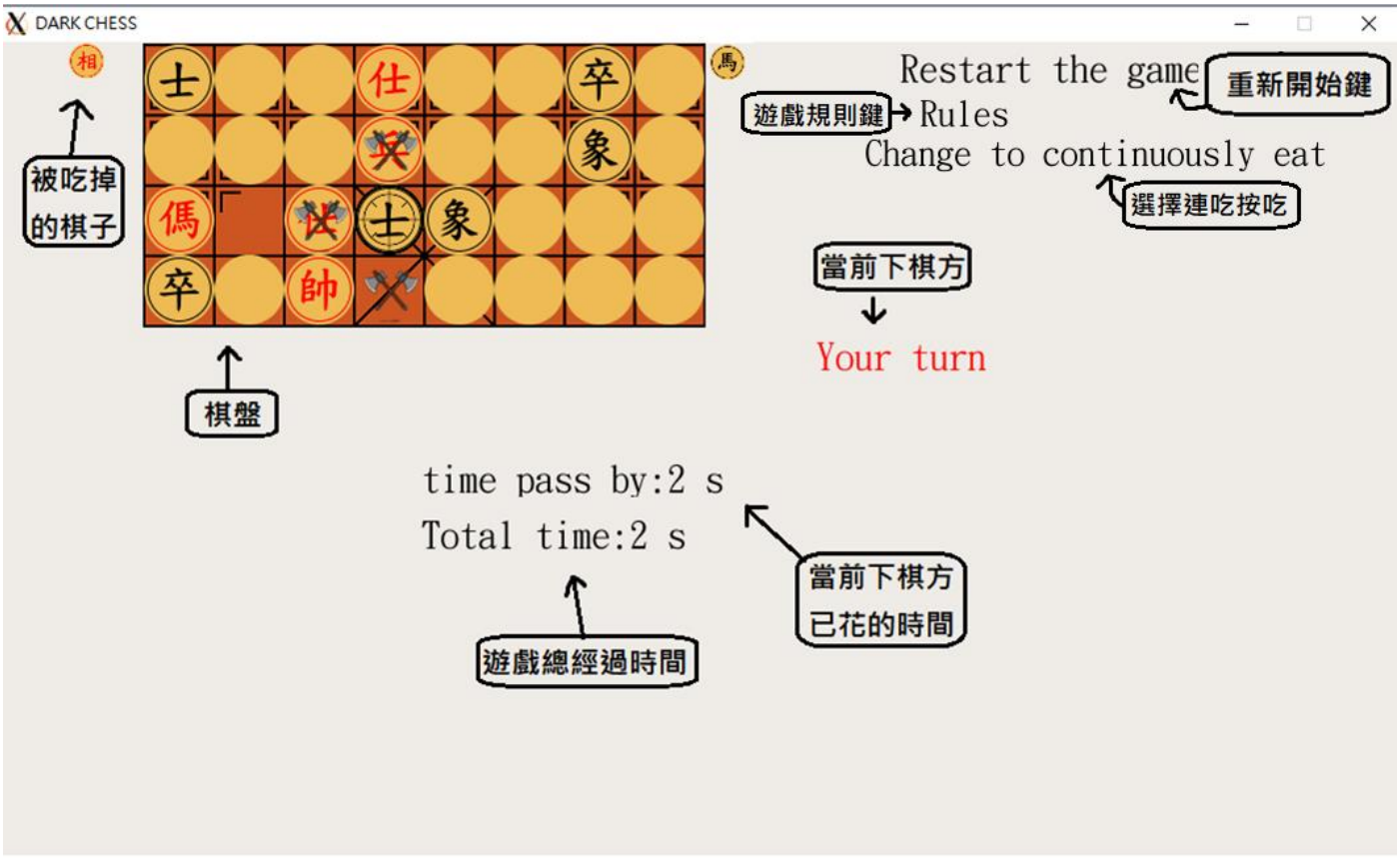
How to compile and run my program:

1. qmake-qt5 -project
2. qmake-qt5 QT+=widgets & QT+=gui
3. make (Do it if you just modified your program. If not, this step is no need.)
4. Open a new terminal -> startxwin
5. Back to the original terminal -> ./test.exe

Explanation of the game interface:



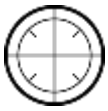
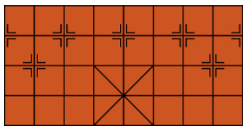
Bonus options

