, WIDTH - 100, WIDTH - 100, true);

SDL\_Rect tmpRect;

tmpRect.x = (WIDTH - tmp->w) / 2;

tmpRect.y = 20;

SDL\_BlitSurface(tmp, NULL, ecran, &tmpRect);

Uint32 bgColor = SDL\_MapRGB(ecran->format, 255, 255, 255);

SDL\_Rect listeRect;

listeRect.x = 50;

listeRect.y = 100;

if (btnListe != NULL) {

for (int i = 0; btnListe[i] != NULL; i++)

delete btnListe[i];

delete btnListe;

}

int nb;

for (nb = 0; (\*listePartie)[nb] != NULL; nb++);

btnListe = new Bouton\*[nb + 1];

btnListe[nb] = NULL;

for (int i = 0; btnListe[i] != NULL; i++) {

string txt = to\_string(i) + ". " + (\*listePartie)[i]->getDate();

if ((\*listePartie)[i]->getJ1() != NULL) {

txt += " Avec " + (\*listePartie)[i]->getJ1()->getNom();

if ((\*listePartie)[i]->getJ2() != NULL)

txt += " Et " + (\*listePartie)[i]->getJ2()->getNom();

}

else if((\*listePartie)[i]->getJ2() != NULL)

txt += " Avec " + (\*listePartie)[i]->getJ2()->getNom();

if (i == selection) {

Uint32 bgColorbis = SDL\_MapRGB(ecran->format, 200, 200, 200);

btnListe[i] = new Bouton(ecran, police, txt, listeRect.x, listeRect.y, WIDTH - 100, 26, bgColorbis, btnFontColor);

}

else {

btnListe[i] = new Bouton(ecran, police, txt, listeRect.x, listeRect.y, WIDTH - 100, 26, bgColor, btnFontColor);

}

btnListe[i]->afficherBtn();

listeRect.y += 28;

}

}

else if (mode == 4) {

SDL\_Surface\* tmp;

tmp = TTF\_RenderText\_Blended(police, "Accueil > Gestionnaire des Joueurs > Nouveau joueur", btnFontColor);

resizeImage(tmp, WIDTH - 100, WIDTH - 100, true);

SDL\_Rect tmpRect;

tmpRect.x = (WIDTH - tmp->w) / 2;

tmpRect.y = 20;

SDL\_BlitSurface(tmp, NULL, ecran, &tmpRect);

SDL\_FreeSurface(tmp);

eb->setText("Defaut");

eb->drawBox();

}

else if (mode == 5) {

SDL\_Surface\* tmp;

tmp = TTF\_RenderText\_Blended(police, "Accueil > Gestionnaire des Joueurs > Modifier joueur", btnFontColor);

resizeImage(tmp, WIDTH - 100, WIDTH - 100, true);

SDL\_Rect tmpRect;

tmpRect.x = (WIDTH - tmp->w) / 2;

tmpRect.y = 20;

SDL\_BlitSurface(tmp, NULL, ecran, &tmpRect);

SDL\_FreeSurface(tmp);

eb->setText((\*listeJoueur)[selection]->getNom());

eb->drawBox();

}

SDL\_Flip(ecran);

}

void UserInterface::dNavBar()

{

SDL\_FillRect(ecran, &navBar, SDL\_MapRGB(ecran->format, 144, 164, 174)); // On met la navBar

if (mode == 0) {

}

else if (mode == 1) {

btnSortir->afficherBtn();

btnGestJoueurs->afficherBtn();

btnGestParties->afficherBtn();

}

else if (mode == 2) {

btnSortir->afficherBtn();

if(btnAddJoueur == NULL)

btnAddJoueur = new Bouton(ecran, police, "Ajouter un joueur", WIDTH + 10, 100, TX - WIDTH - 20, 70, btnColor, btnFontColor);

if(btnUpdateJoueur == NULL)

btnUpdateJoueur = new Bouton(ecran, police, "Modifier le joueur selectionné", WIDTH + 10, 200, TX - WIDTH - 20, 70, btnColor, btnFontColor);

if(btnDeleteJoueur == NULL)

btnDeleteJoueur = new Bouton(ecran, police, "Supprimer le joueur selectionné", WIDTH + 10, 300, TX - WIDTH - 20, 70, btnColor, btnFontColor);

btnAddJoueur->afficherBtn();

btnUpdateJoueur->afficherBtn();

btnDeleteJoueur->afficherBtn();

}

else if (mode == 3) {

btnSortir->afficherBtn();

if(btnNewPartie == NULL)

btnNewPartie = new Bouton(ecran, police, "Créer une nouvelle partie", WIDTH + 10, 100, TX - WIDTH - 20, 70, btnColor, btnFontColor);

if(btnPlayPartie == NULL)

btnPlayPartie = new Bouton(ecran, police, "Jouer à la partie séléctionnée", WIDTH + 10, 200, TX - WIDTH - 20, 70, btnColor, btnFontColor);

if(btnDeletePartie == NULL)

btnDeletePartie = new Bouton(ecran, police, "Supprimer la partie selectionné", WIDTH + 10, 300, TX - WIDTH - 20, 70, btnColor, btnFontColor);

btnNewPartie->afficherBtn();

btnPlayPartie->afficherBtn();

btnDeletePartie->afficherBtn();

}

else if (mode == 4 || mode == 5) {

btnSortir->afficherBtn();

if (btnValider == NULL)

btnValider = new Bouton(ecran, police, "Valider", WIDTH + 10, 500, TX - WIDTH - 20, 70, btnColor, btnFontColor);

btnValider->afficherBtn();

}

}

/\* PartiePlayer \*/

void UserInterface::playPartie(Partie\*& mPartie) {

if (mPartie->getTypePartie() == -1) { // Temporaire, règle 1 obligatoire

mPartie->setTypePartie(1);

