c++ 程序设计报告

计算机学院：刘健琛

2021 年 4 月 29 日

目录

1 作业题目

用 qt 实现黑白棋游戏。

2 开发环境

- qt creator 4.6.2
- MINGW 5.3.0 32bit

3 实现功能

- 8×8 的棋盘与黑白棋子基本实现。
- 黑白棋游戏规则与胜负判断函数的实现。
- 两人单机对战，左右互搏的下棋方法实现。
- 基于 minimax 剪枝算法的人工智能对弈方法实现。
- qmessagebox 的弹出窗口介绍。

4 游戏规则

- 棋局开始时黑棋位于 e4 和 d5，白棋位于 d4 和 e5。
- 黑方先行，双方交替下棋。
- 一步合法的棋步包括：在一个空格新落下一个棋子，并且翻转对手一个或多个棋子。
- 新落下的棋子与棋盘上已有的同色棋子间，对方被夹住的所有棋子都要翻转过来。可以是横着夹，竖着夹，或是斜着夹。夹住的位置上必须全部是对手的棋子，不能有空格。
- 一步棋可以在数个方向上翻棋，任何被夹住的棋子都必须被翻转过来，棋手无权选择不去翻某个棋子。
- 除非至少翻转了对手的一个棋子，否则就不能落子。如果一方没有合法棋步，也就是说不管他下到哪里，都不能至少翻转对手的一个棋子，那他这一轮只能弃权，而由他的对手继续落子直到他有合法棋步可下。
- 如果一方至少有一步合法棋步可下，他就必须落子，不得弃权。
- 棋局持续下去，直到棋盘填满或者双方都无合法棋步可下。

以上游戏规则同时被写在我的个人主页上，点击游戏中的开始按钮后会跳出窗口跳转到我的个人主页。

5 基本程序代码的实现

5.1 画棋盘

```
1
2         void othello::paintEvent(QPaintEvent *event){
3          this->resize(600,400);
4          QPainter painter(this);
5          painter.setRenderHint(QPainter::SmoothPixmapTransform,true);
6          painter.drawPixmap(0,0,400,400,background);
7          int c;
8          if(turn=="b"){
9              c=Whitechess;
10         }
11         else{
12             c=Blackchess;
13         }
14         int potential[8][8];
15         for(int i=0;i<8;i++){
16             for(int j=0;j<8;j++){
17                 potential[i][j]=0;
18             }
19         }
20         if(isGameOver(mainBoard)&&nom){
21             QMessageBox msgBox;
22             if(report(mainBoard,Whitechess)>report(mainBoard,Blackchess))
23                 msgBox.setText("white win.");
24             else
25                 msgBox.setText("black win.");
26             msgBox.exec();
27             nom=false;
28         }
29        vector<posi> nodes =getMove(mainBoard,c);
30        for(int i=0;i<nodes.size();i++){
```

```
            int x=nodes[i].x;
            int y=nodes[i].y;
            potential[x][y]=c;
        }
        for(int i=0;i<8;i++){
            for(int j=0;j<8;j++){
                if(mainBoard->get(i,j)==Whitechess){
                    painter.drawPixmap(0+50*i,0+50*j,50,50,white);
                }
                else if(mainBoard->get(i,j)==Blackchess){
                    painter.drawPixmap(0+50*i,0+50*j,50,50,black);
                }
                if(potential[i][j]==Blackchess){
                    painter.drawPixmap(0+50*i,0+50*j,50,50,hintblack);
                }
                if(potential[i][j]==Whitechess){
                    painter.drawPixmap(0+50*i,0+50*j,50,50,hintwhite);
                }
            }
        }
    }
```

通过一个有 8×8 的数组的棋盘类来存储当前棋盘上的棋子，用下载好的棋盘图片和自己画的棋子图片通过 pixmap 来展示出来。这里可走棋子用圆圈来表示，已走的棋子用黑白实心棋子表现。

5.2 鼠标点击实现下棋

```
    mousex=e->x()/50;
        mousey=e->y()/50;
        if(turn.compare("a")==0&&isGameOver(mainBoard)!=true&&!gameOver){
            if(makeMove(mainBoard,playera,mousex,mousey)==true){
                Last.x=mousex;
                Last.y=mousey;
```

```
7                 if(getMove(mainBoard,playerb).size()!=0){
8                     turn="b";
9                 }
10            }
```