1. 可讓玩家選擇重新開始遊戲
   相比原有 bonus 較容易實作，故加分幅度較少 (up to 2 points)
2. 選定一棋子時，顯示此棋子可移動過去的位置或可吃掉的棋子
   可作為一加分條件 (up to 5%)
3. Add eat piece animation (吃子動畫) and winning animation (up to 5 points)
4. 按 Rules 鍵可顯示遊戲規則
5. 連吃未完全成功，會有 bug

Note:

連吃的部分沒寫成功, 可以連吃, 但吃到最後會有 bug。
其它基本要求都寫好了

Images & Gifs be used

棋子們	**All chesses**
選中棋子時,印在上面	
選中一棋子時,印在可移動過去或吃過去的位子	
某方超過 15 秒未下棋	
紅方勝利	**Red WIN!!**
黑方勝利	**black WIN!!**
吃子時跑出	
棋盤	
遊戲規則	The piece's move/eat should conform to rules ➢將(帥) > 士(仕) > 象(相) > 車(俥) > 馬(傌) > 砲(炮) > 卒(兵) ➢卒(兵) > 將(帥) ➢砲(炮) can eat anyone if between two pieces has exactly one piece ➢Do not move opponent's pieces

Codes

Mainwindow.h

public:

```
QImage chess[15];     //0:back   1~7:由小到大黑棋 8~14:由小到大紅棋

QImage circle_frame,To_Move;                  //載入「選中棋子」  、「可移動位置」  之圖
int dark=1;     //若 dark 為 1,當下就是 invisbly eat, 若為 0 就是 continuisly eat
int num_can_eat=0;          //在 dark 為 1 時, 判斷每次選棋時可以吃子的數量
int a;          //存滑鼠點擊時之 x 座標
int b;          //存滑鼠點擊時之 y 座標
int tie=0;                  //紀錄累積回合雙方皆沒吃子
int press_x=99,press_y=99;              //儲存點擊位置
int turn=99;                            //turn%2=0:紅方   1:黑方
int warning=0;                          //若有無效步數出現,提醒玩家
int AI_turn;                            //turn%2==AI_turn 即輪到電腦下棋
int turn_dicision=0;                    //決定整場第一個下棋者之顏色
int ID_ini[4][8];                       //最初棋位,不隨遊戲進行而變
int ID_status[4][8]={0};          //當前棋位,隨遊戲進行而變，若值=-1 :沒棋、0 : back、else :黑 or 紅棋
int movable[4][8]={0};        //特殊用途的 array, 若 movable[i][j]值為 99, 代表 movable[i][j]是可以選的
QVector<int> eaten[2];          //記錄被吃掉的棋子, eaten[0]存紅方的棋, eaten[1]存黑方的棋
MainWindow(QWidget *parent = 0);
void ini_array();                       //初始化 ID_ini[4][8]
void can_move_eat_or_not(int i,int j,int kind);        //判斷下個步驟可吃或可移動的位子
void update_to_move_array();     //將 movable[i][j]=ID_status[i][j];
void eat_or_move(int eat_x,int eat_y,int eat_ID,int eaten_x,int eaten_y,int eaten_ID);   //執行吃或移動
void AI();                                  //AI 執行函數
int time_turn=0;                            //記錄一方下棋時間
int int_sum_time=0;                         //記錄總下棋時間
QTimer *Timer;                          //計時器
QLabel *player_time;                    //當前下棋方已花的時間
QLabel *sum_time;                       //整個時間
QLabel *timeout;                        //超時
QPushButton *start;                     //按下即重新開始遊戲
QPushButton *help;                      //顯示遊戲規則
```

protected:

```
void draw_plate(QPainter &B);        //畫出棋盤
void draw_chess(QPainter &B);        //畫出棋子
void paintEvent(QPaintEvent *event);
void whose_turn(QPainter &B);     //印出當前輪到誰
```

```
void show_warning(QPainter &C);          //若玩家點錯棋子，秀出警告
void show_gameover(QPainter &C);         //遊戲結束，秀出贏家
void show_tie(QPainter &C);              //若 tie==15，即和局
void mousePressEvent(QMouseEvent *ev);
```

public slots:

```
void counting_slots();     //作為計時函數
void start_slots();        //遊戲開始
void help_slots();   //遊戲規則
void isdark_slots();   //選擇連吃或不連吃
```

main.cpp

```
int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    MainWindow w;
    w.ini_array();                  //一開始就先初始化亮棋盤
    w.show();
    return a.exec();
}
```