}

if (mPartie->isPartieInit()) {

xPartie = 0;

}

else {

if (mPartie->getIsWhiteToPlay()) {

xPartie = -2;

}

else {

xPartie = -1;

}

}

yPartie = 0;

xSelectPartie = -5; // Valeurs par defaut

ySelectPartie = -5;

bool continuer = true;

dPartie(mPartie);

SDL\_Event event;

while (continuer) {

SDL\_WaitEvent(&event);

switch (event.type) {

case SDL\_QUIT:

return;

case SDL\_KEYDOWN:

if (event.key.keysym.sym == SDLK\_ESCAPE)

return;

if (event.key.keysym.sym == SDLK\_LEFT && yPartie > 0) {

yPartie--;

}

else if (event.key.keysym.sym == SDLK\_RIGHT && yPartie < TAILLE - 1) {

yPartie++;

}

else if (event.key.keysym.sym == SDLK\_UP && xPartie > 0) {

xPartie--;

}

else if (event.key.keysym.sym == SDLK\_DOWN && xPartie < TAILLE - 1 && xPartie >= 0) {

xPartie++;

}

else if (event.key.keysym.sym == SDLK\_RETURN) {

if (!mPartie->isPartieInit()) {

if (xSelectPartie == -5 || ySelectPartie == -5) {

if ((xPartie == -2 && mPartie->getPBlanc(yPartie)->getState() == 0) || (xPartie == -1 && mPartie->getPNoir(yPartie)->getState() == 0)) {

xSelectPartie = xPartie;

ySelectPartie = yPartie;

xPartie = 0;

yPartie = 0;

}

}

else if (xSelectPartie == -2) {

if (mPartie->placePiece(mPartie->getPBlanc(ySelectPartie), xPartie, yPartie)) {

xSelectPartie = -5;

ySelectPartie = -5;

if (mPartie->isPartieInit())

xPartie = 0;

else

xPartie = -1;

yPartie = 0;

}

}

else {

if (mPartie->placePiece(mPartie->getPNoir(ySelectPartie), xPartie, yPartie)) {

xSelectPartie = -5;

ySelectPartie = -5;

if (mPartie->isPartieInit())

xPartie = 0;

else

xPartie = -2;

yPartie = 0;

}

}

}

else {

if ((xSelectPartie == -5 || ySelectPartie == -5) && (\*mPartie)(xPartie, yPartie) != NULL) {

if((\*mPartie)(xPartie, yPartie)->getColor() == !mPartie->getIsWhiteToPlay()){

xSelectPartie = xPartie;

ySelectPartie = yPartie;

xPartie = 0;

yPartie = 0;

}

}

else if(xSelectPartie == -5 || ySelectPartie == -5) {} // Empeche de tester le if d'après

else if (mPartie->placePiece((\*mPartie)(xSelectPartie,ySelectPartie), xPartie, yPartie)) {

xSelectPartie = -5;

ySelectPartie = -5;

if (mPartie->isPartieInit())

xPartie = 0;

else

xPartie = -2;

yPartie = 0;

}

}

}

dPartie(mPartie);

if (mPartie->isPartieEnd()) {

cout << "Partie terminé ! " << endl;

int winner = mPartie->getGagnant();

cout << "Le gagnant est le joueur :" ;

if(winner == 0){

cout << mPartie->getJ1()->getNom() << endl;

}

else if(winner == 1){

cout << mPartie->getJ2()->getNom() << endl;

}

else{

cout << "Egalite" << endl;

}

deletePartie(listePartie, mPartie);

continuer = false;

}

break;

case SDL\_MOUSEBUTTONDOWN:

if (btnSetJoueurPartie != NULL) {

if (btnSetJoueurPartie->isClicked(event.button.x, event.button.y)) {

cout << "Rentrez le pseudo svp :" << endl;

string tmp;

cin >> tmp;

for (int i = 0; (\*listeJoueur)[i] != NULL; i++) {

if (tmp == (\*listeJoueur)[i]->getNom()) {

mPartie->addJoueur((\*listeJoueur)[i]);

break;

}

}

if ((\*listePartie)[selection]->getJ1() != NULL && (\*listePartie)[selection]->getJ2() != NULL) { // On supprime le bouton afin d'arreter la detection

delete btnSetJoueurPartie;

btnSetJoueurPartie = NULL;

}

dPartie(mPartie);

}

}

else if (btnSortir->isClicked(event.button.x, event.button.y)) {

continuer = 0;

}

break;

}

}

}

void UserInterface::dPartie(Partie\*& mPartie) {

SDL\_FillRect(ecran, &plateau, SDL\_MapRGB(ecran->format, 207, 216, 220)); // On met l'arriere plan

SDL\_Surface\* tmp;

tmp = TTF\_RenderText\_Blended(police, ("Accueil > Gestionnaire des Parties > Jouer partie du " + mPartie->getDate()).c\_str(), btnFontColor);

resizeImage(tmp, WIDTH - 100, WIDTH - 100, true);

SDL\_Rect tmpRect;

tmpRect.x = (WIDTH - tmp->w) / 2;

tmpRect.y = 20;

SDL\_BlitSurface(tmp, NULL, ecran, &tmpRect);

SDL\_FreeSurface(tmp);

if ((\*listePartie)[selection]->getJ1() == NULL || (\*listePartie)[selection]->getJ2() == NULL) {

if(btnSetJoueurPartie == NULL)

btnSetJoueurPartie = new Bouton(ecran, police, "Ajouter un joueur à la partie (En maintenance ==> console)",150, 450, 600, 100, btnColor, btnFontColor);

btnSetJoueurPartie->afficherBtn();

}

else {

/\* On affiche les pions des joueurs \*/

SDL\_Rect rectBlanc;