5.3 可走棋子判定函数

```
1  vector<posi> othello::isValid(Board *board, int tile, int col, int row){
2      vector<posi> tilesToFlip;
3          if (isOnBoard(col, row) == false || board->get(col, row) != 0) {
4              return tilesToFlip;
5          }
6          int othertile;
7          board->set(col, row, tile);
8
9          if (tile == Blackchess) {
10             othertile = Whitechess;
11         }
12         else {
13             othertile = Blackchess;
14         }
15         for (int i = 0; i < 8; i++) {
16             int x = col;
17             int y = row;
18             int xdirection = Direction[i][0];
19             int ydirection = Direction[i][1];
20             x += xdirection;
21             y += ydirection;
22             if (isOnBoard(x, y) && board->get(x, y) == othertile) {
23                 x += xdirection;
24                 y += ydirection;
25                 if (isOnBoard(x, y) == false) {
26                     continue;
```

```
27                  }
28                  while (board->get(x, y) == othertile) {
29                      x += xdirection;
30                      y += ydirection;
31                      if (isOnBoard(x, y) == false) {
32                          break;
33                      }
34                  }
35                  if (isOnBoard(x, y) == false) {
36                      continue;
37                  }
38                  if (board->get(x, y) == tile) {
39                      while (true) {
40                          x -= xdirection;
41                          y -= ydirection;
42                          if (x == col && y == row) {
43                              break;
44                          }
45                          tilesToFlip.push_back(posi(x, y));
46                      }
47                  }
48              }
49          }
50          board->set(col, row, 0);
51          return tilesToFlip;
52  }
```

通过规则已知这里只有能够将对方棋子反转的下法才可以落子，那么我们就通过向八个方向遍历看能否用两个同色棋子将对方连续棋子夹住。

5.4 minimax 方法

```
1  posi othello::max(Board *mb, int depth, int alpha, int beta, int tile,int ha
2      int best = -10000;
```

```
3       Board *nm=new Board();
4       posi move ;
5       for(int i=0;i<8;i++){
6           for(int j=0;j<8;j++){
7               nm->set(i,j,mb->get(i,j));
8           }
9       }
10      vector<posi> gm=getMove(mb,tile);
11      if(depth==0){
12          for(int i=0;i<getMove(mb,tile).size();i++){
13              nm->set(gm[i].x,gm[i].y,tile);
14              if(evaluate(nm,hard)>best){
15                  best=evaluate(nm,hard);move=gm[i];
16              }
17              nm->set(gm[i].x,gm[i].y,0);
18          }
19          return move;
20      }
21      if(gm.size()==0){
22          return move;
23      }
24      for (int i = 0; i < getMove(mb,tile).size(); i++) {
25          alpha = best>alpha?best:alpha;
26          if(alpha >= beta){
27              break;
28          }
29          nm->set(gm[i].x,gm[i].y,tile);
30          posi next;
31          next=min(nm, depth - 1,alpha, beta, -tile, hard);
32          nm->set(next.x,next.y,-tile);
33          int value = evaluate(nm,hard);
34          if (value < best) {
35            best = value;
```

```
36              move = gm[i];
37            }
38            nm->set(next.x,next.y,0);
39            nm->set(gm[i].x, gm[i].y,0);
40          }
41          return move;
42  }
```

minimax 方法有 min 函数和 max 函数，对当前局面进行评估，这里截取 max 函数，该函数的逻辑是要让落子之后，对方的 min 取值只能最大。而 min 函数同理，通过两边对抗搜索，来寻找对各自最好的情况。搜索多层就可以得到期望的更优选择。

这里的评估函数用一个 8×8 的矩阵加权算得，在这里黑棋的赋值是 1，白棋的赋值是-1（由此一来黑棋用 max 求优，白棋用 min）。

而难度可以由搜索层数和评估矩阵的合理性来决定。

6 游戏说明

在开始玩之前在 qbuttongroup 中选择是 ai 模式还是人人模式，可以在 lineedit 中输入难度，然后点击 start 按钮，跳出窗口介绍规则。此时 start 按钮变成 reset 按钮，再按它就会恢复棋盘。

这之后就可以开始下了，如果是人人模式，可以找一个同学一起下。你在棋盘上选择一个被黑/白圆圈圈出来的棋盘格子点击就可以落子了（如果没有圈圈代表你这一轮没得走），然后可以请你的同学来接着下另一颜色的棋子。双方轮流下直到游戏结束。

如果是人机模式，你默认了先下的黑棋。在你点击完黑棋之后，电脑会自动下出白棋的落子，两方互相下棋，直到游戏结束。

7 收获

通过此次作业，了解了 qt 中的各个控件和使用方法，比如 lineedit，qmessagebox，qlabel，qpixmap 等。

同时，对 minimax 对抗搜索方法有了一定的了解，在此前的作业中了解了蒙特卡洛树搜索，这次作业对 ai 的对抗搜索有了更深的理解。