پروژه ی درس معماری برنامه نویسی پیشرفته

ترم ۰۰/۹۹

موضوع پروژه

بازی شطرنج پیشرفته

مدرس

دکتر صنعتی

دانشجو

حمید مومیوند ۹۷۱۲۳۵۸۰۴۳

محسن تقوی ۹۷۱۲۳۵۸۰۰۸

تابستان ۱۴۰۰

فهرست مطالب

مقدمه

در این برنامه قصد داریم بازی شطرنج پیشرفته را با استفاده از رابط گرافیکی پیاده سازی کنیم

این بازی در بسیاری از قوانین با شطرنج عادی مشترک است اما در مواردی ، قوانینی جدید به بازی افزوده شده است مانند شیوه تعیین برنده بازی

این بازی بصورت آفلاین توسط دو کاربر انجام می پذیرد و تعیین شدن برنده نه از روی مات شدن بلکه از روی تعداد امتیازات مشخص می شود.

حالاتی که باعث گرفتن امتیاز در طول بازی میشوند، شامل ۵دسته هستند: تهدید مهره ی حریف، زدن مهره ی حریف، کیش کردن حریف، کیش و مات کردن حریف و نیمه ی دوم سرباز

شرح پروژه

در این پروژه ۳ بخش اصلی داریم :

۱-صفحه بازی

۲- مهره ها

۳-امتیازات بازی

که در ادامه به توضیح هر بخش می پردازیم

- صفحه بازی

همان طور که همه میدانیم صفحه ی شطرنج دارای ٦٤ خانه بصورت ۸*۸ میباشد که مهره های سفید و سیاه در شروع بازی در دو ردیف ابتدایی و انتهایی صفحه رو در روی یک دیگر قرار می گیرند

برای صفحه بازی کلاس **chessboard** را می سازیم که دارای المان های زیر میباشد :

```
class ChessBoard
{
    private:
        Cell *cells[8][8];
        Player1 * pl1;
        Player2 * pl2;
        char plturn='w';
        bool IsAttack;
        bool Twomovement=false;
        sf::RenderWindow * window;
        void ChangeTurn();
       // bool branches(int a,int b,int c,int d);
        std::string switchname(std::string);//ke1e4->ke4e1
    public:
        ChessBoard(sf::RenderWindow *);
        void SetPlNames(std::string,std::string);
        void move(int a,int b);
        bool GameOver();
        void Moveit(int a,int b,int c,int d);
        void attack(int ,int,int,int);
        void movePiece(std::string);
        bool KingCheck(char );
        void draw();
        bool transformator(sf::Vector2f ,sf::Vector2f pos2);
        ~ChessBoard();
};
```

همان طور که در تصویر مشخص است این کلاس دارای یک آرایه ۸*۸ است که هر عضو آن نماینده یک خانه از ٦٤ خانه ی بازی است

برای هر بازیکن نیز یک شئ تعریف میشود که جلوتر به تشریح آن می پردازیم

Plturn یه کاراکتر است که نوبت بازیکن رو مشخص میکند که پیشفرض شروع نوبت با سفید است

این کلاس یک متد سازنده نیز دارد که آرگومان ورودی آن جهت رندر شدن در صفحه sfml است و مهره ها را در پوزیشن مربوطه در صفحه بازی قرار می دهد

Setplnames دو رشته ورودی گرفته و هر رشته را به نام یکی از بازیکن ها انتساب می دهد

Changeturn هم نوبت بازیکنان را تغییر می دهد.

Moveit این تابع ٤ عدد بعنوان ورودی میگیرد که دو عدد اول مختصات مکان اولیه بوده و دو عدد دوم مختصات مکان بعدی خواهد بود سپس پس از جابه جایی مهره، پوینتر مختصات اولیه را صفر می کند

```
if(cells[c][d]->ptr->GetNamad()=='P')
{
    if(cells[c][d]->ptr->GetColor()=='w')
    {
        if(d==4)
        {
            pl1->AddPoint(3);
        }
        if(d==7)
        {
            //queen or bishop or rook or knight
        }
    }
    else
    {
        if(d==3)
        {
            pl2->AddPoint(3);
        }
        if(d==0)
        {
            //queen or bishop or rook or knight
        }
    }
```

سپس برای هر دو مهره سفید و سیاه رنگ بررسی می کند که اگر مهره سرباز باشد و از نصف زمین گذشته باشد ، به بازیکن ۳ امتیاز می دهد و اگر به پایان صفحه رسیده باشد ، به انتخاب خود می تواند به جای سرباز یک وزیر یا رخ یا فیل و یا اسب انتخاب کند.