SDL\_Rect rectNoir;

rectBlanc.h = CASE\_Y + 8;

rectBlanc.w = 4 \* CASE\_X + 8;

rectBlanc.x = 246;

rectBlanc.y = 120;

rectNoir.h = CASE\_Y + 8;

rectNoir.w = 4 \* CASE\_X + 8;

rectNoir.x = 246;

rectNoir.y = TY - 20 - CASE\_Y;

SDL\_FillRect(ecran, &rectBlanc, SDL\_MapRGB(ecran->format, 255, 255, 255)); // On met l'arriere plan

SDL\_FillRect(ecran, &rectNoir, SDL\_MapRGB(ecran->format, 0, 0, 0)); // On met l'arriere plan

rectBlanc.x += 4;

rectNoir.x += 4;

rectBlanc.y += 4;

rectNoir.y += 4;

rectBlanc.h -= 8;

rectNoir.h -= 8;

rectBlanc.w = CASE\_X;

rectNoir.w = CASE\_X;

for (int i = 0; i < 4; i++) {

/\* On choisi la couleur en fonction de ce qu'on veut (select, paire, etc...) \*/

Uint32 color;

if (xPartie == -2 && i == yPartie) {

color = SDL\_MapRGB(ecran->format, 255, 0, 0);

}

else if (xSelectPartie == -2 && i == ySelectPartie) {

color = SDL\_MapRGB(ecran->format, 0, 255, 0);

}

else if (i % 2 == 0) {

color = SDL\_MapRGB(ecran->format, 246, 228, 151);

}

else {

color = SDL\_MapRGB(ecran->format, 189, 141, 70);

}

SDL\_FillRect(ecran, &rectBlanc, color);

if (xPartie == -1 && i == yPartie) {

color = SDL\_MapRGB(ecran->format, 255, 0, 0);

}

else if (xSelectPartie == -1 && i == ySelectPartie) {

color = SDL\_MapRGB(ecran->format, 0, 255, 0);

}

else if (i % 2 == 0) {

color = SDL\_MapRGB(ecran->format, 246, 228, 151);

}

else {

color = SDL\_MapRGB(ecran->format, 189, 141, 70);

}

SDL\_FillRect(ecran, &rectNoir, color);

if (mPartie->getPBlanc(i) == NULL) {

}

else if (mPartie->getPBlanc(i)->getName() == "Tour") {

if(mPartie->getPBlanc(i)->getState() == 1)

SDL\_SetAlpha(imgBlanc.tour, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgBlanc.tour, NULL, ecran, &rectBlanc);

SDL\_SetAlpha(imgBlanc.tour, SDL\_SRCALPHA, 255);

}

else if (mPartie->getPBlanc(i)->getName() == "Roi") {

if (mPartie->getPBlanc(i)->getState() == 1)

SDL\_SetAlpha(imgBlanc.roi, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgBlanc.roi, NULL, ecran, &rectBlanc);

SDL\_SetAlpha(imgBlanc.roi, SDL\_SRCALPHA, 255);

}

else if (mPartie->getPBlanc(i)->getName() == "Fou") {

if (mPartie->getPBlanc(i)->getState() == 1)

SDL\_SetAlpha(imgBlanc.fou, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgBlanc.fou, NULL, ecran, &rectBlanc);

SDL\_SetAlpha(imgBlanc.fou, SDL\_SRCALPHA, 255);

}

else if (mPartie->getPBlanc(i)->getName() == "Reine") {

if (mPartie->getPBlanc(i)->getState() == 1)

SDL\_SetAlpha(imgBlanc.reine, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgBlanc.reine, NULL, ecran, &rectBlanc);

SDL\_SetAlpha(imgBlanc.reine, SDL\_SRCALPHA, 255);

}

else if (mPartie->getPBlanc(i)->getName() == "Cavalier") {

if (mPartie->getPBlanc(i)->getState() == 1)

SDL\_SetAlpha(imgBlanc.cavalier, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgBlanc.cavalier, NULL, ecran, &rectBlanc);

SDL\_SetAlpha(imgBlanc.cavalier, SDL\_SRCALPHA, 255);

}

else if (mPartie->getPBlanc(i)->getName() == "Pion") {

if (mPartie->getPBlanc(i)->getState() == 1)

SDL\_SetAlpha(imgBlanc.pion, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgBlanc.pion, NULL, ecran, &rectBlanc);

SDL\_SetAlpha(imgBlanc.pion, SDL\_SRCALPHA, 255);

}

if (mPartie->getPNoir(i) == NULL) {

}

else if (mPartie->getPNoir(i)->getName() == "Tour") {

if (mPartie->getPNoir(i)->getState() == 1)

SDL\_SetAlpha(imgNoir.tour, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgNoir.tour, NULL, ecran, &rectNoir);

SDL\_SetAlpha(imgNoir.tour, SDL\_SRCALPHA, 255);

}

else if (mPartie->getPNoir(i)->getName() == "Roi") {

if (mPartie->getPNoir(i)->getState() == 1)

SDL\_SetAlpha(imgNoir.roi, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgNoir.roi, NULL, ecran, &rectNoir);

SDL\_SetAlpha(imgBlanc.roi, SDL\_SRCALPHA, 255);

}

else if (mPartie->getPNoir(i)->getName() == "Fou") {

if (mPartie->getPNoir(i)->getState() == 1)

SDL\_SetAlpha(imgNoir.fou, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgNoir.fou, NULL, ecran, &rectNoir);

SDL\_SetAlpha(imgNoir.fou, SDL\_SRCALPHA, 255);

}

else if (mPartie->getPNoir(i)->getName() == "Reine") {

if (mPartie->getPNoir(i)->getState() == 1)

SDL\_SetAlpha(imgNoir.reine, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgNoir.reine, NULL, ecran, &rectNoir);

SDL\_SetAlpha(imgNoir.reine, SDL\_SRCALPHA, 255);