MainWindow.cpp

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent):

```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
    setFixedSize(1000,600);
    this->setWindowTitle("DARK CHESS");
```

打開一大小為1000*600視窗
並將此視窗取名為DARK CHESS

```
    QFont ft1;
    ft1.setPointSize(20);
```

設定字型大小, 後面會用到

```
    player_time=new QLabel("Player time:0",this);
    player_time->setGeometry(300,300,500,30);
    player_time->setFont(ft1);
```

創立一QLabel負責印當前玩家已花時間

```
    sum_time=new QLabel("Total time: 00:00:00",this);
    sum_time->setGeometry(300,340,500,30);
    sum_time->setFont(ft1);
```

創立一QLabel負責印遊戲總經過時間

```
    timeout=new QLabel("",this);
    timeout->setGeometry(300,400,300,150);
```

創立一Qlabel負責印超時通知

```
    help=new QPushButton("Rules",this);
    help->setGeometry(600,30,200,30);
    help->setFlat(1);
    help->setFont(ft1);
```

創立一Qpushbottom負責顯示遊戲規則

```
    start=new QPushButton("Restart the game",this);
    start->setGeometry(600,5,300,30);
    start->setFont(ft1);
    start->setFlat(1);
```

創立一Qpushbottom負責重新開啟遊戲的事

```
    connect(help,SIGNAL(clicked(bool)),this,SLOT(help_slots()));
    connect(Timer,SIGNAL(timeout()),this,SLOT(counting_slots()));
    connect(start,SIGNAL(clicked(bool)),this,SLOT(start_slots()));
```

遊戲規則觸發事件
計時事件
重啟遊戲事件

```
    setMouseTracking(true);
    setAcceptDrops(true);
    Timer =new QTimer(this);
```

創立計時器

```
}
```

後來在此函數裡還有加上 connect(isdark,SIGNAL(clicked(bool)),this,SLOT(isdark_slots()));
負責處理選擇連吃或暗吃

void MainWindow::mousePressEvent(QMouseEvent *ev)

```
void MainWindow::mousePressEvent(QMouseEvent *ev) {    此函數為玩家最主要使用到的函數!!
  if (ev->button() == Qt::LeftButton) {
    a=ev->x();
    b=ev->y();                                          將點到的值當作a跟b
    if(101<=a&&501>=a&&1<=b&&201>=b){                   只在點選棋盤內的區域時才執行以下動作
      for(int i=0;i<4;i++){
        for(int j=0;j<8;j++){                           跑迴圈找出a跟b對應到的座標, i, j 為座標化後的值
          if ((((101+50*j)<a)&&(a<(151+50*j))&&((1+50*i)<b)&&(b<(51+50*i)))){   Ex : i=1, j=2 代表點選到棋盤第三行第二列的棋子
            if(press_x==i&&press_y==j){
              update_to_move_array();
              press_x=99;                               press_x與press_y為第一次點擊的棋子位置(前一次的i, j保存下來的)
              press_y=99;                               這次的i, j若與第一次的位子一樣, 代表重複點擊棋子
              this->repaint();                          將之還原上一個步驟(movable還原成當前棋盤樣子)
              return;
            }
            if(movable[i][j]==99){                      若已點選一棋子且再點到的棋子是有效的(可吃或可移動)
              if(dark==1){                              若當下為invisibly eat (這個模式我有寫成功)
                eat_or_move(press_x,press_y,ID_status[press_x][press_y],i,j,ID_status[i][j]);
                press_x=99;                             若第二次所選的位子為前一次所選的棋子可以移動過去或吃過去的
                press_y=99;                             就執行eat_or_move, 並把press_x,press_y還原, 防止誤會
                time_turn=0;                            同時因換AI下棋,所以重新計時
                this->repaint();
                AI();
                return;
              }
              if(dark==0){                              若當下為continuosly eat (沒寫成功,成功一半但有bug)
                int num_can_eat=0;
                for(int m=0;m<4;m++){
                  for(int n=0;n<8;n++){                 計算能吃的數量
                    if(movable[m][n]==99&&ID_status[m][n]>0)
                      num_can_eat++;
                  }
                }
                if(num_can_eat==0){
                  press_x=99;                           若已經沒東西吃了,就換AI下棋
                  press_y=99;
                  time_turn=0;
                  turn++;
                  this->repaint();
                  AI();
                  return;
                }
                else{
                  if(ID_status[i][j]>0){
                    eat_or_move(press_x,press_y,ID_status[press_x][press_y],i,j,ID_status[i][j]);
                    //update_to_move_array();
                    turn--;                             如果有棋子可以吃就選擇要吃誰
                    time_turn=0;                        因為eat_or_move裡有turn++
                    can_move_eat_or_not(i,j,ID_status[i][j]);   但因為是連吃模式,吃後還沒換人
                    press_x=i;                          所以有turn--
                    press_y=j;
                    this->repaint();
                  }
                  else{
                    turn++;                             若有得吃,卻選到不合法的棋位
                    AI();                               直接換AI
                    return;
                  }
                }
              }
            }
          }
        }
      }
    }
  }
```

```
else if(ID_status[i][j]==0){
    ID_status[i][j]=ID_ini[i][j];
    update_to_move_array();
    if(turn_dicision==0){
        if (ID_status[i][j]>7&&ID_status[i][j]<15){
            turn=0;
            AI_turn=!turn;
            }
        else{
            turn=1;
            AI_turn=!turn;
        }
        turn_dicision++;
        Timer->start(1000);
    }
    turn++;
    time_turn=0;
    press_x=99;
    press_y=99;
    this->repaint();
    AI();
    return;
}
```

