南开大学 计算机大类

姓名：管湘宇

学号：2311358

班级：网安学院模拟2.2班

2024年5月12日

高级语言程序设计

实验报告

目录

[高级语言程序设计大作业实验报告 2](#_Toc165637585)

[**一.** **作业题目** 2](#_Toc165637586)

[**二.** **开发软件** 2](#_Toc165637587)

[**三.** **课题要求** 2](#_Toc165637588)

[**四.** **主要流程** 3](#_Toc165637589)

[**1． 整体流程** 3](#_Toc165637590)

[**2． 主要棋子的函数以及一些重要函数** 3](#_Toc165637591)

[**五.** **单元测试** 12](#_Toc165637592)

[测试结果 12](#_Toc165637593)

[**六.** **收获** 12](#_Toc165637594)

高级语言程序设计大作业实验报告

1. **作业题目**

实现国际象棋的基本功能

1. **开发软件**

Qt5.14.0

1. **课题要求**
2. 面向对象。
3. 单元测试。
4. 模型部分
5. 验证
6. **主要流程**
   1. **整体流程**

实现思路：

首先绘制棋盘，定义棋子作为类对象，定义棋子的走棋规则作为函数，在成员函数中提供各种棋子走棋基本规则，利用鼠标点击位置判定行棋路线以及是否合法，合法的判断条件出来每个棋子的走棋函数以外，还包括是否有棋子阻挡，吃子是否同色，王是否正在被将等，若合法更新棋盘上棋子状态，每次更新判定是否棋局结束

* 1. **主要棋子的函数以及一些重要函数**

**(1).车**

bool setting::**moveRook**(int n1, int n2, int xp, int yp){

if (xp==myChess[n1].x&&yp==myChess[n1].y)

return false;

if (exist(xp,yp)&&sameColor(n1,n2))

return false;

if (middleChess(n1,xp,yp))

return false;

if (xp!=myChess[n1].x&&yp!=myChess[n1].y)

return false;

return true;

实现判断车是否沿着直线走的情况下目的地为空或异色棋子，及中间是否有阻挡。

**(2).马**

bool setting::**moveKnight**(int n1, int n2, int xp, int yp){

if (xp==myChess[n1].x&&yp==myChess[n1].y)

return false;

if (exist(xp,yp)&&sameColor(n1,n2))

return false;

if (abs(xp-myChess[n1].x)==1&&abs(yp-myChess[n1].y)==2)

return true;

else if (abs(xp-myChess[n1].x)==2&&abs(yp-myChess[n1].y)==1)

return true;

return false;

}//马的走法

实现判断马是否沿着“日”走的情况下目的地为空或异色棋子，马不会被阻挡，无需判断阻挡情况

**(3).象**

bool setting::**moveBishop**(int n1, int n2, int xp, int yp){

if (xp==myChess[n1].x&&yp==myChess[n1].y)

return false;

if (exist(xp,yp)&&sameColor(n1,n2))

return false;

if (middleChess(n1,xp,yp))

return false;

if (abs(xp-myChess[n1].x)==abs(yp-myChess[n1].y))

return true;

return false;

}//象的走法

实现判断象是否沿着斜线走的情况下目的地为空或异色棋子，且不被阻挡

**(4).后**

bool setting::**moveQueen**(int n1, int n2, int xp, int yp){

if (xp==myChess[n1].x&&yp==myChess[n1].y)

return false;

if (exist(xp,yp)&&sameColor(n1,n2))

return false;

if (middleChess(n1,xp,yp))

return false;

if (xp==myChess[n1].x||yp==myChess[n1].y)

return true;

else if (abs(xp-myChess[n1].x)==abs(yp-myChess[n1].y))

return true;

return false;

}//后的走法

实现判断后是否沿着斜线或直线走的情况下目的地为空或异色棋子，且不被阻挡

**(5).王**

bool setting::**moveKing**(int n1, int n2, int xp, int yp){

if (xp==myChess[n1].x&&yp==myChess[n1].y)

return false;

if (exist(xp,yp)&&sameColor(n1,n2))

return false;

if (abs(xp-myChess[n1].x)==1&&abs(yp-myChess[n1].y)==1)

return true;

if (xp==myChess[n1].x&&abs(yp-myChess[n1].y)==1)

return true;

if (abs(xp-myChess[n1].x)==1&&yp==myChess[n1].y)

return true;

if (yp==myChess[n1].y&&abs(xp-myChess[n1].x)==2){ //castling

if (myChess[n1].white&&!whiteCastled)

return castling(n1,xp,yp);

else if (!myChess[n1].white&&!blackCastled)

return castling(n1,xp,yp);

else

return false;