در ادامه این تابع با استفاده از دو حلقه ی تو در تو تهدید های اطراف را بررسی کرده و در صورت وجود داشتن با تابع getwarn() هشدار را ارسال می کند

```
for(int i=0;i<8;i++)
{
    for(int j=0;j<8;j++)
    {
        if(cells[c][d]->ptr->AreSquaresLegal(c, d, i, j,cells))
        {
            if(cells[c][d]->ptr->GetColor()=='w')
            {
                pl1->AddPoint(cells[i][j]->ptr->GetWarn());
            }
            else
            {
                pl2->AddPoint(cells[i][j]->ptr->GetWarn());
            }
        }
    }
}
```

****

تابع attack()

```
}
void ChessBoard::attack(int a, int b, int c, int d)
{
    if(cells[c][d]->ptr->GetColor()=='w')
    {
        pl2->AddPoint(cells[c][d]->ptr->GetPoint());
        pl1->graveyard.push_back(cells[c][d]->ptr);
    }
    else
    {
        pl1->AddPoint(cells[c][d]->ptr->GetPoint());
        pl2->graveyard.push_back(cells[c][d]->ptr);
    }
    cells[c][d]->ptr->SetInGame(false);
    cells[c][d]->ptr=0;
    Moveit(a,b,c,d);

}
```

این تابع برای هر دو بازیکن بررسی می کند اگر حرکت موجب کشته شدن مهره ای بشود ، امتیاز را به مهاجم داده و مهره کشته شده از بازی حذف می کند.

****

تابع movepiece()

این تابع که میتوان گفت اصلی ترین تابع برای حرکت مهره در صفحه بازی است یک رشته بعنوان ورودی دریافت میکند که حرف اول این رشته نشان دهنده ی مهره ای است که قرار است حرکت کند . حرف دوم ستون و حرف سوم اندیس ستون موقعیت فعلی را نشان داده و حروف ٤ و ٥ ستون و اندیس موقعیت بعدی را نشان می دهد. سپس با تعیین این ستون و ها و اندیس ها و هم چنین نوع مهره ، برای هر رنگ حرکات را انجام میدهد.

****

تابع draw()

```
void ChessBoard::draw()
{
    for(int i=0;i<8;i++)
    {
        for(int j=0;j<8;j++)
        {
            cells[i][j]->ptr->draw();
        }
    }
}
```

این تابع برای رسم سلول های صفحه است .(منظور از سلول هر یک از ٦٤ خانه ی صفحه است)

****

تابع transformator()

این تابع دو پوزیشن از جنس vector۲f که کاربر وارد می کند را گرفته و آن ها را به حروفی که برای تابع movepiece نیاز دارد تبدیل می کند.

```
void ChessBoard::transformator(sf::Vector2f pos1,sf::Vector2f pos2)
{
    char a=0;
    char b=0;
    char c=0;
    char d=0;
    char e=0;
    bool find2=false;
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            if (cells[i][j]->ptr != nullptr && cells[i][j]->ptr->GetTextureSprite().getGlobalBounds().contains(pos1))
            {
                find2=true;
                if(cells[i][j]->ptr->GetColor()==plturn)
                {
                    a=cells[i][j]->ptr->GetNamad();
                    std::cout << "a: " << a << std::endl;
                    c=(char)fabs((j+1)-9) + '0';
                    switch (i+1)
                    {
                        case 1:
                            b='a';
                            break;
                        case 2:
                            b='b';
                            break;
                        case 3:
                            b='c';
                            break;
                        case 4:
                            b='d';
                            break;
                        case 5:
                            b='e';
                            break;
                        case 6:
                            b='f';
                            break;
                        case 7:
                            b='g';
```

```
                    case 7:
                        b='g';
                        break;
                    case 8:
                        b='h';
                        break;
                    default:
                        throw std::runtime_error("cant switch case in transformator in pos1");
                        break;
                }
            }
            else
            {
                throw std::runtime_error("not your turn in transformator");
            }

        }
    }
}
if(!find2)
{
    throw std::runtime_error("not choose a piece in first pos in transformator");
}
////////////////////////////////////////////////////////////////////////////////////////////
bool find=false;
for (int i = 0; i < 8; i++)
{
    for (int j = 0; j < 8; j++)
    {
        if (cells[i][j]->ptr != nullptr && cells[i][j]->ptr->GetTextureSprite().getGlobalBounds().contains(pos2))
        {
            find=true;
            if(cells[i][j]->ptr->GetColor()!=plturn)
            {
                e=(char)fabs((j+1)-9)+'0';
                switch (i+1)
                {
                    case 1:
                        d='a';
                        break;
                    case 2:
                        d='b';
                        break;
                    case 3:
                        d='c';
```

