同化棋UI设计文档

朱可仁
naeioi@hotmail.com
naeioi.github.com

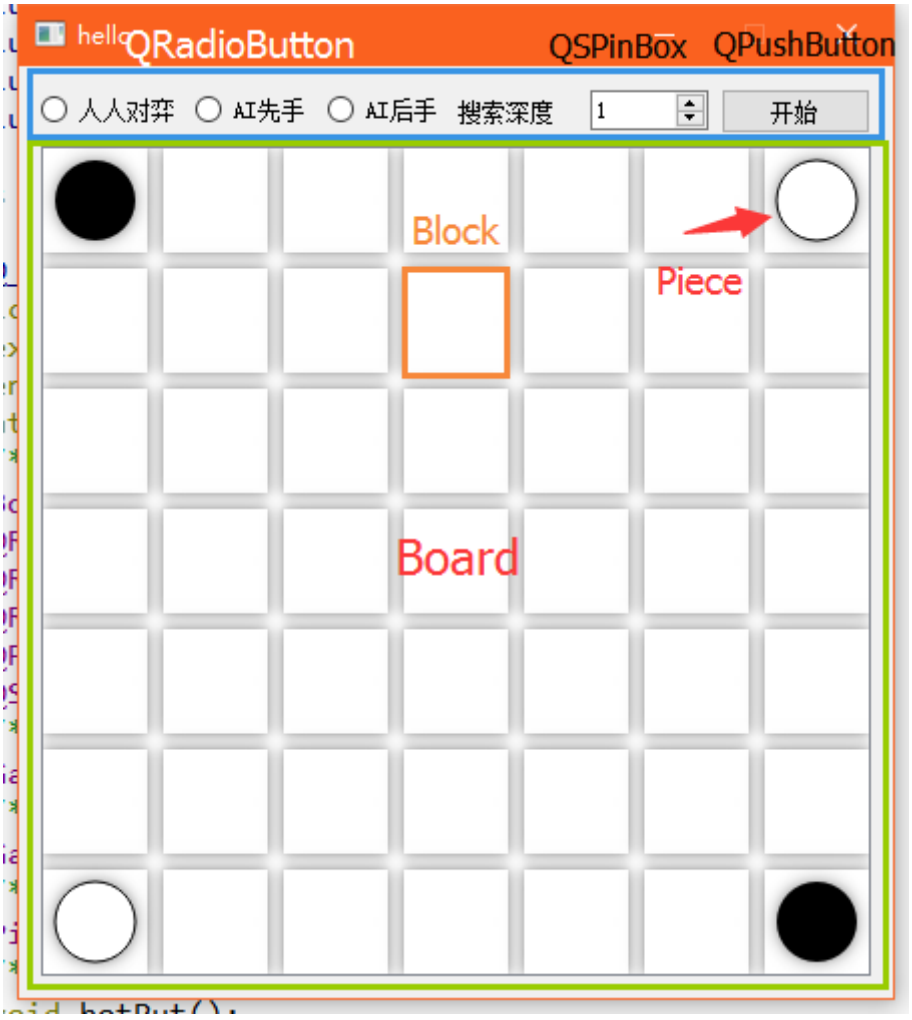
同化棋UI使用 `Qt` 编写。 `Qt` 是一个跨平台的UI框架（wiki链接）。下面就UI的各个组成部分做说明。

文件说明

`release` 目录下是可执行文件，使用 `windeplotqt` 工具生成，可在没有安装Qt的系统上运行。 `src` 是源代码，使用Qt Creator可以打开 `.pro` 工程文件修改。

`GameWrapper` 类

继承自 `QWidget` ，是用于显示的主模块。视图上， `QWidget` 由上下两部分组成。上部是控制区，包含三个 `QRadioButton` 单选选项，一个 `QSpinBox` 搜索深度控制，一个 `QPushButton` 开始游戏按钮。下部是一个 `Board` 类，用来显示棋盘和上面的棋子。



控制区的控件横向排布，因此放在一个表示横向排布的 `QHBoxLayout` 中

```
/* gamewrapper.cpp, GameWrapper::GameWrapper(QWidget *parent) */
QHBoxLayout *loTop = new QHBoxLayout;
/* ... */
loTop->addWidget(manualRadio);  /* 人人对弈Radio */
loTop->addWidget(botFirstRadio); /* AI先手Radio */
loTop->addWidget(userFirstRadio); /* AI后手Radio */
loTop->addWidget(label); /* "搜索深度"标签 */
loTop->addWidget(depthSpin); /* 搜索深度滚动空间 */
loTop->addWidget(startBtn); /* “开始”按钮 */
```