}

else if (mPartie->getPNoir(i)->getName() == "Cavalier") {

if (mPartie->getPNoir(i)->getState() == 1)

SDL\_SetAlpha(imgNoir.cavalier, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgNoir.cavalier, NULL, ecran, &rectNoir);

SDL\_SetAlpha(imgNoir.cavalier, SDL\_SRCALPHA, 255);

}

else if (mPartie->getPNoir(i)->getName() == "Pion") {

if (mPartie->getPNoir(i)->getState() == 1)

SDL\_SetAlpha(imgNoir.pion, SDL\_SRCALPHA, 128);

SDL\_BlitSurface(imgNoir.pion, NULL, ecran, &rectNoir);

SDL\_SetAlpha(imgNoir.pion, SDL\_SRCALPHA, 255);

}

rectBlanc.w = CASE\_X; // Car changé par le BlitSurface...

rectNoir.w = CASE\_X;

if (mPartie->getPBlanc(i)->getState() == 2) {

SDL\_BlitSurface(croix, NULL, ecran, &rectBlanc);

}

if (mPartie->getPNoir(i)->getState() == 2) {

SDL\_BlitSurface(croix, NULL, ecran, &rectNoir);

}

rectBlanc.x += CASE\_X;

rectNoir.x += CASE\_X;

}

/\* On colle le plateau \*/

SDL\_Rect contour;

contour.h = CASE\_Y \* TAILLE + 24;

contour.w = CASE\_X \* TAILLE + 24;

contour.x = 238;

contour.y = 268;

if(mPartie->getIsWhiteToPlay())

SDL\_FillRect(ecran, &contour, SDL\_MapRGB(ecran->format, 255, 255, 255)); // On met l'arriere plan

else

SDL\_FillRect(ecran, &contour, SDL\_MapRGB(ecran->format, 0, 0, 0)); // On met l'arriere plan

contour.h = CASE\_Y \* TAILLE + 8;

contour.w = CASE\_X \* TAILLE + 8;

contour.x = 246;

contour.y = 276;

SDL\_FillRect(ecran, &contour, SDL\_MapRGB(ecran->format, 144, 164, 174)); // On met l'arriere plan

SDL\_Rect caseRect;

caseRect.h = CASE\_Y;

caseRect.w = CASE\_X;

caseRect.x = 250;

caseRect.y = 280;

for (int i = 0; i < TAILLE; i++) {

caseRect.x = 250;

for (int j = 0; j < TAILLE; j++) {

/\* On choisi la couleur en fonction de ce qu'on veut (select, paire, etc...) \*/

Uint32 color;

if (i == xPartie && j == yPartie) {

color = SDL\_MapRGB(ecran->format, 255, 0, 0);

}

else if (i == xSelectPartie && j == ySelectPartie) {

color = SDL\_MapRGB(ecran->format, 0, 255, 0);

}

else if ((j + i) % 2 == 0) {

color = SDL\_MapRGB(ecran->format, 246, 228, 151);

}

else {

color = SDL\_MapRGB(ecran->format, 189, 141, 70);

}

SDL\_FillRect(ecran, &caseRect, color); // On met la case

SDL\_Rect posPiece;

posPiece.y = caseRect.y;

/\* On affiche la pièce correspondante \*/

if ((\*mPartie)(i, j) == NULL) {

}

else if ((\*mPartie)(i, j)->getName() == "Tour") {

posPiece.x = caseRect.x + CASE\_X / 2 - imgBlanc.tour->w / 2;

if ((\*mPartie)(i, j)->getColor() == 0)

SDL\_BlitSurface(imgBlanc.tour, NULL, ecran, &posPiece);

else

SDL\_BlitSurface(imgNoir.tour, NULL, ecran, &posPiece);

}

else if ((\*mPartie)(i, j)->getName() == "Roi") {

posPiece.x = caseRect.x + CASE\_X / 2 - imgBlanc.roi->w / 2;

if ((\*mPartie)(i, j)->getColor() == 0)

SDL\_BlitSurface(imgBlanc.roi, NULL, ecran, &posPiece);

else

SDL\_BlitSurface(imgNoir.roi, NULL, ecran, &posPiece);

}

else if ((\*mPartie)(i, j)->getName() == "Fou") {

posPiece.x = caseRect.x + CASE\_X / 2 - imgBlanc.fou->w / 2;

if ((\*mPartie)(i, j)->getColor() == 0)

SDL\_BlitSurface(imgBlanc.fou, NULL, ecran, &posPiece);

else

SDL\_BlitSurface(imgNoir.fou, NULL, ecran, &posPiece);

}

else if ((\*mPartie)(i, j)->getName() == "Reine") {

posPiece.x = caseRect.x + CASE\_X / 2 - imgBlanc.reine->w / 2;

if ((\*mPartie)(i, j)->getColor() == 0)

SDL\_BlitSurface(imgBlanc.reine, NULL, ecran, &posPiece);

else

SDL\_BlitSurface(imgNoir.reine, NULL, ecran, &posPiece);

}

else if ((\*mPartie)(i, j)->getName() == "Cavalier") {

posPiece.x = caseRect.x + CASE\_X / 2 - imgBlanc.cavalier->w / 2;

if ((\*mPartie)(i, j)->getColor() == 0)

SDL\_BlitSurface(imgBlanc.cavalier, NULL, ecran, &posPiece);

else

SDL\_BlitSurface(imgNoir.cavalier, NULL, ecran, &posPiece);

}

else if ((\*mPartie)(i, j)->getName() == "Pion") {

posPiece.x = caseRect.x + CASE\_X / 2 - imgBlanc.pion->w / 2;

if ((\*mPartie)(i, j)->getColor() == 0)

SDL\_BlitSurface(imgBlanc.pion, NULL, ecran, &posPiece);

else

SDL\_BlitSurface(imgNoir.pion, NULL, ecran, &posPiece);