****

تابع Kingcheck()

این تابع چک می کند که کیش و مات رخ داده است یا ن

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
bool ChessBoard::KingCheck(char cColor)
{
    // Find the king
    int iKingRow;
    int iKingCol;
    for (int iRow = 0; iRow < 8; ++iRow)
    {
        for (int iCol = 0; iCol < 8; ++iCol)
        {
            if (!cells[iRow][iCol]->IsEmpty())
            {
                if (cells[iRow][iCol]->ptr->GetColor() == cColor)
                {
                    if (cells[iRow][iCol]->ptr->GetNamad() == 'K')
                    {
                        iKingRow = iRow;
                        iKingCol = iCol;
                    }
                }
            }
        }
    }
    // Run through the opponent's pieces and see if any can take the king
    for (int iRow = 0; iRow < 8; ++iRow)
    {
        for (int iCol = 0; iCol < 8; ++iCol)
        {
            if (!cells[iRow][iCol]->IsEmpty()) {
                if (cells[iRow][iCol]->ptr->GetColor() != cColor)
                {
                    if (cells[iRow][iCol]->ptr->AreSquaresLegal(iRow, iCol, iKingRow, iKingCol,cells))
                    {
                        return true;
                    }
                }
            }
        }
    }
    return false;
}
```

****

تابع gameover()

این تابع تمام حرکات مهره های خودی را بررسی کرده اگر حرکت هیچ مهره ای جلوی کیش شدن را نگیرد بازی gameover شده و بازیکن بازی را میبازد

```
ol ChessBoard::GameOver()

  // Run through all pieces
  for (int iRow = 0; iRow < 8; ++iRow)
  {
      for (int iCol = 0; iCol < 8; ++iCol)
      {
          // If it is a piece of the current player, see if it has a legal move
          if(!cells[iRow][iCol]->IsEmpty())
          {
              if (cells[iRow][iCol]->ptr->GetColor() == plturn )
              {
                  for (int iMoveRow = 0; iMoveRow < 8; ++iMoveRow)
                  {
                      for (int iMoveCol = 0; iMoveCol < 8; ++iMoveCol)
                      {
                          if (cells[iRow][iCol]->ptr->IsLegalMove(iRow, iCol, iMoveRow, iMoveCol, cells))
                          {
                              // Make move and check whether king is in check
                              Chessman* qpTemp=(cells[iMoveRow][iMoveCol]->ptr) ;
                              sf::Vector2i x=cells[iMoveRow][iMoveCol]->ptr->PosOnGrid;
                              cells[iMoveRow][iMoveCol]->SetNut(cells[iRow][iCol]->ptr);
                              cells[iMoveRow][iMoveCol]->ptr->PosOnGrid=cells[iRow][iCol]->ptr->PosOnGrid;
                              cells[iRow][iCol]->ptr = nullptr;

                              bool bCanMove = !KingCheck(plturn);
                              // Undo the move
                              cells[iRow][iCol]->SetNut(cells[iMoveRow][iMoveCol]->ptr);
                              cells[iRow][iCol]->ptr->PosOnGrid=cells[iMoveRow][iMoveCol]->ptr->PosOnGrid;
                              cells[iMoveRow][iMoveCol]->SetNut(qpTemp);
                              cells[iMoveRow][iMoveCol]->ptr->PosOnGrid=x;
                              if (bCanMove)
                              {
                                  return true;
                              }
                          }
                      }
                  }
              }
          }
      }
  }
  if(plturn=='w')
  {
      pl2->AddPoint(70);
  }
  else
  {
      pl1->AddPoint(70);
  }
  return false;
```

Chessman class

