## 目录



# 获取可能落子的位置

## 在对应文件中加入代码

### Layout.h

```
public:
    bool IsHasPieces(POS pos); //判断对应的位置是否有落子 (黑子或白子)

protected:
    vector<POS> _allMaybePos; //记录所有可以落子的位置
    void GetAllMaybePos();//获取所有可以落子的位置
    bool IsMaybePos(POS pos); //判断是否可以落子
    bool IsLegalPos(POS pos); //判断位置是否合法
```

## Layout.cpp

```
/*
判断对应的位置是否有落子 (黑子或白子)
*/
bool Layout::IsHasPieces(POS pos)
{
    return _layout[pos.row][pos.col] == CP_NOTHING ? false : true;
}

/*
判断位置是否合法
*/
bool Layout::IsLegalPos(POS pos)
{
    if (pos.row >= 0 && pos.row < Game::_gridCount &&
        pos.col >= 0 && pos.col < Game::_gridCount)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

```
/*
功能描述：判断所给的pos位置是否是一个可落子位置
函数名  ：IsMaybePos
输入    ：pos，所判断的位置
输出    ：bret，为真pos位置可落子，为假pos位置不可落子
外部数据：_layout, _player
外部函数：
作者    ：雷一鸣
时间    ：2017-12-26
版本    ：ver 0.1
-----------
修改    ：
版本    ：
```

```
 */
bool Layout::IsMaybePos(POS pos)
{
    bool bret = false;
    POS nextPos;

    for (int row = -1; row <= 1; ++row)
    {
        for (int col = -1; col <= 1; ++col)
        {
            //代表是自己的位置
            if (row == 0 && col == 0)
            {
                continue;
            }

            //设置下一次判断的位置 （8个方向中的一个）
            nextPos.row = pos.row + row;
            nextPos.col = pos.col + col;

            //如果位置合法，并且布局中对方的落子
            if (IsLegalPos(nextPos) && _layout[nextPos.row][nextPos.col] ==
(CHESS_PIECE)(3 - _player))
            {
                for (nextPos.row += row, nextPos.col += col;
                    IsLegalPos(nextPos);
                    nextPos.row += row, nextPos.col += col)
                {
                    //如果是对方的棋子，则继续
                    if (_layout[nextPos.row][nextPos.col] ==
                        (CHESS_PIECE)(3 - _player))
                    {
                        continue;
                    }
                    else if (_layout[nextPos.row][nextPos.col] == _player)
                    {//如果是己方的落子则返回
                        return true;
                    }
                    else
                    {//如果无落子则结束循环
                        break;
                    }
                }
            }
```

```
        }
    }
    return false;
}



/*
获取所有可以落子的位置
*/
void Layout::GetAllMaybePos()
{
    //清空容器
    _allMaybePos.clear();

    POS pos;

    for (pos.row = 0; pos.row < Game::_gridCount; ++pos.row)
    {
        for (pos.col = 0; pos.col < Game::_gridCount; ++pos.col)
        {
            if (!IsHasPieces(pos) && IsMaybePos(pos))
            {
                //存入链表
                _allMaybePos.push_back(pos);
            }
        }
    }

}
```

## 修改对应文件的代码

## 修改绘制 Layout::Draw 函数的代码

```
/*
落子绘制
*/
```

```
void Layout::Draw()
{
    int x, y;

    for (int i = 0; i<Game::_gridCount; ++i)
    {
        for (int j = 0; j<Game::_gridCount; ++j)
        {
            if (_layout[i][j] == CP_BLACK)
            {
                //设置填充色为黑色
                setfillcolor(EGERGB(0, 0, 0));
            }
            else if (_layout[i][j] == CP_WHITE)
            {
                //设置填充色为白色
                setfillcolor(EGERGB(255, 255, 255));
            }
            else
            {
                continue;
            }

            //计算圆心的位置
            x = Game::_xBegin + j * Game::_gridSize + Game::_gridSize / 2;
            y = Game::_yBegin + i * Game::_gridSize + Game::_gridSize / 2;

            //绘制棋子
            fillellipse(x, y, Game::_pieceSize, Game::_pieceSize);
        }
    }


    setfillcolor(EGERGB(255, 255, 0));
    for (int i = 0; i<_allMaybePos.size(); ++i)
    {
        //计算圆心的位置
        y = Game::_yBegin + _allMaybePos[i].row * Game::_gridSize +
Game::_gridSize / 2;
        x = Game::_xBegin + _allMaybePos[i].col * Game::_gridSize +
Game::_gridSize / 2;

        //绘制所有可以落子的位置
        fillellipse(x, y, 5, 5);
```

```
        }

}
```

## 修改绘制 Layout ::Layout()函数的代码

```
Layout::Layout()
{
    //给存放棋子的容器分配内存
    _layout.resize(Game::_gridCount);
    for (int i = 0; i<Game::_gridCount; ++i)
    {
        _layout[i].resize(Game::_gridCount);
    }

    //初始化棋盘格子中的对应位置的内容
    for (int i = 0; i<Game::_gridCount; ++i)
    {
        for (int j = 0; j<Game::_gridCount; ++j)
        {
            _layout[i][j] = CP_NOTHING;
        }
    }

    //设置对应的位置是白色棋子
    _layout[3][3] = _layout[4][4] = CP_WHITE;
    //设置对应的位置是黑色棋子
    _layout[3][4] = _layout[4][3] = CP_BLACK;

    //黑子先手
    _player = CP_BLACK;

    GetAllMaybePos();
}
```

# 运行效果