}

caseRect.x += CASE\_X;

}

caseRect.y += CASE\_Y;

}

}

SDL\_Flip(ecran);

}

/\* CheckEvent \*/

int UserInterface::checkEventMenu(int x, int y)

{

if (x < WIDTH)

return -1;

if (mode == 1) {

if (btnSortir->isClicked(x, y))

return 0;

if (btnGestJoueurs->isClicked(x, y)) {

mode = 2;

dNavBar();

dPlateau();

return -1;

}

if (btnGestParties->isClicked(x, y)) {

mode = 3;

dNavBar();

dPlateau();

return -1;

}

}

else if (mode == 2) {

if (btnSortir->isClicked(x, y)) {

mode = 1;

dNavBar();

dPlateau();

return -1;

}

if (btnAddJoueur->isClicked(x, y)) {

mode = 4;

dPlateau();

dNavBar();

return 5;

}

if (btnUpdateJoueur->isClicked(x, y) && selection != -1) {

mode = 5;

dPlateau();

dNavBar();

return 4;

}

if (btnDeleteJoueur->isClicked(x, y) && selection != -1) {

deleteJoueur(listeJoueur, (\*listeJoueur)[selection]);

selection = -1;

dPlateau();

return 6;

}

}

else if (mode == 3) {

if (btnSortir->isClicked(x, y)) {

mode = 1;

dNavBar();

dPlateau();

return -1;

}

if (btnNewPartie->isClicked(x, y)) {

newPartie(listePartie);

dPlateau();

return 2;

}

if (btnPlayPartie->isClicked(x, y) && selection != -1) {

playPartie((\*listePartie)[selection]);

dPlateau();

dNavBar();

return 7;

}

if (btnDeletePartie->isClicked(x, y) && selection != -1) {

deletePartie(listePartie, (\*listePartie)[selection]);

selection = -1;

dPlateau();

return 1;

}

}

else if (mode == 4 || mode == 5) {

if (btnSortir->isClicked(x, y)) {

mode = 2;

dNavBar();

dPlateau();

return -1;

}

if (btnValider->isClicked(x, y)) {

return 8;

}

}

return -1;

}

int UserInterface::checkEventListe(int x, int y)

{

if (mode != 2 && mode != 3)

return -1;

selection = - 1;

for (int i = 0; btnListe[i] != NULL; i++) {

if (btnListe[i]->isClicked(x, y)) {

selection = i;

break;

}

}

if (selection != -1)

dPlateau();

return 0;

}

int UserInterface::checkEventEditBox()

{

bool isEditionFinish = false;

while (!isEditionFinish) {

int res = eb->start();

if (res == -2) { // Quitter le jeu

return -1;

}

SDL\_Event event;

SDL\_WaitEvent(&event);

if (btnSortir->isClicked(event.button.x, event.button.y)) {

return 1;

}

if ((btnValider->isClicked(event.button.x, event.button.y) || res == 0) && eb->getText().length() > 0) {

if (mode == 4) {

if (!ajouterJoueur(eb->getText(), listeJoueur)) {

dPlateau();

SDL\_Surface\* tmp;

tmp = TTF\_RenderText\_Blended(police, "Le nom est déjà pris !", btnFontColor);

resizeImage(tmp, 250, 250, true);

SDL\_Rect tmpRect;

tmpRect.x = 300;

tmpRect.y = 360;

SDL\_BlitSurface(tmp, NULL, ecran, &tmpRect);

SDL\_FreeSurface(tmp);

}

else

isEditionFinish = true;

}

else {

if (!updateJoueur(eb->getText(), listeJoueur, selection)) {

dPlateau();

SDL\_Surface\* tmp;

tmp = TTF\_RenderText\_Blended(police, "Le nom est déjà pris !", btnFontColor);

resizeImage(tmp, 250, 250, true);

SDL\_Rect tmpRect;

tmpRect.x = 300;

tmpRect.y = 360;

SDL\_BlitSurface(tmp, NULL, ecran, &tmpRect);

SDL\_FreeSurface(tmp);

}

else

isEditionFinish = true;

}

}

}

return 0;

}

/\* Destructeur \*/

UserInterface::~UserInterface()

{

/\* On delete les boutons \*/

if(btnSortir != NULL)

delete btnSortir;

if (btnGestJoueurs != NULL)

delete btnGestJoueurs;

if (btnGestParties != NULL)

delete btnGestParties;

if (btnAddJoueur != NULL)

delete btnAddJoueur;

if (btnUpdateJoueur != NULL)

delete btnUpdateJoueur;

if (btnDeleteJoueur != NULL)

delete btnDeleteJoueur;

if (btnValider != NULL)

delete btnValider;

if (btnPlayPartie != NULL)

delete btnPlayPartie;

if (btnNewPartie != NULL)

delete btnNewPartie;

if (btnDeletePartie != NULL)

delete btnDeletePartie;

if (btnListe != NULL) {

for (int i = 0; btnListe[i] != NULL; i++)

delete btnListe[i];

delete btnListe;

}

/\* On efface les pièces \*/

SDL\_FreeSurface(imgNoir.tour);

SDL\_FreeSurface(imgNoir.roi);

SDL\_FreeSurface(imgNoir.fou);

SDL\_FreeSurface(imgNoir.reine);

SDL\_FreeSurface(imgNoir.cavalier);

SDL\_FreeSurface(imgNoir.pion);

SDL\_FreeSurface(imgBlanc.tour);

SDL\_FreeSurface(imgBlanc.roi);

SDL\_FreeSurface(imgBlanc.fou);

SDL\_FreeSurface(imgBlanc.reine);

SDL\_FreeSurface(imgBlanc.cavalier);

SDL\_FreeSurface(imgBlanc.pion);