棋盘 `Board` 跟空间区呈纵向排布关系，因此放在一个表示纵向排布的 `QVBoxLayout` 中

```
/* gamewrapper.cpp, GameWrapper::GameWrapper(QWidget *parent) */
QVBoxLayout *layout = new QVBoxLayout;
layout->addLayout(loTop);
layout->addWidget(board.View());
```

`GameWrapper` 类中有两个 `Slot` （Qt中 `Slot` 的概念）。 `startBtnClick()` 用来处理“开始”按钮的按下， `userPut(r, c, nr, nc)` 用来接收 `board` 传来的用户落子信息，表示用户通过在 `board` 上点击棋盘将(r, c)处的棋落到了(nr, nc)处。

```
/* gamewrapper.h, class GameWrapper */
public slots:
    void userPut(int r, int c, int nr, int nc);
private slots:
    void startBtnClick();
```

在 `GameWrapper` 的构造函数中使用 `connect` 将信号绑定到这两个 `slot` 上

```
/* gamewrapper.cpp, GameWrapper::GameWrapper(QWidget *parent) */
connect(startBtn, SIGNAL(clicked(bool)), this, SLOT(startBtnClick()));
connect(&board,SIGNAL(userPut(int,int,int,int)),this,SLOT(userPut(int,int,int
,int)));
```

Board 类

`Board` 类是一个棋盘，是棋盘格 `Block` 和棋子 `Piece` 的容器，同时负责响应用户落子操作。对棋盘上的每个点，用 `g[i][j]` 表示它的棋子类型， `blocks[i][j]` 表示棋盘格， `pieces[i][j]` 表示棋子。对于没有落子的点，也放置一个不可见的 `Piece` 。此外，将所有的 `Piece` 和 `Block` 的点击信号绑定到 `blockClick(int, int)` 这个 `slot` 上，用来处理用户落子操作。

```
/* Board.cpp, Board::Board(QObject *parent) */
/* 放好所有棋，只有4颗可见 */
for(int i = 0; i < N; i++)
    for(int j = 0; j < M; j++) {
        blocks[i][j] = new Block(i, j);
        scene.addItem(blocks[i][j]);
        connect(blocks[i][j], SIGNAL(blockClick(int,int)), this, SLOT(blo
ckClick(int,int)));

        pieces[i][j] = new Piece(None, i, j);
        scene.addItem(pieces[i][j]);
        connect(pieces[i][j], SIGNAL(pieceClick(int,int)), this, SLOT(blo
ckClick(int,int)));
    }

/* 边角的4颗棋 */
g[0][0] = g[6][6] = Black;
g[0][6] = g[6][0] = White;

pieces[0][0]->setType(Black);
pieces[0][6]->setType(White);
pieces[6][0]->setType(White);
pieces[6][6]->setType(Black);
```

Block 类

Block 类表示棋盘格，继承自用来表示长方体的 QGraphicsRectItem 。在 Block 类的构造函数中，先设置好自己在棋盘上的坐标 (r, c) ，再设置好自己的位置和大小。

```
Block::Block(int r, int c): r(r), c(c)
{
    setAcceptHoverEvents(true);
    QPoint pos = rc2pos(r, c);
    this->setRect(pos.x(), pos.y(), width, width);
    this->setGraphicsEffect(genShadow());
    this->setBrush(whiteBrush);
    this->setPen(Qt::NoPen);
}
```

Blocks 类定义了 blockClick(r, c) 信号，用于向 Board 传递用户点击 (r, c) 的信号。当发生鼠标点击事件时发送信号。

```
void Block::mousePressEvent(QGraphicsSceneMouseEvent *event) {
    emit blockClick(r, c);
}
```

Piece 类

`Piece` 类用于绘制棋子。 `Piece` 类使用 `pieceType` 表示棋子类型（黑/白）， `lifted` 表示棋子是否被拿起。 `Piece` 类定义了 `pieceClick(r, c)` 信号，跟 `blockClick(r, c)` 信号的作用是一致的。

`ChessStyle` 类

这个类定义了黑白笔刷 `QBrush` 用来填颜色、两个笔 `QPen` 用来描边。 `Block` 和 `Piece` 公用它们显示不同的样式，达到节省内存的目的。这个类还定义了一个 `genShadow()` 生成阴影，因为经过实验，阴影不能公用。