```
class Chessman
{
    protected:
        char color;//color of pieces
        char namad;//symbol of any pieces
        bool InGame=true;//showing that is this piece alive or not
        int point;//point for enemy for get  beated
        int warn;//point for enmey for make this piece in dangerous mode
        sf::Texture texture;
        sf::RenderWindow* window;
        sf::Sprite TextureSprite;

    public:
        Chessman(sf::Vector2i,sf::RenderWindow* ,char);
        Chessman();
        ~Chessman();
        sf::Vector2i PosOnGrid;
        bool IsLegalMove(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, Cell* [8][8]);//
        virtual bool AreSquaresLegal(int a,int  b,int c,int  d ,Cell* [8][8]) = 0;//based on ty
        //void SetParameter(int a,int,char);
        bool GetInGame();
        void SetInGame(bool);
        char GetNamad();
        char GetColor();
        int GetPoint();
        int GetWarn();
        bool DidMove;//is this piece move or not
        virtual void draw();
        virtual sf::Sprite& GetTextureSprite();
        sf::Vector2i GetPosOnGrid() ;
};
#endif /*Chessman*/
```

این کلاس مشخصات مهره ها را در خود دارد از جمله رنگ آن ها ، نوع آن ها و امتیاز آن ها و ...

هم چنین دارای یک سری توابع get و set میباشد که بعنوان مثال تابع get color در صورت فراوانی نوع رنگ مهره را بر می گرداند

```
void Chessman::draw()
{
    sf::Vector2f position;
    if(InGame)
    {
        position.x=(float)((PosOnGrid.x)*(125))+30
        position.y=(float)((PosOnGrid.y)*(125))+20
    }
    TextureSprite.setScale(sf::Vector2f(0.3,0.3));
    TextureSprite.setPosition(position);
    window->draw(TextureSprite);
}
```

تابع شاخص از سایر تابع ها در این کلاس ، تابع draw می باشد که با مشخص کردن پوزیشن هر مهره در صفحه بازی توسط تابع setposition , setscale در sfml ، در صفحه بازی نمایش داده می شوند.

****

حال به بررسی ویژگی های انواع مهره های بازی می پردازیم:

King – شاه

```
#ifndef KING_H
#define KING_H
#include "chessman.h"
class King:public Chessman
{
    public:
        King(sf::Vector2i,sf::RenderWindow *,char);
        King() {};
        virtual ~King();
        bool DidMove = false;
    protected:
        char namad='K';
        int warn=10;
        int point=50;
        bool AreSquaresLegal(int,int,int,int,Cell *[8][8]) override;
};


#endif
```

این کلاس از کلاس chessman ارث می برد

همان طور ک در تصویر نیز مشخص است امتیاز مربوط به این مهره ۵۰ است و نماد آن k است

مهره شاه بر اساس نوع رنگی که توسط سازنده این کلاس مشخص می شود ، پوزیشن خود را در صفحه پیدا میکند

هم چنین تابعی وجود دارد که مجاز بودن حرکت (از نظر خالی بودن مقصد) را بررسی می کند و در واقع اجازه حرکت داشتن یا نداشتن را برای شاه مشخص می کند.

پیاده سازی توابع این کلاس به صورت زیر است:

ers > Hamid Mvd > Desktop > advancedchess-main > King.cpp

```cpp
#include "King.h"
King::King(sf::Vector2i pos,sf::RenderWindow* s,char x):Chessman(pos,s,x)
{
    if(x=='w')
    {
        texture.loadFromFile("/Users/Taylor1989/Desktop/advancedchess/project/White/King.png");
    }
    else
    {
        texture.loadFromFile("/Users/Taylor1989/Desktop/advancedchess/project/Black/King.png");
    }
    TextureSprite.setTexture(texture);
}
bool King::AreSquaresLegal(int iSrcRow,int iSrcCol,int iDestRow,int iDestCol,Cell * cellBoard[8][8])
{
    int iRowDelta = iDestRow - iSrcRow;
    int iColDelta = iDestCol - iSrcCol;
    if (((iRowDelta >= -1) && (iRowDelta <= 1)) && ((iColDelta >= -1) && (iColDelta <= 1)))
    {
        DidMove=true;
        return true;
    }
    return false;
}
King::~King()
{