}

return false;

}

//王的走法

实现判断王是否沿着斜线或直线走一格的情况下目的地为空或异色棋子，其中涉及到易位的特殊规则实现如下

bool whiteCastled=false,blackCastled=false;

bool setting::**castling**(int n, int xp, int yp){

if (myChess[n].white){

if (xp>myChess[n].x){ //short castling

if (!myChess[7].moved&&!myChess[n].moved){

if (!middleChess(n,myChess[7].x,myChess[7].y)){

for (int i=16;i<32;i++){

if (threat(i,myChess[n].x,myChess[n].y))

return false;

if (threat(i,myChess[n].x+1,myChess[n].y))

return false;

if (threat(i,myChess[n].x+2,myChess[n].y))

return false;

}

}

}

}

else{ //long castling

if (!myChess[0].moved&&!myChess[n].moved){

if (!middleChess(n,myChess[0].x,myChess[0].y)){

for (int i=16;i<32;i++){

if (threat(i,myChess[n].x,myChess[n].y))

return false;

if (threat(i,myChess[n].x-1,myChess[n].y))

return false;

if (threat(i,myChess[n].x-2,myChess[n].y))

return false;

}

}

}

}

whiteCastled=true;

}

else{

if (xp>myChess[n].x){ //short castling

if (!myChess[23].moved&&!myChess[n].moved){

if (!middleChess(n,myChess[23].x,myChess[23].y)){

for (int i=0;i<16;i++){

if (threat(i,myChess[n].x,myChess[n].y))

return false;

if (threat(i,myChess[n].x+1,myChess[n].y))

return false;

if (threat(i,myChess[n].x+2,myChess[n].y))

return false;

}

}

}

}

else{ //long castling

if (!myChess[16].moved&&!myChess[n].moved){

if (!middleChess(n,myChess[16].x,myChess[16].y)){

for (int i=0;i<16;i++){

if (threat(i,myChess[n].x,myChess[n].y))

return false;

if (threat(i,myChess[n].x-1,myChess[n].y))

return false;

if (threat(i,myChess[n].x-2,myChess[n].y))

return false;

}

}

}

}

blackCastled=true;

}

return true;

}

其中涉及王是否被将的问题threat函数将在下一部分重要函数中体现

**(6).兵**

bool setting::**movePawn**(int n1, int n2, int xp, int yp){

if (xp==myChess[n1].x&&yp==myChess[n1].y)

return false;

if (n2!=-1&&sameColor(n1,n2))

return false;

if (n2!=-1&&myChess[n1].x==xp&&abs(myChess[n1].y-yp)==1)

return false;

if (xp==myChess[n1].x){

if (myChess[n1].white){

if ((myChess[n1].y-yp)==2&&!myChess[n1].moved)

return true;

else if (myChess[n1].y-yp==1)

return true;

}

else {

if ((yp-myChess[n1].y)==2&&!myChess[n1].moved)

return true;

else if (yp-myChess[n1].y==1)

return true;

}

}

else if (abs(xp-myChess[n1].x)==1){

if (myChess[n1].white&&(myChess[n1].y-yp==1)){

if (n2!=-1)

return true;

else if (enPassant(n1,xp,yp))

return true;

}

else if (!myChess[n1].white&&(yp-myChess[n1].y==1)){

if (n2!=-1)

return true;

else if (enPassant(n1,xp,yp))

return true;

}

}

return false;

}

实现兵只能向前一步或初始时向前两步以及斜吃的操作，吃过路兵函数如下bool setting::**enPassant**(int n1, int xp, int yp){

int n2=-1;

if (myChess[n1].white){

n2=getNum(xp,yp+1);

if (n2>=24&&n2<32){

if (myChess[n2].y==3&&myChess[n2].moved==1){

myChess[n2].dead=true;

myChess[n2].x=-1;

myChess[n2].y=-1;

label[n2]->clear();

black--;

return true;

}

else

return false;

}

else

return false;

}

else{

n2=getNum(xp,yp-1);

if (n2>=8&&n2<16){

if (myChess[n2].y==4&&myChess[n2].moved==1){

myChess[n2].dead=true;

myChess[n2].x=-1;

myChess[n2].y=-1;

label[n2]->clear();

white--;

return true;

}

else

return false;

}

else

return false;