若選中背面棋, 則將ID_ini的值給到ID_status
並將movable還原(執行update_to_move_array())

判斷第一位玩家是代表哪一方的棋子
翻到紅色代表玩家是紅方, 電腦(AI)則為黑方, 反之
一點擊, 遊戲就開始計時
因一開始一定是背面 故放在 if(ID_status[i][j]==0)的條件框框裡
為了只在遊戲一開始進行, 所以把turn_dicision++
之後程式就不會再跑進這個小方框

選完就換下一位(即AI), 並重新計時
並把press_x, press_y還原成99

```
else if(ID_status[i][j]==-1){
    update_to_move_array();
    press_x=99;
    press_y=99;
    warning=1;
    this->repaint();
    return ;
}
```

若選中棋子為背面(ID_status==-1)
還原movable
press_x, press_y還原成99
warning給1, 讓程式執行警告事件

```
else if(turn%2==0&&(ID_status[i][j]>7&&ID_status[i][j]<15)) {
    update_to_move_array();
    can_move_eat_or_not(i,j,ID_status[i][j]);
    press_x=i;
    press_y=j;
    this->repaint();
}
else if (turn%2==1&&(ID_status[i][j]>0&&ID_status[i][j]<8)) {
    update_to_move_array();
    can_move_eat_or_not(i,j,ID_status[i][j]);
    press_x=i;
    press_y=j;
    this->repaint();
}
```

若當前為紅方(黑方)下棋, 且選中之棋子確實是紅方(黑方)
為確保錯亂, 仍先還原update_to_move_array
再執行can_move_eat_or_not
以判斷哪些位子可以讓當前選中的棋子移動過去或吃過去
並把當前被選中的棋子位子紀錄在press_x,press_y

```
else if((turn%2==1&&(ID_status[i][j]>7&&ID_status[i][j]<15))||(turn%2==0&&(ID_status[i][j]>0&&ID_status[i][j]<8))){
    warning=1;
    this->repaint();
}
        }
    }
}
```

若以上條件都沒符合
代表選錯了
使warning=1以執行警告尋錫

void MainWindow::AI()

```
void MainWindow::AI(){   此函數為AI用到的函數!
    if(end==1) return;
    int asd=1;
    while(asd){
        srand( time(NULL) );
        int com = rand()%32;
        int i=com/8;
        int j=com%8;
```

讓AI隨機選棋

當某方棋子被吃完時,會執行end=1

```
        if(AI_turn==0&&eaten[0].size()>8){
            for(int m=0;m<4;m++){
                for(int n=0;n<8;n++){
                    if(ID_status[m][n]>7&&ID_status[m][n]<15){
                        i=m;
                        j=n;
                    }
                }
            }
        }
        if(AI_turn==1&&eaten[1].size()>8){
            for(int m=0;m<4;m++){
                for(int n=0;n<8;n++){
                    if(ID_status[m][n]>0&&ID_status[m][n]<8){
                        i=m;
                        j=n;
                    }
                }
            }
        }
```