}
```

****

مهره وزیر _ Queen

```cpp
#include "chessman.h"
class Queen:public Chessman
{
    public:
        Queen(sf::Vector2i,sf::RenderWindow *,char);
        Queen();
        virtual ~Queen();
    protected:
        char namad='Q';
        int warn=5;
        int point=15;
        bool AreSquaresLegal(int,int,int,int,Cell *[8][8]) override;
};
```

وزیر نیز توابعی مشابه شاه دارد که نحوه پیاده سازی آن را در زیر ملاحظه می کنید :

```
#include "Queen.h"
Queen::Queen(sf::Vector2i pos,sf::RenderWindow *s,char x):Chessman(pos,s,x)
{
    if(x=='w')
    {
        texture.loadFromFile("/Users/Taylor1989/Desktop/advancedchess/project/White/Queen.png");
    }
    else
    {
        texture.loadFromFile("/Users/Taylor1989/Desktop/advancedchess/project/Black/Queen.png");
    }
    TextureSprite.setTexture(texture);
}
Queen::Queen() {}
bool Queen::AreSquaresLegal(int iSrcRow,int iSrcCol,int iDestRow,int iDestCol,Cell *cellBoard[8][8])
{
    if (iSrcRow == iDestRow)
    {
        // Make sure that all invervening squares are empty
        int iColOffset = (iDestCol - iSrcCol > 0) ? 1 : -1;
        for (int iCheckCol = iSrcCol + iColOffset; iCheckCol !=  iDestCol; iCheckCol = iCheckCol + iColOffset)
        {
            if (cellBoard[iSrcRow][iCheckCol] != 0)
            {
                return false;
            }
        }
        return true;
    }
    else if (iDestCol == iSrcCol)
    {
        // Make sure that all invervening squares are empty
        int iRowOffset = (iDestRow - iSrcRow > 0) ? 1 : -1;
        for (int iCheckRow = iSrcRow + iRowOffset; iCheckRow !=  iDestRow; iCheckRow = iCheckRow + iRowOffset)
        {
            if (cellBoard[iCheckRow][iSrcCol] != 0)
```

```
    {
        // Make sure that all invervening squares are empty
        int iRowOffset = (iDestRow - iSrcRow > 0) ? 1 : -1;
        for (int iCheckRow = iSrcRow + iRowOffset; iCheckRow !=  iDestRow; iCheckRow = iCheckRow + iRowOffset)
        {
            if (cellBoard[iCheckRow][iSrcCol] != 0)
            {
                return false;
            }
        }
        return true;
    }
    else if ((iDestCol - iSrcCol == iDestRow - iSrcRow) || (iDestCol - iSrcCol == iSrcRow - iDestRow))
    {
        // Make sure that all invervening squares are empty
        int iRowOffset = (iDestRow - iSrcRow > 0) ? 1 : -1;
        int iColOffset = (iDestCol - iSrcCol > 0) ? 1 : -1;
        int iCheckRow;
        int iCheckCol;
        for (iCheckRow = iSrcRow + iRowOffset, iCheckCol = iSrcCol + iColOffset; iCheckRow !=  iDestRow;iCheckRow = iCheckRow + iRowOffset, iCheckCol = iCheckCol + iColOffset)
        {
            if (cellBoard[iCheckRow][iCheckCol] != 0)
            {
                return false;
            }
        }
        return true;
    }
    return false;
}
```

****

Rook _ رخ

رخ که به صورت افقی و عمودی حرکت می کند و امتیاز آن ۸ است

کد رخ نیز به مانند شاه و وزیر بوده با این تفاوت که پیاده سازی aresquareslegal برای ان متفاوت است:

```
#ifndef ROOK_H
#define ROOK_H
#include "chessman.h"
class Rook:public Chessman
{
    public:
        Rook(sf::Vector2i,sf::RenderWindow *,char);
        Rook() {}
        virtual ~Rook();
        bool DidMove = false;
    protected:
        char namad='R';
        int warn=2;
        int point=8;
        bool AreSquaresLegal(int,int,int,int,Cell *[8][8]) override;
};
```

```
    if(x=='w')
    {
        texture.loadFromFile("Rook.png");
    }
    else
    {
        texture.loadFromFile("Rook.png");
    }
    TextureSprite.setTexture(texture);
}
bool Rook::AreSquaresLegal(int iSrcRow,int iSrcCol,int iDestRow,int iDestCol,Cell *cellBoard[8][8])
{
    if (iSrcRow == iDestRow)
    {
        // Make sure that all invervening squares are empty
        int iColOffset = (iDestCol - iSrcCol > 0) ? 1 : -1;
        for (int iCheckCol = iSrcCol + iColOffset; iCheckCol !=  iDestCol; iCheckCol = iCheckCol + iColOffset)
        {
            if (cellBoard[iSrcRow][iCheckCol] != 0)
            {
                return false;
            }
        }
        DidMove=true;
        return true;
    }
    else if (iDestCol == iSrcCol)
    {
        // Make sure that all invervening squares are empty
        int iRowOffset = (iDestRow - iSrcRow > 0) ? 1 : -1;
        for (int iCheckRow = iSrcRow + iRowOffset; iCheckRow !=  iDestRow; iCheckRow = iCheckRow + iRowOffset)
        {
            if (cellBoard[iCheckRow][iSrcCol] != 0)
            {
                return false;
            }
        }
        DidMove=true;
        return true;
    }
    return false;
}
```

****

فیل _ bishop

فیل که به صورت قطری حرکت می کند و امتیاز آن ۸ است

کد فیل نیز به مانند شاه و وزیر و رخ بوده با این تفاوت که پیاده سازی aresquareslegal برای ان متفاوت است:

```h
#ifndef BISHOP_H
#define BISHOP_H
#include "chessman.h"
class Bishop:public Chessman
{
    public:
        Bishop(sf::Vector2i,sf::RenderWindow*,char);
        Bishop() {}
        virtual ~Bishop();
    protected:
        char namad='B';
        int warn=2;
        int point=8;
        bool AreSquaresLegal(int,int,int,int,Cell *x[8][8]) override;
};
```

```cpp
#include "Bishop.h"
Bishop::Bishop(sf::Vector2i pos,sf::RenderWindow* s,char x):Chessman(pos,s,x)
{
    if(x=='w')
    {
        texture.loadFromFile("White_Bishop.png");
    }
    else
    {
        texture.loadFromFile("Black_Bishop.png");
    }
    TextureSprite.setTexture(texture);
}
bool Bishop::AreSquaresLegal(int iSrcRow,int iSrcCol,int iDestRow,int iDestCol,Cell *cellBoard[8][8])
{
    if ((iDestCol - iSrcCol == iDestRow - iSrcRow) || (iDestCol - iSrcCol == iSrcRow - iDestRow))
    {
        // Make sure that all invervening squares are empty
        int iRowOffset = (iDestRow - iSrcRow > 0) ? 1 : -1;
        int iColOffset = (iDestCol - iSrcCol > 0) ? 1 : -1;
        int iCheckRow;
        int iCheckCol;
        for (iCheckRow = iSrcRow + iRowOffset, iCheckCol = iSrcCol + iColOffset;iCheckRow !=  iDestRow;iCheckRow = iCheckRow + iRowOffset, iCheckCol = iCheckCol + iColOff
        {
            if (cellBoard[iCheckRow][iCheckCol] != 0)
            {
                return false;
            }
        }
        return true;
    }
    return false;
}
```

****

اسب_knight

فیل که به صورت L حرکت می کند و امتیاز آن ۸ است

کد اسب نیز به مانند شاه و وزیر و رخ و فیل بوده با این تفاوت که پیاده سازی aresquareslegal برای ان متفاوت است:

```
#ifndef KNIGHT_H
#define KNIGHT_H
#include "chessman.h"
class Knight:public Chessman
{
    public:
        Knight(sf::Vector2i,sf::RenderWindow *,char);
        Knight() {}
        virtual ~Knight();
    protected:
        char namad='H';
        int warn=2;
        int point=8;
        bool AreSquaresLegal(int,int,int,int,Cell *[8][8]) override;
};
```