}

}//吃过路兵

**(7).判断行棋路线上是否有棋子**

bool setting::**middleChess**(int n, int xp, int yp){

if (myChess[n].x==xp){ //in the same column

if (yp>myChess[n].y){

for (int i=myChess[n].y+1;i<yp;i++)

if (exist(xp,i))

return true;

}

else {

for (int i=yp+1;i<myChess[n].y;i++)

if (exist(xp,i))

return true;

}

}

else if (myChess[n].y==yp){ //in the same row

if (xp>myChess[n].x){

for (int i=myChess[n].x+1;i<xp;i++)

if (exist(i,yp))

return true;

}

else {

for (int i=xp+1;i<myChess[n].x;i++)

if (exist(i,yp))

return true;

}

}

else{ //diagonal

if (xp>myChess[n].x){

if (yp>myChess[n].y){

int j=myChess[n].y+1;

for (int i=myChess[n].x+1;i<xp;i++)

if (exist(i,j))

return true;

else

j++;

}

else{

int j=myChess[n].y-1;

for (int i=myChess[n].x+1;i<xp;i++)

if (exist(i,j))

return true;

else

j--;

}

}

else{

if (yp>myChess[n].y){

int j=myChess[n].y+1;

for (int i=myChess[n].x-1;i>xp;i--)

if (exist(i,j))

return true;

else

j++;

}

else{

int j=myChess[n].y-1;

for (int i=myChess[n].x-1;i>xp;i--)

if (exist(i,j))

return true;

else

j--;

}

}

}

return false;

}//判断行棋路径上有无子的阻挡

**(7).判断王是否正在被将**

bool setting::**threat**(int n, int xk, int yk){

if (myChess[n].dead)

return false;

if (myChess[n].chessType==0){ //rook

if (myChess[n].x==xk||myChess[n].y==yk){

if (middleChess(n,xk,yk))

return false;

else

return true;

}

else

return false;

}

else if (myChess[n].chessType==1){ //knight

if (abs(xk-myChess[n].x)==1&&abs(yk-myChess[n].y)==2)

return true;

else if (abs(xk-myChess[n].x)==2&&abs(yk-myChess[n].y)==1)

return true;

else

return false;

}

else if (myChess[n].chessType==2){ //bishop

if (abs(xk-myChess[n].x)==abs(yk-myChess[n].y)){

if (middleChess(n,xk,yk))

return false;

else

return true;

}

else

return false;

}

else if (myChess[n].chessType==3){ //queen

if (myChess[n].x==xk||myChess[n].y==yk){

if (middleChess(n,xk,yk))

return false;

else

return true;

}

else if (abs(xk-myChess[n].x)==abs(yk-myChess[n].y)){

if (middleChess(n,xk,yk))

return false;

else

return true;

}

else

return false;

}

else if (myChess[n].chessType==4){ //king

if (abs(myChess[n].x-xk)==1&&abs(myChess[n].y-yk)==1)

return true;

else if (myChess[n].x==xk&&abs(myChess[n].y-yk)==1)

return true;

else if (abs(myChess[n].x-xk)==1&&myChess[n].y==yk)

return true;

else

return false;

}

else if (myChess[n].chessType==5){ //pawn

if (myChess[n].white){

if ((myChess[n].y-yk)==1&&abs(myChess[n].x-xk)==1)

return true;

else

return false;

}

else{

if ((yk-myChess[n].y)==1&&abs(xk-myChess[n].x)==1)

return true;

else

return false;

}

}

}

//判断王是否正在被将

1. **单元测试**

测试案例定义

表 1：测试案例

|  |  |  |
| --- | --- | --- |
| **输入** | **输出** | **目的** |
| 尝试各种棋子的常规走法 | 仅在正确情况下移动，其他无效运子情况下均报错 | 验证判定走棋是否合法的函数的正确性 |
| 尝试王车易位 | 仅在正确情况下易位 | 验证王车易位函数正确性 |
| 尝试吃过路兵和升变 | 仅能再正确情况下吃过路兵四种升变均可 | 验证吃过路兵和升变函数 |
| 尝试将杀 | 将杀情况下棋局结束，输出胜利一方 | 验证将杀函数 |
| 尝试和棋 | 在简单预设情况下棋局结束和棋 | 验证和棋函数 |

### 测试结果

除了在某些极端情况,除和棋外均通过验证

1. **收获**

学习了qt框架下的各种控件，以及可视化编程的一些方法，学习了国象的走子和胜负判定的一些实现方法