AI方的棋子剩沒幾顆時
(這裡設定被吃掉的>8顆,即剩不到8顆棋)
若使用隨機讓AI選到自己的棋子會跑很久
故強制讓AI去選棋盤最下面最右邊的自己的棋子

```
        for(int m=0;m<4;m++){
            for(int n=0;n<8;n++){
                if(movable[m][n]==99){
                    eat_or_move(press_x,press_y,ID_status[press_x][press_y],m,n,ID_status[m][n]);
                    press_x=99;
                    press_y=99;
                    time_turn=0;
                    this->repaint();
                    asd=0;
                }
            }
        }
```

當已選定一棋子後
直接強制AI去選可以走過去或吃過去的位子
並將press_x, press_y還原成99
輪到下一位,故重新計時
並把asd改為0, 為了不執行下面的程式

```
        if(!asd) break;
```

若剛剛AI已執行移動或吃的動作
會將asd=1,以下程式就不執行

```
        if(ID_status[i][j]==0){
            ID_status[i][j]=ID_ini[i][j];
            update_to_move_array();
            turn++;
            time_turn=0;
            press_x=99;
            press_y=99;
            this->repaint();
            asd=0;
        }
```

若AI翻到背面, 就翻牌
跟我方翻到背面執行一樣的事

```
else if(ID_status[i][j]==-1){
            update_to_move_array();
            press_x=99;
            press_y=99;
            warning=1;
            this->repaint();
        }
        else if(turn%2==0&&(ID_status[i][j]>7&&ID_status[i][j]<15)) {
            update_to_move_array();
            can_move_eat_or_not(i,j,ID_status[i][j]);
            press_x=i;
            press_y=j;
            this->repaint();
        }
        else if (turn%2==1&&(ID_status[i][j]>0&&ID_status[i][j]<8)) {
            update_to_move_array();
            can_move_eat_or_not(i,j,ID_status[i][j]);
            press_x=i;
            press_y=j;
            this->repaint();
        }
        else if((turn%2==1&&(ID_status[i][j]>7&&ID_status[i][j]<15))||(turn%2==0&&(ID_status[i][j]>0&&ID_status[i][j]<8))){
            warning=1;
            this->repaint();
        }
    }
}
```

這裡都跟前面我方下棋時一樣!!

void MainWindow::whose_turn(QPainter &B)

```
void MainWindow::whose_turn(QPainter &B){
    if (turn==99){
        B.drawText(300,500,QString("Time start. Please choose a chess!!!"));
    }
    else{
        if(turn%2==AI_turn){
            B.drawText(600,180,QString("Computer's turn"));
        }
        else{
            B.drawText(600,180,QString("Your turn"));
        }
    }
}
```

若turn==99 (即初始化的值)
就提醒玩家選棋子

再來判斷目前輪到誰
並印出輪到誰的資訊

void MainWindow::ini_array()

```
void MainWindow::ini_array(){
    ID_ini[0][0]=6;
    ID_ini[0][1]=8;
    ID_ini[0][2]=2;
    ID_ini[0][3]=13;
    ID_ini[0][4]=4;
    ID_ini[0][5]=9;
    ID_ini[0][6]=1;
    ID_ini[0][7]=7;
    ID_ini[1][0]=8;
    ID_ini[1][1]=1;
    ID_ini[1][2]=12;
    ID_ini[1][3]=8;
    ID_ini[1][4]=2;
    ID_ini[1][5]=1;
    ID_ini[1][6]=5;
    ID_ini[1][7]=9;
    ID_ini[2][0]=3;
    ID_ini[2][1]=10;
    ID_ini[2][2]=13;
    ID_ini[2][3]=5;
    ID_ini[2][4]=12;
    ID_ini[2][5]=10;
    ID_ini[2][6]=4;
    ID_ini[2][7]=8;
    ID_ini[3][0]=1;
    ID_ini[3][1]=8;
    ID_ini[3][2]=14;
    ID_ini[3][3]=6;
    ID_ini[3][4]=11;
    ID_ini[3][5]=1;
    ID_ini[3][6]=3;
    ID_ini[3][7]=11;
    update_to_move_array();
}
```