```cpp
#include "Knight.h"
Knight::Knight(sf::Vector2i pos,sf::RenderWindow * s,char x):Chessman(pos,s,x)
{
    if(x=='w')
    {
        texture.loadFromFile("White/Knight.png");
    }
    else
    {
        texture.loadFromFile("Black/Knight.png");
    }
    TextureSprite.setTexture(texture);
}
bool Knight::AreSquaresLegal(int iSrcRow,int iSrcCol,int iDestRow,int iDestCol,Cell *cellBoard[8][8])
{
    // Destination square is unoccupied or occupied by opposite color
    if ((iSrcCol == iDestCol + 1) || (iSrcCol == iDestCol - 1))
    {
        if ((iSrcRow == iDestRow + 2) || (iSrcRow == iDestRow - 2))
        {
            return true;
        }
    }
    if ((iSrcCol == iDestCol + 2) || (iSrcCol == iDestCol - 2))
    {
        if ((iSrcRow == iDestRow + 1) || (iSrcRow == iDestRow - 1))
        {
            return true;
        }
    }
    return false;
}
Knight::~Knight()
{

}
```

****

سرباز_Pawn

سرباز یا مهره ی پیاده دارای حرکت ساده ی رو به جلو است. در هنگام اولین حرکت می تواند دو خانه بپیماید ولی در حرکات بعدی فقط حرکت یه خانه ای دارد . مهره سرباز برای کشتن مهره ی حریف به صورت قطری به سمت چپ جلو و یا سمت راست جلو می تواند حرکت کند . در صورتی که سرباز به اولین خانه حریف برسد ،

بازیکن میتواند سرباز را به وزیر یا فیل عوض کند.

سرباز نیز به مانند مهره های قبل دارای ویژگی ها و توابع اسمی مشترکی است با تفاوت در نحوه پیاده سازی که در ادامه مشاهده خواهیم کرد :

```
class Pawn:public Chessman
{
    public:
        Pawn(sf::Vector2i,sf::RenderWindow*,char);
        Pawn();
        virtual ~Pawn();
        bool DidMove = false;
    protected:
        char namad='P';
        int warn=1;
        int point=3;
        bool AreSquaresLegal(int,int,int,int,Cell *[8][8]) override;
};
```

```
#include "Pawn.h"
Pawn::Pawn(sf::Vector2i pos,sf::RenderWindow *s,char x):Chessman(pos,s,x)
{
    if(x=='w')
    {
        texture.loadFromFile("White/Pawn.png");
    }
    else
    {
        texture.loadFromFile("Black/Pawn.png");
    }
    TextureSprite.setTexture(texture);
}
bool Pawn::AreSquaresLegal(int iSrcRow,int iSrcCol,int iDestRow,int iDestCol,Cell *cellBoard[8][8])
{
    Cell* qpDest = cellBoard[iDestRow][iDestCol];
            if (qpDest == 0)
            {
                // Destination square is unoccupied
                if (iSrcCol == iDestCol)
                {
                    if (GetColor() == 'w')
                    {
                        if (iDestRow == iSrcRow + 1)
                        {
                            DidMove=true;
                            return true;
                        }
                        if (iDestRow == iSrcRow + 2 && !DidMove)
                        {
                            DidMove=true;
                            return true;
                        }
                    }
```