SDL\_FreeSurface(croix);

/\* On delete l'EditBox \*/

delete eb;

/\* On ferme les modules \*/

TTF\_CloseFont(police);

TTF\_Quit();

SDL\_Quit();

}

### Main.cpp

/\*

\* ChessQuito

\*/

/\* Includes STANDARD \*/

#include <iostream>

#include <cstdlib>

#include <stdio.h>

#include <Windows.h>

#include <fstream>

/\* Includes ChessQuito \*/

#include "Joueur.h"

#include "Partie.h"

#include "UserInterface.h"

using namespace std;

/\* Liste fonctions \*/

bool ajouterJoueur(string, Joueur\*\*\*&);

bool updateJoueur(string, Joueur\*\*\*&, int);

void deleteJoueur(Joueur\*\*\*&, Joueur\*);

void newPartie(Partie\*\*\*&);

void deletePartie(Partie\*\*\*&, Partie\*);

void chargerJeu(Partie\*\*\*&, Joueur\*\*\*&);

void saveParties(Partie\*\*\*&);

void saveJoueurs(Joueur\*\*\*&);

//////////////////

// MAIN //

//////////////////

int main(int argc, char \*argv[]) {

/\* On charge les sauvegardes \*/

Joueur\*\*\* listeJoueur = NULL;

Partie\*\*\* listePartie = NULL;

chargerJeu(listePartie, listeJoueur);

/\* Initialisation de l'interface Utilisateur \*/

UserInterface ui(listeJoueur, listePartie);

ui.start();

/\* On enregistre les joueurs + les parties \*/

saveJoueurs(listeJoueur);

saveParties(listePartie);

/\* On supprime les listes de la mémoire \*/

for(int i = 0; (\*listePartie)[i] != NULL;i++){

delete (\*listePartie)[i];

}

delete [](\*listePartie);

for (int i = 0; (\*listeJoueur)[i] != NULL; i++) {

delete (\*listeJoueur)[i];

}

delete[](\*listeJoueur);

return EXIT\_SUCCESS;

}

////////////////////////

// CHESSQUITO //

////////////////////////

/\* Surcharges cout \*/

ostream& operator<<(ostream &flux, Partie const& mPartie)

{

mPartie.affichePlateau(flux);

return flux;

}

/\* Gestion (ajout|suppression) des (joueurs|parties) \*/

bool ajouterJoueur(string nom, Joueur\*\*\*& listeJoueur) {

int i = 0;

for (; (\*listeJoueur)[i] != NULL; i++)

if (nom == (\*listeJoueur)[i]->getNom()) {

return false;

}

/\* On commence agrandir le tableau de pointeur de joueurs \*/

Joueur\*\* tabTmp = new Joueur\*[i + 2];

for (int j = 0; j < i; j++) {

tabTmp[j] = (\*listeJoueur)[j];

}

tabTmp[i] = new Joueur(nom);

tabTmp[i + 1] = NULL; // On met le dernier à NULL afin de pouvoir avoir un etat d'arret lors des tests sur la liste

delete[](\*listeJoueur); // On supprime l'ancien tableau

(\*listeJoueur) = tabTmp; // On assigne la nouvelle addresse du tableau

return true;

}

bool updateJoueur(string nom, Joueur\*\*\*& listeJoueur, int i) {

for (int j = 0; (\*listeJoueur)[j] != NULL; j++)

if (nom == (\*listeJoueur)[j]->getNom()) {

return false;

}

(\*listeJoueur)[i]->setNom(nom);

return true;

}

void deleteJoueur(Joueur\*\*\*& listeJoueur, Joueur\* mJoueur) {

int i = 0;

for (; (\*listeJoueur)[i] != NULL; i++);

int id = 0;

for (; (\*listeJoueur)[id] != mJoueur; id++);

/\* On reduit le tableau de pointeur de joueurs \*/

Joueur\*\* tabTmp = new Joueur\*[i];

for (int j = 0, k = 0; j < i; j++) {

if (id != j) {

tabTmp[k] = (\*listeJoueur)[j];

k++;

}

}

delete mJoueur;

tabTmp[i - 1] = NULL; // On met le dernier à NULL afin de pouvoir avoir un etat d'arret lors des tests sur la liste

delete[](\*listeJoueur); // On supprime l'ancien tableau

(\*listeJoueur) = tabTmp; // On assigne la nouvelle addresse du tableau

}

void newPartie(Partie\*\*\*& listePartie) {

int i = 0;

for (; (\*listePartie)[i] != NULL; i++);

/\* On commence agrandir le tableau de pointeur de partie \*/

Partie\*\* tabTmp = new Partie\*[i + 2];

for (int j = 0; j < i; j++) {

tabTmp[j] = (\*listePartie)[j];

}

tabTmp[i] = new Partie();

tabTmp[i + 1] = NULL; // On met le dernier à NULL afin de pouvoir avoir un etat d'arret lors des tests sur la liste

delete[](\*listePartie); // On supprime l'ancien tableau

(\*listePartie) = tabTmp; // On change l'addresse du tableau pointé par listePartie

}

void deletePartie(Partie\*\*\*& listePartie, Partie\* mPartie) {

int i = 0;

for (; (\*listePartie)[i] != NULL; i++);

int id = 0;

for (; (\*listePartie)[id] != mPartie; id++);

/\* On reduit le tableau de pointeur de joueurs \*/

Partie\*\* tabTmp = new Partie\*[i];

for (int j = 0, k = 0; j < i; j++) {

if (id != j) {

tabTmp[k] = (\*listePartie)[j];

k++;