初始化亮的棋盤

void MainWindow::update_to_move_array()

```
void MainWindow::update_to_move_array(){
    for(int i=0;i<4;i++){
        for(int j=0;j<8;j++){                    將 movable 變成還原成當前棋盤的樣子
            movable[i][j]=ID_status[i][j];
        }
    }
}
```

void MainWindow::show_warning(QPainter &C)

```
void MainWindow::show_warning(QPainter &C){
    if(warning==1){                              當 warning==1 秀出選錯棋子的警告
        C.drawText(0,450,QString("Warning!!!!It's an illegal step!!!!!!!!!"));
        warning=0;
    }
}
```

void MainWindow::can_move_eat_or_not(int i,int j,int kind)

此函數程式碼過長，在此簡介：
針對被選中的棋子去判斷他接下來可以移動去或吃過去的所有可能位子
並將這些位子的 movable[i][j]賦予 99 的值
以供 eat_or_move()去做移動或吃的動作

void MainWindow::eat_or_move(int eat_x,int eat_y,int eat_ID,int eaten_x,int eaten_y,int eaten_ID)

```
void MainWindow::eat_or_move(int eat_x,int eat_y,int eat_ID,int eaten_x,int eaten_y,int eaten_ID){
    QLabel *label=new QLabel();
    QMovie *movie = new QMovie("dataset/eat.gif");
    label->setMovie(movie);
```

宣告一Qlabel, 並載入吃子的動畫

```
    turn++;
    time_turn++;
```

若能執行到這個程式代表有人被吃了,吃完就要換人

```
    if(eaten_ID!=-1){
        if(eaten_ID>7&&eaten_ID<15) eaten[0].push_back(eaten_ID);
        else eaten[1].push_back(eaten_ID);
    }
```

把被吃掉的棋子存到eaten這個vector
eaten[0]存紅色的
eaten[1]存黑色的

```
    if(eaten_ID==-1) tie++;
    else tie=0;
```

若只是移動, 沒有人被吃掉
就累積tie (若tie==15會顯示和局)

```
    if(eaten_ID>0){
        label->show();
        label->setGeometry(-100,230,300,300);
        movie->start();
    }
```

若有人被吃掉
就打開吃子的動畫

```
    ID_status[eat_x][eat_y]=-1;
    ID_status[eaten_x][eaten_y]=eat_ID;
    update_to_move_array();
}
```

吃人的位子變成空位(-1)
被吃的就變成吃人的

void MainWindow::counting_slots()

```
void MainWindow::counting_slots(){
    time_turn++;
    int_sum_time++;
```
每經過一秒就++

```
    QString print_turn_time;
    print_turn_time=QString("time pass by:%2 s").arg(time_turn);
    player_time->setText(print_turn_time);
```
印出當前玩家已花時

```
    QString print_total_time;
    print_total_time=QString("Total time:%1 s").arg(int_sum_time);
    sum_time->setText(print_total_time);
```
印出遊戲總經過時間

```
    if(time_turn==15){
        QLabel *label=new QLabel();
        QMovie *timeup= new QMovie("dataset/timeup.gif");
        label->setMovie( timeup );
        label->show();
        label->setGeometry(100,230,480,270);
        timeup->start();
        Timer->stop();
    }
    else return;
```
若某方下棋超過15秒未有動作
就印出time's up的動畫

```
}
```

void MainWindow::isdark_slots()

```
void MainWindow::isdark_slots(){
    if (dark==1){
        isdark->setText("Change to invisibly eat ");
        this->repaint();
        dark=0;
    }
    else{
        isdark->setText("Change to continuously eat ");
        this->repaint();
        dark=1;
    }
}
```
負責決定連吃或暗吃

void MainWindow::start_slots()

```
void MainWindow::start_slots(){
    start->setText("Restart the game");
    press_x=99;
    press_y=99;
    turn=99;
    warning=0;
    turn_dicision=0;
    eaten[0].clear();
    eaten[1].clear();
    //把所有棋子翻回背面
    for(int i=0;i<4;i++){
        for(int j=0;j<8;j++){
            ID_status[i][j]=0;
        }
    }
    time_turn=0;
    int_sum_time=0;
    this->repaint();
    Timer->start(1000);
}
```

重新開始時用到

在右邊印出Restart the game

按下即重新開始遊戲

所有東西都要初始化

void MainWindow::help_slots()

```
void MainWindow::help_slots(){
    QLabel *label=new QLabel();
    QMovie *Help = new QMovie("dataset/help.gif");
    label->setMovie(Help);
    label->show();
    label->setGeometry(100,230,1138,273);
    Help->start();
}
```