```
                    else
                    {
                        if (iDestRow == iSrcRow - 1)
                        {
                            DidMove=true;
                            return true;
                        }
                        if (iDestRow == iSrcRow - 2 && !DidMove)
                        {
                            DidMove=true;
                            return true;
                        }
                    }
                }
            }
            else
            {
                // Dest holds piece of opposite color
                if ((iSrcCol == iDestCol + 1) || (iSrcCol == iDestCol - 1))
                {
                    if (GetColor() == 'w')
                    {
                        if (iDestRow == iSrcRow + 1)
                        {
                            DidMove=true;
                            return true;
                        }
                    }
                    else
                    {
                        if (iDestRow == iSrcRow - 1)
                        {
                            DidMove=true;
                            return true;
                        }
                    }
                }
            }
            return false;
    }
```

****

حال که مهره ها و تخته بازی را بررسی کردیم ، به کلاس بازیکن می رسیم. ما برای سادگی کار دو کلاس player۱ و player۲ را تعریف میکنیم ولی ویژگی های یکسانی دارند این دو کلاس.

ما برای هر بازیکن ۱۶ مهره در نظر میگیریم که برای افزایش سرعت کار با آنها از نوع پوینتر و حافظه پویا استفاده کرده ایم . برای هر بازیکن نام، امتیازات مثبت و منفی ، یک وکتور از نوع chessman و یک پوینتر از نوع renderwindow جهت کار های گرافیکی در نظر گرفته ایم

توابع گت و ست برای نام بازیکن ، Addpoint,addNpoint,getpoint,getNpoint برای امتیازات مثبت و منفی برای این کلاس ها تعریف میکنیم. مهره ها را که برای هر بازیکن تعریف کردیم ، آن ها را به درون وکتور chessman پوش می کنیم. در پایان نیز حافظه های هر مهره را آزاد میکنیم!

```
class Player1
{
    private:
        int point;//posetive point of player
        short int negativepoint;//negetive point of player
        sf::RenderWindow * window;
    public:
        Player1(sf::RenderWindow *);
        void AddPoint(int);
        int GetPoint();
        void AddNPoint(int);
        short int GetNPoint();
        std::string lastmove;
        ///////////
        Queen * queen;
        King * king;
        Rook * rook1;
        Rook * rook2;
        Knight * knight1;
        Knight * knight2;
        Bishop * bishop1;
        Bishop *bishop2;
        Pawn * pawn1;
        Pawn * pawn2;
        Pawn * pawn3;
        Pawn * pawn4;
        Pawn * pawn5;
        Pawn * pawn6;
        Pawn * pawn7;
        Pawn * pawn8;
        ///////////
   //   std::vector <Chessman *>pieces;
        std::vector <std::string>stack;
        std::vector <Chessman*> graveyard;
        ~Player1();
};
```

```cpp
#include "Player1.h"
#include <string>
Player1::Player1(sf::RenderWindow * window)
{
    point=0;
    negativepoint=0;
    queen =new Queen(sf::Vector2i(4,0),window,'w');
    king=new King(sf::Vector2i(3,0),window,'w');
    rook1=new Rook(sf::Vector2i(0,0),window,'w');
    rook2=new Rook(sf::Vector2i(7,0),window,'w');
    knight1=new Knight(sf::Vector2i(1,0),window,'w');
    knight2=new Knight(sf::Vector2i(6,0),window,'w');
    bishop1=new Bishop(sf::Vector2i(2,0),window,'w');
    bishop2=new Bishop(sf::Vector2i(5,0),window,'w');
    pawn1=new Pawn(sf::Vector2i(0,1),window,'w');
    pawn2=new Pawn(sf::Vector2i(1,1),window,'w');
    pawn3=new Pawn(sf::Vector2i(2,1),window,'w');
    pawn4=new Pawn(sf::Vector2i(3,1),window,'w');
    pawn5=new Pawn(sf::Vector2i(4,1),window,'w');
    pawn6=new Pawn(sf::Vector2i(5,1),window,'w');
    pawn7=new Pawn(sf::Vector2i(6,1),window,'w');
    pawn8=new Pawn(sf::Vector2i(7,1),window,'w');
    this->window=window;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Player1::AddPoint(int a)
{
    point+=a;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int Player1::GetPoint()
{
    return point;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Player1::AddNPoint(int a)
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Player1::AddNPoint(int a)
{
    negativepoint+=a;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
short int Player1::GetNPoint()
{
    return negativepoint;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
Player1::~Player1()
{
    delete king;
    delete queen;
    delete rook1;
    delete rook2;
    delete knight1;
    delete knight2;
    delete bishop1;
    delete bishop2;
    delete pawn1;
    delete pawn2;
    delete pawn3;
    delete pawn4;
    delete pawn5;
    delete pawn6;
    delete pawn7;
    delete pawn8;
}


////////////////////////////////////////////////////////////////////
```

منابع

*YouTube

*www.sfml-dev.org

*cpp reference

*deitel and deitel

*stackoverflow

*github