}

}

delete mPartie;

tabTmp[i - 1] = NULL; // On met le dernier à NULL afin de pouvoir avoir un etat d'arret lors des tests sur la liste

delete[](\*listePartie); // On supprime l'ancien tableau

(\*listePartie) = tabTmp; // On assigne la nouvelle addresse du tableau

}

/\* Fonctions de chargement des joueurs et parties \*/

void chargerJeu(Partie\*\*\*& listePartie, Joueur\*\*\*& listeJoueur) {

cout << "Chargement des Joueurs .";

/\* On charge les joueurs \*/

ifstream fichier("save/Joueurs.txt", ios::in); // on ouvre en lecture

if (fichier) // si l'ouverture a fonctionné

{

string contenu; // déclaration d'une chaîne qui contiendra la ligne lue

int i = 0;

cout << ".";

while (!fichier.eof()) {

fichier.ignore(1000, '\n');

i++;

}

listeJoueur = new Joueur\*\*;

(\*listeJoueur) = new Joueur\*[i];

(\*listeJoueur)[i - 1] = NULL;

fichier.seekg(0, std::ios::beg);

for (int j = 0; j < i - 1; j++) {

cout << ".";

getline(fichier, contenu, '\n');

(\*listeJoueur)[j] = new Joueur(contenu);

}

cout << endl;

fichier.close();

}

else {

cout << "Impossible de charger les joueurs !" << endl;

listeJoueur = new Joueur\*\*;

(\*listeJoueur) = new Joueur\*[1];

(\*listeJoueur)[0] = NULL;

}

cout << "Chargement des Parties .";

/\* On charge maintenant les parties \*/

ifstream fichier2("save/Parties.txt", ios::in); // on ouvre en lecture

if (fichier2) // si l'ouverture a fonctionné

{

string contenu; // déclaration d'une chaîne qui contiendra la ligne lue

cout << ".";

int i = 0;

while (!fichier2.eof()) {

fichier2.ignore(1000, '\n');

i++;

}

listePartie = new Partie\*\*;

(\*listePartie) = new Partie\*[i];

(\*listePartie)[i - 1] = NULL;

fichier2.seekg(0, std::ios::beg);

for (int j = 0; j < i - 1; j++) {

cout << ".";

/\* On charge chaque fichier \*/

string fileName;

getline(fichier2, contenu, '\n');

fileName = "save/parties/" + contenu + ".txt";

ifstream tmp(fileName, ios::in); // on ouvre en lecture

/\* On parcours les différentes propriétées du fichier \*/

// Date

getline(tmp, contenu, '\n');

(\*listePartie)[j] = new Partie(contenu);

// Joueur 1

getline(tmp, contenu, '\n');

for (int k = 0; (\*listeJoueur)[k] != NULL; k++) {

if ((\*listeJoueur)[k]->getNom() == contenu) {

(\*listePartie)[j]->addJoueur((\*listeJoueur)[k]);

break;

}

}

// Joueur 2

getline(tmp, contenu, '\n');

for (int k = 0; (\*listeJoueur)[k] != NULL; k++) {

if ((\*listeJoueur)[k]->getNom() == contenu) {

(\*listePartie)[j]->addJoueur((\*listeJoueur)[k]);

break;

}

}

// Type de partie

getline(tmp, contenu, '\n');

(\*listePartie)[j]->setTypePartie(atoi(contenu.c\_str()), true);

// IsWhiteToPlay

getline(tmp, contenu, '\n');

(\*listePartie)[j]->setIsWhiteToPlay(atoi(contenu.c\_str()));

// Tableaux d'initialisation

for (int k = 0; k < 4; k++) {

getline(tmp, contenu, '\_'); // Piece par Piece

if (contenu == "NULL") {

tmp.ignore(5, ' ');

(\*listePartie)[j]->setPBlanc(k, NULL);

}

else {

int color;

int state;

tmp >> color;

tmp.ignore(1); // UNDERSCORE

tmp >> state;

tmp.ignore(1); // SPACE

if (contenu == "Tour") {

(\*listePartie)[j]->setPBlanc(k, new Tour(color, state));

}

else if (contenu == "Fou") {

(\*listePartie)[j]->setPBlanc(k, new Fou(color, state));

}

else if (contenu == "Roi") {

(\*listePartie)[j]->setPBlanc(k, new Roi(color, state));

}

else if (contenu == "Reine") {

(\*listePartie)[j]->setPBlanc(k, new Reine(color, state));

}

else if (contenu == "Cavalier") {

(\*listePartie)[j]->setPBlanc(k, new Cavalier(color, state));

}

else if (contenu == "Pion") {

(\*listePartie)[j]->setPBlanc(k, new Pion(color, state));

}

}

}

tmp.ignore(2, '\n');

for (int k = 0; k < 4; k++) {

getline(tmp, contenu, '\_'); // Piece par Piece

if (contenu == "NULL") {

tmp.ignore(5, ' ');

(\*listePartie)[j]->setPNoir(k, NULL);

}

else {

int color;

int state;

tmp >> color;

tmp.ignore(1); // UNDERSCORE

tmp >> state;

tmp.ignore(1); // SPACE

if (contenu == "Tour") {

(\*listePartie)[j]->setPNoir(k, new Tour(color, state));

}

else if (contenu == "Fou") {

(\*listePartie)[j]->setPNoir(k, new Fou(color, state));

}

else if (contenu == "Roi") {

(\*listePartie)[j]->setPNoir(k, new Roi(color, state));

}

else if (contenu == "Reine") {

(\*listePartie)[j]->setPNoir(k, new Reine(color, state));

}

else if (contenu == "Cavalier") {

(\*listePartie)[j]->setPNoir(k, new Cavalier(color, state));

}

else if (contenu == "Pion") {

(\*listePartie)[j]->setPNoir(k, new Pion(color, state));

}

}

}

tmp.ignore(2, '\n');

// Chargement du plateau

for (int l = 0; l < TAILLE; l++) {

for (int k = 0; k < TAILLE; k++) {

getline(tmp, contenu, '\_'); // Piece par Piece

if (contenu == "NULL") {

tmp.ignore(5, ' ');