印出遊戲規則

void MainWindow::paintEvent(QPaintEvent *event)

```
void MainWindow::paintEvent(QPaintEvent *event) {
    QFont ft;
    ft.setPointSize(20);
    QPainter B(this); // Create a painter on this widget
    QPainter C(this);
    QPen pen,pen1; // Initialize a pen
    B.setRenderHint(QPainter::Antialiasing, true); // Set antialiasing
    pen.setColor(Qt::red);
    pen1.setColor(Qt::blue);
    B.setFont(ft);
    C.setFont(ft);
    C.setPen(pen);
    B.setPen(pen1);
    draw_plate(B);
    draw_chess(B);
    whose_turn(B);
    show_warning(C);
    show_gameover(C);
    show_tie(C);
}
```

印刷大雜匯

void MainWindow::draw_plate

```
void MainWindow::draw_plate(QPainter &B) {
    QImage image;
    image.load("./dataset/plate.png");
    B.drawImage(100,0,image);
}
```

印出棋盤格

void MainWindow::show_gameover

```
void MainWindow::show_gameover(QPainter &C){
    QLabel *label=new QLabel();
    QMovie *redwin = new QMovie("dataset/redwin.gif");
    QMovie *blackwin = new QMovie("dataset/blackwin.gif");
```

創立QMovie去存紅方跟黑方獲勝的動畫

```
    if(eaten[0].size()==16){
        label->setMovie(blackwin);
        label->show();
        label->setGeometry(100,230,400,400);
        blackwin->start();
        C.drawText(300,400,QString("Black WIN!!!!!!"));
        eaten[0].clear();
        end=1;
    }
    if(eaten[1].size()==16){
        label->setMovie(redwin);
        label->show();
        label->setGeometry(100,230,400,400);
        redwin->start();
        C.drawText(300,400,QString("RED WIN!!!!!!"));
        eaten[1].clear();
        end=1;
    }
```

只要其中一方棋子被吃完
就印出另一方獲勝的動畫

```
}
```

void MainWindow::show_tie

```
void MainWindow::show_tie(QPainter &C){
    if(tie==15){
        C.drawText(300,400,QString("TIE!!!!!!"));
        Timer->stop();
    }
}
```

秀出和局
和局後停止計時

void MainWindow::draw_chess(QPainter &B)

```
void MainWindow::draw_chess(QPainter &B){
    To_Move.load("./dataset/ToMove");
    circle_frame.load("./dataset/frame");
    chess[0].load("./dataset/back");       //back
    chess[1].load("./dataset/black7");     //卒
    chess[2].load("./dataset/black6");     //砲
    chess[3].load("./dataset/black5");     //黑馬
    chess[4].load("./dataset/black4");     //黑車
    chess[5].load("./dataset/black3");     //象
    chess[6].load("./dataset/black2");     //士
    chess[7].load("./dataset/black1");     //將
    chess[8].load("./dataset/red7");       //兵
    chess[9].load("./dataset/red6");       //炮
    chess[10].load("./dataset/red5");      //黑馬
    chess[11].load("./dataset/red4");      //紅車
    chess[12].load("./dataset/red3");      //相
    chess[13].load("./dataset/red2");      //仕
    chess[14].load("./dataset/red1");      //帥
```

在Qimage chess[]載入棋子

```
    for(int m=0;m<4;m++){
        for(int n=0;n<8;n++){
            if(ID_status[m][n]!=-1)
                B.drawImage(101+50*n,1+50*m,chess[ID_status[m][n]]);
            if(movable[m][n]==99){
                B.drawImage(101+50*n,1+50*m,To_Move);
            }
    }
    }
```

當下棋盤是甚麼棋子
就印甚麼
若movable==99.就代表可以移動過去
就印 

```
    for(int i=0;i<2;i++){
        for(int j=0;j<eaten[i].size();j++){
            if(i==0) B.drawImage(QRect(70,1+26*j,25,25),chess[eaten[i][j]]);
            else B.drawImage(QRect(505,1+26*j,25,25),chess[eaten[i][j]]);
        }
    }
```

在兩邊印出被吃掉的棋子

```
    B.drawImage(101+50*press_y,1+50*press_x,circle_frame);
}
```

當棋子被選中
印出