(\*(\*listePartie)[j])(l, k) = NULL;

}

else {

int color;

int state;

tmp >> color;

tmp.ignore(1); // UNDERSCORE

tmp >> state;

tmp.ignore(1); // SPACE

if (contenu == "Tour") {

(\*(\*listePartie)[j])(l, k) = new Tour(color, state);

}

else if (contenu == "Fou") {

(\*(\*listePartie)[j])(l, k) = new Fou(color, state);

}

else if (contenu == "Roi") {

(\*(\*listePartie)[j])(l, k) = new Roi(color, state);

}

else if (contenu == "Reine") {

(\*(\*listePartie)[j])(l, k) = new Reine(color, state);

}

else if (contenu == "Cavalier") {

(\*(\*listePartie)[j])(l, k) = new Cavalier(color, state);

}

else if (contenu == "Pion") {

(\*(\*listePartie)[j])(l, k) = new Pion(color, state);

}

}

}

tmp.ignore(2, '\n');

}

// Fin de la boucle

tmp.close();

}

}

else {

cout << "Impossible de charger la sauvegarde !" << endl;

listePartie = new Partie\*\*;

(\*listePartie) = new Partie\*[1];

(\*listePartie)[0] = NULL;

}

}

void saveParties(Partie\*\*\*& listePartie) {

/\* On commence par creer un registre des parties \*/

ofstream fichier("save/Parties.txt", ios::out | ios::trunc); //déclaration du flux et ouverture du fichier

if (fichier) // si l'ouverture a réussi

{

for (int i = 0; (\*listePartie)[i] != NULL; i++) {

fichier << (\*listePartie)[i]->getDate() << endl;

}

fichier.close(); // on referme le fichier

}

else { // sinon

cout << "Erreur durant l'enregistrement !" << endl;

return;

}

for (int i = 0; (\*listePartie)[i] != NULL; i++) {

string fileName = "save/parties/" + (\*listePartie)[i]->getDate() + ".txt";

ofstream fichier(fileName, ios::out | ios::trunc); //déclaration du flux et ouverture du fichier

if (fichier) // si l'ouverture a réussi

{

/\* On commence par enregistrer les différentes propriétées de la partie \*/

fichier << (\*listePartie)[i]->getDate() << endl;

if ((\*listePartie)[i]->getJ1() == NULL)

fichier << "\_é!#charSpeciauxà@\_" << endl;

else

fichier << (\*listePartie)[i]->getJ1()->getNom() << endl;

if ((\*listePartie)[i]->getJ2() == NULL)

fichier << "\_é!#charSpeciauxà@\_" << endl;

else

fichier << (\*listePartie)[i]->getJ2()->getNom() << endl;

fichier << (\*listePartie)[i]->getTypePartie() << endl

<< (\*listePartie)[i]->getIsWhiteToPlay() << endl;

/\* On enregistre les deux tableaux de pièce\*/

for (int j = 0; j < 4; j++) {

if ((\*listePartie)[i]->getPBlanc(j) != NULL)

fichier << (\*listePartie)[i]->getPBlanc(j)->getName() << "\_" << (\*listePartie)[i]->getPBlanc(j)->getColor() << "\_" << (\*listePartie)[i]->getPBlanc(j)->getState() << " ";

else

fichier << "NULL\_0\_0 ";

}

fichier << endl;

for (int j = 0; j < 4; j++) {

if ((\*listePartie)[i]->getPNoir(j) != NULL)

fichier << (\*listePartie)[i]->getPNoir(j)->getName() << "\_" << (\*listePartie)[i]->getPNoir(j)->getColor() << "\_" << (\*listePartie)[i]->getPNoir(j)->getState() << " ";

else

fichier << "NULL\_0\_0 ";

}

fichier << endl;

/\* On enregistre le plateau \*/

for (int j = 0; j < TAILLE; j++) {

for (int k = 0; k < TAILLE; k++) {

if ((\*(\*listePartie)[i])(j, k) != NULL)

fichier << (\*(\*listePartie)[i])(j, k)->getName() << "\_" << (\*(\*listePartie)[i])(j, k)->getColor() << "\_" << (\*(\*listePartie)[i])(j, k)->getState() << " ";

else

fichier << "NULL\_0\_0 ";

}

fichier << endl;

}

fichier.close(); // on referme le fichier

}

else { // sinon

cout << "Erreur durant l'enregistrement !" << endl;

return;

}

}

cout << "Fin de l'enregistrement ! " << endl;

}

void saveJoueurs(Joueur\*\*\*& listeJoueur) {

ofstream fichier("save/Joueurs.txt", ios::out | ios::trunc); //déclaration du flux et ouverture du fichier

if (fichier) // si l'ouverture a réussi

{

for (int i = 0; (\*listeJoueur)[i] != NULL; i++) {

fichier << (\*listeJoueur)[i]->getNom() << endl;

}

fichier.close(); // on referme le fichier

}

else { // sinon

cout << "Erreur durant l'enregistrement !" << endl;

return;

}

}

///////////////////

// SDL //

///////////////////

/\* Redimmensionne l'image \*/

void resizeImage(SDL\_Surface\*& img, const double newwidth, const double newheight, bool x)

{

// Zoom function uses doubles for rates of scaling, rather than

// exact size values. This is how we get around that:

double zoomx = newwidth / (float)img->w;

double zoomy = newheight / (float)img->h;

SDL\_Surface\* sized = NULL;

// This function assumes no smoothing, so that any colorkeys wont bleed.

if (x)

sized = zoomSurface(img, zoomx, zoomx, SMOOTHING\_OFF);

else

sized = zoomSurface(img, zoomy, zoomy, SMOOTHING\_OFF);

SDL\_FreeSurface(img);

img = sized;

}